Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
FG Komplexitätstheorie und Effiziente Algorithmen

TECHNISCHE UNIVERSITÄT
ILMENAU

On Statistical Data Compression

DISSERTATION

von Dipl.-Ing. Christopher Mattern,
geboren am 15. März 1986 in Erfurt,

zum Erlangen des akademischen Grades

DOCTOR RERUM NATURALIUM (DR. RER. NAT.),

eingereicht am 22. September 2015
an der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau.

Gutachter:

1. Univ.-Prof. Dr. Martin Dietzfelbinger, TU-Ilmenau;
2. Jan Østergaard, Associate Professor, Aalborg University;
3. Joel Veness, PhD, Google DeepMind.

Die Verteidigung fand am 18. Januar 2016 statt.

urn:nbn:de:gbv:ilm1-2016000010

Abstract

The ongoing evolution of hardware leads to a steady increase in the amount of data that is processed, transmitted and stored. Data compression is an essential tool to keep the amount of data manageable. Furthermore, techniques from data compression have many more applications beyond compression, for instance data clustering, classification and time series prediction.

In terms of empirical performance statistical data compression algorithms rank among the best. A statistical data compressor processes an input text letter by letter and performs compression in two stages — modeling and coding. During modeling a model estimates a probability distribution on the next letter based on the past input. During coding an encoder translates this probability distribution and the next letter into a codeword. Decoding reverts this process. Note that the model is exchangeable and its actual choice determines a statistical data compression algorithm. All major models use a mixer to combine multiple simple probability estimators, so-called elementary models.

In statistical data compression there is an increasing gap between theory and practice. On the one hand, the "theoretician's approach" puts emphasis on models that allow for a mathematical code length analysis to evaluate their performance, but neglects running time and space considerations and empirical improvements. On the other hand the "practitioner's approach" focuses on the very reverse. The family of PAQ statistical compressors demonstrated the superiority of the "practitioner's approach" in terms of empirical compression rates.

With this thesis we attempt to bridge the aforementioned gap between theory and practice with special focus on PAQ. To achieve this we apply the theoretician's tools to practitioner's approaches: We provide a code length analysis for several common and practical modeling and mixing techniques. The analysis covers modeling by relative frequencies with frequency discount and modeling by exponential smoothing of probabilities. For mixing we consider linear and geometrically weighted averaging of probabilities with Online Gradient Descent for weight estimation. Our results show that the models and mixers we consider perform nearly as well as idealized competitors that may adapt to the input. Experiments support our analysis. Moreover, our results add a theoretical justification to modeling and mixing from PAQ and generalize methods from PAQ. Ultimately, we propose and analyze Context Tree Mixing (CTM), a generalization of Context Tree Weighting (CTW). We couple CTM with modeling and mixing techniques from PAQ and obtain a theoretically sound compression algorithm that improves over CTW, as shown in experiments.

Zusammenfassung

Im Zuge der stetigen Weiterentwicklung moderner Rechentechnik wächst auch die Menge an zu verarbeitenden Daten. Es gilt diese Datenmengen zu verwalten, zu übertragen und zu speichern. Dafür ist Datenkompression unerlässlich. Techniken aus der Datenkompression kommen auch in anderen Bereichen zum Einsatz, z. B. beim Klassifizieren und Clustern von Daten oder in der Zeitreihenvorhersage.

Gemessen an empirischen Kompressionsraten zählen Statistische Datenkompressionsalgorithmen zu den Besten. Statistische Datenkompressionsalgorithmen verarbeiten einen Eingabetext buchstabenweise. Dabei verfährt man für jeden Buchstaben in zwei Phasen — Modellierung und Kodierung. Während der Modellierung schätzt ein Modell, basierend auf dem bereits bekannten Text, eine Wahrscheinlichkeitsverteilung für den nächsten Buchstaben. Ein Kodierer überführt die Wahrscheinlichkeitsverteilung und den Buchstaben in ein Codewort. Umgekehrt ermittelt der Dekodierer aus der Wahrscheinlichkeitsverteilung und dem Codewort den kodierten Buchstaben. Die Wahl des Modells bestimmt im wesentlichen den statistischen Datenkompressionsalgorithmus, das Modell ist also von zentraler Bedeutung. Ein Modell mischt typischerweise viele einfache Wahrscheinlichkeitsschätzer.

In der statistischen Datenkompression driften Theorie und Praxis auseinander. Theoretiker legen Wert auf Modelle, die eine mathematische Codelängenanalyse zulassen, vernachlässigen aber Laufzeit, Speicherbedarf und empirische Verbesserungen; Praktiker verfolgen den gegenteiligen Ansatz. Die PAQ-Algorithmen haben eindrucksvoll die Überlegenheit des praktischen Ansatzes verdeutlicht.

Diese Arbeit soll Theorie und Praxis annähren. Dazu wird das Handwerkszeug des Theoretikers, die Codelängenanlyse, auf Algorithmen des Praktikers angewendet. In dieser Arbeit werden Wahrscheinlichkeitsschätzer, basierend auf gealterten relativen Häufigkeiten und basierend auf exponentiell geglätteten Wahrscheinlichkeiten, analysiert. Weitere Analysen decken Methoden ab, die Wahrscheinlichkeitsverteilungen durch gewichtetes arithmetisches und geometrisches Mitteln mischen und Gewichte mittels Gradientenverfahren bestimmen. Die Analysen zeigen, dass sich die betrachteten Verfahren ähnlich gut wie idealisierte adaptive Vergleichsverfahren verhalten. Methoden aus PAQ werden durch die Ergebnisse dieser Arbeit erweitert und mit einer theoretischen Basis versehen. Experimente stützen die Analyseergebnisse. Ein weiterer Beitrag dieser Arbeit ist Context Tree Mixing (CTM), eine Verallgemeinerung von Context Tree Weighting (CTW). Durch die Kombination von CTM mit Methoden aus PAQ entsteht ein theoretisch fundierter Kompressionsalgorithmus, der in Experimenten besser als CTW komprimiert.

Acknowledgement

First of all I would like to thank my advisor, Prof. Martin Dietzfelbinger. He assisted me by proof-reading various papers, by technical discussions and he helped me to simplify some of the proofs found in this thesis. I would also like to thank the reviewers, Jan Østergaard and Joel Veness, for their comments on this work and helpful discussions. Not only my advisor chew over and discussed my work, but also my colleagues, thanks to you Martin Aumüller, Sascha Grau and Michael Rink (listed alphabetically).

Also there are many other people who have not directly been involved in my work, but still contributed in some way. So I would like to thank my wife, Anne Mattern, who beared with me while writing this thesis. Moreover, I would like to thank my mother, Bärbel Mattern, and my grandparents, Anne-Marie Haag and Hans Haag. Your support allowed me to graduate, so I was able to become a PhD candidate.

— *Thank you, Christopher*

Contents

CHAPTER 1

Introduction

1.1 Background

Driven by the evolution of hardware over the last decades the amount of data that is processed, transmitted and stored is steadily growing. This trend is likely to continue. To keep the amount of data manageable, especially in bottleneck situations, such as the distribution of documents, software, images, videos, . . . over the Internet, a reduction of the amount of data is essential: We can save resources, time and money. On a computer we typically represent data, or a data stream, as a sequence of fixed-size letters (bits, bytes, . . .). A reduction of such a data stream, i. e. the process of mapping a data stream to a preferably smaller data stream is called *data compression*. Given the compressed version of a data stream, we normally want to restore the original data stream without losing any information, which leads to *lossless data compression*. For image, audio and video compression we can discard information that isn't important for human perception, leading to lossy data compression; but even such techniques employ lossless compression at some stage.

In this work we concentrate on recent developments in *statistical data compression*, a particularly efficient class of lossless data compression methods. Besides statistical data compression, *dictionary-based data compression* and *transform-based data compression* are the other two major lossless data compression techniques.[1] The simple yet powerful idea behind statistical data compression is to encode frequent letters in fewer bits than infrequent letters. Similarly, in dictionary-based data compression we substitute words with shorter references to a dictionary. Lempel-Ziv-based algorithms are the most prominent examples. Transform-based data compression is rather different: A sequence of transforms, for instance the Burrows Wheeler Transform or Sort Transform followed by Move To Front encoding, maps the in-

[1]General information on lossless (statistical) data compression given in this chapter can be found in standard textbooks, such as [68, 80, 81], so we only give citations beyond these basics.

put data stream to a representation that is easy to compress. To actually achieve compression we apply another lossless data compression technique, typically statistical data compression, to the transformed output. It is common wisdom that empirically top-performing compression algorithms fall into the category of statistical data compression, whereas dictionary- and transform-based data compression is empirically inferior, but also less demanding in terms of processing power and memory. Since hardware limitations are likely to become less severe in the future, statistical data compression will become more and more popular. This trend is already observable: Nowadays, in 2015, compression software for every day use, like WinZIP [2] and 7-Zip [1] already employs statistical data compression (more precisely variants of [84]), which was unimaginable in the 1990s, due to relatively restrictive hardware.

At this point we want to emphasize that data compression has a wider range of application beyond bare compression, examples include: The creation of natural language trees from a text translated to different languages and the inference of evolutionary trees from genomes [21, 51, 93], image recognition [21, 24], the clustering of music by similarity [20, 21], the classification of articles (text) [32], spam filtering [13], hardware branch prediction [19], stock market prediction [6], time series prediction [47, 76] and policy evaluation [90]. (A good overview of applications of data compression beyond compression can be found in [21].) So (statistical) data compression is applicable in clustering, classification and forecasting and improvements in data compression are likely to translate into improvements in various fields of application.

1.2 Crash Course – Statistical Data Compression

We will now take a closer look at the principles of statistical data compression, to be able to clearly outline the goals of this work. In statistical data compression we process the input data stream $x_1x_2\ldots x_n$ (the letters are supposed to be from a finite alphabet) letter by letter and perform compression for each letter in two stages: *Modeling* and *coding*, see Figure 1.1. Let us now consider the procedure for a single letter x_t, where $1 \leqslant t \leqslant n$. During modeling a statistical model predicts a probability distribution p on the next letter x_t, given $x_1x_2\ldots x_{t-1}$, the preceding, already processed, portion of input; during coding a coder computes a codeword, typically a binary word, for the next letter x_t. Thereby, the length of a codeword for a letter depends on the probability which the model estimated for that letter, highly probable letters correspond to short codewords, less probable letters receive long code-

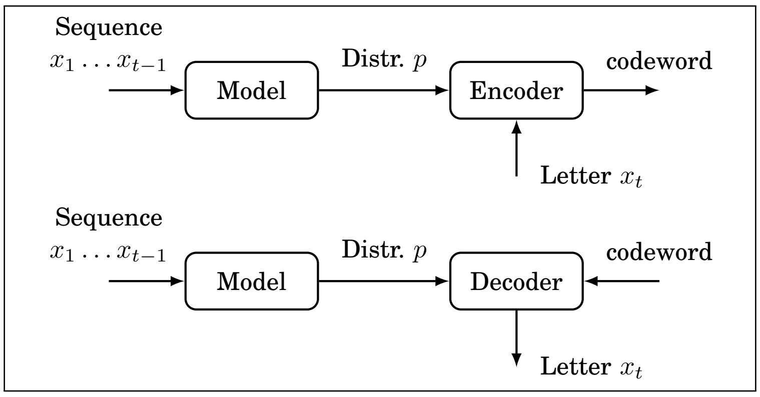


Figure 1.1: The procedure of statistical data compression during compression (top) and decompression (bottom).

words. Arithmetic Coding is the de-facto standard coder, since the implied codeword length is very close to optimal, i. e. $-\log p(x_t)$. Decompression is the very reverse. Based on the already decompressed sequence $x_1x_2 \dots x_{t-1}$ of letters the model, which is identical to the model from compression, predicts a probability distribution p on the next letter. The decoder examines the compressed data stream and uses the estimated probability distribution to decode the next codeword. As a result we obtain the next letter x_t of the uncompressed data stream.

Since there exist efficient coding algorithms, modeling is the main concern in statistical data compression. In fact, different models make up different statistical data compression algorithms. The three most popular families of statistical data compression algorithms are Prediction by Partial Matching (PPM) [22, 84, 23, 106], Context Tree Weighting (CTW) [98, 101] and PAQ (pronounced "pack") [56, 52, 54]. These three algorithms use a common approach for modeling: They pass numerous simple probability estimates (distributions) to a *mixer* (or multiple mixers) in order to obtain a single probability distribution for coding. These simple probability estimates are usually defined by simple closed-form expressions, therefore we introduce the term *elementary model* to refer to such a simple estimator. Furthermore, elementary models are conditioned on a context, to make their predictions more accurate. (Consider the probability of the letter "u" in English text: In general its probability is rather low, but if we read the letter "q", i. e. use a one letter context, then it is almost surely followed by "u".) Consequently, the novel term *context mixing* was established with PAQ to emphasize this type of estimation procedure. We believe that it is misleading to call only PAQ-based algorithms context mixing-algorithms, since the PPM and CTW family work similar.

1.3 A Gap between Theory and Practice

Statistical data compression has been tackled from two different viewpoints, that is, from a theoretic viewpoint and from a practical viewpoint. On the one hand, in a theoretic approach we put most effort into a model design that allows for a mathematical analysis, but neglects algorithmical and practical aspects like running time, memory requirements and adjustments that improve the empirical performance. The goal of the mathematical analysis (code length analysis) is to show that a proposed model performs not much worse than an idealized competing model. On the other hand, in a practical approach we let the theoretic feasibility fade from the spotlight and focus on the algorithmical and practical aspects. CTW and its derivatives are a shining example for the theoretic approach, whereas PPM, although there exists some theoretic work [4, 15], and especially PAQ are examples for the practical approach.

PAQ is even more different from CTW and PPM, since it has sounded the bell to a change in paradigm (which we mentioned earlier) [80]: Instead of major refinements to a compression algorithm or the proposal of novel compression algorithms from time to time many frequent and small changes in elementary modeling and mixing, additional heuristics, etc. sum up to considerable empirical improvements and many different PAQ variants. The superior empirical performance of PAQ becomes apparent when we compare the bare compression ratio of the best performing algorithms (known to the author) from each of the major three classes on a standard data set, the Calgary Corpus. The CTW family achieves an average compression of 2.10 bits per character (bpc), for the PPM family we have 1.82 bpc and finally 1.73 bpc for PAQ.[2] The comparison is not entirely fair, since PAQ has models for data, e. g. fax images, contained in the Calgary Corpus, whereas PPM and CTW don't. However, this is one of the strengths of PAQ, it is designed to mix *arbitrary* models, whereas PPM and CTW by design *cannot*. As we can see, the practical approach leads to superior empirical performance, but usually doesn't have a sound theoretical basis to explain its success. This is the blessing and curse of PAQ, leading to an increasing gap between theory and practice.

[2] Each file is compressed individually, the compression rates for every file (in bpc) are summed and divided by the number of files. For CTW we use the results of [10]; PPM measured using PPMonstr J Rev. 1, a revision of [84], from `http://compression.ru/ds/`, accessed 2014-07-01; PAQ measured using PAQ8L from `http://mattmahoney.net/dc/paq.html`, accessed 2014-07-01.

1.4 Our Contribution and Thesis Structure

With this work we want to make first steps towards bridging the aforementioned gap between theory and practice with an emphasis on PAQ. To do so, we add a sound theoretical basis to the latest elementary modeling and mixing algorithms of PAQ. The PAQ evolution of these building blocks started in 2002 with PAQ1 and eventually ended in 2005 with PAQ7. Later PAQ variants mainly added preprocessing tricks and additional models for specific data types. Thus, by "latest elementary modeling and mixing algorithms" we refer to PAQ variants not older than PAQ7. Our methodology to obtain a theoretical basis is to provide a mathematical analysis of elementary modeling and mixing in the spirit of the theoretical approach to data compression, as mentioned in the previous section. Whenever possible we try to give results as general as possible and sometimes generalize approaches from PAQ.

The remainder of this thesis is structured as follows: In Chapter 2 we introduce notation, discuss the basics of elementary modeling, context modeling, mixing and our approach to a code length analysis. Based on this we present the central algorithms in statistical data compression and discuss relevant literature. We cover Arithmetic Coding, PPM, CTW, PAQ and the less well-known algorithms DMC and DEPLUMP. In Chapter 3 and Chapter 4 we cover elementary modeling and mixing. After discussion existing approaches from literature we provide a code length analysis for methods that work well in practice but lack a theoretic basis (see below). In Chapter 5 we propose Context Tree Mixing, a novel statistical data compression algorithm and provide a code length analysis (see below). Chapter 6 summarizes the results of this thesis and offers a perspective for future work.

Chapter 3, Chapter 4 and Chapter 5 contain our main contributions to the state of the art in statistical data compression, which are the following.

Chapter 3: Elementary Modeling. We provide a code length analysis for two widespread families of elementary models: First, we consider elementary models that assign probabilities by relative letter frequencies and that apply a frequency discount (frequencies get multiplied by a number from $[0, 1)$, when the frequency sum exceeds a threshold); Second, we consider elementary models that assign probabilities by exponential smoothing. We work out a code length analysis for each method that shows that the coding performance of these elementary models is close to that of a Piecewise Stationary Model that partitions the input sequence into segments and predicts a fixed distribution within each segment. (Our results on elementary modeling by smoothing apply to binary input sequences only; for other results there is no restriction.) Furthermore, in this chapter we add a theoretic basis to the PAQ approach to elementary modeling, since PAQ employs variations of the smoothing approach. We support our results by an experimental study. The main results of Chapter 3 have previously been published in [60, 61].

Chapter 4: Mixing. In this chapter we systematically propose and analyze two methods for mixing, at that we proceed in two steps: First, we introduce two slightly different information theoretic minimization problems. The minimizer of either problem defines a mixture distribution that allows for the (non-)linear weighted combination of multiple distributions. In this way we introduce the Linear Mixture Distribution (linear weighted averaging of probabilities) and the Geometric Mixture Distribution (normalized geometrically weighted averaging of probabilities). Second, either approach relies on a set of given weights. We adopt Online Gradient Descent to estimate these weights and provide a code length analysis. By our analysis either mixture distribution coupled with Online Gradient Descent provides a coding performance close to that of a Switching Mixer. A Switching Mixer partitions the input sequence into segments and switches back and forth between the distributions to mix across segment boundaries. An experimental study supports our results. Moreover, we add a theoretic justification and code length guarantees to PAQ's ad-hoc neural network mixing, since we show that it is a special form of the Geometric Mixture Distribution coupled with Online Gradient Descent. The results in Chapter 3 are a polished version of results published earlier in [58, 59].

Chapter 5: Context Tree Mixing. We propose Context Tree Mixing (CTM) and conduct a code length analysis. CTM is a statistical data compression algorithm that generalizes CTW: In CTW at least mixing is fixed, CTM allows for arbitrary mixers *and* arbitrary elementary models. Our theoretic results show that CTM achieves a coding performance close to that of a sequence of Predictive Context Trees (PCT), if elementary models and mixers are chosen carefully. A PCT groups the letters of the input sequence by context and predicts a fixed distribution per context (the way of "context-grouping" is induced by some context tree). We extend our theoretical results by analyzing CTM coupled PAQ-approaches to elementary modeling and mixing. Our results show that this particular CTM configuration enjoys code length guarantees stronger than those of CTW, since its coding performance is close to that of an arbitrary sequence of PCTs, not just close to that of a single PCT. Experiments also indicate improved empirical performance.

CHAPTER 2

Fundamentals

CHAPTER OVERVIEW

In this chapter we introduce the basic notation we use within the remainder of this work. Taking this as a basis we formally introduce models, mixers, context modeling and sketch our approach to their theoretical analysis (code length analysis). We end this chapter with a description of Prediction by Partial Matching, Context Tree Weighting and PAQ, a short summary of Arithmetic Coding and two less popular statistical data compression algorithms: Dynamic Markov Coding and DEPLUMP .

2.1 Basic Notation

Sets, Families, Segments and Partitions. We typeset sets using uppercase calligraphic letters and use $|\mathcal{S}|$ to denote the cardinality of a set $\mathcal{S}$. For objects $x_j, x_k, \ldots$ of identical type (numbers, letters, ...) with labels $j, k, \ldots \in \mathcal{S}$ we call the indexed multiset $\{x_i\}_{i\in\mathcal{S}}$ a *family* (of numbers, letters, ...). Given integers a and b, a *segment* $a{:}b$ is the set $\{a, a+1, \ldots, b\}$ of integers (or the empty set, if $b < a$). A *partition* $\mathcal{P}$ of some segment $a{:}b$ is a set of non-overlapping segments whose union equals $a{:}b$.

Vectors, Unit Simplex, Projection and Gradients. Vectors are typeset using lowercase boldface letters. Within this work we use column vectors only, "T" denotes the transpose operator, so $(z_1, z_2, \ldots, z_m)^\mathsf{T}$ is a column vector. The i-th component of vector $\boldsymbol{z}$ is z_i and the euclidean norm of a real-valued vector $\boldsymbol{z}$ is $|\boldsymbol{z}|$. We denote the m-dimensional unit simplex as

$$\Delta := \{\boldsymbol{z} \in \mathbb{R}^m \mid z_1 + \cdots + z_m = 1 \text{ and } z_1, \ldots, z_m \geqslant 0\}.$$

The function $\operatorname{proj} : \boldsymbol{z} \mapsto \boldsymbol{v}$ maps any $\boldsymbol{z} \in \mathbb{R}^m$ to the point $\boldsymbol{v} \in \Delta$ closest to $\boldsymbol{z}$ in the euclidean sense, that is, $\operatorname{proj}(\boldsymbol{z})$ maps to the minimizer of $\min_{\boldsymbol{u}\in\Delta}|\boldsymbol{u} - \boldsymbol{z}|$. For a differentiable function $f\colon \mathcal{D} \to \mathbb{R}$, where $\mathcal{D} \subseteq \mathbb{R}^m$ is an open set, let $\nabla_{\boldsymbol{z}} f := (\partial f/\partial z_1, \ldots, \partial f/\partial z_m)$ denote the gradient of f w. r. t. $\boldsymbol{z}$. A *p-vector* is a vector whose components are probability distributions.

Alphabets, Sequences and Segments (of Sequences). For segment $1{:}n$ we denote a sequence $(x_1, x_2, \dots, x_n)$ of objects of identical type by $x_{1:n} := x_1 x_2 \dots x_n$, we further let $x_{a:b} := x_a x_{a+1} \dots x_b$ denote a *segment* (subsequence) of $x_{1:n}$; if $a{:}b = \emptyset$, we have $x_{a:b} = \phi$, the empty sequence. We define $x_{<a} := x_{1:a-1}$ and use $x_{a:\infty}$ to denote a sequence $x_a x_{a+1} \dots$ of infinite length. Unless stated differently, we only consider sequences over some alphabet $\mathcal{X} := \{0, 1, \dots, N-1\}$ of cardinality $N \geqslant 2$; a letter denotes an element of $\mathcal{X}$. Moreover, we let $|x_{a:b}| := \max\{b-a+1, 0\}$ denote the sequence length. For a set $\mathcal{S} = \{i_1, i_2, \dots\}$ of integers $i_1 < i_2 < \dots$ and a function $e : i \mapsto e(i)$ let $\langle e(i) \rangle_{i \in \mathcal{S}}$ denote the sequence $x_1 x_2 \dots$, where $x_t = e(i_t)$.

Distributions, Code Length, (Empirical) Entropy and KL-Divergence. In general we consider probability distributions on $\mathcal{X}$ only. Let $\log := \log_2$ and $\ln := \log_e$. For a distribution p, a sequence $p_{1:n}$ of distributions, a letter x and a sequence $x_{1:n}$ we define the "code lengths" $\ell(x; p) := \log \frac{1}{p(x)}$,

$$\ell(x_{1:n}; p) := \sum_{1 \leqslant t \leqslant n} \log \frac{1}{p(x_t)} \quad \text{and} \quad \ell(x_{1:n}; p_{1:n}) := \sum_{1 \leqslant t \leqslant n} \log \frac{1}{p_t(x_t)}. \tag{2.1}$$

We call these quantities code length(s), since encoding a letter x with probability $p(x)$ ideally requires $\log \frac{1}{p(x)}$ bits. There exist coding algorithms that approximate the ideal code length arbitrarily close (see Section 2.4.1). A sequence $x_{1:n}$ has *empirical entropy* $h(x_{1:n}) := \ell(x_{1:n}; p)$, where the distribution p holds the relative letter frequencies of $x_{1:n}$. We use the standard notation $H(p)$ for the entropy of distribution p, and $D(p \parallel q)$ for the Kullback-Leibler-Divergence (KL-Divergence) of the distributions p and q, that is

$$H(p) = \sum_{x \in \mathcal{X}} p(x) \cdot \log \frac{1}{p(x)} \quad \text{and} \quad D(p \parallel q) = \sum_{x \in \mathcal{X}} p(x) \cdot \log \frac{p(x)}{q(x)}. \tag{2.2}$$

For $p(x) = 0$ we set $p(x) \cdot \log \frac{1}{p(x)} = 0$ in (2.2) and $\log \frac{1}{p(x)} = \infty$ in (2.1) to simplify notation. The conventions and notation above follow standard literature [26].

2.2 Models, Mixers and Redundancy

Formal Definitions. As already discussed in Section 1.2, (elementary) models are a fundamental building block in statistical data compression algorithms, so we specify:

Definition 2.2.1 *A* model MDL *maps every sequence $x_{1:t}$ of length $t \geqslant 0$ to a probability distribution p; we call p the prediction of model* MDL *(given input sequence $x_{1:t}$). We define $\mathsf{MDL}(x_{1:t}) := p$, we introduce the shorthand $\mathsf{MDL}(x; x_{1:t}) := p(x)$ and we further define the code length*

$$\ell(x_{a:b}; \mathsf{MDL}) := \sum_{a \leqslant t \leqslant b} \log \frac{1}{\mathsf{MDL}(x_t; x_{<t})}, \textit{ for } 1 \leqslant a, b \leqslant n.$$

There is a wide range of possibilities for specifying a model. On the one hand, a simple closed-form expression suffices to specify a model; on the other hand, a specification may be a sophisticated function of multiple (sub-) models and parameters. Nevertheless, a particularly simple type of model, which we already mentioned in Section 1.2, is of great importance: An *elementary model* is defined in terms of a simple closed-form expression depending only on $x_{1:t}$ and possibly some parameters. For instance, for the parameter $f_0 > 0$, we consider the probability assignment

$$\mathsf{RF}(x; x_{1:t}) := \frac{|\{i \mid x_i = x \text{ and } 1 \leqslant i \leqslant t\}| + f_0}{t + N \cdot f_0}, \tag{2.3}$$

based on relative letter frequencies, to be an elementary model.

Let us now turn to mixers. A mixer combines predictions of a finite number of models $\mathsf{MDL}_1, \mathsf{MDL}_2, \dots$. In time step t we may think of this as follows: We feed the past sequence $x_{<t}$ and the predictions of all models up to step t,

$$\boldsymbol{p}_{1:t} = \langle (\mathsf{MDL}_1(x_{<i}), \mathsf{MDL}_2(x_{<i}), \dots)^{\mathsf{T}} \rangle_{1 \leqslant i \leqslant t},$$

into the mixer. Using the past sequence $x_{<t}$ and the past p-vectors $\boldsymbol{p}_{<t}$ the mixer determines its internal state, for instance a weight vector. Given the internal state the mixer produces a single distribution by combining all distributions from $\boldsymbol{p}_t = (\mathsf{MDL}_1(x_{<t}), \mathsf{MDL}_2(x_{<t}), \dots)^{\mathsf{T}}$. We now give a formal definition:

Definition 2.2.2 *A* mixer MIX *maps a sequence $\boldsymbol{p}_{1:t}$ of p-vectors, where $t > 0$, with fixed dimension and a sequence $x_{<t}$ to a probability distribution p. We say p is the* prediction *of* MIX *given* mixer input *$\boldsymbol{p}_{1:t}$ and input sequence $x_{<t}$; we define $\mathsf{MIX}(x_{<t}, \boldsymbol{p}_{1:t}) := p$, the shorthand $\mathsf{MIX}(x; x_{<t}, \boldsymbol{p}_{1:t}) := p(x)$ and we further define the code length*

$$\ell(x_{a:b}; \mathsf{MIX}, \boldsymbol{p}_{1:b}) := \sum_{a \leqslant t \leqslant b} \log \frac{1}{\mathsf{MIX}(x_t; x_{<t}, \boldsymbol{p}_{1:t})}, \textit{ for } 1 \leqslant a, b \leqslant n.$$

When the mixer input $\boldsymbol{p}_{1:t}$ may be determined by a function $\mathrm{in}(x_{<t})$, for all $t \geqslant 1$, then by setting $\mathsf{MDL}(x_{<t}) = \mathsf{MIX}(x_{<t}, \mathrm{in}(x_{<t}))$, mixer MIX essentially induces a new model MDL, a so-called *mixture model*. In turn, this mixture model may serve as part of another mixer input, etc.; following this scheme we may stack (mixture) models almost arbitrarily.

To breathe life into Definition 2.2.2, consider Beta-Weighting [49] as an example. Beta-Weighting combines distributions by weighted linear averaging. Hence, given the weight vector $\boldsymbol{w}_t = (w_{1,t}, \dots, w_{m,t})^\mathsf{T}$ in the t-th step the prediction of Beta-Weighting is given by

$$\mathsf{BETA}(x; x_{<t}, \boldsymbol{p}_{1:t}) := w_{1,t} \cdot p_{1,t}(x) + \cdots + w_{m,t} \cdot p_{m,t}(x), \tag{2.4}$$

where $\boldsymbol{p}_t = (p_{1,t}, \dots, p_{m,t})^\mathsf{T}$. In step 1 the weight vector is $\boldsymbol{w}_1 = (\frac{1}{m}, \dots, \frac{1}{m})^\mathsf{T}$, for a step $t > 1$ we may state the weight $w_{i,t+1}$ of model i recursively,

$$w_{i,t+1} = w_{i,t} \cdot \frac{p_{i,t}(x_t)}{\mathsf{BETA}(x_t; x_{<t}, \boldsymbol{p}_{1:t})}.$$

Later, in Section 2.4.3 will discuss a well-known statistical compression algorithm, – Context Tree Weighting – that recursively stacks Beta-Weighting.

Note that the code length $\ell(x_{a:b}; \mathsf{MDL})$ may not only depend on the segment $x_{a:b}$, but also on the segment $x_{<a}$, since a prediction on x_a typically depends on $x_{<a}$. The same applies to $\ell(x_{a:b}; \mathsf{MIX}, \boldsymbol{p}_{1:b})$.

Idealization vs. Reality. We note that Definition 2.2.1 and Definition 2.2.2 do not claim non-zero probabilities. Based on the outcome space of a model's prediction we may further distinguish two types of models, namely *idealized models* and *non-idealized models*.

For a given sequence $x_{1:n}$ an idealized model assigns probability 0 to some letter(s), but still guarantees $\mathsf{MDL}(x_t; x_{<t}) > 0$, for $1 \leqslant t \leqslant n$. Idealized models must have prior knowledge of the sequence $x_{1:n}$, and thus may assign probability 0 to some letter *different* from x_t in step t. Of course, they are of little use in practice. However, we still consider idealized models, since they act as competitors in a code length analysis (see below).

In contrast, a non-idealized model always guarantees $\mathsf{MDL}(x; x_{<t}) > 0$, for any letter x and all t. Any model whose prediction is supposed to be fed into an encoder will guarantee $\mathsf{MDL}(x; x_{<t}) > 0$, for every letter x, and thus be a non-idealized model. (In advance, we usually do not know which letter appears next.) Similar considerations apply to (non-)idealized mixers.

Redundancy and Code Length Analysis. A major portion of this work is devoted to code length analysis or redundancy analysis, for either elementary models, mixers or more complex (mixture) models. We will now briefly explain our approach and bring it in line with the existing literature. To simplify the following discussion consider a model M, which could be an elementary model, a mixture model induced by a mixer, etc. and a competitor C. We may view M and C as models in the sense of Definition 2.2.1.

Let us now describe our approach to code length analysis. Our central goal in a code length analysis is to compare the code length $\ell(x_{1:n}; \mathsf{M})$ of model M to the code length $\ell(x_{1:n}; \mathsf{C})$ of competitor C assuming ideal encoding (a probability p corresponds to code length $-\log p$). For a code length analysis we strive to show that model M is close to competitor C. That is, we provide a code length bound of type

$$\ell(x_{1:n}; \mathsf{M}) \leqslant (1+\delta) \cdot \ell(x_{1:n}; \mathsf{C}) + r(\mathsf{C}, \mathsf{M}, x_{1:n}), \tag{2.5}$$

for some constant $\delta \geqslant 0$. If $\delta = 0$ (most results from literature are of this form), then the amount of bits model M requires beyond competitor C is called *redundancy*, so the term $r(\mathsf{C}, \mathsf{M}, x_{1:n})$ is a bound on the *(pointwise) redundancy* of the model M w. r. t. the competitor C for a given sequence $x_{1:n}$. The case $\delta > 0$ provides a weaker guarantee, since the code length $\ell(x_{1:n}; \mathsf{M})$ of model M may differ from the code length $\ell(x_{1:n}; \mathsf{C})$ of competitor C up to a constant factor $1+\delta$, apart from the additive term $r(\mathsf{C}, \mathsf{M}, x_{1:n})$. This type of guarantee is still useful, since achieving $\delta = 0$ sometimes requires to have prior knowledge on $x_{1:n}$, which often is infeasible (for instance, we will encounter this situation in Section 3.4). Furthermore, if $\ell(x_{1:n}; \mathsf{C})$ is small, a bound with $\delta > 0$ might actually improve over a bound with $\delta = 0$ that has a relatively large (compared to $\ell(x_{1:n}; \mathsf{C})$) additive redundancy term.

Example 2.2.3 *We consider the model* RF *from (2.3) with* $f_0 = 1$*. This model is also known as the Laplace-Estimator, hence we write* LP*. If the letter* x *has appeared* i *times in* $x_{1:t}$*, model* LP *predicts* $\mathsf{LP}(x; x_{1:t}) = \frac{i+1}{t+N}$*, so the code length for a sequence* $x_{1:n}$*, where letter* x *has frequency* $f(x)$*, is*

$$\ell(x_{1:n}; \mathsf{LP}) = \log \prod_{1 \leqslant t \leqslant n} \frac{1}{\mathsf{LP}(x_t; x_{<t})} = \log \frac{N \cdot (N+1) \cdot \ldots \cdot (n+N-1)}{\prod_{x \in \mathcal{X}} 1 \cdot 2 \cdot \ldots \cdot f(x)}.$$

By $n+t \leqslant (n+1) \cdot t$*, for* $1 \leqslant t < N$*, and* $\ell(x_{1:n}; p) \geqslant h(x_{1:n}) \geqslant \log \frac{n!}{\prod_{x \in \mathcal{X}} f(x)!}$*, for any distribution* p*, we obtain*

$$\ell(x_{1:n}; \mathsf{LP}) \leqslant \ell(x_{1:n}; p) + (N-1)\log(n+1), \tag{2.6}$$

a code length bound in the spirit of (2.5). The model LP *has redundancy at most* $r(p, \mathsf{LP}, x_{1:n}) = (N-1)\log(n+1)$ *w. r. t. any fixed distribution* p*.*

To maximize the effectiveness of our results, we consider code length bounds of type (2.5) that hold for any sequence $x_{1:n}$ (hence, these hold in the worst case) and for all M and C that satisfy given conditions. Example 2.2.3 illustrates bounds of type (2.5).

The approach we pursue may also be interpreted in another way. We may view a refined version of bound (2.5) as follows. Essentially, conditions on M span a class $\mathcal{M}$ of models, similarly conditions on C span a class $\mathcal{C}$ of competitors. Let $r'(\mathcal{C}, \mathcal{M}, n) \geqslant r(\mathsf{C}, \mathsf{M}, x_{1:n})$, for all sequences $x_{1:n}$, models $\mathsf{C} \in \mathcal{C}$ and $\mathsf{M} \in \mathcal{M}$. Thus, r' is a uniform bound on the redundancy of class $\mathcal{M}$ w. r. t. class $\mathcal{C}$ for sequences of length n in the sense of

$$\sup_{\substack{\mathsf{M}\in\mathcal{M},\\ x_{1:n}}} \left(\ell(x_{1:n}; \mathsf{M}) - (1+\delta) \cdot \inf_{\mathsf{C}\in\mathcal{C}} \ell(x_{1:n}; \mathsf{C}) \right) \leqslant r'(\mathcal{C}, \mathcal{M}, n). \tag{2.7}$$

Whenever $r'(\mathcal{C}, \mathcal{M}, n) = o(n)$ and $\delta = 0$, the average per-letter redundancy vanishes with increasing n, so the class $\mathcal{M}$ (or strategy M, if $|\mathcal{M}| = 1$) is said to be *asymptotically optimal*, or *universal*, w. r. t. class $\mathcal{C}$. Hence, the ultimate goal is to design models that work well in practice and guarantee asymptotic optimality w. r. t. a (wide and/or powerful) class of competitors, preferably with an average redundancy as small as possible. Example 2.2.4 illustrates bounds of type (2.7).

Example 2.2.4 *If we define class $\mathcal{C} := \{p \mid p$ is distribution$\}$ of competitors, then (2.6) implies the guarantee*

$$\sup_{x_{1:n}} \left(\ell(x_{1:n}; \mathsf{LP}) - \inf_{p\in\mathcal{C}} \ell(x_{1:n}; p) \right) \leqslant (N-1)\log(n+1),$$

in spirit of (2.7). The redundancy of LP *w. r. t. class $\mathcal{C}$ on sequences of length n may uniformly be bounded by $r'(\mathcal{C}, \{\mathsf{LP}\}, x_{1:n}) = (N-1)\log(n+1)$.*

In either of the above settings we assumed that the sequence $x_{1:n}$ is arbitrary and made no assumptions on a possible origin, which is called the *deterministic view* in information theory [64]. In contrast, the *probabilistic view* assumes that there exists a data-generating mechanism, the *source*, which draws a sequence $x_{1:n}$ at random according to its inherent *source distribution* P (here P is a distribution on sequences of length n). A common goal in this situation is to design and analyze models that attain low expected redundancy $\sup_{\mathsf{M}\in\mathcal{M},\text{source } P} \mathrm{E}\left[\ell(x_{1:n}; \mathsf{M}) + \log P(x_{1:n})\right]$ for a class of sources and arbitrary $n \geqslant 1$. (We take the expectation over the probability space induced by the source distribution P.) We will use the probabilistic view only once within this work, namely to derive a generalized form of PAQ-mixing, see Chapter 4.

2.3 Context Modeling

In Section 1.2 we already mentioned that (elementary) models and mixers may be conditioned on contexts. In fact, every major statistical compression algorithm employs context conditioning in some way. At this point we introduce the reader to our concept for context modeling, which will serve as a basis for our discussion of the algorithms PPM, CTW and PAQ in Section 2.4.

Contexts, Context Histories and Context Trees. If for some sequence $x_{1:t}$ we have $x_{t-d:t-1} = c_{1:d}$, or $c_{1:d} = \phi$, we say *the letter x_t has length-d context c*. When possible, we write c in place of $c_{1:d}$ to avoid a cluttered notation. Let

$$\mathcal{T}_c(x_{a:b}) := \{a \leqslant t \leqslant b \mid x_t \text{ has context } c\}$$

be the set of time indices s. t. the letter x_t, for $a \leqslant t \leqslant b$, has context c. (Notice that $\mathcal{T}_c(x_{a:b})$ depends on the segment $x_{a-|c|:a-1}$ and is independent of x_b.) The *context history* of context c for $x_{a:b}$ is $x^c_{a:b} := \langle x_t \rangle_{t \in \mathcal{T}}$, where $\mathcal{T} = \mathcal{T}_c(x_{a:b})$. Below we provide Example 2.3.1 (top) to clarify this notation.

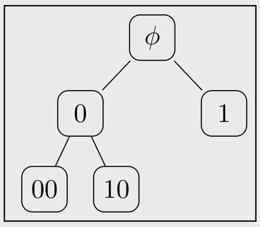
Example 2.3.1 **(a)** *Fix sequence $x_{1:n} = 101100101101$ of length 12. First, we consider our basic notation on context histories. For context $c = 0$ we have*

$$\mathcal{T}_c(x_{1:n}) = \{3, 6, 7, 9, 12\} \text{ and } x^c_{1:n} = x_3x_6x_7x_9x_{12} = 10111;$$

for the segment $x_{3:8}$ and context $c = 0$ we get

$$\mathcal{T}_c(x_{3:8}) = \{3, 6, 7\} \text{ and } x^c_{3:8} = x_3x_6x_7 = 101.$$

(b) *We now turn towards context trees, the binary context tree on the right has set $\mathcal{C}_I = \{\phi, 0\}$ of non-leaf contexts, set $\mathcal{C}_L = \{1, 00, 10\}$ of leaf contexts and set $\mathcal{C} = \{\phi, 0, 1, 00, 10\}$ of contexts. Specifying any of these sets suffices to uniquely determine the corresponding tree.*



A *context tree* is an N-ary tree, where every node either has N children or is a leaf. Every node is labeled with a unique context, so we may refer to node or context interchangeably. The root context is ϕ, a non-leaf context c has child contexts $0c, 1c, \dots (N-1)c$. For a finite context tree, the set of all leaf contexts is $\mathcal{C}_L$, the set of all non-leaf contexts is $\mathcal{C}_I$ and $\mathcal{C} := \mathcal{C}_L \cup \mathcal{C}_I$. Notice that specifying either $\mathcal{C}_L$, $\mathcal{C}_I$ or $\mathcal{C}$ suffices to uniquely determine a context tree. One typically specifies the structure of a context tree via $\mathcal{C}_L$, the so-called proper suffix-set [101]. (Often the underlying tree is ignored and one only considers $\mathcal{C}_L$.) Example 2.3.1 (bottom) illustrates this notation.

Context-conditioned Modeling and Mixing. We consider the t-th step in the course of a statistical data compressor on some input sequence $x_{1:n}$. The past sequence $x_{<t}$ is known and let the letter x_t have context c. For this context a *context conditioned model* MDL^c maps the context history $x^c_{<t}$ to its prediction $\mathsf{MDL}^c(x^c_{<t})$. We write MDL^c to indicate that the model inputs the context history $x^c_{<t}$ (not $x_{<t}$) and also to indicate that the model MDL^c may depend on the context c. For instance, at some context the context conditioned model may be RF with parameter $f_0 = 1$ and at some other context the context conditioned model may be RF with parameter $f_0 = 5$. Hence the model parameters may vary from context to context.

For the context c of x_t a *context conditioned mixer* MIX^c maps the context history $x^c_{<t}$ and its mixer input $\mathrm{in}^c(x_{<t})$ to the prediction $\mathsf{MIX}^c(x^c_{<t}, \mathrm{in}^c(x_{<t}))$. The function in^c maps the input sequence $x_{<t}$ to a sequence of p-vectors. At this point we leave this function unspecified, since different choices make up different statistical data compression algorithms. In Section 2.4.2 and Section 2.4.3 provide concrete examples. (Typically, the mixer input $\mathrm{in}^c(x_{<t})$ will hold predictions of other context-conditioned models.) Similarly to models, a context conditioned mixer MIX^c and the mixer input $\mathrm{in}^c(x_{<t})$ may also depend on the context c. To clarify our notation we refer the reader to Example 2.3.2.

Example 2.3.2 *Consider the binary sequence $x_{1:n} = 1010010101$ of length 10, context set $\mathcal{C} = \{0, 1\}$, models $\{\mathsf{MDL}^c\}_{c\in\mathcal{C}}$ and mixers $\{\mathsf{MIX}^c\}_{c\in\mathcal{C}}$ associated to every context $c \in \mathcal{C}$ and time step $t = 8$ (i. e. $x_{<t}$ is known, $x_t, x_{t+1}, \dots$ is unknown). The situation at context $c = 0$ for the whole sequence $x_{1:n}$ (left) and for the t-th step (right) is depicted below:*

	1	2	3	4	5	6	7	8	9	10
$x_{1:n}$	1	0	1	0	0	1	0	1	0	1
$x^c_{1:n}$			1		0	1		1		1
$\mathrm{in}^c(x_{1:n})$			$\boldsymbol{p}_1$		$\boldsymbol{p}_2$	$\boldsymbol{p}_3$		$\boldsymbol{p}_4$		$\boldsymbol{p}_5$

	1	2	3	4	5	6	7	8
$x_{<t}$	1	0	1	0	0	1	0	?
$x^c_{<t}$			1		0	1		
$\mathrm{in}^c(x_{<t})$			$\boldsymbol{p}_1$		$\boldsymbol{p}_2$	$\boldsymbol{p}_3$		$\boldsymbol{p}_4$

The letter x_8 has context $c = 0$ with context history $x^c_{<8} = 101$, so we obtain the prediction $\mathsf{MDL}^c(x^c_{<t}) = \mathsf{MDL}^0(101)$; furthermore, we have $|\mathcal{T}_c(x_{1:t})| = 4$, so we encounter mixer input $\mathrm{in}^c(x_{<t}) = \boldsymbol{p}_{1:4}$. Consequently, the mixer MIX^c predicts $\mathsf{MIX}^c(x^c_{<t}, \mathrm{in}^c(x_{<t})) = \mathsf{MIX}^0(101, \boldsymbol{p}_{1:4})$.

2.4 Algorithms in Statistical Data Compression

The previous sections provided the tools we require to thoroughly discuss the three major statistical data compression algorithms, PPM, CTW and PAQ. Before we immerse into details on these algorithms in Section 2.4.2 (PPM), Section 2.4.3 (CTW) and Section 2.4.4 (PAQ), we give a short primer on Arithmetic Coding (AC) in Section 2.4.1. AC is of less importance for the rest of this work, so the reader may just skim over Section 2.4.1, to get the general idea. Still, we decided to include a short description of AC, since it is such a fundamental algorithm. For completeness we include a short description of two less popular statistical data compression algorithms, Dynamic Markov Coding and DEPLUMP, in Section 2.4.5.

2.4.1 Arithmetic Coding

Arithmetic Coding (AC) is a coding algorithm in statistical data compression. For encoding, it translates a distribution p and a letter x to a codeword of length close to the ideal code length $\log \frac{1}{p(x)}$. For decoding, it restores the letter x given the distribution p and the codeword. AC was developed in the late 1970s [70, 74] and popularized since then. Nowadays, AC is the de facto standard coding algorithm in statistical data compression. (A common implementation-variant of AC is called Range Coding [57, 82].) We now give a simplified description of AC which follows introductions to AC within textbooks, such as [68, 80, 81]. Numerous tutorial-style papers on AC are freely available [50, 79, 107]. An in-depth discussion and redundancy analysis is beyond the scope of this work. It is worthwhile to note that there exists an alternative to AC, viz. Asymmetric Binary Coding and generalizations thereof, for a non-binary alphabet [28, 29, 30].

Encoding and Decoding. Roughly speaking, an arithmetic encoder injectively maps a sequence $x_{1:n}$ and a sequence $p_{1:n}$ of probability distributions to a subinterval $[l, h)$ of the real interval $[0, 1)$. To encode $x_{1:n}$ it finally outputs a bitstring that represents a number from $[l, h)$. AC constructs the interval $[l, h)$ stepwise.

Let us now consider the t-th encoding step. We are given a distribution p_t, a letter x_t to encode and the interval $[l_{t-1}, l_{t-1}+r_{t-1})$ from the previous step. (Initially we have $l_0 = 0$ and $r_0 = 1$.) The distribution p_t partitions the real interval $[0, 1)$ s. t. every letter x corresponds to a subinterval of width $p_t(x)$. Typically, the letter 0 is represented by $[0, p_t(0))$, the letter 1 is represented by $[p_t(0), p_t(0) + p_t(1))$, ... and the letter x is represented by the subinterval

$$\left[\sum_{y<x} p_t(y), \sum_{y\leqslant x} p_t(y)\right) .$$

For encoding we transfer the partition of $[0,1)$ induced by p_t to the interval $[l_{t-1}, l_{t-1}+r_{t-1})$: Letter 0 is represented by $[l_{t-1}, l_{t-1}+r_{t-1}\cdot p_t(0))$, letter 1 is represented by $[l_{t-1}+r_{t-1}\cdot p_t(0), l_{t-1}+r_{t-1}\cdot(p_t(0)+p_t(1))$,.... To encode x_t the next interval $[l_t, l_t+r_t)$ becomes the subinterval of $[l_{t-1}, l_{t-1}+r_{t-1})$ that represents x_t, so we set

$$l_t := l_{t-1} + r_{t-1}\cdot \sum_{y<x_t} p_t(y) \quad\text{and}\quad r_t := r_{t-1}\cdot p_t(x_t). \tag{2.8}$$

After encoding $x_{1:n}$ we obtain the interval $[l,h) = [l_n, l_n+r_n)$. (To signal "end-of-file" we may introduce an end-of-file letter or prepend the arithmetic code with the sequence length, if known in advance.) The encoder outputs a number $c\in[l,h)$, by writing its binary representation $0.b_1b_2\dots b_m$. For any interval $[l,h)$ we can find c s. t. it takes $\log\frac{1}{h-l}+O(1)$ bits to write c.

Example 2.4.1 *The table below shows the course of AC on a small sample:*

		Encoding:		**Decoding:** $c = 0.046875$	
t	p_t	x_t	$[l_t,\ l_t+r_t)$	$[l_{t-1},\ l_{t-1}+r_{t-1})$	$\frac{c-l_{t-1}}{r_{t-1}}$
0	$(0.1, 0.5, 0.4)$	0	$[0.00000, 0.10000)$	$[0.00000, 1.00000)$	0.04688
1	$(0.3, 0.2, 0.5)$	1	$[0.03000, 0.05000)$	$[0.00000, 0.10000)$	0.46875
2	$(0.7, 0.2, 0.1)$	1	$[0.04400, 0.04800)$	$[0.03000, 0.05000)$	0.84375
3	$(0.5, 0.2, 0.3)$	2	$[0.04680, 0.04800)$	$[0.04400, 0.04800)$	0.71875
4	$(0.2, 0.7, 0.1)$	0	$[0.04680, 0.04704)$	$[0.04680, 0.04800)$	0.06250

Above we depict distribution p_t as $(p_t(0), p_t(1), p_t(2))$. Encoding yields the interval $[l,h) = [0.04680, 0.04704)$, so we output the binary form of

$$c = 0.046875 = 0.0000110000000_{\text{binary}} \quad (= 0.b_1b_2\dots b_m),$$

using $m = 13$ bits, whereas the ideal code length is $-\sum_{1\leqslant t\leqslant n}\log p_t(x_t) = 12.0246\dots$ bits. (Notice a technical detail: We pad the output with trailing zeros, such that we have $0.b_1b_2\dots b_m z_1 z_2\dots \in [l,h)$, for any bitstring $z_{1:\infty}$. This guarantees correct decoding, no matter what bits $z_1z_2\dots$ the decoder reads past the actual end of input; this is necessary to make the code self-delemiting.)

Decoding is straightforward, first, the decoder reads the number c and sets $l_0 = 0$ and $r_0 = 1$. Now consider the t-th step, the interval $[l_{t-1}, l_{t-1} + r_{t-1})$ and the distribution p_t are known. Just as in encoding, p_t partitions the interval $[l_{t-1}, l_{t-1}+r_{t-1})$. To identify the next letter $x_t = x$, we simply find the subinterval of $[l_{t-1}, l_{t-1} + r_{t-1})$ (determined by x) that encloses c:

$$\sum_{y<x} p_t(y) \leqslant \frac{c - l_{t-1}}{r_{t-1}} < \sum_{y \leqslant x} p_t(y).$$

The enclosing subinterval will correspond to x. It remains to maintain l_t and r_t using (2.8). Example 2.4.1 illustrates encoding and decoding.

Redundancy. An idealized implementation of AC that operates on arbitrary precision real numbers only suffers from at most 2 bits of redundancy regardless of the input length [26]. In reality computations use B-bit integer arithmetic and have to take special care of finite-precision effects. Depending on the exact setting of interest the per-letter redundancy is negligible, ranging from $O(1/2^B)$ bits per letter [42, Theorem 1] to $O(B/2^B)$ [77, Appendix A] bits per letter. Experiments indicate redundancy in scale of 10^{-4} for $B = 16$ [42]. Nowadays we have $B = 32$ or $B = 64$, so we have good reasons to assume that the coding redundancy is negligible.

2.4.2 Prediction by Partial Matching

Prediction by Partial Matching (PPM), introduced in 1984 [23], is the first widely accepted statistical data compression algorithm. In contrast to CTW and PAQ, PPM typically operates on a non-binary alphabet. The underlying principle of PPM is to combine predictions of several elementary models conditioned on order-d contexts by switching from high to low orders. (Most PPM variants bound the maximum order, although PPM* does not [22].) Switching is explicitly signaled by encoding a so-called escape-letter “esc”, which extends the original alphabet. On the one hand, various PPM variants such as PPMII [84] and PPMDP [85] offer excellent empirical performance, on the other hand, there is no code length analysis in the deterministic view (covering any PPM variant) that provides an explanation.

In general, there are two equivalent approaches to describe PPM: First, an approach following the original work we sketched above; second, an approach that expresses PPM as a *sequential recursive mixture*. The former approach is widespread and easy to follow, many textbooks and papers are based upon this. However, it suffers from a severe disadvantage, namely, it neither cleanly separates modeling and coding, nor does it follow the mixture model scheme of Section 1.2. We refer to this as the *textbook approach*. In contrast, the latter approach is less well-known (although some researchers rely on it, e. g. [4, 15]), but it separates modeling and coding and provides a mixture model.

Below we first introduce the reader to basic elementary modeling and context modeling in PPM, in order to cover the textbook approach to PPM. Thereafter, we show in a clear way how to turn the textbook approach into a sequential recursive mixture. Altogether we concentrate on the principles of PPM, rather than discussing particular PPM variants. Some historical notes conclude this section.

Elementary Modeling and Context Modeling. Let us first consider elementary modeling and associated context modeling, before we proceed with the actual description of PPM. PPM employs an elementary model that predicts a distribution $\mathsf{MDL}(x_{1:t}, \mathcal{E})$ on $\mathcal{X}$ and may *exclude* letters during probability assignment in the following way:

$$\begin{aligned} &\mathsf{MDL}(x; x_{1:t}, \mathcal{E}) = 0, \text{ if } x \in \mathcal{E} \text{ or } x \text{ does not appear in } x_{1:t}, \\ &\mathsf{MDL}(x; x_{1:t}, \mathcal{E}) > 0, \text{ otherwise.} \end{aligned} \tag{2.9}$$

The set of excluded letters is $\mathcal{E}$. Since excluded letters receive zero probability, even if they appear in $x_{1:t}$, the probability of some non-excluded letters increases[1]. Typically, the exclusion is organized s. t. for all letters $y \notin \mathcal{E}$ appearing in $x_{1:t}$ we have $\mathsf{MDL}(x; x_{1:t}, \mathcal{E} \cup \{y\}) > \mathsf{MDL}(x; x_{1:t}, \mathcal{E})$, for x appearing in $x_{1:n}$ and $x \notin \mathcal{E} \cup \{y\}$.

Table 2.1: Elementary model and escape probabilities for PPM Variants A, B [23], C [67], D [43]. For a sequence $x_{1:n}$ and the set $\mathcal{E}$ of excluded letters, let $f(x)$ be the frequency of letter x, if x is not excluded; 0, otherwise, let $F = \sum_{x \in \mathcal{X}} f(x)$ be the length of $x_{1:n}$ ignoring excluded letters and let $M := |\{x \in \mathcal{X} \mid f(x) > 0\}|$ be the number of non-excluded distinct letters in $x_{1:n}$.

Algorithm	$\mathsf{MDL}(x; x_{1:n}, \mathcal{E})$	$e(x_{1:n})$
PPMA	$\frac{f(x)}{F}$	$\frac{1}{1+F}$
PPMB	$\frac{\max\{0, f(x)-1\}}{F-M}$	$\frac{M}{F}$
PPMC	$\frac{f(x)}{F}$	$\frac{M}{F+M}$
PPMD	$\frac{f(x)-1/2}{F-M/2}$	$\frac{M}{2F}$

Since PPM extends the alphabet with an artificial escape letter esc, we cannot directly apply the elementary model MDL for context modeling. As a workaround we define a function e that maps a sequence to a number

[1]Notice that $\mathsf{MDL}(x_{1:t}, \mathcal{E})$ with all letters from $x_{1:t}$ excluded will not be a distribution anymore (by (2.9) all letters have probability zero) however, this situation will never arise and thus, we may assume this situation to be forbidden.

```
Function PPMEncode ($x, x_{<t}$)
if $t = 1$ then Encode ($x, u(x_{<t})$);
$\ell \leftarrow$ length, at most $D$, of longest
  context $c$ of $x_t$ with non-empty
  context history;
for $d = \ell, \ell - 1, \dots, 0$ do
    $c \leftarrow x_{t-d:t-1}$;
    if $x \in \mathcal{X}_c(x_{<t})$ then
        Encode ($x, \mathsf{MDL}^c(x^c_{<t}, \mathcal{E})$);
        return;
    end
    Encode (esc, $\mathsf{MDL}^c(x^c_{<t}, \mathcal{E})$);
    $\mathcal{E} \leftarrow \mathcal{X}_c(x_{<t})$;
end
Encode ($x, u(x_{<t})$);
```

```
Function PPMDecode ($x_{<t}$)
if $t = 1$ then return Decode ($u(x_{<t})$);
$\ell \leftarrow$ length, at most $D$, of longest
  context $c$ of $x_t$ with non-empty
  context history;
$\mathcal{E} \leftarrow \emptyset$;
for $d = \ell, \ell - 1, \dots, 0$ do
    $c \leftarrow x_{t-d:t-1}$;
    $x \leftarrow$ Decode ($\mathsf{MDL}^c(x^c_{<t}, \mathcal{E})$);
    if $x \neq \mathrm{esc}$ then return $x$;
    $\mathcal{E} \leftarrow \mathcal{X}_c(x_{<t})$;
end
return Decode ($u(x_{<t})$);
```

Figure 2.1: Pseudocode for encoding a letter x (PPMEncode) and decoding a letter (PPMDecode), given the past sequence $x_{<t}$; for distribution u see (2.11).

in $[0, 1]$. This function estimates the probability $e(x^c_{<t})$ of the escape letter esc, given a context history $x^c_{<t}$. Based on the escape probability we define a context model MDL^c that predicts a distribution on $\mathcal{X} \cup \{\mathrm{esc}\}$,

$$\mathsf{MDL}^c(x; x^c_{<t}, \mathcal{E}) := \begin{cases} e(x^c_{<t}), & \text{if } x = \mathrm{esc}, \\ (1 - e(x^c_{<t})) \cdot \mathsf{MDL}(x; x^c_{<t}, \mathcal{E})), & \text{otherwise}, \end{cases} \tag{2.10}$$

after observing context history $x^c_{<t}$. Different choices of $e(\cdot)$ and MDL make up different PPM-variants, see Table 2.1 for examples. There exist many other variants, as we will explain at the end of this section. Below we do not rely on a particular choice of $e(\cdot)$ and MDL, so we omit it.

Textbook Approach — Encoding and Decoding a Letter. To encode (or decode) a letter x, given past sequence $x_{<t}$ PPM utilizes the procedures given in Figure 2.1, where

$$\mathcal{X}_c(x_{<t}) := \{y_i \mid x^c_{<t} = y_{1:m} \text{ and } 1 \leqslant i \leqslant m\}$$

and the distribution u is defined to be

$$u(x; x_{<t}) := \begin{cases} 0, & \text{if } x \text{ appeared in } x_{<t}, \\ 1/(N - |\{x_1, x_2, \dots, x_{t-1}\}|), & \text{otherwise}. \end{cases} \tag{2.11}$$

When no letters have previously been processed, i. e. $t = 1$, encoding and decoding trivially rely on uniform distribution u. Let us now consider the usual situation, $t > 1$. During encoding, we first identify the longest context of x_t with non-empty context history of length $\ell \leqslant D$ (the maximum context length D is a parameter of PPM) and the set $\mathcal{E}$ is set to $\emptyset$. Starting at the longest context, we work on parent contexts, in order of decreasing length $d = \ell, \ell - 1, \dots, 0$: Let c be the current length-d context of x_t; if the letter x appeared before in the current context c, we encode x and stop; if the letter x did not appear in the current context c, we encode esc to tell the decoder about this situation. After we signaled a switch from the current length d-context c to the next length $(d-1)$-context, we know that no letter which appeared in context c matches x. Consequently, we exclude all those letters from future probability assignments by setting $\mathcal{E}$ to $\mathcal{X}_c(x_{<t})$. (We automatically exclude letters from child contexts of c, since these letters appear in context c as well.) This process stops when we reach a context that contains x; if no such context exists (i. e. the letter x did not appear in the segment $x_{<t}$) we use the distribution u for coding. Example 2.4.2 illustrates the encoding process.

Example 2.4.2 *Suppose that PPMA (see Table 2.1) with paramter $D = 2$ processes a sequence over the alphabet $\{0, 1, 2, 3\}$ and reaches time step t, where x_t has context 10. We assume the letter frequencies, depicted below left, at contexts $\phi, 0$ and 10 and resulting elementary model prediction, escape probability and excluded letters $\mathcal{E}$, depicted below right (we denote a distribution p as $(p(0), \dots, p(3))$):*

$c \backslash x$	0	1	2	3
10	0	2	0	0
0	2	2	3	0
ϕ	4	3	3	0

c	$\mathsf{MDL}(x^c_{<t}, \mathcal{E})$	$e(x^c_{<t})$	$\mathcal{E}$
10	$(0,\ 1,\ 0,\ 0)$	$1/3$	$\emptyset$
0	$(2/5, 0, 3/5, 0)$	$1/6$	1
ϕ	$(0,\ 0,\ 0,\ 0)$	1	$0,1,2$
$u(x_{<t})$:	$(0,\ 0,\ 0,\ 1)$		

To encode letter 2 we encode esc in context 10, where $\mathcal{E} = \emptyset$, with probability

$$\mathsf{MDL}^{10}(\mathrm{esc}; x^{10}_{<t}, \emptyset) = e(x^{10}_{<t}) = 1/3,$$

followed by letter 2 in context $c = 0$, where $\mathcal{E} = \{1\}$, with probability

$$\mathsf{MDL}^0(0; x^0_{<t}, \{1\}) = (1 - e(x^0_{<t})) \cdot \mathsf{MDL}^0(0; x^0_{<t}, \{1\}) = 5/6 \cdot 3/5 = 1/2,$$

so we assign probability $1/3 \cdot 1/2 = 1/6$ to letter 2. Overall, PPMA predicts $p = \mathsf{PPM}(x_{<t}) = \mathsf{PPM}_2(x_{<t})$ given by $(p(0), \dots, p(3)) = (2/18, 12/18, 3/18, 1/18)$.

With encoding in mind, decoding is straightforward. First, initializing ℓ and $\mathcal{E}$ mirrors encoding. Now let c be the length-ℓ context of x_t. Based on the prediction of the context model $\mathsf{MDL}^c(x^c_{<t}, \mathcal{E})$ we decode a letter x. If $x = \mathrm{esc}$, then we exclude all letters from the current context by setting $\mathcal{E} = \mathcal{X}_c(x_{<t})$ and repeat decoding at the parent context of c. (This process stops, when we decode a letter different from esc.) If $x \neq \mathrm{esc}$, then we have decoded the next letter $x_t = x$.

From the pseudocode in Figure 2.1 and the text above it is not hard to check that the letter x receives probability $\mathsf{PPM}(x; x_{<t}) = \mathsf{PPM}_\ell(x; x_{<t})$, where for $-1 \leqslant d \leqslant \ell$ we have

$$\mathsf{PPM}_d(x; x_{<t}) := \begin{cases} \mathsf{MDL}^c(x; x^c_{<t}, \mathcal{E}), & \text{if } d \geqslant 0 \text{ and } x \in \mathcal{X}_c(x_{<t}), \\ \mathsf{MDL}^c(\mathrm{esc}; x^c_{<t}, \mathcal{E}) \cdot \mathsf{PPM}_{d-1}(x; x_{<t}), & \text{if } d \geqslant 0 \text{ and } x \notin \mathcal{X}_c(x_{<t}), \\ u(x; x_{<t}), & \text{if } d = -1, \end{cases} \tag{2.12}$$

where c is the length-d context of x_t, and for the length-$(d+1)$ context c' of x_t we set

$$\mathcal{E} = \begin{cases} \emptyset, & \text{if } d = \ell, \\ \mathcal{X}_{c'}(x_{<t}), & \text{otherwise.} \end{cases}$$

Sequential Recursive PPM Mixture. The distribution (2.12) computed by PPM does not seem to be a mixture distribution. However, with a little extra effort, we can observe that the first two cases in (2.12) collapse to the linear mixture

$$\mathsf{PPM}_d(x; x_{<t}) = (1 - e(x^c_{<t})) \cdot \mathsf{MDL}(x; x^c_{<t}, \mathcal{E}) + e(x^c_{<t}) \cdot \mathsf{PPM}_{d-1}(x; x^c_{<t}). \tag{2.13}$$

Note that the escape probability essentially plays the role of a weight in a weighted linear average of two probability distributions.

Let us now justify (2.13). Keep in mind that we have $c = x_{t-d:t-1}$, since c is the length-d context of x_t. In the first case of (2.12), the letter x appears within the context history $x^c_{<t}$ and we conclude (2.13) by adding

$$\begin{aligned} \mathsf{PPM}_d(x; x_{<t}) &\overset{(2.12)}{=} \mathsf{MDL}^c(x; x^c_{<t}, \mathcal{E}) \overset{(2.10)}{=} (1 - e(x^c_{<t})) \cdot \mathsf{MDL}(x; x^c_{<t}, \mathcal{E}) \\ 0 &= e(x^c_{<t}) \cdot \mathsf{PPM}_{d-1}(x; x_{<t}). \end{aligned}$$

To see that $\mathsf{PPM}_{d-1}(x; x_{<t}) = 0$ we distinguish:

$$\begin{aligned} d = 0 &\implies \mathsf{PPM}_{d-1}(x; x_{<t}) \overset{(2.12)}{=} u(x; x_{<t}) \overset{(2.11)}{=} 0, \quad \text{since } x \in \{x_1, \ldots, x_{t-1}\}; \\ d > 0 &\implies \mathsf{PPM}_{d-1}(x; x_{<t}) \overset{(2.12)}{=} \mathsf{MDL}^{c'}(x; x^{c'}_{<t}, \mathcal{E}) \overset{(2.9)}{=} 0, \end{aligned}$$

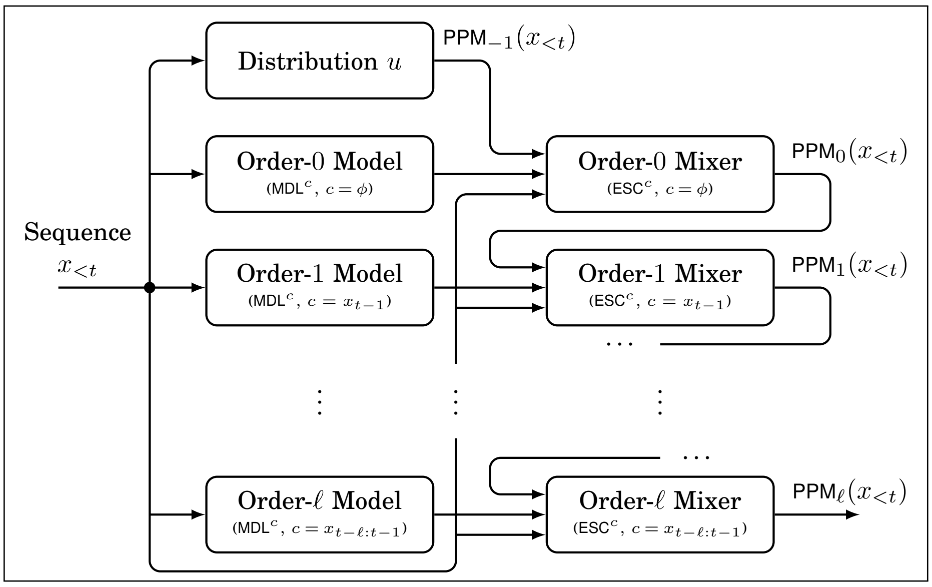

Distribution u
$\mathsf{PPM}_{-1}(x_{<t})$
Order-0 Model
(MDL^c, $c=\phi$)
Order-0 Mixer
(ESC^c, $c=\phi$)
$\mathsf{PPM}_0(x_{<t})$
Sequence
$x_{<t}$
Order-1 Model
(MDL^c, $c=x_{t-1}$)
Order-1 Mixer
(ESC^c, $c=x_{t-1}$)
$\mathsf{PPM}_1(x_{<t})$
Order-ℓ Model
(MDL^c, $c=x_{t-\ell:t-1}$)
Order-ℓ Mixer
(ESC^c, $c=x_{t-\ell:t-1}$)
$\mathsf{PPM}_\ell(x_{<t})$

Figure 2.2: Structure of the sequential recursive PPM mixture.

since $x \in \mathcal{E} = \mathcal{X}_c(x_{<t})$, where $c' = x_{t-d+1:t-1}$ is length-$(d-1)$ context of x_t. In case two of (2.12), the letter x does not appear in context history $x^c_{<t}$, and we get

$$\mathsf{PPM}_d(x; x_{<t}) \overset{(2.12)}{=} \mathsf{MDL}^c(\mathrm{esc}; x^c_{<t}) \cdot \mathsf{PPM}_{d-1}(x; x_{<t}) \overset{(2.10)}{=} e(x^c_{<t}) \cdot \mathsf{PPM}_{d-1}(x; x_{<t}),$$
$$0 \overset{(2.9)}{=} (1 - e(x^c_{<t})) \cdot \mathsf{MDL}^c(x; x^c_{<t}, \mathcal{E}).$$

Again, adding the above equations yields (2.13).

Summary. PPM has an integer parameter $D \geqslant 0$, the maximum context length, and employs an elementary model MDL and its context model counterpart MDL^c, see (2.9) and (2.10). For escape probability assignment $e(\cdot)$ mixing in PPM uses the context-conditioned mixer

$$\mathsf{ESC}^c(x; x_{<t}, \boldsymbol{p}_{1:t}) := e(x_{<t}) \cdot u(x) + (1 - e(x_{<t})) \cdot v(x), \text{ where } \boldsymbol{p}_t = (u, v). \tag{2.14}$$

Given time steps $\mathcal{T} = \mathcal{T}_c(x_{1:t})$, the mixer ESC^c associated to context c has mixer input

$$\mathrm{in}^c(x_{<t}) = \langle (\mathsf{MDL}^c(x^c_{<i}, \mathcal{E}^c_i), \mathsf{PPM}_{|c|-1}(x_{<i})) \rangle_{i \in \mathcal{T}},$$

where $\mathcal{E}_i^c$ is the set of excluded letters at context c during time step i. For the length-d context $c = x_{t-d:t-1}$ of x_t PPM defines the recursive probability assignment

$$\mathsf{PPM}_d(x_{<t}) := \begin{cases} \mathsf{ESC}^c(x_{<t}^c, \mathrm{in}^c(x_{<t})), & \text{if } d \geqslant 0, \\ u(x_{<t}), & \text{otherwise} \end{cases} \tag{2.15}$$

and predicts the distribution $\mathsf{PPM}(x_{<t}) = \mathsf{PPM}_\ell(x_{<t})$, where $\ell \leqslant D$ is the length of longest context of x_t with non-empty context history. The actual choice of escape probability assignment and elementary modeling distinguishes different PPM variants.

If we break apart the tail recursion in the mixture (2.15) (the mixer input in^c depends on PPM_{d-1}!), we observe the following: At the bottom level of recursion PPM mixes the distribution u and the prediction of the order-0 context model MDL^c, where $c = \phi$, resulting in the distribution $\mathsf{PPM}_0(x_{<t})$. At the next level of recursion, PPM mixes the prediction of the order-1 context model MDL^c, where $c = x_{t-1}$ and $\mathsf{PPM}_0(x_{<t})$, resulting in the distribution $\mathsf{PPM}_1(x_{<t})$. This process continues up to the longest context with non-empty context history (length ℓ). Figure 2.2 illustrates this process. Common encoding and decoding routines, as in Figure 2.1, are efficient implementations that interleave mixing, modeling and coding to avoid the explicit computation of the distribution $\mathsf{PPM}(x_{<t})$.

Historical Notes. The earliest PPM variants, PPMA and PPMB, were introduced in [23], followed by PPMC [67], PPMD [43] (see Table 2.1) and, later, PPME [5]. Variants PPMP and PPMX [106] use an approximate poisson process model to estimate escape probabilities. All of these PPM variants mainly differ in the choice of escape probability, although there are minor differences in elementary modeling. In contrast to these members of the PPM family, PPM* [22] does not limit the maximum context length. Subsequently, two dissertations [4, 15] systematically investigated various PPM variations, e. g. local order estimation (choose ℓ in Figure 2.1 based on more sophisticated criteria), blending (recursively mix distributions without excluding letters) and update exclusions (if letter x_t was coded in length-d context, do not update statistics of contexts with length less than d), just to name a few. PPMZ [12], a carefully engineered PPM variant, broke the traditional approach for escape estimation: Instead of using a closed form expression on the escape probability, it employed a context model for escape probability estimation. Such a procedure is termed *secondary escape estimation*. Many other carefully chosen refinements to PPMD, resulted in PPMII [84] and its implementation PPMonstr (and the stripped-down version

PPMd). A major innovation of PPMII is *information inheritance*, a heuristic to initialize and maintain statistics for a novel child context based on statistics from its parent context. Up until now, the most recent member of the PPM family is PPMDP [85], a PPM variant that tunes parameters via Online Gradient Descent (OGD). Out of all PPM algorithms we mentioned, PPMII, followed by PPMDP, are state of the art. These two variants offer excellent empirical compression performance.

2.4.3 Context Tree Weighting

In 1983 Rissanen proposed the CONTEXT algorithm [73] to combine elementary models within a context tree by picking just a single elementary model. An improvement which was suggested later is that instead of selecting a single model, mixing multiple elementary models improves performance and still retains good code length guarantees [102]. This gave birth to Context Tree Weighting (CTW). The probability assignment in CTW may be stated in two equivalent ways, by using a *block probability recursive mixture*, or, alternatively, using a *sequential recursive mixture*. The former approach follows the original paper [101] and is well known. Unfortunately, it does not fit the mixture model scheme of Section 1.2. The latter approach was introduced as an implementation technique [105, 78] and received less attention. However, it resembles the mixture model scheme of Section 1.2.

Below we first give an introduction to CTW based on block probabilities, following the original work [101], and discuss code length guarantees. Thereafter, we explain how to arrive at the sequential recursive CTW mixture, summarize CTW and give some historical notes on CTW variations.

Block Probability Recursive CTW Mixture. Like any statistical data compression algorithm CTW is essentially defined by a sequential probability assignment rule. We will now explain how CTW assigns the probability $\mathsf{CTW}(x; x_{<t})$ to letter x after processing the segment $x_{<t}$. Our explanation is based upon probability estimates on sequences, rather than letters. Such a probability estimate is called *block probability* (of some sequence). In the following we restrict our view to a binary alphabet. The extension to a non-binary alphabet is straightforward [8, 100].

Consider a context tree of depth D (a parameter of CTW) with 2^d contexts on level $d \in \{0, 1, \dots, D\}$. Every leaf- and non-leaf context c is labeled with a probability $P_e^c(x_{<t})$ and every non-leaf context c is labeled with a probability $P_w^c(x_{<t})$, to be defined shortly. For some context c these number(s) estimate the block probability of the context history $x_{<t}^c$, given $x_{<t}$. (Note that these numbers are probabilities, not distributions; and that they are probabilities

on sequences, not on letters!) The block probabilities rely on the elementary model RF with parameter $f_0 = {}^1\!/_N$ (see (2.3)). This elementary model is called the Krichevsky-Trofimov-Estimator, we write KT to refer to this model. The model KT assigns probabilities by

$$\mathsf{KT}(x; x_{1:t}) := \frac{|\{i \mid x_i = x \text{ and } 1 \leqslant i \leqslant t\}| + \frac{1}{2}}{t + \frac{N}{2}}. \tag{2.16}$$

For $\mathcal{T} = \mathcal{T}_c(x_{<t})$ the block probability $P_e^c(x_{<t})$ only depends on the estimates (2.16) by

$$P_e^c(x_{<t}) := \prod_{i \in \mathcal{T}} \mathsf{KT}(x_i; x_{<i}^c). \tag{2.17}$$

The block probability $P_w^c(x_{<t})$ depends on $P_e^c(x_{<t})$ and on probabilities P_w^{0c} and P_w^{1c} from child contexts of c, and is defined by

$$P_w^c(x_{<t}^c) := \begin{cases} P_e^c(x_{<t}), & \text{if } |c| = D, \\ \frac{1}{2} P_w^{0c}(x_{<t}) P_w^{1c}(x_{<t}) + \frac{1}{2} P_e^c(x_{<t}), & \text{if } |c| < D. \end{cases} \tag{2.18}$$

Example 2.4.3 *We consider CTW for a binary alphabet with context tree depth* $D = 2$ *on input* $x_{<t} = 1011101$ *with initial context* $x_{-1:0} = 11$ *(we supply* $x_{-1:0}$ *to make the context of* x_1 *and* x_2 *well-defined), see "Block Probability Recursive CTW Mixture" in Figure 2.3. For instance the context* 11 *induces context history* 1010 *and*

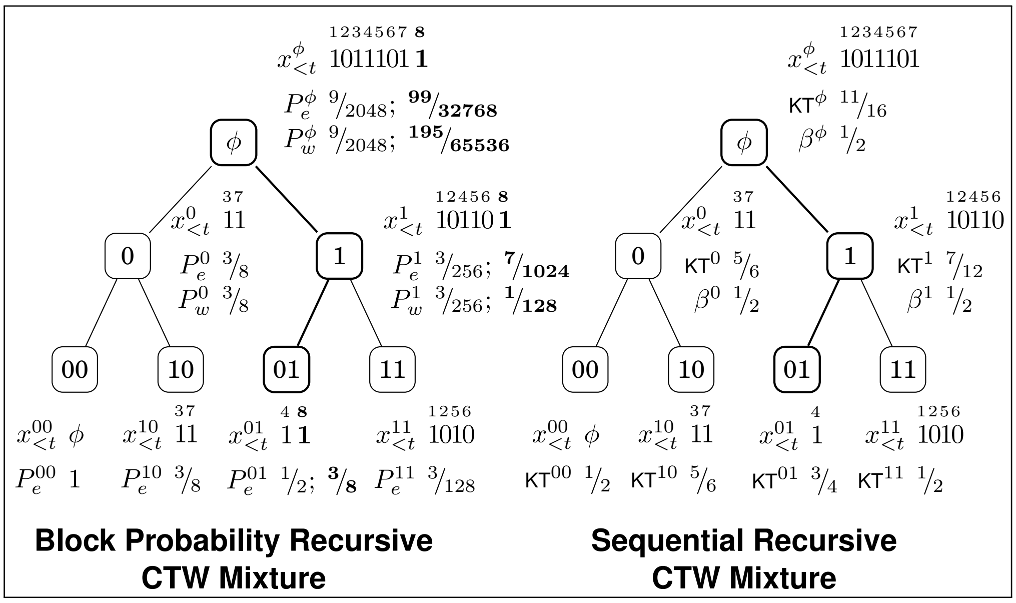
$$\begin{aligned} P_e^{11}(1010) &= \mathsf{KT}^{11}(1; \phi) \cdot \mathsf{KT}^{11}(0; 1) \cdot \mathsf{KT}^{11}(1; 10) \cdot \mathsf{KT}^{11}(0; 101) \\ &= \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{3}{8} = \frac{3}{128}, \end{aligned}$$

context 1 *has weighted probability*

$$\begin{aligned} P_w^1(101101) &= \frac{P_w^{01}(1) \cdot P_w^{11}(1010) + P_e^1(101101)}{2} \\ &= \frac{1/2 \cdot 3/128 + 3/256}{2} = \frac{3}{256}. \end{aligned}$$

To compute $\mathsf{CTW}(1; x_{<t})$ *we first save the value of* $P_w^{\phi}(x_{<t}) = 9/2048$*, compute* $P_w^{\phi}(x_1 \dots x_{t-1} 1) = 195/65536$ *(in Figure 2.3 contexts* c *with changing values of* P_e^c, P_w^c *and* x^c *and changed values themself are typeset bold) and finally obtain*

$$\mathsf{CTW}(1; x_{<t}) = \frac{P_w^{\phi}(10111011)}{P_w^{\phi}(1011101)} = \frac{195/65536}{9/2048} = \frac{65}{96}.$$



1234567 **8**
$x^{\phi}_{<t}$ 1011101 **1**
P^{ϕ}_e $^9/_{2048}$; $\mathbf{^{99}/_{32768}}$
P^{ϕ}_w $^9/_{2048}$; $\mathbf{^{195}/_{65536}}$
ϕ
37
$x^0_{<t}$ 11
0
P^0_e $^3/_8$
P^0_w $^3/_8$
12456 **8**
$x^1_{<t}$ 10110 **1**
1
P^1_e $^3/_{256}$; $\mathbf{^7/_{1024}}$
P^1_w $^3/_{256}$; $\mathbf{^1/_{128}}$
00
10
01
11
$x^{00}_{<t}$ ϕ
P^{00}_e 1
37
$x^{10}_{<t}$ 11
P^{10}_e $^3/_8$
4 **8**
$x^{01}_{<t}$ 1 **1**
P^{01}_e $^1/_2$; $\mathbf{^3/_8}$
1256
$x^{11}_{<t}$ 1010
P^{11}_e $^3/_{128}$
Block Probability Recursive CTW Mixture
1234567
$x^{\phi}_{<t}$ 1011101
KT^{ϕ} $^{11}/_{16}$
β^{ϕ} $^1/_2$
ϕ
37
$x^0_{<t}$ 11
0
KT^0 $^5/_6$
β^0 $^1/_2$
12456
$x^1_{<t}$ 10110
1
KT^1 $^7/_{12}$
β^1 $^1/_2$
00
10
01
11
$x^{00}_{<t}$ ϕ
KT^{00} $^1/_2$
37
$x^{10}_{<t}$ 11
KT^{10} $^5/_6$
4
$x^{01}_{<t}$ 1
KT^{01} $^3/_4$
1256
$x^{11}_{<t}$ 1010
KT^{11} $^1/_2$
Sequential Recursive CTW Mixture

Figure 2.3: Binary CTW with context tree depth $D = 2$ for the input $x_{<t} = 1011101$, where we assume the initial context $x_{-1}x_0 = 11$. Every node is labeled with its context c, and for every context c we depict: Context history $x^c_{<t}$ (number i above a bit b of $x^c_{<t}$ means $b = x_i$); estimated block probability $P^c_e = P^c_e(x_{<t})$; weighted block probability $P^c_w = P^c_w(x_{<t})$, where boldface numbers are values for the sequence $x_1x_2 \ldots x_{t-1}1 = 10111011$; weight $\beta^c = \beta^c(x^c_{<t})$ and elementary model estimate KT^c given by the probability $\mathsf{KT}(1; x^c_{<t})$ of a 1-bit.

Given the above definitions we may finally define the probability assignment used by CTW:

$$\mathsf{CTW}(x; x_{<t}) := \frac{P^{\phi}_w(x_1 \ldots x_{t-1}x)}{P^{\phi}_w(x_{<t})}. \tag{2.19}$$

For an illustration see Example 2.4.3 and Figure 2.3 (left part).

Theoretical Properties. CTW is guaranteed to perform not much worse (to be quantified shortly) than the following competing scheme:

Definition 2.4.4 *For a context tree of depth D with leaf context set $\mathcal{C}_L$ and family $\{p^c\}_{c\in\mathcal{C}_L}$ of associated probability distributions let $\mathsf{PCT}\langle\mathcal{C}_L, \{p^c\}_{c\in\mathcal{C}_L}\rangle$ denote a* Prediction Context Tree (PCT) model. *For a sequence $x_{-D+1:n}$ this PCT predicts $\mathsf{PCT}(x_{1:t}) = p^c$, where x_{t+1} has context c; the sequence $x_{1:n}$ receives code length*

$$\ell(x_{1:n}; \mathsf{PCT}) = \sum_{c\in\mathcal{C}_L} \ell(x^c_{1:n}; p^c).$$

It might look strange that Definition 2.4.4 specifies the segment $x_{-D+1:0}$, but this segment does not contribute to the code length $\ell(x_{1:n}; \mathsf{PCT})$. The explanation is simple, if we do not specify $x_{-D+1:0}$, then the contexts of $x_1, x_2, \dots x_D$ would be undefined and in turn $\mathsf{PCT}(x; x_{<t})$ would be undefined for $1 \leqslant t \leqslant D$. In the remainder of this work we always assume that $x_{-D+1:0}$ is known and omit it. Creating this situation in reality just requires to transmit the first D letters $x_{-D+1:0}$ of an input in $O(\log N)$ bits each. Alternatively, we can assume an arbitrary fixed context $x_{-D+1:0}$ for every input $x_{1:n}$.

A PCT is frequently called Tree Source [101] or Prediction Suffix Tree [10]. In the original work [101] on CTW it was shown that for a sequence $x_{1:n}$ and an arbitrary PCT model $\mathsf{PCT}\langle \mathcal{C}_L, \{p^c\}_{c\in\mathcal{C}_L}\rangle$ we have

$$\ell(x_{1:n}; \mathsf{CTW}) \leqslant \ell(x_{1:n}; \mathsf{PCT}) + |\mathcal{C}_L| \gamma\left(\frac{n}{|\mathcal{C}_L|}\right) + \Gamma_D(\mathcal{C}_L), \tag{2.20}$$

where

$$\gamma(z) := \begin{cases} z, & \text{if } 0 \leqslant z \leqslant 1, \\ \frac{1}{2}\log z + 1, & \text{if } z > 1, \end{cases}$$

$$\Gamma_D(\mathcal{C}_L) := |\{c \mid c \text{ is a context in the context tree induced by } \mathcal{C}_L \text{ and } |c| < D\}|. \tag{2.21}$$

We may interpret the terms in the bound (2.20) as follows: $|\mathcal{C}_L|\gamma({}^{n}/_{|\mathcal{C}_L|})$ accounts for estimating the probability distributions $\{p^c\}_{c\in\mathcal{C}_L}$ at each leaf context from $\mathcal{C}_L$ and $\Gamma_D(\mathcal{C}_L)$ bounds the cost of estimating the tree structure of the given PCT (in fact one may encode the tree structure in exactly $\Gamma_D(\mathcal{C}_L)$ bits [101]). In Chapter 5 we will take a closer look. One should note that the actual PCT is unknown to CTW and that bound (2.20) holds simultaneously for all PCTs of depth at most D.

Sequential Recursive CTW Mixture. We will now refine the formulation of CTW based on block probabilities to obtain a sequential recursive mixture, similar to (2.13), based on Beta-Weighting (see (2.4)). Such a refinement was partially done before, as a matter of an efficient CTW implementation, but not exactly in the way we do it here. (Even though suggested in [101] it is cumbersome and inefficient to implement CTW using block probabilities. One should use conditional probabilities [78, 105].) Our refinement is close to [78, 105] with modifications due to [49]. At the end of this section we will explain similarities and differences to previous work.

First, for the length-d context c of x_t, where $0 \leqslant d \leqslant D$, we define the model

$$\mathsf{CTW}_d(x; x_{<t}) := \frac{P_w^c(x_1x_2\dots x_{t-1}x)}{P_w^c(x_{<t})}, \text{ where } c = x_{t-d:t-1}. \tag{2.22}$$

(For P_w^c and P_e^c see (2.18) and (2.17).) Clearly, by (2.19) CTW predicts the distribution $\mathsf{CTW}(x_{<t}) = \mathsf{CTW}_0(x_{<t})$. In the following we will turn definition (2.22) of CTW_d into a recursive formulation. In the base case, when $d = D$, we obtain

$$\begin{aligned}\mathsf{CTW}_d(x; x_{<t}) &\overset{(2.22)}{=} \frac{P_w^c(x_1x_2\dots x_{t-1}x)}{P_w^c(x_{<t})}\\ &\overset{(2.18)}{=} \frac{P_e^c(x_1x_2\dots x_{t-1}x)}{P_e^c(x_{<t})} \overset{(2.17)}{=} \mathsf{KT}(x; x_{<t}^c);\end{aligned} \tag{2.23}$$

in the recursive case, when $d < D$, we have

$$\begin{aligned}\mathsf{CTW}_d(x; x_{<t}) &\overset{(2.22)}{=} \frac{P_w^c(x_1\dots x_{t-1}x)}{P_w^c(x_{<t})}\\ &\overset{(2.18)}{=} \frac{1}{2}\frac{P_w^{0c}(x_1\dots x_{t-1}x)P_w^{1c}(x_1\dots x_{t-1}x)}{P_w^c(x_{<t})} + \frac{1}{2}\frac{P_e^c(x_1\dots x_{t-1}x)}{P_w^c(x_{<t})}\\ &= \underbrace{\frac{1}{2}\frac{P_w^{0c}(x_{<t})P_w^{1c}(x_{<t})}{P_w^c(x_{<t})}}_{=:\,\beta^c(x_{<t})} \cdot \frac{P_w^{0c}(x_1\dots x_{t-1}x)P_w^{1c}(x_1\dots x_{t-1}x)}{P_w^{0c}(x_{<t})P_w^{1c}(x_{<t})}\\ &\qquad + \frac{1}{2}\frac{P_e^c(x_{<t})}{P_w^c(x_{<t})} \cdot \frac{P_e^c(x_1\dots x_{t-1}x)}{P_e^c(x_{<t})}.\end{aligned} \tag{2.24}$$

To proceed, we make two observations: (i) By plugging the definition of β^c from (2.24) into case $|c| < D$ of (2.18) we get

$$P_w^c(x_{<t}) \overset{(2.18)}{=} \beta^c(x_{<t}) \cdot P_w^c(x_{<t}) + \frac{P_e^c(x_{<t}^c)}{2} \implies 1 - \beta^c(x_{<t}) = \frac{1}{2}\frac{P_e^c(x_{<t})}{P_w^c(x_{<t})}. \tag{2.25}$$

(ii) If x_t *does not have* context z, then $P_w^z(x_1\dots x_{t-1}x) = P_w^z(x_{<t})$, for any letter x. (One may easily prove this by induction on $|z| = D, D-1, \dots, 0$; However, it is also evident from Figure 2.3: x_t has contexts from $\mathcal{C} = \{\phi, 1, 01\}$ and block probabilities $P_w^c(x_1\dots x_{t-1}1)$ and $P_w^c(x_{<t})$ match, for all contexts except those from $\mathcal{C}$.) Since x_t has length-$(d+1)$ context $c' = x_{t-d-1:t-1}$ we conclude

$$\frac{P_w^{0c}(x_1\dots x_{t-1}x)P_w^{1c}(x_1\dots x_{t-1}x)}{P_w^{0c}(x_{<t})P_w^{1c}(x_{<t})} = \frac{P_w^{c'}(x_1\dots x_{t-1}x)}{P_w^{c'}(x_{<t})} \overset{(2.22)}{=} \mathsf{CTW}_{d+1}(x; x_{<t}). \tag{2.26}$$

Let us now step back to (2.24) and apply the two earlier observations (i) and (ii): We plug $P_e^c(x_1 \dots x_{t-1}x)/P_e^c(x_{<t}) = \mathsf{KT}(x; x^c_{<t})$ (by (2.17)) and equations (2.25) and (2.26) into (2.24), the result is

$$\mathsf{CTW}_d(x; x_{<t}) = \beta^c(x_{<t}) \cdot \mathsf{CTW}_{d+1}(x; x_{<t}) + (1 - \beta^c(x_{<t})) \cdot \mathsf{KT}(x; x^c_{<t}). \quad (2.27)$$

At this point we successfully turned definition (2.22) of $\mathsf{CTW}_d(x_{<t})$ into a sequential recursive mixture that does not rely on block probabilities. If $d = D$, then CTW_d collapses to the elementary model KT, see (2.23); if $d < D$, then CTW_d is a linear mixture, see (2.27), which combines the model CTW_{d+1}, depending on length-$(d+1)$ context of x_t, and the elementary model KT conditioned on the length-d context c of x_t. Example 2.4.5 demonstrates how to compute the recursive mixture.

Example 2.4.5 *We consider exactly the same situation as in Example 2.4.3 and want to compute* $\mathsf{CTW}(1; x_{<t})$*. The part "Sequential Recursive CTW Mixture" (right) in Figure 2.3 matches the situation of "Block Probability Recursive CTW Mixture" (left), except that we employ the modified representation of CTW described in the text above. So contexts do not hold block probabilities, but estimates* $\mathsf{KT}(1; x^c_{<t})$ *of the* KT *elementary model and mixture weights* $\beta^c(x_{<t})$*. Since* $x_{<t} = 1011101$*, the letter* x_t *has contexts* ϕ, 1, 01*, we obtain*

$$\begin{aligned}
\mathsf{CTW}_2(1; x^{01}) &= \mathsf{KT}(1; x^{01}_{<t}) = \frac{3}{4} \\
\mathsf{CTW}_1(1; x^1) &= \beta^1(x_{<t}) \cdot \mathsf{CTW}_2(1; x_{<t}) + (1 - \beta^1(x_{<t})) \cdot \mathsf{KT}_1(1; x^1_{<t}) \\
&= \frac{1}{2} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{7}{12} = \frac{2}{3} \\
\mathsf{CTW}(1; x) &= \beta^\phi(x_{<t}) \cdot \mathsf{CTW}_1(1; x_{<t}) + (1 - \beta^\phi(x_{<t})) \cdot \mathsf{KT}(1; x^\phi_{<t}) \\
&= \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{11}{16} = \frac{65}{96},
\end{aligned}$$

just as in Example 2.4.3.

Notice that the mixture weight $\beta^c(x_{1:t})$ itself can be computed recursively: If $t = 0$, then we have $x_{1:t} = \phi$ and $P_e^c(\phi) = P_w^c(\phi) = 1$ for every context c (one may easily prove this by induction on $|c| = D, D-1, \dots, 0$, keeping in mind that the empty product evaluates to 1), so

$$\beta^c(\phi) = \frac{1}{2} \frac{P_w^{0c}(\phi) P_w^{1c}(\phi)}{P_w^c(\phi)} = \frac{1}{2}. \quad (2.28)$$

If $t > 0$, we use (2.22) and (2.26) with $x = x_t$ and get

$$\beta^c(x_{1:t}) = \frac{1}{2}\frac{P_w^{0c}(x_{1:t})P_w^{1c}(x_{1:t})}{P_w^c(x_{1:t})} = \beta^c(x_{<t}) \cdot \frac{P_w^{0c}(x_{1:t})P_w^{1c}(x_{1:t})}{P_w^{0c}(x_{<t})P_w^{1c}(x_{<t})} \cdot \frac{P_w^c(x_{<t})}{P_w^c(x_{1:t})}$$
$$\overset{(2.22),\,(2.26)}{=} \beta^c(x_{<t}) \cdot \frac{\mathsf{CTW}_{d+1}(x_t; x_{<t})}{\mathsf{CTW}_d(x_t; x_{<t})}. \tag{2.29}$$

By looking closely at the mixture formula (2.27) and at the recursive weight definition (2.28), (2.29), we see that this is a special case of Beta-Weighting for a mixer input of dimension $m = 2$, cf. (2.4).

As we already noted, similar refinements (of the block probability recursive mixture (2.19) to the sequential recursive mixture (2.23) and (2.27)) have previously been proposed in [78, 105]. More precisely, for $d < D$ (2.22) was transformed into

$$\mathsf{CTW}_d(x; x_{<t}) = \frac{\alpha^c(x_{<t})}{1+\alpha^c(x_{<t})} \cdot \mathsf{CTW}_{d+1}(x; x_{<t}) + \frac{1}{1+\alpha^c(x_{<t})} \cdot \mathsf{KT}(x; x_{<t}^c),$$
$$\alpha^c(x_{<t}) := \frac{P_e^c(x_{<t})}{P_w^{0c}(x_{<t})P_w^{1c}(x_{<t})} \quad \left(= \frac{1-\beta_c(x_{<t})}{\beta_c(x_{<t})}\right),$$

where the term $\alpha^c(x_{<t})$ may be stated recursively, similar to $\beta^c(x_{<t})$ in (2.28) and (2.29). In [49] Kufleitner proposed Beta-Weighting to combine an arbitrary number of models (not in the scope of CTW) using a linear mixture. He pointed out that Beta-Weighting has its roots in CTW, without giving an exact link. Our refinement of CTW makes the connection explicit. In Chapter 4 we will examine Beta-Weighting in greater detail.

Summary. CTW has an integer parameter $D \geqslant 0$, the maximum context length, and utilizes the elementary model KT and the mixture BETA. Any context c of depth at most D has a context model $\mathsf{KT}^c = \mathsf{KT}$, where a context c of depth less than D also has a context conditioned mixer $\mathsf{BETA}^c = \mathsf{BETA}$. Based on time steps $\mathcal{T} = \mathcal{T}_c(x_{1:t})$, the mixer BETA^c has the mixer input

$$\mathrm{in}^c(x_{<t}) = \langle \mathsf{KT}^c(x_{<i}^c), \mathsf{CTW}_{|c|+1}(x_{<i})\rangle_{i\in\mathcal{T}}.$$

For any length-d context $c = x_{t-d:t-1}$ of x_t CTW defines the probability assignment

$$\mathsf{CTW}_{|c|}(x_{<t}) = \begin{cases} \mathsf{BETA}^c(x_{<t}^c, \mathrm{in}^c(x_{<t})), & \text{if } 0 \leqslant |c| < D, \\ \mathsf{KT}^c(x_{<t}^c), & \text{if } |c| = D \end{cases} \tag{2.30}$$

and predicts $\mathsf{CTW}(x_{<t}) = \mathsf{CTW}_0(x_{<t})$.

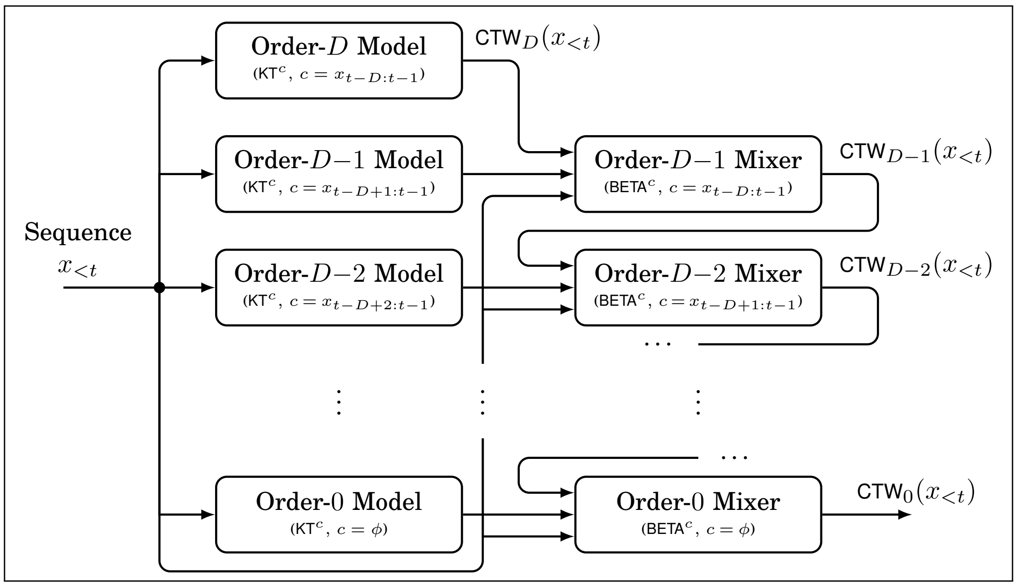


Figure 2.4: Structure of the sequential recursive CTW mixture.

If we expand the recursion in (2.30), we observe the following: At the bottom level of recursion CTW combines prediction of an order-D model (KT$^c(x_{<t})$, where $c = x_{t-D:t-1}$) and the prediction of an order-$(D-1)$ model (KT$^c(x_{<t})$, where $c = x_{t-D+1:t-1}$) using an order-$(D-1)$ mixer (BETA^c, where $c = x_{t-D+1:t-1}$), resulting in the distribution CTW$_{D-1}(x_{<t})$; at the next level of recursion CTW combines the prediction CTW$_{D-1}(x_{<t})$ and the prediction of an order-$(D-2)$ model, resulting in the distribution CTW$_{D-2}(x_{<t})$, and so on. We sketch this situation in Figure 2.4.

It is interesting to note that the structure (the way models and mixers interact) of PPM and CTW mixture models share great similarity, cf. Figure 2.2 and Figure 2.4. To the best of our knowledge, this has not been observed before.

Historical Notes. As we already stated, the most influential work on CTW is [101], although there exists prior work on CTW for an N-ary alphabet [100]. CTW for a binary alphabet, coupled with implementation techniques to handle N-ary alphabets by alphabet decomposition (every letter receives a binary codeword) [103, 88], improves the empirical compression over CTW for a N-ary alphabet [8]. Consequently, most CTW research assumes a binary alphabet. Extensions to the original work include removing the dependence of CTW on a full length-D context of the first D input letters (these actually do not have D preceding letters, i. e. do not have a context of length D!) and on the parameter D itself [98]. Recently, Veness proposed Context Tree

Switching (CTS) [91], a CTW-spinoff that replaces the mixture within CTW, resulting in similar theoretical properties and improved empirical performance. An extension to CTS is SkipCTS [10], which allows for "*don't-care*"-letters within a context (tree). SkipCTS consequently extends code length guarantees to a wider class of context trees and, in turn, improves empirical performance. Similarly to other statistical compression algorithms CTW is demanding in terms of processing power and memory requirements, so researchers tried to cut down time and space complexity [103, 104, 78].

2.4.4 PAQ

Among CTW and PPM, PAQ is the most recent member of the major three statistical data compression algorithms. The PAQ family of algorithms was introduced in 2002 [54] and has undergone heavy development since then [53, 80]. As a result there are countless PAQ variants, and it is impossible to sketch "the PAQ algorithm". Nevertheless, there are two things all PAQ variants have in common, namely that they work on a binary alphabet and that they push the modeling-mixing approach to the extreme: Mixers combine a large number (tens and even up to hundreds) of context models to compute a single prediction. At that context models use a generous interpretation of what makes up a context. In Figure 2.5 we show the typical layout of a PAQ-based mixture model.

There is no PAQ model people have agreed on; rather, there are numerous variations. Still, there are key techniques shared by many PAQ variants. One may divide the key techniques into elementary modeling, mixing and other common key components. Below we first describe common key components (not related to elementary modeling), followed by a description of PAQ-style elementary modeling and mixing. Out of many elementary modeling and mixing approaches, corresponding to different PAQ variants, we restrict our attention to the most mature variants of PAQ7, since these remained unchanged from 2005 on. We conclude this section with historical notes that offer further references to the reader.

Key Components. Despite elementary modeling and mixing, the superior empirical performance of PAQ further rests on three pillars: data transforms, a huge amount of data-specific context models and special modeling techniques. Let us now summarize these. (For further reading we recommend [53, 52, 80].)

Various PAQ variants attempt to detect the underlying data type and, depending on it, enable special context models and/or apply data-specific

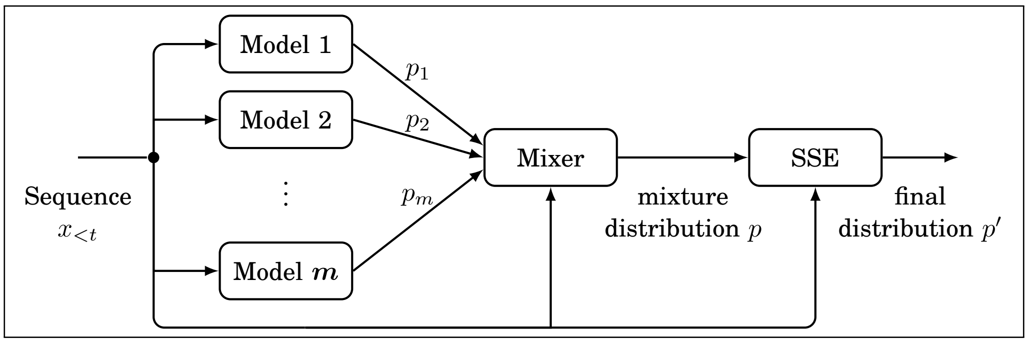


Figure 2.5: Typical architecture of a PAQ-based mixture model: Multiple context models predict the distributions $p_1, p_2, \dots, p_m$ on the next bit x_t of sequence $x_{<t}$; a mixer combines these distributions, this yields distribution p, which gets refined by a so-called Secondary Symbol Estimation (SSE) to produce the distribution p', which may be fed into a coding algorithm. The mixer and SSE may input $x_{<t}$ as well, since their operation can depend on a context.

reversible transforms. For instance, common words in text files may be replaced by shorter phrases or machine code in executables is (reversibly) altered to produce machine code that is easier to compress; in addition we may enable context models that are specialized for text/executable files. Some PAQ variants even revert Huffman coding in JPEG images and utilize specialized context models instead. Despite data-specific context models, PAQ employs a large amount of "generic" context models, e. g. various types of order-d context models (as in PPM), sparse context models (contexts that contain "don't cares") and a match model (the prediction largely depends on the length of current context). Apart from the PAQ approaches to elementary modeling and mixing (see below) there are two special modeling techniques found in many PAQ variants. First, there are context models conditioned on predicted distributions (and possibly other contexts), rather than just letters from the input sequence. This technique was derived from SEE in [84] and is called Secondary Symbol Estimation (SSE) [53, 80]. A precise treatment of SSE is beyond the scope of this work, see [52, 53, 80] for further reading. Second, sometimes a context history is not directly processed by an elementary model to obtain a prediction. Instead, a Finite State Machine (FSM) tracks the context history and, in turn, its current state is used as a context to look up an appropriate elementary model.

It is natural that removing various tricks and heuristics from PAQ will degrade its performance and might render PAQ inferior to PPM or CTW. So one may argue that these tricks are the only advantages of PAQ. However, in Chapter 5 we will see that in the vast majority of cases PAQ-style elemen-

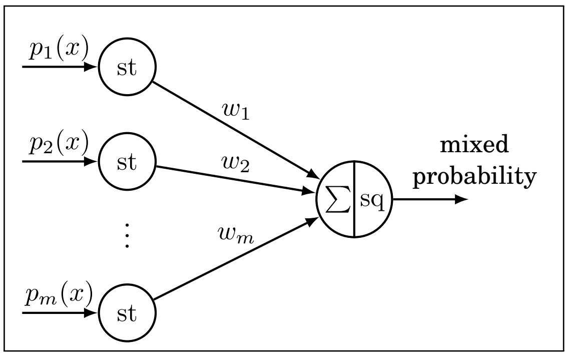


Figure 2.6: Logistic Mixing in PAQ interpreted as neural network.

tary modeling and mixing alone is superior to traditional approaches.

Elementary Modeling. Unlike most widespread attempts to elementary modeling, PAQ does no more use relative-frequency based elementary models. Instead, it adopts a technique we term Probability Smoothing (PS). Given an initial distribution p and a sequence $\alpha_{1:\infty}$ of numbers from $(0,1)$, PS is defined by the sequential probability assignment rule

$$\mathsf{PS}(x;x_{1:t}) := \begin{cases} \alpha_t \cdot \mathsf{PS}(x;x_{<t}) + 1 - \alpha_t, & \text{if } t > 0 \text{ and } x_t = x, \\ \alpha_t \cdot \mathsf{PS}(x;x_{<t}), & \text{if } t > 0 \text{ and } x_t \neq x, \\ p(x), & \text{if } t = 0 \end{cases}.$$

The sequence $\alpha_{1:\infty}$ is called *smoothing rate sequence*, hence the naming PS. For reasons of implementation efficiency PAQ commonly employs a fixed smoothing rate, $\alpha = \alpha_1 = \alpha_2 = \ldots$, where α is close to one. Another common choice is to increase smoothing rates and leave them fixed from some stage on: $\alpha_1 < \alpha_2 < \cdots < \alpha_k = \alpha_{k+1} = \ldots$. In Chapter 3 we provide a code length analysis for various smoothing rate sequences considering a binary alphabet.

Logistic Mixing. For mixing PAQ draws strength from a neural-network approach (see [75] for a basic introduction to neural networks) called Logistic Mixing (LM) [53, 80]. Structurally, the neural network is simple: It consists of m input neurons, one neuron per distribution to mix, and a single output neuron. Every input neuron is connected with the single output neuron and the connecting edge owns a real-valued weight. Figure 2.6 sketches the setting. Every input neuron has the transfer function

$$\mathrm{st}(z) := \ln \frac{z}{1-z} \text{ (“stretch”), for } 0 < z < 1; \tag{2.31}$$

the output neuron relies on dot product activation and on the logistic transfer function

$$\mathrm{sq}(z) := \frac{1}{1+e^{-z}} \text{ (“squash”), for } z \in \mathbb{R}. \tag{2.32}$$

If we mix distributions $p_1, p_2, \dots, p_m$, bit x receives the mixed probability

$$\mathrm{sq}\left(w_1 \cdot \mathrm{st}(p_1(x)) + \cdots + w_m \cdot \mathrm{st}(p_m(x))\right). \tag{2.33}$$

Note that the functions st and sq are the inverse of each other, so one may interpret mixing as follows: First, bit probabilities get transformed nonlinearly, then combined by a weighted linear average, and finally the transform is reversed. PAQ tunes the neural network weights via OGD, minimizing code length, rather than square error.

Summary. PAQ operates on a binary alphabet and employs the elementary model PS and LM for mixing. In addition to the enormous number of context models, a large set of heuristics and tricks contributes to the superior empirical performance of PAQ.

Historical Notes. In 2002 PAQ1 was introduced [54] as a simplified realization of the neural network-based compression algorithm proposed in [55]. Later PAQ variants improve compression by adding more (specialized) context models, preprocessing tricks and refinements to key components (see [53, 80] for a thorough discussion): Early variants (PAQ1, PAQ2, PAQ3) use elementary modeling and weighting based on relative frequencies and employ SSE since PAQ2. Later refinements (PAQ4, PAQ5, PAQ6) tune weights via OGD, PAQ6 [56] adds FSM-based elementary models. Starting from PAQ7, elementary modeling and mixing rely on the algorithms given above.

2.4.5 Others

Besides PPM, CTW and PAQ there are two other statistical data compression algorithms worth mentioning: Dynamic Markov Coding (DMC) and DEPLUMP. Both algorithms have quite a shadowy existence in the domain of statistical data compression. (DMC never gained as much attention as, e. g. PPM or CTW; DEPLUMP was published in 2010, thus is rather novel at the time of writing this thesis.) In terms of empirical compression, both, DMC and DEPLUMP are inferior to state of the art PPM and PAQ variants. As a matter of completeness we now sketch the statistical model of DMC and DEPLUMP.

Dynamic Markov Coding. DMC (also called Dynamic Markov Compression) pursues a fundamentally different approach to context modeling: Instead of conditioning elementary models on contexts, DMC conditions elementary models on states of a (huge) dynamically built FSM.

As initially proposed in [25] DMC operates on a binary alphabet. All states within the FSM have an associated elementary model and two outgoing edges, one edge corresponds to a 0-bit (the "0-edge"), the other edge corresponds to a 1-bit (the "1-edge"). (An outgoing edge points to a follow-up state, depending on an input bit.) DMC uses a relative frequency based elementary model similar to RF. Context modeling in DMC works as follows: Initially the FSM has a fixed structure and we use the elementary model associated to a predefined state, the initial state, to obtain a prediction on the first input bit. After we know about the input bit x, we examine a so-called *cloning condition*. If the cloning condition is satisfied, we perform cloning: We copy the state pointed to by the x-edge, including its outgoing transitions and elementary model, and alter the x-edge to point to the newly created state. No matter whether or not cloning took place, we follow the x-edge to reach the follow-up state which we use for prediction in the next round.

In general there is little literature on DMC available, possibly due to a lack of theoretical understanding. Present theoretical results [9, 14, 15] aim to characterize in which way states correspond to conditioning contexts. Unfortunately, there exist no code length guarantees for DMC, "the main justification of DMC is that it works" [80]. The DMC description we gave above lacks mixing, and it is currently unknown if an equivalent characterization of DMC (similar to the refinement of PPM in Section 2.4.2) that utilizes a mixture exists at all. A practical consideration, taken care of in [89], is the extension of DMC to a non-binary alphabet. It is interesting to note that the influential work [13] popularized DMC as a classifier in email spam filtering.

DEPLUMP. The DEPLUMP Algorithm was introduced in [34]. (PLUMP abbreviates Power Law Unbounded Markov Prediction, naming the compressor DEPLUMP turns this into a pun.) DEPLUMP assumes that the sequence we encounter is the outcome of a stochastic process called Sequence Memorizer (SM). Based on this DEPLUMP approximates the assumed SM, the approximation yields a sequential prediction rule.

DEPLUMP shares great similarities with PPM: It computes a recursive sequential mixture, similar to the PPM mixture (2.15). Elementary modeling is based on relative letter frequencies and mixing relies on a weighted linear average (just as PPM's escape weighting (2.14) and CTW's Beta-Weighting (2.4)). However, several features of DEPLUMP differ from (classical) PPM: DEPLUMP neither makes use of exclusion, nor does it bound context length.

As a consequence (of the unbounded context length) it requires time $O(n^2)$ for a sequence of length n, which is inappropriate for statistical data compression. So follow-up work [7] ironed out that deficit and proposed a refined DEPLUMP variant that requires time $O(n)$. Weights in the underlying recursive mixture depend on parameters (which may vary depending on context) that DEPLUMP tunes via Online Optimization techniques, such as OGD. Furthermore, some DEPLUMP variants update parameters probabilistically (a random experiment determines whether or not to update a parameter).

Although DEPLUMP is promising in terms of empirical compression performance, not much literature is available. (As we already stated, we believe that the reason is DEPLUMP's novelty.) Currently there exist no published code length guarantees for DEPLUMP.

2.5 Summary

In this chapter we laid out the fundamentals of this thesis. We have established the basic notation that will be with us throughout the remainder of this work. On this basis we subsequently introduced the two key components of every statistical data compressor, that is, models and mixers, to the reader.

As we have seen, a common recipe in statistical data compression is context modeling, i. e. conditioning models and mixers on contexts. In turn, we supplied additional notation in this respect. The technical evaluation of models, mixers and ultimately of whole statistical data compressors relies on the comparison of these entities to idealized competitors via a code length analysis (or redundancy analysis). Since there exist different approaches to a code length analysis, we have thoroughly described our approach and aligned it with literature. We do not assume that a source generated the sequences we encounter, rather we view sequences as arbitrary and individual. Our approach to a code length analysis aligns the coding performance of a model to that of an idealized competitor in terms of a multiplicative factor ("the model's code length is off any competitor's code length at most by the factor $1 + \delta \ldots$") and an additive redundancy term ("...plus some additive redundancy").

Based on the decomposition of a statistical data compressor's model into (context conditioned) submodels and mixers we provided descriptions of the models found in the three major statistical compression algorithms: PPM, CTW and PAQ. The framework we fit PPM and CTW to (i. e. the sequential recursive mixture) gives a unifying view on these algorithms and reveals similarities (see Figure 2.2 and Figure 2.4). Furthermore, we sketched the models of DMC and DEPLUMP and gave a short introduction to the standard coding algorithm in statistical data compression, AC.

CHAPTER 3

Elementary Modeling

CHAPTER OVERVIEW

In the present chapter we discuss elementary modeling techniques in great detail. We first introduce the general problem of elementary modeling to the reader, including practical limitations and common approaches. We provide a survey of elementary modeling techniques, followed by a description and a code length analysis of several elementary models. In doing so, we concentrate on elementary models that are widely used in practice, but yet poorly understood in a theoretical sense: Relative Frequencies with Discount, Probability Smoothing and Relative Frequencies with Smoothing. An experimental study that supports our code length bounds and a short summary end this chapter.

3.1 Introduction

The Setting. Sequential probability assignment is a key feature of any statistical data compression algorithm. More precisely, the model of a statistical data compressor attempts to solve the following fundamental problem:

For the sequence $x_1x_2\dots x_n$ of n letters, revealed letter by letter, estimate a distribution p_t on the t-th letter, given the sequence $x_1x_2\dots x_{t-1}$ known so far.

As we have seen in Chapter 2, the most basic approach to this problem relies on simple, closed-form expressions for sequential probability assignment, previously introduced as elementary models. In a statistical data compressor an elementary model associated to some context c solves the above problem for the context history $x^c_{<t}$, rather than for the input sequence $x_{<t}$; however, the general problem statement remains unchanged, so we may just consider elementary modeling for some input sequence $x_{<t}$. In this chapter we cover several practical and widespread approaches to elementary modeling. Unfortunately these approaches lack a sound theoretical basis. We will introduce such a theoretical basis by providing a code length analysis.

Design Constraints. Possible solutions to the above problem are limited by common complexity constraints.[1] More precisely, an elementary model can be considered practical if it may be implemented s. t. the following is true: It takes constant time to predict a single probability and constant time to update its state (e. g. increment the frequency of a letter), given the next letter x_t. Regarding space, the implementation ideally requires M words of memory, where $M \leqslant N$ is the number of distinct letters in the sequence $x_{1:n}$ (over an alphabet of size N). For small alphabets, such as a binary alphabet, a space requirement of $N + O(1)$ memory words is acceptable as well. Despite the aforementioned complexity constraints an elementary model should be able to adapt to varying statistics. For instance, for a sequence like $00\dots011\dots1$ an elementary model should quickly increase the probability of letter 1 right after the transition from the 0-block to the 1-block. This type of desired behavior takes locality into account and is likely to improve compression [43].

Elementary Modeling by Relative Letter Frequencies. Most practical approaches to elementary modeling follow a common design pattern. That is, they maintain an approximate frequency $f_t(x)$ of a letter x in the already processed sequence $x_{<t}$ (we use the term "frequency" in a wider sense, viz. to refer to frequency estimates and to refer to the actual frequency of some letter in a sequence) and assign probability

$$p_t(x) := \frac{f_t(x)}{F_t}, \text{ where } F_t = \sum_{x \in \mathcal{X}} f_t(x), \tag{3.1}$$

to the letter x in step t. For various reasons $f_t(x)$ does not need to match the actual frequency of letter x: Typically, the probability $p_t(x)$ must be non-zero, regardless of x, so $f_t(x)$ has to be non-zero as well, even if x does not appear in $x_{<t}$; a common workaround is to set $f_t(x)$ to the actual frequency of letter x within $x_{<t}$ plus some small positive additive constant. In addition, the frequencies $f_t(\cdot)$ may get discounted to achieve adaptivity. For instance, after we observe the letter $x_t = x$, we may multiply all frequencies by a discount factor from $(0, 1)$, before incrementing the frequency of letter x. In this way recent letters receive a higher weight than letters that appeared in earlier stages.

Simplicity is not the only virtue of relative frequency-based elementary modeling. It is easy to tailor an implementation so that we match the given

[1]We measure running time and space complexity in terms of the word-RAM model, where we assume the register size to be sufficiently large (typically roughly $\log n$, for input sequences of length n).

running time and space constraints. Furthermore, as we described above, we may incorporate an aging strategy with little effort and (typically) benefit from improved compression.

Piecewise Stationary Models. As we will see in this chapter, we can supplement the pleasant practical features of relative frequency-based elementary models with code length guarantees that support adaptivity. In particular, we compare the code length of several elementary models to that of a competitor that supports adaptivity, namely a Piecewise Stationary Model (PWS):

Definition 3.1.1 *A Piecewise Stationary Model* $\mathsf{PWS}\langle\mathcal{P}, \{p_s\}_{s\in\mathcal{P}}\rangle$ *for sequences of length* n *is given by a partition* $\mathcal{P}$ *of the set* $1{:}n$ *and a family* $\{p_s\}_{s\in\mathcal{P}}$ *of probability distributions. The PWS predicts* $\mathsf{PWS}(x_{<t}) = p_s$*, where* $s \in \mathcal{P}$ *is the unique segment with* $t \in s$*. The sequence* $x_{1:n}$ *receives code length*

$$\ell(x_{1:n}; \mathsf{PWS}) = \sum_{s\,=\,a:b\in\mathcal{P}} \ell(x_{a:b}; p_s). \tag{3.2}$$

Let us take a closer look at the idea behind a PWS, to do so fix a PWS $\mathsf{PWS}\langle\mathcal{P}, \{p_s\}_{s\in\mathcal{P}}\rangle$ for sequence length n. The model PWS essentially divides a given sequence $x_{1:n}$ into segments according to the partition $\mathcal{P}$. For time steps t from segment $s = a{:}b \in \mathcal{P}$ the model PWS predicts a fixed distribution, i. e. we have $p_s = \mathsf{PWS}(x_{<a}) = \mathsf{PWS}(x_{<a+1}) = \cdots = \mathsf{PWS}(x_{<b})$. Consequently, PWS assigns code length $\ell(x_{a:b}; p_s)$ to the segment $x_{a:b}$ and code length (3.2) to the sequence $x_{1:n}$.

3.2 Previous Work

We now summarize several approaches to elementary modeling from literature. Each section below corresponds to a particular approach to elementary modeling. For all approaches we first sketch the general idea and subsequently discuss relevant literature. The reader should keep in mind that any given code length guarantee considers the deterministic view, unless stated differently, and assumes a competing PWS for sequences of length n given by $\mathsf{PWS}\langle\mathcal{P}, \{p_s\}_{s\in\mathcal{P}}\rangle$.

Interestingly, we may group the approaches below into two families: On the one hand, there is the family of “practitioner’s approaches” consisting of Relative Frequencies with Discount, Relative Frequencies with Smoothing and Probability Smoothing. We may characterize this family by high practical relevance (low space and time complexity, meeting the design constraints we gave previously; widespread use and appealing empirical performance), but the lack of code length guarantees (especially w. r. t. PWS). On the other hand, there is the family of “theoretician’s approaches” made up of Relative Frequencies with Windows, Transition Diagrams and Partition Tree Weighting. We can support these methods by code length guarantees, often w. r. t. PWS, but unfortunately the time and space complexity is beyond the acceptable range. Elementary modeling based on Finite State Machines (FSMs) is a hybrid: In most cases an implementation is fast and efficient, however, known code length guarantees only consider PWS with $|\mathcal{P}| = 1$. So known theoretic results do not support the ability to adapt and, to our best knowledge, there is no experimental evaluation.

Relative Frequencies with Discount. Classical relative frequency-based elementary models, such as the Laplace- (model RF from (2.3) with $f_0 = 1$) and Krichevsky-Trofimov-Estimator (model RF from (2.3) with $f_0 = 1/2$) [48] are well-known and well understood [18, 26]. Refinements thereof that periodically discount letter frequencies using a *discount factor* λ enjoy great popularity in statistical data compression: Given the initial frequencies $f_1(\,\cdot\,)$ we use the prediction (3.1) and for $t > 0$ we update the frequencies according to

$$f_{t+1}(x) := \begin{cases} \lambda \cdot f_t(x) + 1, & \text{if } t > 0 \text{ and } x = x_t, \\ \lambda \cdot f_t(x), & \text{if } t > 0 \text{ and } x \neq x_t, \end{cases} \quad \text{where } \lambda \in [0, 1), \tag{3.3}$$

when a discount takes place in step t; otherwise, we do a “usual update” and set $f_{t+1}(x) = f_t(x) + 1$, if $x = x_t$, or $f_{t+1}(x) = f_t(x)$, if $x \neq x_t$. Often, frequencies have to be integer, so (3.3) is slightly altered to support rounding and to guarantee non-zero frequencies.

In the 1970s several researchers introduced Adaptive Huffman Coding [31, 33], which relies on the estimation of letter frequencies or, equivalently, letter probabilities. (Adaptive Huffman Coding constructs a coding tree for the t-th letter based on letter frequency/probability predictions due to the segment $x_{<t}$; in fact, Adaptive Huffman Coding, is a statistical compressor with order-0 model that substitutes AC with Huffman Coding.) An early idea for adaptive frequency (probability) estimation is to halve the integer letter frequencies every B letters [33]. A more general version that allows for discount factors other than $\frac{1}{2}$ was later analyzed in [42]. Unfortunately, the code length analysis given there utilizes a competitor based on the weighted empirical entropy: The competitor partitions the input sequence into blocks of length B, in the i-th block it assigns weight

$$w_i(x) := \lambda \cdot w_{i-1}(x) + |\{t \mid x_t = x \text{ for } (i-1)B < t \leqslant iB\}|, \text{ for } i \geqslant 1,$$

and code length $\log \frac{\sum_{y\in\mathcal{X}} w_i(y)}{w_i(x)}$ to letter x (appearing in block i). It is not clear how to quantify if this competitor may perform well for inputs with varying statistics. Compared to a PWS, this type of competing scheme is hard to interpret, so the overall results remain hard to interpret, as well. Moreover, to trigger a frequency discount every B letters, we require an additional frequency counter. This space overhead might not seem to be a big problem, however, a statistical compressor maintains millions of context conditioned elementary models and for each context only few distinct letters appear. For instance, in PPM typically $60\ \%$ to $80\ \%$ of all contexts just contain a single letter [84]. To cut down space complexity a common approach is to scale down letter frequencies when their sum exceeds a threshold [107]. In this way there is no need for an additional frequency counter per elementary model. Due to its simplicity and appealing practical performance this scheme can be found in most practical implementations of statistical data compression algorithms.

A discount factor $\lambda = 0$ completely discards all previous statistics and the frequency discount becomes a frequency reset. Such a harsh behavior is typically undesirable in practice, since retaining some statistics tends to improve compression. However, if we time resets carefully, then we at least obtain good code length guarantees. A KT estimator that resets frequencies in intervals of exponentially increasing length has redundancy $O(|\mathcal{P}|\cdot\sqrt{n\log n})$ w. r. t. any PWS [83].

Relative Frequencies with Smoothing. A special borderline case of Relative Frequencies with Discount is Relative Frequencies with Smoothing: By allowing non-integer letter frequencies, it is possible to apply the frequency discount (3.3) in *every* frequency update.

In experiments that consider input sequences with varying statistics frequency smoothing with a fixed discount factor was found to perform remarkably well compared to more sophisticated elementary models [92]. Several frequency smoothing strategies coupled with CTW have been studied experimentally [69]. The basic idea is to use discount factor $\alpha\cdot t^{-\beta}$ for various choices of $\alpha,\beta\in[0,1)$ in the t-th step. Results indicate notable compression improvements over basic CTW with elementary model KT. Similarly, the CTW descendant CTS [91] relies on frequency smoothing with a fixed discount factor to improve compression. To estimate bit probabilities early PAQ variants [56, 54] employed a variation of frequency smoothing with integer frequencies: In the t-th step we set $f_{t+1}(x) = f_t(x) + 1$, if $x = x_t$; and only smooth the frequency of the other bit, $f_{t+1}(x) = \lfloor f_t(x)/2+1\rfloor$, if $x \neq x_t$.

An implementation of frequency smoothing requires multiplying all frequencies by the discount factor λ and to increment the frequency corresponding to the novel letter by 1. Equivalently (prediction (3.1) remains unchanged), we may omit the multiplication of all frequencies by λ and instead use an increment of λ^{-1} for the first frequency update, an increment of λ^{-2} for the second frequency update, etc. This technique was suggested in [3].

Probability Smoothing. Another idea for probability prediction is to apply smoothing not to frequencies, but to probabilities. Given a probability distribution (i. e. the prediction of the previous step or some initial estimate) and a novel letter we compute an updated distribution as follows: First we multiply all probabilities with the smoothing rate $\alpha \in (0, 1)$ and afterwards we increment the probability of the novel letter by $1-\alpha$. The smoothing rate α may vary from step to step.

To our best knowledge this common-sense approach was first mentioned in [43] and nowadays enjoys great popularity in PAQ-based algorithms. In reality an implementation of probability smoothing has to work with finite precision arithmetic. Taking this into account [65] showed that a K-state FSM approximation of probability smoothing for a binary alphabet has redundancy $O(K^{-2/3} \cdot n)$ w. r. t. PWS with $|\mathcal{P}| = 1$.

Finite State Machines. An elementary model based on integer letter frequencies essentially maps an input segment $x_{<t}$ to a set of frequencies, which we may view as a state. Moreover, for every practical elementary model the number of such states is finite, since the amount of usable memory for frequency data is finite. Having this in mind an obvious idea is to directly construct a Finite State Machine (FSM) for elementary modeling: Every state has N outgoing edges and an associated probability distribution. An outgoing edge (one for each letter) points to a follow-up state. In the t-th step the FSM is in some state (for $t = 1$ this is the initial state) and uses the distribution of this state as prediction. When the upcoming letter x_t becomes known, the FSM follows the corresponding outgoing edge and reaches the state for step $t + 1$. Before we discuss previous work, let us parenthesize a remark: All of the following results apply to binary sequences, FSM elementary models with K states and a competing PWS with a single segment, i. e. $|\mathcal{P}| = 1$.

A simple idea is to use an FSM that counts the frequency of 0- and 1-bits and returns to its initial state when the total frequency exceeds r (so all frequency data is lost) [71]. The FSM has $K = \binom{r}{2}$ states, one state for each possible frequency pair, and guarantees redundancy $O(\frac{\log K}{\sqrt{K}} \cdot n)$. A different idea is to simulate probability smoothing (see previous section) via an FSM [65]. Any K-state FSM predictor has redundancy at least $\Theta(K^{-4/5} \cdot n)$ [65]. Later, this lower bound was improved [44] and the authors proposed an FSM construction algorithm: Given per-bit redundancy R, the algorithm constructs an FSM with per-bit redundancy close to R. Other approaches rely on randomization, the FSM proposed in [66] simulates an Imaginary Sliding Window (see next section) and attains expected redundancy $O(\frac{\log K}{K} \cdot n)$. All major PAQ variants employ either randomized or non-randomized FSM elementary models. To improve compression the distribution associated to each state is estimated online using an elementary model, rather than being fixed [80] (also see Section 2.4.4 on PAQ).

Relative Frequencies with Windows. Frequency discount and frequency smoothing slowly age *all* previous letters. To put emphasis only on a fixed amount of past letters (rather than all letters) we can alternatively compute relative frequencies (3.1) based on the last w letters, which we store in a sliding window. (When a new letter enters the window, we increment its frequency and decrement the frequency of the oldest letter that leaves the sliding window.)

The sliding-windows approach and its time-, space- and redundancy-tradeoffs have previously been studied in the probabilistic view [77] considering a stationary memoryless Bernoulli Source. The expected redundancy per letter is $O(N/w)$. If we allow for a randomization [77] suggests using an Imaginary Sliding Window (ISW) to eliminate the space overhead of a sliding window: In the t-th step we increment the frequency of the next letter x_t and assume that the letter x, chosen at random with probability roughly $p_t(x)$, leaves the ISW, and thus we decrement its frequency by 1. By choosing the bit-width of a frequency counter appropriately, the expected redundancy remains unchanged. Another work [86] provides results on a windowed version of the KT-Estimator considering binary, non-stationary Bernoulli sources (probabilistic view). The expected redundancy of the t-th bit may be bounded from above by the KL-Divergence of the source distributions in steps t and $t-1$.

Transition Diagrams. Several researchers constructed models that explicitly aim towards low redundancy w. r. t. PWS based on *transition diagrams*. A transition diagram is a set of nodes that grows while stepping through the input $x_{1:n}$. Every node holds an elementary model and possibly some auxiliary data (e. g. node creation time). At time instant t a node represents some segment $a{:}t$ and the associated elementary model predicts based on segment $x_{a:t}$. (So a transition diagram is actually composed of multiple elementary models.) Unfortunately, it is unknown which node (segment) with associated elementary model will offer the best prediction. A solution to this problem is to mix predictions of all nodes.

In the probabilistic view it was shown [63] that for any fixed $\delta > 0$ and large enough n the expected redundancy w. r. t. PWS is at least

$$(1-\delta)\cdot\left(\frac{(N-1)|\mathcal{P}|}{2}+|\mathcal{P}|-1\right)\cdot\log n.$$

Furthermore, the authors provided algorithms that achieve this bound in polynomial time per letter for $|\mathcal{P}| = O(1)$ or exponential time per letter when $|\mathcal{P}|$ is unknown. Later two more practical transition diagram approaches have been proposed [97]. They require either time $O(n^2)$ and space $O(n)$ or time $O(n^3)$ and space $O(n^2)$. Both methods have redundancy $O(|\mathcal{P}|\cdot N\log n)$, where the more complex method enjoys slightly better code length guarantees. In experiments both algorithms perform similarly. To further trade complexity for redundancy one may assign a life time to nodes, such that there are at most $O(\log n)$ nodes at any time [99]. As a consequence the redundancy increases to $O(|\mathcal{P}|\cdot N\log^2 n)$ while the running time drops to $O(n\log n)$. Along a similar line the number of nodes may even be cut down to $O(1)$, thus the running time becomes $O(n)$ [83]. The redundancy of this scheme may be tuned, depending on parameters. In the same work the authors proposed a $O(n^2)$ time and $O(n)$ space method that improves code length guarantees over [97] while retaining identical complexity.

Partition Tree Weighting. Partition Tree Weighting (PTW) is a meta-algorithm that turns any elementary model with redundancy $O(g(n))$ (where g is a non-decreasing and concave function) w. r. t. PWS with $|\mathcal{P}| = 1$ to an elementary model with redundancy

$$O\left(|\mathcal{P}|\cdot g\left(\frac{n}{|\mathcal{P}|\cdot\log n}\right)\cdot\log n\right)$$

w. r. t. arbitrary PWS. For instance the KT elementary model within the PTW framework has redundancy $O(|\mathcal{P}|\cdot N\log^2(n))$. The improved guarantee comes at its price: We must lift the number of elementary models from 1 to $O(\log n)$, thus running time increases by a factor of $O(\log n)$.

3.3 Our Contribution

The literature survey we gave in Section 3.2 underpins the gap between theory and practice (here, in terms of elementary models) which we discussed in Section 1.3: Most elementary models that are widely used because they admit an efficient implementation – Relative Frequencies with Discount, Relative Frequencies with Smoothing and Probability Smoothing – have no thorough theoretical basis. In contrast, techniques that have a theoretical basis (and often low redundancy w. r. t. PWS) – Relative Frequencies with Windows, Transition Diagrams and Partition Tree Weighting – are beyond the scope of practical running time and space requirements. (FSMs are somewhat in between, code length guarantees only consider PWS with $|\mathcal{P}| = 1$ and no deep experimental study is available.)

In the remainder of this chapter we concentrate on the aforementioned group of practical elementary models and provide a code length analysis w. r. t. PWS in the deterministic view. All of our results are novel, to our best knowledge there exist no code length guarantees that cover any of the elementary models above. The majority of this chapter has previously been published [60, 61]. In the remainder of this chapter we present a polished version thereof.

Section 3.4. We provide a code length analysis of Relative Frequencies with Discount w. r. t. the empirical entropy (Section 3.4.2) and subsequently generalize our analysis to the class of PWS with either bounded (away from 0) or unbounded (arbitrary) letter probabilities (Section 3.4.3). Our analysis is based on mild assumptions and holds for a N-ary alphabet.

Section 3.5. In Section 3.5.2 we first provide a code length analysis of Probability Smoothing w. r. t. the empirical entropy and in Section 3.5.3 we proceed by generalizing the code length analysis to the class of PWS. These results holds for binary sequences and smoothing rates that satisfy a set of mild assumptions. Finally, in Section 3.5.4, we apply our analysis machinery to provide code length bounds for two particular smoothing rate choices, that is, a fixed smoothing rate and a slowly increasing smoothing rate.

Section 3.6. We observe that Relative Frequencies with Smoothing is an instance of Probability Smoothing, hence we adopt our analysis method to provide a code length analysis w. r. t. PWS for a binary alphabet.

Section 3.7. All of the aforementioned elementary models share great similarity. Based on approximations or the choice of parameters we may transform one elementary model to become the other (e. g. by a particular choice of parameters Probability Smoothing becomes Relative Frequencies with Smoothing), as we will explain in this section.

Section 3.8. We provide the results of an experimental study on Relative Frequencies with Discount, Probability Smoothing and Relative Frequencies with Smoothing to support our code length bounds and to judge on their tightness. All experiments consider a binary alphabet and take place in a worst-case setting, since our bounds are worst-case bounds. Despite the alignment of theoretic results and measurements, we discuss the measured redundancy behavior of either elementary model.

```
Function Init()
create Array f [0..$N-1$];
for i $= 0$ to $N-1$ do f[i] $\leftarrow f_1$(i);
F $\leftarrow f_1(0) + \cdots + f_1(N-1)$;
return (f, F);
```

```
Function Predict(f[0..$N-1$], F)
create Array p[0..$N-1$];
for i $= 0$ to $N-1$ do p[i]$\leftarrow$f[i]/F;
return p;
```

```
Function Update(f[0..$N-1$], F, x)
if F $+ d > T$ then
    F $\leftarrow 0$;
    for i $= 0$ to $N-1$ do
        f[i] $\leftarrow \lfloor \lambda \cdot$ f[i]$\rfloor$;
        if f[i]=0 then f[i] $\leftarrow 1$;
        F $\leftarrow$ F + f[i];
    end
end
f[x] $\leftarrow$ f[x] $+ d$;
F $\leftarrow$ F $+ d$;
```

Figure 3.1: Pseudocode for a typical implementation of RFD consisting of: The function `Init` to set up initial frequencies, the function `Predict` to obtain relative letter frequencies as a prediction and the function `Update` to maintain relative letter frequencies.

3.4 Relative Frequencies with Discount

We now present and analyze a common variant of Relative Frequencies with Discount (RFD). Before we start, let us sketch the roadmap. In Section 3.4.1 we first precisely describe the RFD elementary model by means of pseudocode, explain its operation and identify and discuss its parameters. Subsequently, we conduct a code length analysis that we split into two parts. In Section 3.4.2 the empirical entropy serves as a competitor and we only consider the situation when a discount does not affect the probability assignment for letters $x_1, x_2, \ldots, x_n$. Section 3.4.3 provides an extension: First, we allow for an arbitrary number of discounts (that may actually affect probability assignment); second, we generalize the competitor to become a PWS with either bounded or unbounded letter probabilities.

3.4.1 The Setting

Algorithm RFD. The pseudocode in Figure 3.1 summarizes a typical implementation of RFD and will serve as the basis for our code length analysis. RFD requires an array f[0..$N-1$] to store letter frequencies, so a letter x has frequency f[x]. All frequencies are non-zero integers. For efficiency the variable F stores the sum of all letter frequencies. Thus, the state of RFD is completely captured by the pair (f, F). Naturally RFD assigns probability f[x]/F to letter x (see `Predict`). After we observe the next letter x, the implementation must update the frequencies (cf. `Update`): If F is not too large, in particular F $+ d \leqslant T$, for some threshold T, we simply increase `f[x]` and `F` by an integer d. However, if `F` is too large, we perform a *discount* before increasing `f[x]` and `F`, that is, we replace the content of `f[i]` by $\lfloor \lambda \cdot \texttt{f[}i\texttt{]} \rfloor$ for some $0 \leqslant \lambda < 1$ (e. g. an integer division), for all letters i. As a fixup we assign value 1 to any array cell which now has value 0. The variable `F` is adjusted to hold the sum of all frequencies. This completes the discount and we can now increase `f[x]` and `F` by d, as usual. A discount reduces the influence of old statistics and limits the memory for frequency storage to $O(N \log T)$ bits.

Parameters of RFD. The pseudocode in Figure 3.1 has 4 parameters, in particular, frequency increment d, discount factor λ, limit T on the sum of all letter frequencies and the initial frequencies $f_1(\,\cdot\,)$. For a code length analysis, we make the following assumptions on these parameters throughout Section 3.4:

Assumption 3.4.1 *The parameters of RFD satisfy:*

(a) *d is a positive integer,*
(b) *λ is a real from $[0, 1)$,*
(c) *T is an integer s. t. $(1-\lambda)\cdot(T-N) = d \cdot i$, for some positive integer i,*
(d) *$f_1(\,\cdot\,)$ maps to positive integers s. t. $\sum_{x\in\mathcal{X}} f_1(x) \leqslant d + (1-\lambda)\cdot N + \lambda \cdot T$.*

Assumption 3.4.1 does not impose severe restrictions. Assumption 3.4.1 (a) claims a non-zero frequency increment; Assumption 3.4.1 (b) disallows a discount factor of $\lambda = 1$, which would not discount frequencies at all. Assumption 3.4.1 (c) essentially introduces a lower bound on T depending on the parameters d and λ, that is $T \geqslant N + \frac{d}{1-\lambda}$; furthermore, $1-\lambda$ is forced to be a multiple of $d/(T-N)$, which is just a technical restriction to simplify our analysis. Finally, Assumption 3.4.1 (d) calls for non-zero initial frequencies, $f_1(x) > 0$, for all letters x, and limits the sum F_1 of initial letter frequencies.

Model RFD. When we consider the operation of RFD on a given input $x_{1:n}$ we actually observe a sequence of function calls. There is a single call to `Init`, followed by a sequence of `Predict`- and `Update`-function calls,

$$\begin{aligned}(\texttt{f},\texttt{F}) \leftarrow \texttt{Init()},\ \texttt{p} \leftarrow \texttt{Predict(f,F)},\ (\texttt{f},\texttt{F}) \leftarrow \texttt{Update(f,F,}x_1\texttt{)}, \ldots,\\ \texttt{p} \leftarrow \texttt{Predict(f,F)},\ (\texttt{f},\texttt{F}) \leftarrow \texttt{Update(f,F,}x_n\texttt{)}.\end{aligned} \tag{3.4}$$

The phrase *step* t uniquely refers to the t-th pair of `Predict`- and `Update`-operations. We say a *discount* takes place in step t, if `Update` executes the lines 2 to 7 (see Figure 3.1) in step t. Having the above notion in mind, we can finally define “the prediction of RFD in step t”.

Table 3.1: Notation used for the code length analysis of RFD in Section 3.4.

Symbol	Definition	Symbol	Definition
$x_{1:n}$	input sequence	F_t	sum of freq. estimates $f_t(x)$
N	alphabet size		(`F` in t-th `Predict`-call)
M	number of distinct letters in $x_{1:n}$	$\mathsf{RFD}(x; x_{<t})$	prob. estimate $f_t(x)/F_t$
$c(x)$	frequency of letter x in $x_{1:n}$	d	integer frequency increment
$h(x_{1:n})$	$\sum_{x\in\mathcal{X}} c(x)\log({}^{n}/_{c(x)})$	λ	discount factor from $[0, 1)$
$f_t(x)$	freq. estimate for x based on $x_{<t}$	T	limit on total frequency F_t
	(`f[`x`]` in t-th `Predict`-call)	L	$(T - N)/d$

Definition 3.4.2 *Let $f_t(x)$ be the value of array cell* `f[`x`]` *and let F_t be the value of variable* `F`*, both at the beginning of step t in operation sequence (3.4). The model* RFD *is defined by the prediction*

$$\mathsf{RFD}(x; x_{<t}) := \frac{f_t(x)}{F_t}.$$

3.4.2 Relative Frequencies with Discount vs. Empirical Entropy

Probability Assignment. We are now ready to start the code length analysis of RFD. Initially, just the empirical entropy will serve as a competitor. Subsequently, we will extend the analysis to cover a competing PWS. Our starting point is not only simple in terms of the competitor, but also in terms of how RFD operates: We assume that there is at most one discount in step n. In this situation the prediction of RFD collapses to

$$\mathsf{RFD}(x; x_{<t}) = \frac{f_1(x) + d \cdot |\{i \mid x_i = x \text{ and } 1 \leqslant i < t\}|}{F_1 + (t-1)\cdot d}. \tag{3.5}$$

As a reminder for the reader, Table 3.1 summarizes the most important symbols for the analysis.

Analysis. First, we require a technical lemma on the empirical entropy.

Lemma 3.4.3 *We have* $\binom{n}{c(0),\ldots,c(N-1)} \leqslant 2^{h(x_{1:n})}$.

Proof. Case $n = 0$ is trivial, let $n > 0$. By the Multinomial Theorem we get

$$1 = \left(\sum_{x\in\mathcal{X}} \frac{c(x)}{n}\right)^n \geqslant \binom{n}{c(0),\ldots,c(N-1)} \cdot \prod_{x\in\mathcal{X}} \left(\frac{c(x)}{n}\right)^{c(x)}$$

$$= \binom{n}{c(0), \ldots, c(N-1)} \cdot 2^{-h(x_{1:n})},$$

rearranging completes the proof. □

Notice that by using Stirling's formula the above bound can be improved (cf. [26, Lemma 17.5.1]). However, in the worst case the improvement is small (multiplicative factors independent of n), thus we omit it and use the simpler version. We now proceed with a code length guarantee for RFD.

Proposition 3.4.4 *If the operation sequence (3.4) contains at most one discount in step n and $N \leqslant F_1 < \infty$ (i. e. Assumption 3.4.1 (d) does not need to hold!), then*

$$\ell(x_{1:n}; \mathsf{RFD}) \leqslant h(x_{1:n}) + \left(M - 1 + \frac{F_1 - 1}{d}\right) \log(n) + \log\left(F_1/d \cdot e^{F_1/d} \cdot d^M\right).$$

Proof. Our plan is to bound $\prod_{1 \leqslant t \leqslant n} \mathsf{RFD}(x_t; x_{<t})$ from below by separately treating numerator and denominator; finally, we combine the results. The numerator $f_t(x)$ and denominator F_t of $\mathsf{RFD}(x; x_{<t})$ are given by (3.5).

Numerator. By Assumption 3.4.1 (d) we have $f_1(x) \geqslant 1$, for any letter x, so

$$\begin{aligned}\prod_{1 \leqslant t \leqslant n} f_t(x_t) \overset{\text{A3.4.1 (d)}}{\geqslant} \prod_{x \in \mathcal{X}} \prod_{1 \leqslant i < c(x)} (1 + di) &= \prod_{x \in \mathcal{X}} \frac{c(x)!}{c(x)} \prod_{1 \leqslant i < c(x)} \frac{di+1}{i} \\ &\geqslant \prod_{x \in \mathcal{X}} c(x)! \cdot \prod_{1 \leqslant i \leqslant n-M} \frac{di+1}{i} \cdot \left(\frac{M}{n}\right)^M,\end{aligned} \tag{3.6}$$

where we used $\prod_{1 \leqslant i < c(x), x \in \mathcal{X}} \frac{di+1}{i} \geqslant \prod_{1 \leqslant i \leqslant n-M} \frac{di+1}{i}$, since $\frac{di+i}{i}$ decreases with i, and $\prod_{x \in \mathcal{X}} c(x) \leqslant (n/M)^M$, by the Arithmetic-Geometric-Mean inequality.

Denominator. We have

$$\prod_{1 \leqslant t \leqslant n} F_t = F_1 \cdot \prod_{1 \leqslant i < n} (F_1 + di) = n! \cdot \frac{F_1}{n} \cdot \prod_{1 \leqslant i < n} \frac{di + F_1}{i}. \tag{3.7}$$

Combination. We require two simple inequalities: First,

$$\prod_{1 \leqslant i < n} \frac{di + F_1}{di + 1} \leqslant \prod_{1 \leqslant i < n} \left(1 + \frac{F_1 - 1}{di}\right) \leqslant e^{(F_1-1)/d \cdot H_n} \leqslant (en)^{(F_1-1)/d}, \tag{3.8}$$

where we used $1+z \leqslant e^z$ and $H_n = \sum_{1\leqslant i\leqslant n} i^{-1} \leqslant \ln(en)$; Second,

$$\prod_{n-M<i<n} \frac{di+1}{i} \leqslant d^{M-1} \cdot \prod_{n-M<i<n} \frac{i+1}{i} = \frac{n \cdot d^{M-1}}{n-M+1} \leqslant M \cdot d^{M-1}. \tag{3.9}$$

We combine (3.6) and (3.7) and use (3.8), (3.9) and $M^{M-1} \geqslant 1$, to obtain

$$\prod_{1\leqslant t\leqslant n} \mathsf{RFD}(x_t; x_{<t}) \overset{(3.6),\,(3.7)}{\geqslant} \frac{\prod_{x\in\mathcal{X}} c(x)!}{n!} \cdot \frac{M^M}{F_1 n^{M-1}} \cdot \prod_{1\leqslant i<n} \frac{di+1}{di+F_1} \cdot \prod_{n-M<i<n} \frac{i}{di+1}$$
$$\overset{(3.8),\,(3.9)}{\geqslant} \left[\binom{n}{c(0),\dots,c(N-1)} \cdot n^{M-1+(F_1-1)/d} \cdot F_1/d \cdot e^{F_1/d} \cdot d^M\right]^{-1}.$$

We apply Lemma 3.4.3 and take the logarithm of the above inequality. □

Discussion. Let us compare the code length bound of Proposition 3.4.4 to similar results from the literature. If we set $d = 1$ and $f_1(x) = 1$, for all letters x, then we obtain the Laplace estimator LP, which satisfies (cf. Example 2.2.3 and [26, 18])

$$\ell(x_{1:n}; \mathsf{LP}) \leqslant h(x_{1:n}) + (N-1)\cdot \log(n) + o(N). \tag{3.10}$$

In contrast, the bound of Proposition 3.4.4 becomes

$$\ell(x_{1:n}; \mathsf{LP}) \leqslant h(x_{1:n}) + (N+M-2)\cdot \log(n) + O(N). \tag{3.11}$$

In case of the Laplace-Estimator the sequence $x_{1:n}$ that maximizes the redundancy $\ell(x_{1:n};\mathsf{LP}) - h(x_{1:n})$ satisfies $M = 1$ [18]. On the one hand, for this worst-case situation, bounds (3.10) and (3.11) yield the same leading redundancy term $(N-1)\cdot\log(n)$ on their r. h. s.; on the other hand, when we do not know about the worst-case input and just consider a uniform version of bound (3.11) (by plugging in $M \leqslant N$), then the leading redundancy term of (3.11) becomes $2(N-1)\cdot\log(n)$ and is off by a factor of 2 compared to (3.10). Similarly, we may set $d = 2$ and $f_1(x) = 1$, for all letters x, to obtain the KT estimator KT which guarantees [8, Theorem 7]

$$\ell(x_{1:n}; \mathsf{KT}) \leqslant h(x_{1:n}) + \frac{N-1}{2}\cdot \log(n) + \log N, \tag{3.12}$$

whereas a uniform version of our bound (we plugged in $M \leqslant N$) yields

$$\ell(x_{1:n}; \mathsf{KT}) \leqslant h(x_{1:n}) + 3\cdot\frac{N-1}{2}\cdot \log(n) + O(N). \tag{3.13}$$

Again, the leading redundancy term in (3.13) is three times as large as its counterpart in (3.12). This inaccuracy is the price we have to pay to obtain code length guarantees that hold for a class of elementary model strictly larger than $\{\mathsf{LP}, \mathsf{KT}\}$. (As we noted, we can tighten our bounds slightly by improving Lemma 3.4.3. However, with the above proof technique it seems impossible to cut down the leading factor of the $\log n$-term for a bound that holds uniformly for all sequences.)

3.4.3 Relative Frequencies with Discount vs. Piecewise Stationary Models

Time Spans between Discounts. For code length guarantees it turns out to be crucial to estimate distances between adjacent discounts within the operation sequence (3.4). So let us take a closer look at time steps that trigger a discount. First, we define

$$L := \frac{T-N}{d}, \tag{3.14}$$

on which we will rely shortly. The first discount in operation sequence (3.4) takes place in step t, whenever the value F_t of variable `F` satisfies $F_t + d > T$. Hence, if `F` has value F_1 after `Init` (at the beginning of step 1), then the first discount takes place in step $t = (F_t - F_1)/d + 1$, since in each step $1, 2, \dots, t-1$ variable `F` increases by d. Similarly, if the most recent discount before step t took place in step t' and `F` had value $F_{t'+1}$ after step t' (at the beginning of step $t'+1$), then the distance $t' - t$ between these discounts is $(F_t - F_{t'+1})/d + 1$. In Figure 3.2 we depict this situation graphically. As we observe, dealing with distances between adjacent discounts requires dealing with F_t.

Lemma 3.4.5 *In operation sequence (3.4) the letter frequency sum satisfies*

(a) $N \leqslant F_1$ *and* $N + d \leqslant F_t$, *for* $t > 1$,
(b) $F_t \leqslant T$, *for any time step* t, *and*
(c) $F_t \leqslant d + (1-\lambda)N + \lambda T$, *if* $t = 1$ *or a discount took place in step* $t-1$.

Proof. **(a)** Assumption 3.4.1 (d) implies $F_1 \geqslant N$. For step $t > 1$, consider the situation at the end of the $(t-1)$-th call to `Update`: Immediately before line 10 we have `F` $\geqslant N$ (since every letter has positive frequency and `F` stores the letter frequency sum), thereafter we have `F` $\geqslant N + d$ and `F` $= F_t$.

(b) *and* **(c)** We use induction on t to prove both bounds.

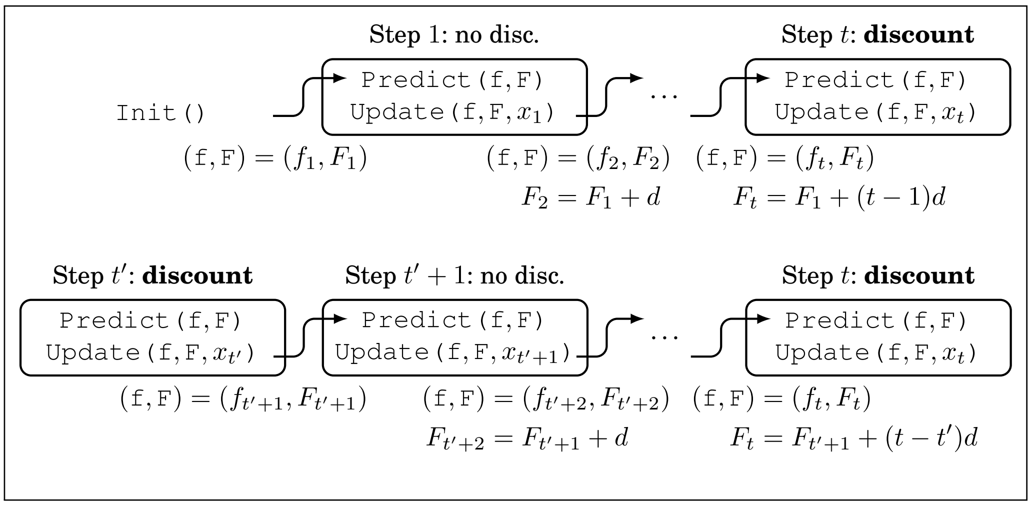


Figure 3.2: Time spans between discounts. Top: first discount; Bottom: a subsequent discount. An arrow represents the "transmission" of RFD's state information,i. e. the pair (f, F), from the Update-call in one step to a succeeding step.

Base: $t = 1$. We consider the situation right after Init and take advantage of Assumption 3.4.1 (d) and Assumption 3.4.1 (c), so we obtain

$$F_1 \overset{\text{A3.4.1 (d)}}{\leqslant} d + (1-\lambda)N + \lambda T = T + d - (1-\lambda)(T-N) \overset{\text{A3.4.1 (c)}}{\leqslant} T, \qquad (3.15)$$

which proves **(b)** and **(c)**. (Assumption 3.4.1 (c) implies $(1-\lambda)(T-N) \geqslant d$.)

Step: $t > 1$. We consider the Update-call in step $t-1$ and distinguish:

Case 1: No discount took place. At the beginning of this Update-call we get $F_{t-1} + d \leqslant T$, so $F_t = F_{t-1} + d$ and $F_t \leqslant T$, which proves **(b)**.

Case 2: A discount took place. Function Update executes lines 2 to 7, so

$$F_t = d + \sum_{x \in \mathcal{X}} \max\{1, \lfloor \lambda f_{t-1}(x) \rfloor\} \leqslant d + (1-\lambda)N + \lambda F_{t-1}.$$

The inequality is due to $\max\{1, \lfloor \lambda z \rfloor\} \leqslant 1 - \lambda + \lambda z$, for $z \geqslant 1$ and $0 \leqslant \lambda \leqslant 1$. From $F_{t-1} \leqslant T$ (by the induction hypothesis) we conclude **(c)**. Based on **(c)** we conclude **(b)** analogously to (3.15). □

By the upper and lower bounds on F_t we can immediately bound the distance between adjacent discounts (or the distance between Init and the first discount) depending on L and λ:

Lemma 3.4.6 *For a discount in step t of operation sequence (3.4) we have*

(a) $(1-\lambda)L \leqslant t \leqslant L + 1$, *if this is the first discount, or*

(b) $(1-\lambda)L \leqslant t - t' \leqslant L$, *if the previous discount took place in step t'.*

Proof. **(a)** *First Discount.* Since no discount takes place in steps $1, 2, \dots, t-1$, we have $F_t = F_1 + (t-1)d$, or equivalently

$$t = \frac{F_t - F_1}{d} + 1. \tag{3.16}$$

We plug $F_t > T - d$ (see `Update` in Figure 3.1) and $F_1 \leqslant d + (1-\lambda)N + \lambda T$ (by Assumption 3.4.1 (d)) into (3.16) which yields

$$t > \frac{T - d - (d + (1-\lambda)N + \lambda T)}{d} + 1 \overset{(3.14)}{=} (1-\lambda)L - 1 \overset{\text{A3.4.1 (c)}}{\Longrightarrow} t \geqslant (1-\lambda)L,$$

since $(1-\lambda)L$ is an integer (by Assumption 3.4.1 (c)). Analogously, we plug $F_t \leqslant T$ (by Lemma 3.4.5 (b)) and $F_1 \geqslant N$ (by Lemma 3.4.5 (a)) into (3.16) and obtain

$$t \leqslant \frac{T - N}{d} + 1 = L + 1.$$

(b) *Subsequent Discount.* Similarly to (a), we may write

$$t - t' = \frac{F_t - F_{t'+1}}{d} + 1 \tag{3.17}$$

Since $F_{t'+1}$ and F_1 share the same upper bound (cf. Lemma 3.4.5 (c) and Assumption 3.4.1 (d)), the lower bound on $t - t'$ matches the lower bound on t from (a). For the upper bound we plug $F_t \leqslant T$ (by Lemma 3.4.5 (b)) and $F_{t'+1} \geqslant N + d$ (by Lemma 3.4.5 (a)) into (3.17) and get

$$t - t' \leqslant \frac{T - (N + d)}{d} + 1 = L,$$

which concludes the proof. □

At this point the role of L is clear: It essentially determines the maximum distance between adjacent discounts and, jointly with λ, it also determines the minimum distance. (Note that the maximum distance may be bounded by a function of $(1-\lambda)L$, however this will be of little use, so we stick with the simpler estimate depending on L only.) Based on the minimum distance, we may now argue on the number of discounts.

Lemma 3.4.7 *Operation sequence (3.4) contains at most $\frac{n}{(1-\lambda)L}$ discounts.*

Proof. By Lemma 3.4.6 a discount takes place at least every $(1-\lambda)L$ steps and the claim immediately follows. □

Analysis: PWS with Unbounded Letter Probabilities. In Section 3.4.2 we analyzed RFD in a rather restrictive setting: The empirical entropy served as a competitor and we did not allow for frequency discounts that affect predictions. We will now lift these limitations, by generalizing the competitor to become a PWS and by allowing for an arbitrary number of discounts. The basis for the aforementioned enhancement is a slightly refined version of Proposition 3.4.4:

Lemma 3.4.8 *For any distribution p and steps $a, a+1, \dots, b$ of operation sequence (3.4), with at most one discount in step b, we have: If $a = 1$, then*

$$\ell(x_{a:b}) \leqslant \ell(x_{a:b}; p) + L\log(eL) - (b-a+1)\log(eL) + A\log L + B + C.$$

If $a > 1$, then

$$\ell(x_{a:b}) \leqslant \ell(x_{a:b}; p) + L\log(eL) - (b-a+1)\log(eL) + A\log L + B.$$

The terms A, B and C are defined as follows:

$$A := \frac{(d+1)N + d}{d}, \quad B := \frac{\frac{L+1}{L}N + d}{d}\log e \quad \text{and} \quad C := \frac{\frac{L-1}{L}d + \frac{(d+1)N}{L}}{d}\log e.$$

These terms may be bounded from above by constants independent of L, depending on N and d only.

Proof. For brevity let $n = b-a+1$. For the proof we first show how to apply Proposition 3.4.4 in the current situation, then we simplify the resulting code length bound and finally, we treat the cases $a = 1$ and $a > 1$.

Applying Proposition 3.4.4. For $a \leqslant t \leqslant b$ the prediction $\mathsf{RFD}(x_{<t})$ *only* depends on $x_{a:t-1}$ and on the value of $(\mathtt{f}, \mathtt{F})$ at the beginning of step a, that is $\mathtt{f}[x] = f_a(x)$ and $\mathtt{F} = F_a$. If we set $y_{1:n} = x_{a:b}$, choose initial conditions $f'_1 = f_a$ and $F'_1 = F_a$ and consider the corresponding RFD instance RFD', then we get

$$\mathsf{RFD}'(y_{<t-a+1}) = \mathsf{RFD}(x_{<t}) \implies \ell(y_{1:n}; \mathsf{RFD}') = \ell(x_{a:b}; \mathsf{RFD}), \tag{3.18}$$

since the probability assignment is deterministic. We may now apply Proposition 3.4.4 for RFD' and sequence $y_{1:n}$[2] and plug in (3.18) , $h(y_{1:n}) \leqslant \ell(x_{a:b}; p)$

[2]Assumption 3.4.1 may not be fully satisfied in this situation: $f'_1 = f_a$ might violate Assumption 3.4.1 (d), since $F'_1 = F_a$ might exceed $d + (1-\lambda)N + \lambda T$. Luckily, Proposition 3.4.4 does not rely on this and holds nontheless.

and $M \leqslant N$, resulting in

$$\ell(x_{a:b}; \mathsf{RFD}) \overset{\text{P3.4.4}}{\leqslant} \ell(x_{a:b}; p) + \underbrace{(N-1)\log n}_{U} + \underbrace{\frac{F_1}{d}\log(en)}_{V} + \underbrace{\log\frac{F_1}{d}}_{W} + N\log d. \tag{3.19}$$

Simplification. We bound the terms in (3.19) by using the following,

$$\begin{aligned}
\frac{F_1}{d} &= \frac{F_n-(n-1)d}{d} \overset{\text{L3.4.5 (b)}}{\leqslant} \frac{T+d}{d} - n \overset{(3.14)}{=} \frac{N+d}{d} + L - n, \qquad (3.20)\\
U &= (N-1)\log L + (N-1)\log\frac{n}{L},\\
V &\overset{(3.20)}{\leqslant} \left(\frac{N+d}{d} + L - n\right)\log en\\
&\overset{n\geqslant 1}{\leqslant} (L-n)\log eL + \frac{N+d}{d}\log eL + \left(\frac{N}{d}+L\right)\log\frac{n}{L},\\
W &\overset{\substack{(3.20),\\ n\geqslant 1}}{\leqslant} \log\left(\frac{N}{d}+L\right) \leqslant \log L + \frac{N\log e}{dL}.
\end{aligned}$$

We continue by plugging the estimates on U, V and W into (3.19) and obtain

$$\begin{aligned}
\ell(x_{a:b}; \mathsf{RFD}) \leqslant{}& \ell(x_{a:b}; p) + L\log eL - n\log en + A\log L\\
&+ \underbrace{\frac{N+d}{d}\log e + \frac{N\log e}{dL}}_{B} + \underbrace{(A+L-2)\log\frac{n}{L}}_{X}. \qquad (3.21)
\end{aligned}$$

It remains bound X from above. To do so we distinguish two cases:

Case 1: $a = 1$. From $n \leqslant L+1$ (Lemma 3.4.6) we get $\log\frac{n}{L} \leqslant \frac{\log e}{L}$, so $X \leqslant C$.

Case 2: $a > 1$. From $n \leqslant L$ (Lemma 3.4.6) we conclude $X \leqslant 0$. □

Now, we have developed the tools of trade to extend the code length guarantees on RFD to PWS. Let us fix an arbitrary PWS with parameters $(\mathcal{P}, \{p_s\}_{s\in\mathcal{P}})$. For a single segment $s = a{:}b$ from $\mathcal{P}$ Lemma 3.4.8 allows us to bound the code length $\ell(x_{a:b}; \mathsf{RFD})$ in terms of the distribution p_s. So to bound $\ell(x_{1:n}; \mathsf{RFD})$ it suffices to sum the bounds for every individual segment from $\mathcal{P}$. Unfortunately, there is a complication: Lemma 3.4.8 is only applicable when among the steps $a, a+1, \dots, b$ there is at most one discount in step b. In general, this may not be the case. Luckily, we can overcome this limitation with ease: If in segment $a{:}b$ there are discounts in steps $j < k < \dots$, then we split $\ell(x_{a:b}; \mathsf{RFD}) = \ell(x_{a:j}; \mathsf{RFD}) + \ell(x_{j+1:k}; \mathsf{RFD}) + \dots$ and bound the individual terms. In other words, it suffices to sum the bound of Lemma 3.4.8 over all segments from

$$\mathcal{P}' := \{u{:}v \mid \text{there exists } a{:}b \in \mathcal{P} \text{ s. t.: 1. } a \leqslant u \leqslant v \leqslant b,$$

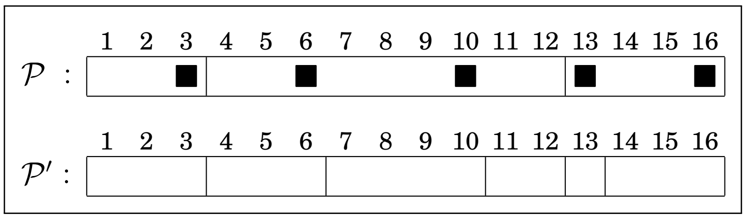


Figure 3.3: PWS partition $\mathcal{P} = \{1{:}3, 4{:}12, 13{:}16\}$ and discounts in steps 3, 6, 10, 13, 16, denoted as "■", induce partition $\mathcal{P}' = \{1{:}3, 4{:}6, 7{:}10, 11{:}12, 13{:}13, 14{:}16\}$.

2. $u = a$ or there is a discount in step $u - 1$ and (3.22)
3. $v = b$ or there is a discount in step $v\}$.

See Figure 3.3 for an example. With the above notion in mind, we derive the following:

Theorem 3.4.9 *Suppose that Assumption 3.4.1 holds for the RFD model* RFD. *For any PWS competitor* PWS$\langle \mathcal{P}, \{p_s\}_{s\in\mathcal{P}}\rangle$ *and terms A, B and C from Lemma 3.4.8 we have*

$$\ell(x_{1:n}; \mathsf{RFD}) \leqslant \ell(x_{1:n}; \mathsf{PWS}) \\ + |\mathcal{P}|\,[L \log eL + A \log L + B] + C + n \left[\frac{\lambda \log eL}{1-\lambda} + \frac{A \log L + B}{(1-\lambda)L}\right].$$

Proof. For the proof it suffices to sum the bound of Lemma 3.4.8 for all segments from partition $\mathcal{P}'$, see (3.22), so

$$\begin{aligned} \ell(x_{1:n}) - \ell(x_{1:n}; \mathsf{PWS}) &\overset{\text{L3.4.8}}{\leqslant} C + \sum_{s\in\mathcal{P}'} (L \log eL - |s| \log eL + A \log L + B) \\ &= |\mathcal{P}'|\,[L \log eL + A \log L + B] + C - n \log eL. \qquad (3.23) \end{aligned}$$

When there is no discount, then $|\mathcal{P}'| = |\mathcal{P}|$, furthermore, every additional discount increases $|\mathcal{P}'|$ by at most one. The maximum number of discounts is given in Lemma 3.4.7, this implies $|\mathcal{P}'| \leqslant |\mathcal{P}| + \frac{n}{(1-\lambda)L}$, which we plug into (3.23) and rearrange, to conclude the proof. □

Discussion. Let us take a closer look at Theorem 3.4.9. For fixed alphabet size N and discount factor λ bounded away from 1, that is $\lambda \leqslant \lambda' < 1$, the redundancy of RFD w. r. t. a PWS with partition $\mathcal{P}$ collapses to

$$O\left(|\mathcal{P}| \cdot L \log L + \frac{n}{L} \cdot (\lambda L \log L + \log L)\right).$$

The redundancy essentially consists of two parts: First, redundancy introduced by the complexity (i. e. the number $|\mathcal{P}|$ of segments) of the competitor, secondly, redundancy due to discounts. Intuitively, it should be hard to go head to head with a sophisticated competitor ($|\mathcal{P}|$ is large). Theorem 3.4.9 confirms this, every segment for the competitor costs $O(L \log L)$ bits. Keep in mind that Theorem 3.4.9 holds for any competitor. Hence, it holds for the competitor that maximizes the redundancy, so we have a worst-case bound. We will shortly sketch a possible worst-case situation. In total there are $\Theta(\frac{n}{L})$ discounts and for every discount we have to pay a price of $O(\lambda L \log L + \log L)$ bits. We now give a disastrous scenario to interpret the cost per discount. To do so, consider a single discount. (The example can be generalized to any number of discounts.) Before the discount we encountered the sequence $00\dots0$, and $11\dots1$ afterwards. Right after the discount RFD will still assign high probability to letter 0. However, in this scenario it would be wise to discard the statistics from the previous segment. The smaller parameters λ and L, the better RFD will perform in this scenario, this adds redundancy $O(\lambda L \log L)$. But even if $\lambda = 0$, we still have to pay $O(\log L)$ bits for adaption. Combining this gives redundancy $O(\lambda L \log L + \log L)$ bits.

As seen in the previous section, we can vary the overall redundancy by tuning parameters L and λ. If we choose $\lambda = c/L$ (keep in mind that λ has to be bounded away from 1!), the bound of Theorem 3.4.9 becomes

$$O\left(|\mathcal{P}| \cdot L \log L + \frac{cn \log L}{L}\right). \tag{3.24}$$

Unfortunately, it is hard to tune the parameter L, since this requires knowledge of desirable $|\mathcal{P}|$. At this point we just give an example to illustrate the influence of the parameters. The choice of $L = \sqrt{n}$ for $c = O(1)$ asymptotically minimizes (3.24), when $|\mathcal{P}|$ is $O(1)$, and we still guarantee sublinear redundancy whenever $|\mathcal{P}| = o(\sqrt{n}/\log n)$. For the given parameter configuration we get redundancy $O(|\mathcal{P}| \cdot \sqrt{n} \log n)$.

Analysis: PWS with Bounded Letter Probabilities. Typically we do not know the sequence length n in advance, thus we cannot easily tune parameters λ and L for sublinear redundancy for specific $|\mathcal{P}|$. To iron out this deficit, we give a weaker version of Theorem 3.4.9, with the following rationale: It is unfair to compare RFD to a (sequence of) *arbitrary* probability distributions, since RFD can assign at most probability $\frac{T-N+1}{T} < 1$, but not probability 1, to any letter. Therefore the per-letter code length is bounded away from 0. So it is fairer to choose a competitor based on probability distributions with probabilities bounded away from 1. This leads to the following result.

Corollary 3.4.10 *Fix $0 < \varepsilon \leqslant \frac{1}{N}$ and an arbitrary PWS competitor* $\mathsf{PWS}\langle \mathcal{P}, \{p_s\}_{s\in\mathcal{P}}\rangle$*, where $p_s(x) \geqslant \varepsilon$, for all segments $s \in \mathcal{P}$ and any letter x. If we choose*

$$\delta := \frac{1}{\varepsilon \log e} \cdot \left[\frac{\lambda \log eL}{1-\lambda} + \frac{A \log L + B}{(1-\lambda)L}\right],$$

with A, B and C from Lemma 3.4.8 and if Assumption 3.4.1 holds for the RFD model RFD*, then we have*

$$\ell(x_{1:n}; \mathsf{RFD}) \leqslant (1+\delta)\cdot \ell(x_{1:n}; \mathsf{PWS}) + |\mathcal{P}|\,[L \log eL + A \log L + B].$$

Proof. For an arbitrary distribution p that satisfies $p(x) \geqslant \varepsilon$ we have

$$\ell(x;p) \geqslant \log \frac{1}{1-\varepsilon} \geqslant \varepsilon \log e \implies \ell(x_{1:n}; \mathsf{PWS}) \geqslant n\varepsilon \log e. \tag{3.25}$$

To end the proof, we combine the bound of Theorem 3.4.9 with

$$n\left[\frac{\lambda \log eL}{1-\lambda} + \frac{A \log L + B}{(1-\lambda)L}\right] = \delta \cdot n\varepsilon \log e \overset{(3.25)}{\leqslant} \delta \cdot \ell(x_{1:n}; \mathsf{PWS}). \qquad \square$$

Discussion. Corollary 3.4.10 states that the code length of RFD will be within $1+\delta$ times the code length of the competing PWS plus $O(|\mathcal{P}| \cdot L \log L)$ bits of redundancy. For fixed ε, fixed alphabet size N and λ bounded away from 1, i. e. $\lambda \leqslant \lambda' < 1$ (these are mild restrictions, overall), we get

$$\delta = O\left(\lambda \log L + \frac{\log L}{L}\right),$$

which should ideally be as small as possible. Indeed, for $\lambda = o(\log^{-1} L)$, we can decrease δ by increasing L. So δ can be made arbitrarily small at the cost of increasing the additive $O(|\mathcal{P}| \cdot L \log L)$ bits of redundancy.

3.5 Probability Smoothing

In this section we present our results on Probability Smoothing (PS). We first introduce the model, discuss its parameters (Section 3.5.1) and subsequently provide a code length analysis for *binary sequences*. For the analysis we first consider a special case, that is, in Section 3.5.2 we compare PS to the empirical entropy. Subsequently, in Section 3.5.3 we generalize the analysis to cover PWS-competitors as well. Finally, in Section 3.5.4 we provide actual smoothing rate choices along with corresponding code length bounds.

3.5.1 The Setting

Model PS. We now turn towards PS, allowing for varying smoothing rates and for an arbitrary initial prediction. To be more specific we pin down the probability assignment rule as follows:

Definition 3.5.1 *For a sequence $\alpha_{1:\infty}$ of smoothing rates, where $0 < \alpha_1, \alpha_2, \dots < 1$, and a probability distribution p, where $p(0), p(1) > 0$, we define the model $\mathsf{PS}\langle\alpha_{1:\infty}, p\rangle$ by its prediction rule*

$$\mathsf{PS}(x; x_{1:t}) = \begin{cases} \alpha_t \cdot \mathsf{PS}(x; x_{<t}) + 1 - \alpha_t, & \text{if } t > 0 \text{ and } x = x_t, \\ \alpha_t \cdot \mathsf{PS}(x; x_{<t}), & \text{if } t > 0 \text{ and } x \neq x_t \\ p(x), & \text{if } t = 0. \end{cases}$$

Parameters of PS. Two parameters influence the behavior of PS, that is, the smoothing rates $\alpha_{1:\infty}$ and the initial distribution p. The smoothing rates control the adaption of PS, large α_t's give high weight to old letters and low weight to recent letters, the converse holds for small α_t's. The initial distribution p mostly determines the early predictions, when t is small (what "small" precisely translates to depends on the smoothing rates). Throughout our analysis (Section 3.5) we assume the following on these parameters:

Assumption 3.5.2 *We consider PS for* bit sequences *and parameters s. t.:*

(a) *For all smoothing rates we have $\frac{1}{2} < \alpha_1, \alpha_2, \dots < 1$,*
(b) *for $t > 1$ smoothing rates satisfy $(1 - \alpha_{t-1}) \cdot \alpha_t \leqslant 1 - \alpha_t$, and*
(c) *the initial distribution p guarantees $p(0) \leqslant p(1)$.*

Assumption 3.5.2 (a) is a major prerequisite for our analysis, since we will rely on smoothing rates that are sufficiently large. Furthermore, Assumption 3.5.2 (b) may be rewritten as

$$\alpha_t \leqslant \frac{1}{2 - \alpha_{t-1}},$$

which implies that smoothing rates must not increase too fast, see Figure 3.4. Assumption 3.5.2 (c) is just a minor technical condition (which we may actually assume w. l. o. g.; if it does not hold, we simply flip 0-bits and 1-bits) to simplify the analysis. As we will see later, the link between smoothing rates and code length guarantees can be expressed depending on products of smoothing rates. Given the smoothing rate sequence $\alpha_{1:\infty}$, we define

$$\beta_0 := 1 \text{ and } \beta_t := \alpha_1 \cdot \ldots \cdot \alpha_t, \text{ for } t > 0. \tag{3.26}$$

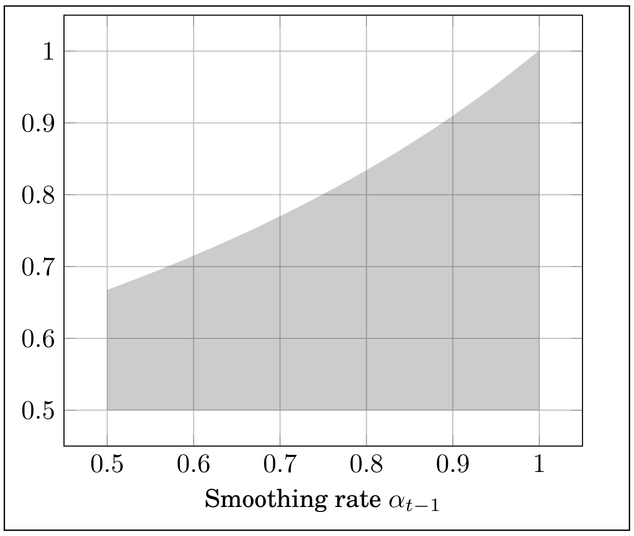


Figure 3.4: Set of admissible smoothing rates implied by Assumption 3.5.2: Given a smoothing rate $\frac{1}{2} < \alpha_{t-1} = \alpha < 1$ the follow-up smoothing rate α_t must lie in $\{\alpha' \mid \frac{1}{2} < \alpha' \leqslant \frac{1}{2-\alpha}\}$.

3.5.2 Probability Smoothing vs. Empirical Entropy

The Plan. In the remainder of Section 3.5.2 we will bound the code length of model $\mathsf{PS}\langle\alpha_{1:\infty}, p\rangle$ in terms of the empirical entropy for arbitrary bit sequences of length n. Our strategy is to identify the sequence $x_{1:n}$ that maximizes the redundancy

$$r(x_{1:n}; \mathsf{PS}) := \ell(x_{1:n}; \mathsf{PS}) - h(x_{1:n}). \tag{3.27}$$

To do so, we partition the set of all 2^n bit sequences into deterministic and non-deterministic sequences: A sequence $x_{1:n}$ is called *deterministic*, if $x_1 = x_2 = \dots = x_n$; otherwise $x_{1:n}$ is called *non-deterministic*. Based on this partition we identify the maximizer of (3.27) for deterministic and non-deterministic sequences. So the set of candidates for possible maximizers of (3.27) boils to just *two* candidates. In Table 3.2 we summarize the most important notation which we rely on in Section 3.5. With a slight abuse of notation, we use $H(p)$ to denote the binary entropy function given a *probability* p, rather than a distribution p. We only use this convention within Section 3.5.2 and Section 3.5.3.

Analysis. For the analysis it is crucial to understand the way probability assignment works for deterministic sequences. As an example consider the probability assignment for a 1-bit while PS operates on a deterministic sequence $x_{1:n} = 00\dots0$ of 0-bits. Initially, we have $\mathsf{PS}(1; \phi) = p(1)$ and after

Table 3.2: Notation used for the code length analysis of PS within Section 3.5.

Symbol	Definition	Symbol	Definition
$x_{1:n}$	input bit sequence, deterministic, if $x_1 = \cdots = x_n$	$\mathsf{PS}(x; x_{<t})$	PS' prediction, see Definition 3.5.1
		α_t	smoothing rate from $(\frac{1}{2}, 1)$ in step t
$h(x_{1:n})$	empirical entropy of $x_{1:n}$	β_t	$\alpha_1 \cdot \ldots \cdot \alpha_t$ (recall that $\beta_0 = 1$)
$H(p)$	$p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$	p	initial distribution, $\mathsf{PS}(x; \phi) = p$

observing the first 0-bit, the probability of a 1-bit decreases by a factor of α_1 (see Definition 3.5.1), so we have $\mathsf{PS}(1; x_1) = \alpha_1 \cdot p(1)$. Consequently, after observing a 0-bit $t-1$ times, the probability of a 1-bit shrinks by a factor of $\alpha_1 \cdot \ldots \cdot \alpha_{t-1} = \beta_{t-1}$ and we obtain

$$\mathsf{PS}(1; x_{<t}) = p(1) \cdot \beta_{t-1} \text{ and } \mathsf{PS}(0; x_{<t}) = 1 - p(1) \cdot \beta_{t-1}, \text{ for } x_{<t} = 00\ldots0. \tag{3.28}$$

With this in mind, it is easy to identify the deterministic sequence of maximum redundancy.

Lemma 3.5.3 *The sequence $x_{1:n} = 00\ldots0$ (only 0-bits) maximizes (3.27), among all deterministic sequences.*

Proof. For any deterministic sequence we have $h(x_{1:n}) = 0$, so we prove $\ell(00\ldots0; \mathsf{PS}) \geqslant \ell(11\ldots1; \mathsf{PS})$. For a deterministic sequence of 0-bits we get

$$\ell(00\ldots0; \mathsf{PS}) \overset{(3.28)}{=} \sum_{0 \leqslant t < n} \log \frac{1}{1 - p(1)\beta_t} \overset{\text{A3.5.2 (c)}}{\geqslant} \sum_{0 \leqslant t < n} \log \frac{1}{1 - p(0)\beta_t} \overset{(3.28)}{=} \ell(11\ldots1; \mathsf{PS}).$$

□

We now shift our attention to non-deterministic sequences, for which the following holds:

Lemma 3.5.4 *The sequence $00\ldots01$ (a single 1-bit at the end) maximizes (3.27), among all non-deterministic sequences.*

We split off the following technical statement, before we proceed with the proof of Lemma 3.5.4.

Lemma 3.5.5 *Any non-deterministic sequence $x_{1:n}$ satisfies*

$$h(x_{1:n}) - h(x_{2:n}) \geqslant \begin{cases} nH\left(\frac{1}{n}\right), & \text{if } x_{2:n} \text{ is deterministic,} \\ nH\left(\frac{1}{n}\right) - (n-1)H\left(\frac{1}{n-1}\right), & \text{otherwise.} \end{cases}$$

Proof. We have $n \geqslant 2$; let $1-p$ be the relative frequency of x_1 in $x_{1:n}$, thus

$$h(x_{1:n}) - h(x_{2:n}) = nH(p) - (n-1)H\left(\tfrac{n}{n-1}\cdot p\right) =: f(p).$$

We distinguish two cases:

Case 1: $x_{2:n}$ is deterministic. We have $p = \frac{n-1}{n}$, so $f(p) = nH\left(\frac{n-1}{n}\right) = nH\left(\frac{1}{n}\right)$.

Case 2: $x_{2:n}$ is non-deterministic. Since $H(p)$ is concave, $H'(p)$ is decreasing and $f'(p) = n\left[H'(p) - H'\left(\frac{n}{n-1}\cdot p\right)\right] \geqslant 0$, i. e. $f(p)$ is increasing and minimal for minimum p. Since $x_{1:n}$ is non-deterministic the minimum value of p is $\frac{1}{n}$ and we get $f(p) \geqslant f\left(\frac{1}{n}\right) = nH\left(\frac{1}{n}\right) - (n-1)H\left(\frac{1}{n-1}\right)$. □

We now proceed with the proof of the main technical lemma of this section.

Proof of Lemma 3.5.4. We have $n \geqslant 2$. By induction on n we prove

$$r(x_{1:n}; \mathsf{PS}) \leqslant \log\frac{1}{u\beta_{n-1}} + \sum_{0\leqslant t<n-1} \log\frac{1}{1-u\beta_t} - nH\left(\tfrac{1}{n}\right),$$

where $u := \max\{p(0), p(1)\}$, for arbitrary p. This proof does *not* rely on Assumption 3.5.2 (c)! (If actually $u = p(1)$, then by (3.27), (3.28) and by $h(00\ldots01) = nH(\frac{1}{n})$ the r. h. s. of the inequality equals $r(00\ldots01; \mathsf{PS})$.)

Base: $n = 2$. We have $x_{1:n} \in \{01, 10\}$ and in either case it holds that

$$\ell(x_{1:n}; \mathsf{PS}) = \log\frac{1}{p(x_1)\beta_1 p(x_2)} = \log\frac{1}{u\beta_1} + \log\frac{1}{1-u}$$

and furthermore, $h(x_{1:n}) = nH\left(\frac{1}{n}\right) = 2$, so the claim follows.

Step: $n > 2$. We distinguish:

Case 1: $x_{2:n}$ is non-deterministic. By $y_{1:n-1} = x_{2:n}$ and $\mathsf{PS}'\langle\alpha'_{1:\infty}, p'\rangle$, where $\alpha'_t = \alpha_{t+1}$, $\beta'_t = \alpha'_1 \cdot \ldots \cdot \alpha'_t$ and $p' = \mathsf{PS}(x_{\leqslant 1})$ we may write

$$r(x_{1:n}; \mathsf{PS}) = \log\frac{1}{p(x_1)} + r(y_{1:n-1}, \mathsf{PS}') - (h(x_{1:n}) - h(x_{2:n})).$$

If we let $u' = \max\{p'(0), p'(1)\}$, Lemma 3.5.5 and the induction hypothesis (it is easy to see that Assumption 3.5.2 holds for PS' as well) yield

$$r(x_{1:n}; \mathsf{PS}) \overset{\text{I. H.,}\atop\text{L3.5.5}}{\leqslant} \log\frac{1}{p(x_1)} + \log\frac{1}{u'\beta'_{n-2}} + \sum_{0\leqslant t<n-2} \log\frac{1}{1-u'\beta'_t} - nH(\tfrac{1}{n}). \quad (3.29)$$

Since we want to bound (3.29) from above, we must choose x_1 s. t. $p(x_1)$ is minimal (and the r. h. s. of (3.29) is maximal). To do so, we w. l. o. g. assume $u' = p'(1)$ (the other case is symmetric) and distinguish:

Case 1a: $x_1 = 0$. For a distribution q, with $q(0) > 0$, we have $p(x_1) = q(0)$,

$$p'(0) = \alpha_1 q(0) + 1 - \alpha_1 \text{ and } p'(0) \leqslant \frac{1}{2} \implies q(0) \leqslant \frac{\alpha_1 - \frac{1}{2}}{\alpha_1}.$$

(Notice the subtle detail: $\alpha_1 \leqslant \frac{1}{2}$ implies $q(0) \leqslant 0$, which contradicts $q(0) > 0$ and would make Case 1a impossible; however we assumed $\alpha_1 > \frac{1}{2}$ in Assumption 3.5.2 (a)) Furthermore, we have $q(0) \leqslant \frac{1}{2}$.

Case 1b: $x_1 = 1$. For a distribution r, with $r(1) > 0$, we have $p(x_1) = r(1)$,

$$p'(1) = \alpha_1 r(1) + 1 - \alpha_1 \text{ and } p'(1) \geqslant \frac{1}{2} \implies r(1) \geqslant \frac{\alpha_1 - \frac{1}{2}}{\alpha_1}.$$

Since $q(0) \leqslant r(1)$ (i. e. Case 1a minimizes $p(x_1)$) we may now w. l. o. g. assume that Case 1a occurred. In this situation we have $u = \max\{q(0), q(1)\} = q(1)$ (since $q(0) \leqslant \frac{1}{2}$) and $u' = p'(1) = \alpha_1 u$, so

$$u'\beta'_t = u\beta_{t+1} \quad \text{and} \quad p(x_1) = 1 - u\beta_0 \quad (\beta_0 = 1). \tag{3.30}$$

It remains to plug (3.30) into (3.29) and to rearrange.

Case 2: $x_{2:n}$ is deterministic. We will reduce this case to Case 1. W. l. o. g. we assume $x_{1:n} = 100\ldots0$, set $y_{1:n} = 010\ldots0$ ($y_{2:n}$ is non-deterministic!) and for the reduction we prove $r(x_{1:n}; \mathsf{PS}) \leqslant r(y_{1:n}; \mathsf{PS})$. We have $h(x_{1:n}) = h(y_{1:n}) = nH(\frac{1}{n})$, so it suffices to show $\ell(x_{1:n}; \mathsf{PS}) \leqslant \ell(y_{1:n}; \mathsf{PS})$, which we now do by showing $\ell(x_{a:b}; \mathsf{PS}) \leqslant \ell(y_{a:b}; \mathsf{PS})$, for all segments $a{:}b \in \{1{:}2, 3{:}n\}$:

Segment $1{:}2$. We obtain $\ell(x_{1:2}; \mathsf{PS}) = \ell(y_{1:2}; \mathsf{PS})$, since

$$\ell(x_{1:2}; \mathsf{PS}) = \log \frac{1}{p(1)\alpha_1 p(0)} \text{ and } \ell(y_{1:2}; \mathsf{PS}) = \log \frac{1}{p(0)\alpha_1 p(1)}.$$

Segment $3{:}n$. For $0 < z < 1$ we define

$$\begin{aligned} f(z) &:= \sum_{2 \leqslant t < n} \log \frac{1}{1 - z\beta_t/\beta_2}, \\ u &:= \mathsf{PS}(1; x_{1:2}) = \alpha_1\alpha_2 \cdot \mathsf{PS}(1; \phi) + (1 - \alpha_1)\alpha_2, \\ v &:= \mathsf{PS}(1; y_{1:2}) = \alpha_1\alpha_2 \cdot \mathsf{PS}(1; \phi) + 1 - \alpha_2 \end{aligned}$$

and may write $\ell(x_{3:n}; \mathsf{PS}) = f(u)$ and $\ell(y_{3:n}; \mathsf{PS}) = f(v)$ (sequences $x_{3:n}$ and $y_{3:n}$ are deterministic sequences of 0-bits). Since $f(z)$ is increasing and by Assumption 3.5.2 (b) we have $u \leqslant v$, we conclude $\ell(x_{3:n}; \mathsf{PS}) \leqslant \ell(y_{3:n}; \mathsf{PS})$. □

Lemma 3.5.3 and Lemma 3.5.4 imply the main result of this section. (We omit the proof, since it is trivial.)

Theorem 3.5.6 *Suppose that Assumption 3.5.2 holds. Either sequence* $00\dots0$ *(only* 0*-bits) or* $00\dots1$ *(a single* 1*-bit at the end) maximizes (3.27).*

Discussion. The above theorem is a remarkable result: Out of all 2^n possible candidates for a maximizer of (3.27), we just need to compare the redundancy incurred by two sequence, $00\dots0$ and $00\dots01$, to precisely identify the worst-case input. (If we have $p(0) > p(1)$, the corresponding candidates are $11\dots1$ (only 1-bits) and $11\dots10$ (a single 0-bit at the and), by symmetry.) Even better, the redundancy for these candidates may be evaluated by simple formulas,

$$r(00\dots0;\mathsf{PS}) \overset{(3.28)}{=} \sum_{0\leqslant t<n} \log\frac{1}{1-p(1)\beta_t},$$

and

$$r(00\dots01;\mathsf{PS}) \overset{(3.28)}{=} \sum_{0\leqslant t<n-1} \log\frac{1}{1-p(1)\beta_t} + \log\frac{1}{p(1)\beta_{n-1}} - nH\left(\tfrac{1}{n}\right),$$

If we take the maximum of these formulas and rearrange, Theorem 3.5.6 implies

$$\ell(x_{1:n};\mathsf{PS}) \leqslant h(x_{1:n}) + \max\left\{\log\frac{1}{1-p(1)\beta_{n-1}}, \log\frac{1}{p(1)\beta_{n-1}} - nH(\tfrac{1}{n})\right\} + \sum_{0\leqslant t<n-1} \log\frac{1}{1-p(1)\beta_t},$$

so the code length (and hence, the redundancy) may be bounded by a function depending on products $\beta_t = \alpha_1 \cdot \ldots \cdot \alpha_t$ of smoothing rates and depending on the initial distribution p.

3.5.3 Probability Smoothing vs. Piecewise Stationary Models

Analysis. We generalize the code length analysis for PS (w. r. t. PWS) similarly to the way we did for RFD. First, we consider the contribution of a single segment of the competing PWS to the total redundancy.

Lemma 3.5.7 *We have* $r(x_{a+1:b}; \mathsf{PS}) \leqslant \log \dfrac{1}{p(0)\beta_{b-1}} + \sum_{a<t<b} \log \dfrac{1}{1-\beta_t/\beta_a}$.

Proof. We define $y_{1:b-a} := x_{a+1:b}$ and $\mathsf{PS}'\langle\alpha'_{1:\infty}, p'\rangle$, for smoothing rates $\alpha'_t = \alpha_{a+t}$, furthermore $\beta'_t = \alpha_{a+1} \cdot \ldots \cdot \alpha_t$, and initial prediction $p' = \mathsf{PS}(x_{1:a})$. So we get $r(x_{a+1:b}; \mathsf{PS}) = r(y_{1:b-a}, \mathsf{PS}')$. Below we only consider the case $p'(0) \leqslant p'(1)$, the other case follows similarly. It is easy to see that Assumption 3.5.2 holds for PS'. We bound $r(y_{1:b-a}, \mathsf{PS}')$ by distinguishing:

Case 1: $x_{a+1:b}$ is deterministic. From Theorem 3.5.6 we conclude

$$
\begin{aligned}
r(y_{1:b-a}, \mathsf{PS}') &\overset{\text{T3.5.6}}{\leqslant} \log \frac{1}{p'(0)} + \sum_{0<t<b-a} \log \frac{1}{1-p'(1)\beta'_t} \\
&\leqslant \log \frac{1}{p(0)\beta_a} + \sum_{a<t<b} \log \frac{1}{1-\beta_t/\beta_a},
\end{aligned} \tag{3.31}
$$

where we used $p'(0) \geqslant \beta_a p(0)$ and $\beta'_t = \beta_{t+a}/\beta_a$ to get inequality (3.31).

Case 2: $x_{a+1:b}$ is non-deterministic. Similarly to the previous case we get

$$
\begin{aligned}
r(y_{1:b-a}, \mathsf{PS}') &\overset{\text{T3.5.6}}{\leqslant} \log \frac{1}{p'(1)\beta'_{b-a-1}} + \sum_{0\leqslant t<b-a-1} \log \frac{1}{1-p'(1)\beta'_t} - (b-a)H\left(\tfrac{1}{b-a}\right) \\
&\leqslant \log \frac{1}{p(1)\beta_{b-1}} + \sum_{a<t<b-1} \log \frac{1}{1-\beta_t/\beta_a},
\end{aligned} \tag{3.32}
$$

where we used $\log \frac{1}{p'(1)} - (b-a)H\left(\frac{1}{b-a}\right) \leqslant 0$ (by $p'(1) \geqslant \frac{1}{2}$ and by $b-a \geqslant 2$, since $x_{a+1:b}$ is non-deterministic) and $\beta'_t = \beta_{t+a}/\beta_a$ to yield inequaliy (3.32).

Both bounds, (3.31) and (3.32), may further be bounded from above by the claimed inequality, since $p(0)\beta_a, \beta_{b-1}p(1) \geqslant p(0)\beta_{b-1}$ (Assumption 3.5.2 (c)). (For the case $p'(0) \geqslant p'(1)$ we similarly have $p(1)\beta_a, \beta_{b-1}p(0) \geqslant p(0)\beta_{b-1}$.) □

Based on the redundancy of a single segment we can easily argue on the total redundancy of all segments (equivalently, the total code length of all segments) that belong to the partition $\mathcal{P}$ of some PWS, by summing the contribution in redundancy of every individual segment. The result is the following theorem:

Theorem 3.5.8 *Suppose that Assumption 3.5.2 holds for the PS model* PS. *For an arbitrary PWS competitor* $\mathsf{PWS}\langle\mathcal{P}, \{p_s\}_{s\in\mathcal{P}}\rangle$ *we have*

$$
\ell(x_{1:n}; \mathsf{PS}) \leqslant \ell(x_{1:n}; \mathsf{PWS}) + |\mathcal{P}| \cdot \log \frac{1}{p(0)\beta_{n-1}} + \sum_{a+1:b\in\mathcal{P}} \sum_{a<t<b} \log \frac{1}{1-\beta_t/\beta_a}.
$$

Proof. For any segment $s = a{+}1{:}b \in \mathcal{P}$ we have

$$\begin{aligned}\ell(x_{a+1:b}; \mathsf{PS}) - \ell(x_{a+1:b}; \mathsf{PWS}) &\overset{\text{D3.1.1}}{=} \ell(x_{a+1:b}; \mathsf{PS}) - \ell(x_{a+1:b}; p_s)\\ &\overset{(3.27)}{\leqslant} r(x_{a+1:b}; \mathsf{PS}),\end{aligned} \tag{3.33}$$

so we apply Lemma 3.5.7 to every segment of $\mathcal{P}$ and sum over all segments:

$$\begin{aligned}\ell(x_{1:n}; \mathsf{PS}) - \ell(x_{1:n}; \mathsf{PWS}) &\overset{(3.33)}{\leqslant} \sum_{a+1:b\in\mathcal{P}} r(x_{a+1:b}; \mathsf{PS})\\ &\overset{\text{L3.5.7}}{\leqslant} \sum_{a+1:b\in\mathcal{P}} \left(\log \frac{1}{p(0)\beta_{b-1}} + \sum_{a<t<b} \log \frac{1}{1-\beta_t/\beta_a} \right).\end{aligned}$$

We use $\beta_{b-1} \geqslant \beta_{n-1}$ and rearrange to conclude the proof. □

Discussion. Theorem 3.5.8 suggests that for a good redundancy guarantee the products of smoothing rates may neither be too large, nor too small: On the one hand, the term $\log \frac{1}{\beta_{n-1}}$ may dominate the redundancy, whenever $\beta_{n-1} = \alpha_1 \cdot \ldots \cdot \alpha_{n-1}$ is close to 0; on the other hand, the sum of terms $\log \frac{1}{1-\beta_t/\beta_a}$ dominates the redundancy, when $\beta_t/\beta_a = \alpha_{a+1} \cdot \ldots \cdot \alpha_t$ is close to 1. So depending on the exact choice of smoothing rates it is crucial to balance the contribution of these terms. We will explore this aspect for particular smoothing rate sequences in the next section.

3.5.4 Choice of Smoothing Rates

Fixed Smoothing Rate. A straightforward choice for the smoothing rates is to use the same smoothing rate α in every step,

$$\alpha = \alpha_1 = \alpha_2 = \ldots, \quad \text{for } \frac{1}{2} < \alpha < 1. \tag{3.34}$$

This leads to a simple and fast implementation, since no smoothing rate sequence needs to be computed or stored.

Recall that our results rely on Assumption 3.5.2, so we must verify that the required prerequisites are met: Obviously Assumption 3.5.2 (a) is satisfied, Assumption 3.5.2 (b) holds automatically,

$$(1-\alpha_{t-1}) \cdot \alpha_t \leqslant 1 - \alpha_t \overset{\alpha_{t-1}=\alpha_t=\alpha}{\iff} 0 \leqslant 1 - 2\alpha + \alpha^2 = (1-\alpha)^2.$$

(Recall that Assumption 3.5.2 (c) is just a technical condition which we may assume w. l. o. g.)

We require the following lemma for the analysis:

Lemma 3.5.9 *For* $0 < \alpha < 1$ *we have* $\sum_{t \geqslant 1} \log \frac{1}{1-\alpha^t} \leqslant \frac{(\pi \log e)^2}{6 \log \frac{1}{\alpha}}$.

Proof. Since $\log \frac{1}{1-\alpha^z}$ is decreasing in z and integrable for z in $[0, \infty)$ we may bound the series by an integral,

$$\sum_{t \geqslant 1} \log \frac{1}{1-\alpha^t} \leqslant \int_0^\infty \log \frac{1}{1-\alpha^z} \mathrm{d}z = \log e \int_0^\infty \sum_{j \geqslant 1} \frac{\alpha^{jz}}{j} \mathrm{d}z. \tag{3.35}$$

The equality in (3.35) follows from the series expansion $\ln \frac{1}{1-y} = \sum_{j \geqslant 1} y^j/j$, for $|y| < 1$. To end the proof, it remains to bound the integral in (3.35) as follows (recall that $\sum_{j \geqslant 1} j^{-2} = \pi^2/6$):

$$\int_0^\infty \sum_{j \geqslant 1} \frac{\alpha^{jz}}{j} \mathrm{d}z = \sum_{j \geqslant 1} \frac{1}{j} \int_0^\infty \alpha^{jz} \mathrm{d}z = \frac{\log e}{\log \frac{1}{\alpha}} \sum_{j \geqslant 1} \frac{1}{j^2} = \frac{\pi^2 \log e}{6 \log \frac{1}{\alpha}}. \qquad \square$$

Corollary 3.5.10 *Consider the PS model* $\mathsf{PS}\langle \alpha_{1:\infty}, p \rangle$ *for smoothing rate choice (3.34) and* $p(0) \leqslant p(1)$. *For any PWS competitor* $\mathsf{PWS}\langle \mathcal{P}, \{p_s\}_{s \in \mathcal{P}} \rangle$ *we have*

$$\ell(x_{1:n}; \mathsf{PS}) \leqslant \ell(x_{1:n}; \mathsf{PWS}) + |\mathcal{P}| \cdot \left[\log \frac{1}{p(0)} + \frac{(\pi \log e)^2}{6 \log \frac{1}{\alpha}} + (n-1) \log \frac{1}{\alpha} \right].$$

Proof. It is easy to see that Assumption 3.5.2 is satisfied (recall the discussion at the beginning of this section). We have $\beta_t = \alpha^t$, thus for the segment $a{+}1{:}b \in \mathcal{P}$ we combine

$$\sum_{a<t<b} \log \frac{1}{1-\beta_i/\beta_a} = \sum_{0<t-a<b-a} \log \frac{1}{1-\alpha^{t-a}} \overset{\text{L3.5.9}}{\leqslant} \frac{(\pi \log e)^2}{6 \log \frac{1}{\alpha}}$$

and $\log \beta_{n-1} = (n-1) \log \alpha$ with Theorem 3.5.8 to conclude the proof. $\square$

By the choice $\alpha = e^{-\pi/\sqrt{6(n-1)}}$ we minimize the code length bound of Corollary 3.5.10 and guarantee $\alpha > \frac{1}{2}$ (Assumption 3.5.2 (a)), when $n \geqslant 5$. The optimal choice gives redundancy at most

$$|\mathcal{P}| \cdot \left[\frac{2\pi \log e}{\sqrt{6}} \cdot \sqrt{n} + \log \frac{1}{p(0)} \right] < |\mathcal{P}| \cdot \left[3.701 \cdot \sqrt{n} + \log \frac{1}{p(0)} \right]. \tag{3.36}$$

Slowly Varying Smoothing Rate. It is impossible to choose an optimal fixed smoothing rate when n is unknown. A standard technique to handle this situation is the doubling trick, which will increase the $\sqrt{n}$-term in (3.36) by a factor of $\sqrt{2}/(\sqrt{2}-1) \approx 3.41$. However, we can do better by slowly increasing the smoothing rate step-by-step, which only leads to a factor $\sqrt{2} \approx 1.41$. The smoothing rate choice we propose is

$$\alpha_t = e^{-\pi/\sqrt{12(t+1)}}, \text{ for } t \geqslant 1. \tag{3.37}$$

Just as in the previous section, we must make sure that (3.37) satisfies Assumption 3.5.2. Since this process is more technical, we wrap it into the following lemma.

Lemma 3.5.11 *Smoothing rate choice (3.37) satisfies Assumption 3.5.2.*

Proof. Since smoothing rates increase, we have $\alpha_t \geqslant \alpha_1 = 0.536\dots > \frac{1}{2}$, hence Assumption 3.5.2 (a) is satisfied. We now show

$$0 \leqslant \frac{1}{\alpha_t} + \alpha_{t-1} - 2, \quad \text{for } t > 1, \tag{3.38}$$

which is equivalent to Assumption 3.5.2 (b). Using the inequalities $e^{-z} \geqslant 1 - z$, for $e^z = \alpha_{t-1}$, and $e^z \geqslant 1 + z + \frac{z^2}{2}$, for $e^z = \frac{1}{\alpha_t}$, and using (3.37) we bound the r. h. s. of (3.38) from below as follows,

$$\frac{1}{\alpha_t} + \alpha_{t-1} - 2 \geqslant \frac{\pi}{\sqrt{12(t+1)}} + \frac{\pi^2}{24(t+1)} - \frac{\pi}{\sqrt{12t}}. \tag{3.39}$$

It remains to show that the r. h. s. of (3.39) is non-negative:

$$\begin{aligned}
& & 0 &\leqslant \frac{\pi}{\sqrt{12(t+1)}} + \frac{\pi^2}{24(t+1)} - \frac{\pi}{\sqrt{12t}} \\
&\Longleftrightarrow & 0 &\leqslant \frac{\pi}{2\sqrt{12}(t+1)} - \frac{\sqrt{t+1}-\sqrt{t}}{\sqrt{t(t+1)}} \\
&\Longleftarrow & 0 &\leqslant \frac{\pi}{2\sqrt{12}(t+1)} - \frac{1}{2t\sqrt{t+1}} && (3.40) \\
&\Longleftrightarrow & \frac{\sqrt{t+1}}{t} &\leqslant \frac{\pi}{\sqrt{12}}. && (3.41)
\end{aligned}$$

Inequality (3.41) holds for all $t > 1$, so (3.38) holds. To obtain the the stronger condition (3.40) we utilized $\sqrt{t+1} \leqslant \sqrt{t} + \frac{1}{2\sqrt{t}}$ (by a Taylor expansion and concavity of $\sqrt{z}$). □

We see that (3.37) meets the requested criteria, so we may now conclude a corresponding redundancy bound.

Corollary 3.5.12 *Consider the PS model* $\mathsf{PS}\langle\alpha_{1:\infty}, p\rangle$ *for smoothing rate choice (3.37) and* $p(0) \leqslant p(1)$. *For any PWS competitor* $\mathsf{PWS}\langle\mathcal{P}, \{p_s\}_{s\in\mathcal{P}}\rangle$ *we have*

$$\ell(x_{1:n}; \mathsf{PS}) \leqslant \ell(x_{1:n}; \mathsf{PWS}) + |\mathcal{P}| \cdot \left[\log \frac{1}{p(0)} + \frac{2\pi \log e}{\sqrt{3}} \cdot \sqrt{n}\right]. \tag{3.42}$$

Proof. By Lemma 3.5.11 Assumption 3.5.2 is satisfied. For the proof we bound the terms in Theorem 3.5.8 depending on β_t's from above. We have

$$\beta_t \overset{(3.26)}{=} \alpha_1 \cdot \ldots \cdot \alpha_t = \exp\left(-\frac{\pi}{\sqrt{12}} \sum_{1<i\leqslant t+1} i^{-1/2}\right) \tag{3.43}$$

First, observe that

$$\sum_{1<i\leqslant n} i^{-1/2} \leqslant \int_1^n \frac{\mathrm{d}z}{\sqrt{z}} \leqslant 2\sqrt{n} \overset{(3.43)}{\Longrightarrow} \log \frac{1}{\beta_{n-1}} \leqslant \frac{\pi \log e}{\sqrt{3}} \sqrt{n}, \tag{3.44}$$

second, for $a < t < b$ we have $\beta_t/\beta_a = \alpha_{a+1} \cdot \ldots \cdot \alpha_t \leqslant (\alpha_{n-1})^{t-a}$, since $t < n$ and $\alpha_1, \alpha_2, \ldots$ is increasing, consequently we obtain

$$\sum_{a<t<b} \log \frac{1}{1 - \beta_i/\beta_a} \leqslant \sum_{a<t<b} \log \frac{1}{1 - (\alpha_{n-1})^{t-a}} \overset{\text{L3.5.9}}{\leqslant} \frac{(\pi \log e)^2}{6 \log \frac{1}{\alpha_{n-1}}} = \frac{\pi \log e}{\sqrt{3}} \sqrt{n}. \tag{3.45}$$

We combine (3.44) and (3.45) with Theorem 3.5.8 to end the proof. □

3.6 Relative Frequencies with Smoothing

We now turn to the final probability estimation scheme of this chapter, Relative Frequencies with Smoothing (RFS). Similarly to the previous sections, we first define the model, discuss its parameters and finally provide a code length analysis w. r. t. PWS for *bit sequences*. In contrast to the previous sections on RFD and PS we proceed more quickly, since RFS may be viewed as a variation of PS (see below).

Model RFS. In this section we consider RFS where the smoothing rate λ is fixed in every step, more formally:

Definition 3.6.1 *For an integer $m \geqslant 0$, smoothing rate $\lambda \in (0,1)$ and a probability distribution p with non-zero probabilities let*

$$F_t := \sum_{0 \leqslant i < m+t} \lambda^i \quad \textit{and} \quad f_t(x) := \begin{cases} \lambda \cdot f_{t-1}(x) + 1, & \textit{if } t > 1 \textit{ and } x_{t-1} = x, \\ \lambda \cdot f_{t-1}(x), & \textit{if } t > 1 \textit{ and } x_{t-1} \neq x, \\ p(x) \cdot F_1, & \textit{if } t = 1. \end{cases}$$

We define the model $\mathsf{RFS}\langle \lambda, p, m \rangle$ *by its prediction* $\mathsf{RFS}(x; x_{<t}) = \frac{f_t(x)}{F_t}$.

Note that F_t, the sum of all frequencies, is a geometric series, so we have

$$F_t = \frac{1 - \lambda^{m+t}}{1 - \lambda} \quad \text{and} \quad F_t = \lambda F_{t-1} + 1, \text{ for } t > 1. \tag{3.46}$$

Parameters of RFS. RFS has three parameters, the initial prediction p, the smoothing rate λ and an integer m. Just as in case of PS, the smoothing rate λ determines the weight of past letters, by increasing λ, the weight of past letters increases and vice-versa. All parameters interact to set up the initial frequencies f_1: It is easy to see that f_1 is chosen so that p is the initial prediction, $\mathsf{RFS}(x; \phi) = p(x)$. The parameters m and λ control the magnitude of $f_1(x)$, by increasing m and/or λ, the magnitude of $f_1(x)$ increases. So by increasing this magnitude RFS will take more time to adapt to observed statistics (what "more time" precisely translates to also depends on λ, since it controls how oblivious RFS acts).

A Link to PS. Below we will provide an analysis of RFS for binary sequences which is based on a crucial observation, that is, we may regard RFS as an instance of probability smoothing for a particular choice of smoothing rates $\alpha_{1:\infty}$. To observe this consider an input sequence $x_{1:t}$, letter $x = x_t$ and the (probability) smoothing rate

$$\alpha_t := \frac{\lambda F_t}{\lambda F_t + 1}. \tag{3.47}$$

Based on Definition 3.6.1 we have

$$\begin{aligned} \mathsf{RFS}(x; x_{1:t}) = \frac{f_{t+1}(x)}{F_{t+1}} &= \frac{\lambda F_t}{\lambda F_t + 1} \cdot \frac{f_t(x)}{F_t} + \frac{1}{\lambda F_t + 1} \\ &= \alpha_t \cdot \mathsf{RFS}(x; x_{<t}) + 1 - \alpha_t. \end{aligned} \tag{3.48}$$

In the opposite case, if $x \neq x_t$, we must drop the additive term $1 - \alpha_t$ in (3.48). So Definition 3.6.1 is equivalent to

$$\mathsf{RFS}(x; x_{1:t}) = \begin{cases} \alpha_t \cdot \mathsf{RFS}(x; x_{<t}) + 1 - \alpha_t, & \text{if } t > 0 \text{ and } x = x_t, \\ \alpha_t \cdot \mathsf{RFS}(x; x_{<t}), & \text{if } t > 0 \text{ and } x \neq x_t, \\ p(x), & \text{if } t = 0, \end{cases} \tag{3.49}$$

matching PS, cf. Definition 3.5.1.

Verifying Assumption 3.5.2. Due to the equivalence of RFS and PS we may apply our analysis machinery of Section 3.5.4 to RFS. However, we first need to make sure that the smoothing rate choice (3.47) satisfies Assumption 3.5.2. In particular we must verify Assumption 3.5.2 (a) and Assumption 3.5.2 (b). (Recall that Assumption 3.5.2 (c) is just a technical condition which we may assume w. l. o. g..)

Let us first consider Assumption 3.5.2 (a), i. e. any smoothing rate α_t, see (3.47), must be bigger than $\frac{1}{2}$. Since F_t is increasing in t, the smoothing rate α_t also increases with t. Consequently, we have

$$\frac{1}{2} < \alpha_t \overset{\alpha_1 \leqslant \alpha_t}{\iff} \frac{1}{2} < \alpha_1 \overset{(3.47)}{=} \frac{\lambda F_1}{\lambda F_1 + 1} \iff 1 < \lambda F_1 \overset{(3.46)}{=} \sum_{1 \leqslant i \leqslant m+1} \lambda^i, \tag{3.50}$$

so we require $1 < \lambda F_1$, in order to satisfy Assumption 3.5.2 (a). Obviously, if $m = 0$, then (3.50) cannot hold, so this case is forbidden. If $m > 0$ we may easily satisfy (3.50) by increasing λ and/or m. For instance, for $m = 1$ (3.50) equals

$$1 < \lambda^2 + \lambda \iff \frac{\sqrt{5} - 1}{2} < \lambda,$$

so λ must exceed the golden ratio to satisfy (3.50) and to make Assumption 3.5.2 (a) valid.

It remains to check Assumption 3.5.2 (b), i. e. the smoothing rates must satisfy $(1 - \alpha_{t-1}) \cdot \alpha_t \leqslant 1 - \alpha_t$, for $t > 1$. By substituting (3.47) and by using (3.46), we observe that Assumption 3.5.2 (b) is always satisfied, since the condition of consideration turns into

$$\begin{aligned} (1 - \alpha_{t-1}) \cdot \alpha_t \leqslant 1 - \alpha_t &\overset{(3.47)}{\iff} \frac{1}{\lambda F_{t-1} + 1} \cdot \frac{\lambda F_t}{\lambda F_t + 1} \leqslant \frac{1}{\lambda F_t + 1} \\ &\overset{(3.46)}{\iff} \frac{\lambda}{\lambda F_t + 1} \leqslant \frac{1}{\lambda F_t + 1}. \end{aligned} \tag{3.51}$$

Analysis. We will carry out the analysis of RFS in brief, since it mostly follows the lines of Section 3.5.4. First, we require a technical statement.

Lemma 3.6.2 *For* $1 \leqslant a \leqslant b$ *and* $0 < \lambda < 1$ *we have* $\frac{1-\lambda^a}{1-\lambda^b} \geqslant \frac{a}{b}$.

Proof. Let $f(z) := \ln \frac{1-\lambda^z}{z}$. It suffices to prove that $f(a) \geqslant f(b)$. We obtain

$$f'(z) = \frac{g(\lambda^z) - 1}{z(1-\lambda^z)}, \text{ where } g(y) := y + y \ln \frac{1}{y}, \text{ for } 0 < y < 1.$$

The given range of y implies $0 \leqslant g'(y) = \ln \frac{1}{y}$, so $g(\lambda^z) \leqslant g(1) = 1$, since g is increasing. In turn we have $f'(z) \leqslant 0$, so f is decreasing. □

Corollary 3.6.3 *Consider the RFS model* $\mathsf{RFS}\langle \lambda, p, m\rangle$ *s. t. the initial distribution* p *satisfies* $p(0) \leqslant p(1)$ *and the parameters* $m \geqslant 1$ *and* λ *satisfy* $\sum_{1 \leqslant i \leqslant m+1} \lambda^i > 1$. *For any PWS competitor* $\mathsf{PWS}\langle \mathcal{P}, \{p_s\}_{s\in\mathcal{P}}\rangle$ *we have*

$$\ell(x_{1:n}; \mathsf{RFS}) \leqslant \ell(x_{1:n}; \mathsf{PWS}) + |\mathcal{P}| \cdot \left[\log \frac{n}{p(0)} + \frac{(\pi \log e)^2}{6 \log \frac{1}{\lambda}} + (n-1) \log \frac{1}{\lambda}\right].$$

Proof. By (3.49) RFS is equivalent to PS with smoothing rate α_t chosen according to (3.47). Furthermore, Assumption 3.5.2 is satisfied, in particular:

$$p(0) \leqslant p(1) \implies \text{Assumption 3.5.2 (c) is satisfied,}$$

$$1 < \sum_{1 \leqslant i \leqslant m+1} \lambda^i \text{ (for } m \geqslant 1) \overset{(3.50)}{\implies} \text{Assumption 3.5.2 (a) is satisfied and}$$

$$(3.51) \implies \text{Assumption 3.5.2 (b) is satisfied.}$$

Consequently, the code length bound given in Theorem 3.5.8 holds. For the remainder of this proof we simplify the terms depending on β_t's in that bound. By $F_{t+1} = \lambda F_t + 1$ we get

$$\beta_t = \alpha_1 \cdot \ldots \cdot \alpha_t \overset{(3.26),\,(3.47)}{=} \frac{\lambda F_1}{F_2} \frac{\lambda F_2}{F_3} \cdots \frac{\lambda F_t}{F_{t+1}} = \frac{\lambda^t F_1}{F_{t+1}} \overset{(3.46)}{=} \lambda^t \cdot \frac{1-\lambda^{m+1}}{1-\lambda^{m+t}}. \tag{3.52}$$

From (3.52) we conclude

$$\beta_t \overset{(3.52),\,\text{L3.6.2}}{\geqslant} \lambda^t \cdot \frac{m+1}{m+t} \overset{m\geqslant 1}{\geqslant} \frac{\lambda^t}{t+1} \text{ and } \frac{\beta_t}{\beta_a} \overset{(3.52)}{=} \lambda^{t-a} \cdot \frac{1-\lambda^{m+a}}{1-\lambda^{m+t}} \leqslant \lambda^{t-a}, \text{ for } a \leqslant t.$$

The above inequalities imply

$$\log \frac{1}{\beta_{n-1}} \leqslant \log \frac{n}{\lambda^{n-1}} \text{ and } \sum_{a<t<b} \log \frac{1}{1-\beta_t/\beta_a} \leqslant \sum_{a<t<b} \log \frac{1}{1-\lambda^{t-a}} \overset{\text{L3.5.9}}{\leqslant} \frac{(\pi \log e)^2}{6 \log \frac{1}{\lambda}}.$$

It remains to combine Theorem 3.5.8 and the above inequalities. □

Discussion. For $t \to \infty$ we have $F_t \to \frac{1}{1-\lambda}$, thus $\alpha_t \to \lambda$, i. e. we expect RFS to perform similar to PS with a fixed smoothing rate $\alpha = \lambda$, when the input is large enough. Corollary 3.6.3 reflects this behavior, the derived code length bound differs from that of Corollary 3.5.10 only by the additive term $|\mathcal{P}| \log n$. Furthermore, the smoothing rate

$$\lambda = e^{-\pi/\sqrt{6(n-1)}}$$

minimizes the r. h. s. of the code length bound we gave in Corollary 3.6.3. In turn, this minimizer coincides with the minimizer of the code length bound from Corollary 3.5.10.

3.7 Overall Relation

RFD, RFS and PS are closely related elementary models. On the one hand, we may view RFS as a variation of RFD which we obtain by slight modifications; on the other hand, PS with a fixed smoothing rate is both, a special case, and an asymptotic approximation of RFS. Let us now discuss the details.

Our starting point is RFD, which we slightly alter to yield RFS. If we modify RFD such that we allow for non-integer letter frequencies, then we may change the letter frequency discount (see the pseudocode in Figure 3.1)

```
f[i] ← $\lfloor \lambda \cdot$ f[i]$\rfloor$;
**if** f[i]=*0* **then** f[i] ← $1$;
```

to simply become `f[i]` $\leftarrow \lambda \cdot$ `f[i]`. Furthermore, if we ignore Assumption 3.4.1, and choose $d = 1$ (the frequency increment is 1) and $T = 0$ (a discount takes place in every step), then we just turned RFD into RFS.

Next, let us discuss the relation of RFS and PS with a fixed smoothing rate. As we learned in Section 3.6, in general, RFS with (frequency) smoothing rate λ is equivalent to PS with *varying* (probability) smoothing rate

$$\alpha_t = \frac{\lambda F_t}{\lambda F_t + 1}.$$

Moreover, by a particular choice of parameters, RFS with smoothing rate λ also is equivalent to PS with *fixed* smoothing rate $\alpha_1 = \alpha_2 = \cdots = \lambda$. That particular parameter choice is as follows: We choose the initial frequencies f_1 s. t.

$$F_1 = \sum_{x \in \mathcal{X}} f_1(x) = \frac{1}{1-\lambda}. \tag{3.53}$$

By this choice we get $F_2 = \lambda \cdot F_1 + 1 = \frac{1}{1-\lambda}$, so $F_1 = F_2 = \cdots = \frac{1}{1-\lambda}$ and

$$\alpha_t \overset{(3.47)}{=} \frac{\lambda F_t}{\lambda F_t + 1} \overset{(3.53)}{=} \frac{\lambda \cdot \frac{1}{1-\lambda}}{\lambda \cdot \frac{1}{1-\lambda} - 1} = \lambda,$$

which means that for the given parameter choice RFS is equivalent to PS with fixed smoothing rate $\alpha_t = \lambda$, as we stated. In addition, as we explained at the end of Section 3.6, this equivalence characterizes the asymptotic behavior of RFS. Namely, if F_1 is a polynomial in λ of type $1+\lambda+\lambda^2+\ldots$, then F_t approaches $\frac{1}{1-\lambda}$, as t increases, and the particular choice of parameters we gave above resembles this situation in the asymptotic case ($t \to \infty$). So the asymptotic behavior of RFS with (frequency) smoothing rate λ matches that of PS with fixed (probability) smoothing rate λ.

3.8 Experiments

We now present the results of an experimental study on the tightness of our code length bounds for RFD, RFS and PS in comparison to PWS and binary input sequences. (We restrict our attention to binary sequences, which allows us to compare results across distinct elementary models, e. g. RFD to RFS. This is necessary since our results on RFS and PS only hold for binary sequences.) To do so, we measure the redundancy of RFD, RFS and PS w. r. t. PWS in a worst-case setting (since all of our bounds are worst-case bounds) and compare the measured redundancy to the maximum redundancy implied by every individual code length bound. We also compare the measured worst-case redundancy of RFD, RFS and PS to each other.

Our experiments evaluate RFD, RFS and PS based on artificial data. We defer experiments on real world data to Chapter 5, where we study the interaction of elementary modeling and mixing within a CTW-based statistical data compressor. This course of action is accounted for by good reason, that is, any major statistical data compression algorithm combines elementary modeling and mixing. Consequently, the actual compression for real-world data depends on the interaction of elementary modeling and mixing, rather than the pure performance of a single elementary model on real-world data. Hence, we omit such experiments.

In the following we sketch the experimental setup in Section 3.8.1 and provide experimental results and an associated evaluation in Section 3.8.2.

Table 3.3: Parameter configuration, redundancy bounds w. r. t. PWS whose partition has s segments in $1{:}n$ (there are $s-1$ switches) for all elementary models we consider in our experiments. Column n_0 indicates the minimum sequence length so that the corresponding redundancy bound holds given the specified parameters. For the terms A, B and C see Lemma 3.4.8.

Model	Parameters	Redundancy Bound	n_0
RFD	$L = \frac{T-N}{d} = \lceil\sqrt{n}\rceil$, $\lambda = \frac{8}{L}$ $d = 1$, $f_1(0) = f_1(1) = 1$	$s \cdot [L \log eL + A \log L + B] + C$ $+\ n \cdot \left[\frac{\lambda \log eL}{1-\lambda} + \frac{A \log L + B}{(1-\lambda)L}\right]$	65
RFS	$\lambda = e^{-\pi/\sqrt{6(n-1)}}$ $p(0) = \frac{1}{2}$, $m = 1$	$s \cdot \left[\frac{2\pi \log e}{\sqrt{6}} \cdot \sqrt{n} + \log 2n\right]$	9
PS1	$p(0) = \frac{1}{2}$, $\alpha = e^{-\pi/\sqrt{6(n-1)}}$	$s \cdot \left[\frac{2\pi \log e}{\sqrt{6}} \cdot \sqrt{n} + 1\right]$	5
PS2	$p(0) = \frac{1}{2}$, $\alpha_t = e^{-\pi/\sqrt{12(t+1)}}$	$s \cdot \left[\frac{2\pi \log e}{\sqrt{3}} \cdot \sqrt{n} + 1\right]$	1

3.8.1 Experimental Setup

The Worst-Case Redundancy. Recall that our comparison should be carried out in a worst case setting. More precisely, for an elementary model MDL (e. g. PS with smoothing rate choice from Corollary 3.5.12) we measure the worst-case redundancy

$$r(n) := \max_{x_{1:n}} \left(\ell(x_{1:n}; \mathsf{MDL}) - \min_{\mathsf{PWS}} \ell(x_{1:n}; \mathsf{PWS})\right), \text{ for } n_0 \leqslant n \leqslant T, \tag{3.54}$$

which brings face to face the worst-case binary input $x_{1:n}$ and the best competing PWS. To obtain a meaningful graphical representation we must only consider PWS that share a single fixed partition (for our final choice of that partition $\mathcal{P}$ see (3.56)). To evaluate the tightness of our code length bounds we compare $r(n)$ to the corresponding bound on redundancy $\ell(x_{1:n}; \mathsf{MDL}) - \ell(x_{1:n}; \mathsf{PWS})$ (that holds for arbitrary $x_{1:n}$ and arbitrary PWS) implied by a code length bound. In Table 3.3 we list all elementary models of consideration along with the implied redundancy bounds. For RFS and PS1 we have chosen smoothing rates s. t. the corresponding redundancy bounds get minimized, similarly, in case of RFD, the parameter choice minimizes the corresponding redundancy bounds in the big-Oh-sense for $|\mathcal{P}| = O(1)$. Notice that we have to impose a lower bound n_0 on the sequence length n in order to satisfy all prerequisites for the redundancy bounds (e. g. in case of PS1 the smoothing rate α has to be larger than $\frac{1}{2}$ in order to satisfy Assumption 3.5.2 (a)).

Unfortunately, a direct computation of (3.54) is intractable since there

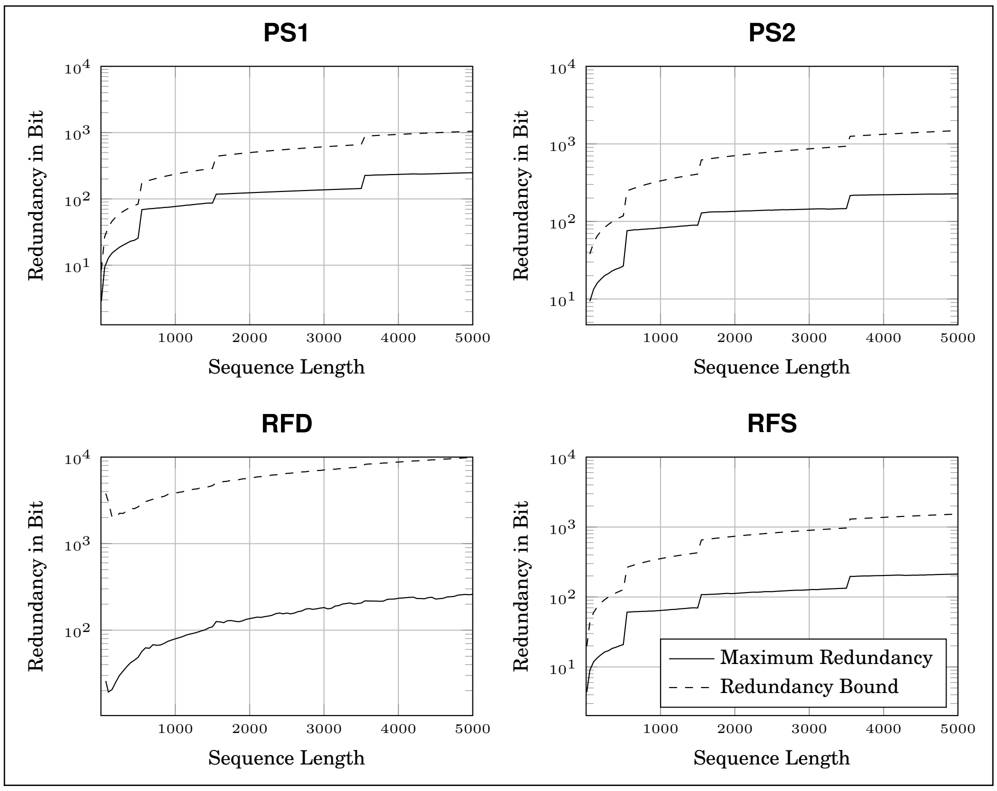


Figure 3.5: Redundancy bounds and approximate worst-case redundancy $r(t)$ for elementary models from Table 3.3.

are exponentially many inputs $x_{1:n}$. To lift this limitation we approximate (3.54). Our general idea is to take the maximum in (3.54) over a large, but finite, set of sequences $x_{1:n}$ that allow to easily approximate the term $\min_{\mathsf{PWS}} \ell(x_{1:n}; \mathsf{PWS})$. In the following we describe this procedure. Therein, we will rely on the notion of generating parameters: We say a partition $\mathcal{P}$ and distributions $p_1, p_2, \dots, p_{|\mathcal{P}|}$ *generate* an input $x_{1:n}$, if for the i-th segment $a{:}b \in \mathcal{P}$ every letter within $x_{a:b}$ is drawn at random according to p_i.

A Single Input. Suppose we are given a partition $\mathcal{P}$, the distributions p_1, $p_2, \dots, p_{|\mathcal{P}|}$ and a generated input $x_{1:n}$. We now explain how we approximate the innermost term in (3.54),

$$\ell(x_{1:n}; \mathsf{MDL}) - \min_{\mathsf{PWS}} \ell(x_{1:n}; \mathsf{PWS}), \tag{3.55}$$

in the current situation. It is obvious that a good approximation to the PWS that minimizes the code length for $x_{1:n}$ will predict the distribution p_i for any letter within the i-th segment of $\mathcal{P}$. If we let PWS' be such a competitor, then, for all $n \in 1{:}T$, we approximate (3.55) via

$$\ell(x_{1:n}; \mathsf{MDL}) - \ell(x_{1:n}; \mathsf{PWS}').$$

All Inputs. It remains to generalize our approximation to the set of all possible inputs. To approximately recover (3.54), we should ideally take the maximum of (3.55) over any input $x_{1:n}$ that may be generated by $\mathcal{P}$, p_1, p_2, …, $p_{|\mathcal{P}|}$, for all distributions $p_1, p_2, \dots, p_{|\mathcal{P}|}$ and $n \in 1{:}T$. As an approximate solution, we take the maximum over a finite subset of inputs samples across the whole input space. We generate the subset of inputs we consider as follows: For every p_1, p_2, …, $p_{|\mathcal{P}|} \in \{p \mid p(0) \in \{\varepsilon, 2\varepsilon, \dots, 1\}\}$ and $n \in \{1, 50, 100, \dots, T\}$ we generate R inputs. Our parameter setup is

$$\begin{aligned}
&\mathcal{P} = \{1{:}500, 501{:}1500, 1501{:}3500, 3501{:}T\},\\
&T = 5000,\\
&R = 50 \quad \text{and}\\
&\varepsilon = 0.1.
\end{aligned} \tag{3.56}$$

By the choice of parameters we take the maximum in (3.54) over $R/\varepsilon^{|\mathcal{P}|} =$ 500,000 inputs for every $n \in \{1, 50, 100, \dots, T\}$.

3.8.2 Evaluation

Evaluation — Tightness of Bounds for RFD, RFS and PS. Let us first consider PS1, PS2 and RFS. For these elementary models our bounds provide a loose upper estimate on the measured worst-case redundancy. In case of PS1 the bound is tighter, compared to PS2 and RFS. This behavior is not surprising, since the code length analysis for PS2 and RFS is based on the code length analysis of PS1, incorporating more simplifications. Across PS1, PS2 and RFS the bounds get more loose, as we step from segment to segment (e. g. from the first segment $1{:}500$ to the second segment $501{:}1500$). The explanation is simple: For the code length analysis w. r. t. PWS we simply concatenated a bound on the worst-case code length w. r. t. the empirical entropy (see Theorem 3.5.6 and Theorem 3.5.8). Albeit, we do not know whether or not the worst-case situation w. r. t. the empirical entropy may occur in two adjacent PWS segments at the same time. Our experiments indicate that this may not be the case.

We turn to RFD. Unfortunately, the redundancy bound for RFD turns out to be pretty loose. The term

$$n \cdot \left[\frac{\lambda \log eL}{1-\lambda} + \frac{A \log L + B}{(1-\lambda)L}\right] = \frac{n \log n}{\sqrt{n} - 8} \cdot O(1) \tag{3.57}$$

dominates the bound for small n and induces two effects. First, it causes the redundancy bound to be “v-shaped”; second, it is the main reason for the looseness we observe. Recall that the term (3.57) accounts for the redundancy due to rescales. Hence, to fight the looseness a good starting point would be to handle the effect of rescales in a more fine grained manner. Overall, our results on RFD are more useful to estimate the asymptotic worst-case behavior of RFD, rather than the worst-case behavior on finite inputs.

Evaluation — Worst-Case Redundancy across RFD, RFS and PS. In contrast to the bounds, the actual worst case redundancy measurements turn out to be more uniform in case of PS1, PS2 and RFS; RFD seems to be an exception.

For PS1, PS2 and RFS the redundancy increases slowly within a segment of $\mathcal{P}$, but increases more drastically across the transition between adjacent segments of $\mathcal{P}$. The jump in redundancy at segment boundaries is especially pronounced when n is large, since the smoothing rate is large and thus it gets harder to age old statistics and to track changing statistics. If we evaluate the measured worst-case redundancy overall, then we obtain the ranking RFS, PS2 and PS1 in order of increasing measured worst-case redundancy. For instance for $n = 5000$, RFS accumulates 213 bits, PS2 wastes 226 bits and PS1 charges 248 bits. In our experiments this ranking remains intact, as long as n is large enough.

As we already stated, the behavior of RFD deviates from that of the former three elementary models. This is not surprising, since PS1, PS2 and RFS share great similarity to each other, whereas RFD is rather different. Most notably, we observe almost no steep increase in measured worst-case redundancy across segment boundaries, but a steady and noisy uniform increase in redundancy, overall. We may explain this effect as follows: RFD conducts discounts in intervals of length roughly equal to $(1 - \lambda) \cdot L \approx \lceil\sqrt{n}\rceil - 8$. Throughout our experiments, the interval length varies from 1 (for $n = 65$) to roughly 63 (for $n = 5{,}000$), so it is much smaller than the length of a segment from $\mathcal{P}$. Hence, RFD continuously tracks the input, no matter if this effect improves (track changing statistics across segment transitions) or degrades (track perturbations in the input within a segment) compression. Consequently, the measured worst-case redundancy should increase at a steady rate.

3.9 Summary

In this chapter we considered one of the fundamental problems every statistical data compressor needs to solve, that is, elementary modeling. Common approaches to elementary modeling may be subdivided into two categories, the “practitioner’s approach” and the “theoretician’s approach”. The latter category typically enjoys powerful code length guarantees, but is out of practical scope, due to running time and space constraints, while the former category meets practical requirements, but lacks a sound theoretical basis.

We took a closer look at some widespread methods that fall into the claim “practitioner’s approaches”, i. e. elementary models RFD, RFS and PS. For these methods we provided a code length analysis. Our results reveal that these elementary models are not only appealing in terms of their low running time and space requirements, but also in terms of code length guarantees w. r. t. arbitrary competing PWS. For PS and RFS our results are valid when compressing bit sequences, whereas our analysis of RFD holds for sequences over a non-binary alphabet as well.

Finally, an experimental study supports the results of our code length analysis. The code length bounds for elementary models RFS and PS provide a loose upper estimate on the worst-case redundancy w. r. t. a PWS and the accuracy declines as the number of PWS-segments increases. Our code length bounds for RFD are more loose and provide an asymptotic estimate on the worst-case redundancy, rather than a tight estimate for relatively short inputs. Nevertheless, we provide a solid theoretical basis for a variety of widespread, practically appealing and relevant elementary models.

CHAPTER 4

Mixing

CHAPTER OVERVIEW

In this we consider how to mix probability distributions. For this purpose we summarize relevant approaches in statistical data compression, Machine Learning and Online Optimization. We propose and solve two divergence optimization problems in the probabilistic view that lead to two methods for the weighted combination of probability distributions, that is, the Linear Mixture Distribution and the Geometric Mixture Distribution. In addition, we provide a code length analysis for the combination of either mixing technique with Online Gradient Descent for weight estimation. In the analysis, a Switching Mixer that may switch back and forth between components of its mixer input over time serves as competitor. It is important to note that Geometric Mixing is a generalization of Logistic Mixing used in PAQ, so we provide a theoretical basis and a generalization of the PAQ-approach to mixing. We end this chapter with an experimental study that supports our code length bounds.

4.1 Introduction

The Setting. In Chapter 3 it was explained that all major statistical data compression algorithms select a subset of elementary models, based on a context, and mix their predictions, to produce a single prediction for coding. Hence, in addition to elementary modeling, mixing multiple predictions is another fundamental building block in statistical data compression. We devote this chapter to mixing, i. e. we study the following problem:

Assume a sequence $x_1x_2\dots x_n$ of letters and a sequence $\boldsymbol{p}_1\boldsymbol{p}_2\dots\boldsymbol{p}_n$ of m-dimensional p-vectors is revealed step-by-step. Estimate a distribution p_t on the t-th letter, given sequences $x_1x_2\dots x_{t-1}$ and $\boldsymbol{p}_1\boldsymbol{p}_2\dots\boldsymbol{p}_t$ known so far.

Notice that during round t a mixer is aware of the past sequence $x_{<t}$ and all

p-vectors up to, and including, the t-th p-vector, that is $\boldsymbol{p}_{1:t}$. In general, the sequences $x_{<t}$ and $\boldsymbol{p}_{<t}$ determine the state of the mixer, e. g. a weight vector. This state finally determines how to mix the distributions given by $\boldsymbol{p}_t$.

Mixing in a statistical data compressor such as CTW or PPM involves the above problem at individual contexts, e. g. the input sequence $x_{1:n}$ actually is a context history $x^c_{1:n}$, for the context c of consideration. However, the general problem statement remains unchanged, so we omit context-conditioning to simplify notation, similarly to Chapter 3. (In Chapter 5 we will study how context-conditioned elementary modeling and mixing interacts in a CTW-like statistical data compressor.) Asides from mixing in such a special setting, it may be beneficial to mix the predictions of *arbitrary* models. PAQ has demonstrated that this approach is very successful. If such models are of opposing nature, then one may add up the advantages of individual models without cumulating the disadvantages [49].

In the remainder of this chapter we study PAQ Logistic Mixing, which is very efficient in practice, and provide a code length analysis and a generalization to a non-binary alphabet. Furthermore, we treat another practical technique, linear mixing, in a similar fashion. Similarly to Chapter 3 our main goal is to provide a theoretical basis for mixing methods that perform well in practice.

Design Constraints. Since in this chapter we concentrate on practical mixing methods, we must sketch common complexity constraints that characterize practical methods. The constraints are similar to that of an elementary model. First, we must be able to implement a mixer with little memory, in practice m memory words should suffice for a mixer that combines m probability distributions. (For reasons of efficiency, using a small constant amount of additional space on top of these m words is still acceptable.) Despite space constraints, practical methods are also limited by running time constraints. The implementation of a mixer must allow to mix m distributions and to update its internal state (e. g. update weights) in time $O(m)$. Just as in elementary modeling, mixing needs to be adaptive, more precisely, adaptive in the following sense: Consider a scenario where up to some point in the input sequence the first component of the mixer input provides good predictions, and afterwards the second component of the mixer input provides good predictions. In this situation a mixer should first produce mixed predictions that are close (in a code length-sense) to the first component and shortly after the aforementioned transition it should produce mixed predictions that are close to the second component.

Mixing based on Mixture Distributions. Given a weight vector $\boldsymbol{w} \in \Delta$ and a p-vector $\boldsymbol{p} = (p_1, \dots, p_m)^\mathsf{T}$ a *mixture distribution* $p_{\boldsymbol{w}}$ maps $\boldsymbol{p}$ to a single distribution $p_{\boldsymbol{w}}(\boldsymbol{p})$. We write $p_{\boldsymbol{w}}(x; \boldsymbol{p})$ to denote the probability of letter x. Typically this mapping can be represented by a simple closed form formula (or an algorithm) that involves some kind of weighted averaging. Given such a formula, we write $\{p_{\boldsymbol{w}}\}_{\boldsymbol{w} \in \Delta}$ to denote a class of mixture distributions.

Most common approaches to mixing rely on mixture distributions in the following way. First, we fix a class of mixture distributions and an initial weight vector $\boldsymbol{w}_1 \in \Delta$. Now consider the t-th step during the course of a mixer on the input sequence $x_{1:n}$ with mixer input $\boldsymbol{p}_{1:n}$. At the beginning of step t the weight estimate $\boldsymbol{w}_t$ determines a mixture distribution $p_{\boldsymbol{w}_t}$ from the class of mixture distributions. By this choice the mixer predicts the distribution $p_{\boldsymbol{w}_t}(\boldsymbol{p}_t)$ on the next letter x_t. After x_t is known, the mixer updates its current weight, which yields $\boldsymbol{w}_{t+1}$ and we may proceed with the next step.

The Switching Mixer. Later in this chapter we will provide a code length analysis for various mixers to add theoretical support for the desirable property of adaptivity. More precisely, we will compare the code length of a mixer to that of a Switching Mixer (SWM), which we now introduce formally:

Definition 4.1.1 *A* Switching Mixer $\mathsf{SW}\langle \mathcal{P}, \{m_s\}_{s \in \mathcal{P}} \rangle$ *for sequences of length n and m-dimensional p-vectors is given by a partition $\mathcal{P}$ of the set $1{:}n$ and a family $\{m_s\}_{s \in \mathcal{P}}$ of integers from $\{1, 2, \dots, m\}$. For a sequence $\boldsymbol{p}_{1:n} = \langle (p_{1,t}, p_{2,t}, \dots, p_{m,t})^\mathsf{T} \rangle_{1 \leqslant t \leqslant n}$ of p-vectors,* SW *induces the prediction* $\mathsf{SW}(x_{<t}, \boldsymbol{p}_{1:t}) = p_{m_s,t}$*, where $s \in \mathcal{P}$ is the unique segment with $t \in s$. A sequence $x_{1:n}$ receives the code length*

$$\ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) = \sum_{s \in \mathcal{P}, t \in s} \ell(x_t; p_{m_s,t}). \tag{4.1}$$

Whenever clear from context, we will omit the sequence length n and the dimension m of the p-vectors.

Let us now sketch the idea behind a SWM. Suppose that we are given $\mathsf{SW}\langle \mathcal{P}, \{m_s\}_{s \in \mathcal{P}} \rangle$, a sequence $x_{1:n}$ along with a sequence $\boldsymbol{p}_{1:n}$ of p-vectors. Now fix a segment $s = a{:}b \in \mathcal{P}$, for this segment a SWM has an associated number $m_s = j$. For the letter x_t from segment $x_{a:b}$ (i. e. $t \in a{:}b$), the SWMs prediction is the j-th component of $\boldsymbol{p}_t = (p_{1,t}, p_{2,t}, \dots, p_{m,t})$, that is $p_{j,t}$. So in this situation, the prediction is $\mathsf{SW}(x_{<t}, \boldsymbol{p}_{1:t}) = p_{j,t}$, the letter x_t receives code length $\ell(x_t; p_{j,t})$ and, consequently, the whole sequence $x_{1:n}$ receives code length (4.1).

In a code length analysis we often consider an arbitrary SWM with a partition of size $|\mathcal{P}|$ as competitor. Hence our results must also hold for the optimal SWM with a partition of size $|\mathcal{P}|$. This competitor is able to adapt to the input in the following sense: It may partition the input according to an optimal partition and use the best component of the mixer input in every input segment. (The choice of the partition and of the mixer input component for every segment minimizes the code length the SWM assigns to the input sequence.) A code length guarantee for some mixer that supports low redundancy w. r. t. such a competitor implies that the mixer should also be able to adapt to the input.

4.2 Previous Work

In the following we discuss approaches that are applicable to mixing in statistical data compression. We may classify these approaches into two categories, based on their origin. On the one hand, there are techniques that directly originate in statistical data compression, i. e. the Switching Algorithm and its descendants, Beta-Weighting[1] and Logistic Mixing. On the other hand, there are problem settings in Machine Learning that are closely related to mixing in statistical data compression, namely Prediction with Expert Advice and Online Convex Programming (OCP). Our survey of previous work intentionally concentrates on the methods that are directly related to statistical data compression, since these methods have proven to work well in practice and one of these methods — Logistic Mixing — will be of central importance in the remainder of this chapter. Nevertheless, we also provide a bit more thorough discussion of OCP, since we will later rely on techniques from OCP. In our survey Prediction with Expert Advice plays a minor role. A complete discussion of the huge number of algorithms that apply to a more general setting is beyond the scope of this work.

The Switching Algorithm and its Descendants. The problem of combining arbitrary models in statistical data compression was first investigated in [94, 95, 96].[2] However, the setup was rather restrictive, since the main

[1]In the domain of statistical data compression Beta-Weighting was introduced as an implementation technique for CTW, there also is a relation to Machine Learning, as we will explain later.

[2]The *Switching Distribution* [91] from the Machine Learning community matches the approaches of the Switching Algorithm while allowing for the combination of more than two models in $O(m)$ time per step. These improvements over the Switching algorithm are due to a clever choice of w_{SW} in (4.2). Unfortunately this hint arrived after the completion of this work, hence we do not include the algorithm.

intention was to combine just *two* models. This approach turns out to be successful, in particular, when the two models are of opposing nature, e. g. CTW and a LZ77-alike model [95] or CTW and PPM* [96].

In [95] the authors proposed three algorithms, namely, the *Switching Algorithm*, the *Snake Algorithm* and the *Reduced Complexity Algorithm*. To explain the algorithms consider the t-th step while we process an input sequence $x_{1:n}$: We are given a sequence $\boldsymbol{p}_{1:t}$ of 2-dimensional p-vectors (since we combine two models) and must produce a single prediction, which may be used for coding. In the t-th step the Switching Algorithm defines a distribution over all 2^t SWMs (see Definition 4.1.1) and computes the block probability

$$P_w(x_1x_2\dots x_{t-1}x) = \sum_{\mathsf{SW}} w_{\mathsf{SW}} \cdot \mathsf{SW}(x; x_{<t}, \boldsymbol{p}_{1:t}) \cdot \prod_{1\leqslant i<t} \mathsf{SW}(x_i; x_{<i}, \boldsymbol{p}_{1:i}) \tag{4.2}$$

of the sequence $x_1x_2\dots x_{t-1}x$ accordingly. (The number of SWMs to weight over grows with t.) Finally, the Switching Algorithm predicts the distribution p s. t. $p(x) = P_w(x_1x_2\dots x_{t-1}x)/P_w(x_{<t})$. The Switching Algorithm relies on an efficient computation of (4.2). In the t-th step this process requires $O(t)$ space to store intermediate information (from previous steps) and time $O(t)$ to compute a summation over $O(t)$ terms. So to process a sequence of length n we require time $O(n^2)$ and space $O(n)$. For an input sequence of length n the Switching Algorithm has redundancy $O(|\mathcal{P}| \cdot \log n)$ w. r. t. an arbitrary competing SWM with partition $\mathcal{P}$.

Since the complexity requirements are beyond practical scope, the time and space payloads have been cut down by the introduction of the Snake Algorithm and the Reduced Complexity Algorithm. The Snake Algorithm is a coarse approximation of the Switching Algorithm using only $O(n)$ time and $O(1)$ space. The Reduced Complexity Algorithm runs in time $O(s\cdot n)$ and requires $O(s)$ space, where s is a parameter. By the choice of s we may turn the Reduced Complexity Algorithm into the Switching Algorithm or into the Snake Algorithm or we may even interpolate between the two. There are no known code length guarantees for the Snake Algorithm or the Reduced Complexity Algorithm.

Beta-Weighting. As seen in Section 2.4.3, Beta-Weighting originates from CTW [105, 78], where it is used to recursively combine the prediction of two models along a path in the context tree of CTW.

As proposed in [49], one may generalize Beta-Weighting (out of the scope of CTW) to mix more than just two distributions. The generalized form of Beta-Weighting is specified by

$$\mathsf{BETA}(x; x_{<t}, \boldsymbol{p}_{1:t}) := w_{1,t} \cdot p_{1,t}(x) + \cdots + w_{m,t} \cdot p_{m,t}(x), \text{ for } \boldsymbol{p}_t := (p_{1,t}, \dots, p_{m,t}),$$

$$w_{i,t+1} = w_{i,t} \cdot \frac{p_{i,t}(x_t)}{\mathsf{BETA}(x_t; x_{<t}, \boldsymbol{p}_{1:t})}, \text{ if } t > 0, \text{ and } w_{i,1} := \frac{1}{m}. \tag{4.3}$$

(Recall that we already discussed Beta-Weighting, see (2.4) for a more verbose description.) To process a sequence $x_{1:n}$, given a mixer input $\boldsymbol{p}_{1:n}$ of m-dimensional p-vectors, Beta-Weighting requires time $O(m \cdot n)$ and space $O(m)$. Furthermore, Beta-Weighting satisfies

$$\prod_{1 \leqslant t \leqslant n} \mathsf{BETA}(x_t; \boldsymbol{p}_{1:t}, x_{<t}) = \sum_{1 \leqslant i \leqslant m} \frac{1}{m} \prod_{1 \leqslant t \leqslant n} p_{i,t}(x_t), \text{ where } \boldsymbol{p}_t = (p_{1,t}, \dots, p_{m,t})^{\mathsf{T}} \tag{4.4}$$

(this may be seen by induction on n) and

$$\ell(x_{1:n}; \mathsf{BETA}, \boldsymbol{p}_{1:n}) \leqslant \ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \log m, \tag{4.5}$$

for all SWMs with partition $\{1{:}n\}$. Let us discuss (4.4), which is an important property of Beta-Weighting. The l. h. s. of (4.4) is the block probability Beta-Weighting assigns to the whole sequence $x_{1:n}$. Each term $\prod_{1 \leqslant t \leqslant n} p_{i,t}(x_t)$ on the r. h. s. is the block probability that the i-th component of the mixer input assigns to $x_{1:n}$. So the block probability assignment of Beta-Weighting matches the block probability given by an equally weighted average of the block probabilities corresponding to either mixer input component. By bounding (4.4) from below and by taking the logarithm it is easy to obtain (4.5). To better understand the code length guarantee (4.5) consider the prediction of such a SWM SW: Given the p-vector $\boldsymbol{p}_t = (p_{1,t}, p_{2,t} \dots, p_{m,t})^{\mathsf{T}}$ in the t-th step it predicts $\mathsf{SW}(x_{<t}, \boldsymbol{p}_{1:t}) = p_{i,t}$ for some i that is fixed and independent of t. So, in every step SW just forwards the i-th component of its mixer input. On the one hand, if we have a set of m models and mix their predictions using Beta-Weighting, then the coding cost of some sequence w. r. t. the mixed prediction exceeds the coding cost of the sequence w. r. t. the best model, out of the m models, only by $\log m$ bits, regardless of the sequence length n. On the other hand, by (4.5) Beta-Weighting is not guaranteed to have low redundancy w. r. t. a SWM with an arbitrary partition $\mathcal{P}$. (Recall from the discussion in Section 4.1 that such a competitor may adapt to changes in the input.) So there is no theoretical result that supports Beta-Weighting to have low redundancy w. r. t. such an adaptive competitor, although there exists empirical evidence [49].

Logistic Mixing. At this point let us just revise Logistic Mixing (LM) in brief. For more information we refer the reader to the description of PAQ in Section 2.4.4. There is not much literature available on LM; major references are [53, 80].

LM combines an arbitrary number of distributions on a binary alphabet using a single layer neural network and tunes the network weights via Online Gradient Descent: If we let $\boldsymbol{w}_t = (w_{1,t}, \dots, w_{m,t})$ be the weight vector at the beginning of step t and let $\boldsymbol{p}_{1:t}$ be the mixer input up to the t-th step, and consider a sequence $\alpha_{1:\infty}$ of real numbers from $(0,1)$ (step size sequence for Online Gradient Descent), then PAQ mixing is defined by

$$\mathsf{LM}(x; x_{<t}, \boldsymbol{p}_{1:t}) := \mathrm{sq}\Bigg(\textstyle\sum_{1\leqslant i\leqslant m} w_{i,t} \cdot \mathrm{st}(p_{i,t}(x)) \Bigg), \text{ where } \boldsymbol{p}_t = (p_{1,t}, \dots, p_{m,t}),$$
$$w_{i,t+1} = w_{i,t} + \alpha_t \cdot (1 - \mathsf{LM}(x_t; x_{<t}, \boldsymbol{p}_{1:t})) \cdot \mathrm{st}(p_{i,t}(x)),\ t > 0,\ \boldsymbol{w}_1 \in \mathbb{R}^m.$$

(See equations (2.32) and (2.31) in Chapter 2 for the functions st and sq and Section 2.4.4 some more illustrations.) For the sake of implementation efficiency PAQ utilizes a fixed step size sequence, $\alpha_1 = \alpha_2 = \dots$, similar to PS.

To process a sequence of length n, given a corresponding sequence of p-vectors with dimension m, LM takes time $O(m \cdot n)$ and requires m real-valued weights (the weights do not need to form a point within the unit simplex). LM shows great performance in practice, but no code length guarantees are known.

Prediction with Expert Advice. At this point let us first discuss the general concept of Prediction with Expert Advice and afterwards explain how we may cast mixing in statistical data compression into this framework. There exists a huge amount of algorithms and relevant literature and it is beyond the scope of this work to provide a thorough survey. For a comprehensive overview we suggest [18], which we are guided by in this section.

The problem of Prediction with Expert Advice may be stated as a round-wise process. In every round we must provide a *decision*, based on the *advice* of m experts. An algorithm for this problem combines the experts' advice into a single decision (the space of possible advice and decisions is identical). After this compound decision is made, an opponent reveals his *response* and a (initially fixed) *loss function* maps the compound decision and the response to a loss. Finally, the expert algorithm updates its internal state (e. g. we adjust weights associated to individual experts) to take the loss it just suffered into account.

Ultimately, we want to obtain expert algorithms whose cumulative loss is close to the loss of the best expert in hindsight. The additional loss an expert algorithm suffers over the best expert is called *regret*. More wider notations of regret are *shifting regret*, where we compare the expert algorithm to the best sequence of experts, or *adaptive regret*, where the regret of the expert algorithm w. r. t. the best expert during an arbitrary period $a{:}b$ is of interest. Note that shifting regret induces competitors that may adapt to the input.[3] This property translates to competitors induced by adaptive regret as well, since guarantees on the adaptive regret imply guarantees on the shifting regret. Depending on the exact setting (especially depending on properties of the loss function) there exist expert algorithms that run in time $O(m \cdot n)$, require space $O(m)$ and typically attain regret guarantees ranging from $O(n^{2/3})$ over $O(\log n)$ to $O(1)$ (the constants in the big-Oh notation typically depend on the number of experts and on the loss function) for n rounds of the expert prediction problem.

Given the above outline of Prediction with Expert Advice, it is straightforward to translate mixing from statistical data compression to this framework: In the t-th round we must provide a probability distribution p (decision), given the distributions predicted by every model (expert advice). The opponent responds the t-th letter x_t of the input sequence and the coding cost $-\log p(x_t)$ of the t-th letter acts as the loss function, thus regret directly corresponds to redundancy. (So a sequence $\boldsymbol{p}_{1:n}$ of p-vectors stores the expert advice for rounds $1, 2, \dots$ and the opponent is given by the input sequence $x_{1:n}$.) It is interesting to note that in the data compression setting Beta-Weighting is equivalent to several expert algorithms such as the Exponentially Weighted Average Forecaster, the Greedy Forecaster or the Aggregation Forecaster, see [18, Section 9.2] for details.

Online Convex Programming. We follow the lines of the previous section, that is, we first introduce the general approach of Online Convex Programming (OCP) to the reader and afterwards discuss how to link mixing in statistical data compression and OCP. We cover OCP a bit more thoroughly than Prediction with Expert Advice, since in Section 4.5 we will adopt and modify OCP analysis techniques.

The goal in OCP (introduced in [108]) is to solve a very general optimization task: First, we fix a feasible set $\mathcal{S} \subseteq \mathbb{R}^m$, which is supposed to be compact and convex. Then, a sequence $c_1, c_2, \dots : \mathcal{S} \to \mathbb{R}$ of n convex cost functions is revealed to us step by step. In the t-th step we know about $c_{<t}$ and we must choose a point $\boldsymbol{w}_t \in \mathcal{S}$ before the t-th cost function c_t becomes known and we have to pay the cost $c_t(\boldsymbol{w}_t)$. A strategy $\boldsymbol{w}_{1:n}$ (e. g. computed by an algorithm) incurs cost $C(\boldsymbol{w}_{1:n}) = \sum_{1 \leqslant t \leqslant n} c_t(\boldsymbol{w}_t)$, which should ideally be as small as possible. As a theoretic measure of quality algorithms for OCP should perform almost as well as an idealized competing scheme,

[3]Similar to SWMs, the competitor "best sequence of experts" partitions the input into segments and predicts using a fixed expert per segment. Both, the partitioning and the expert per segment may be chosen to minimize the cumulative loss of the competitor.

e. g. the best constant strategy with cost $C^* = \min_{\boldsymbol{w}\in\mathcal{S}} \sum_{1\leqslant t\leqslant n} c_t(\boldsymbol{w})$. Similarly to Prediction with Expert Advice, the cost $C(\boldsymbol{w}_{1:n}) - C^*$ a strategy $\boldsymbol{w}_{1:n}$ charges above a competitor (or a class of competitors) is called regret. There exist the extensions shifting regret and adaptive regret [38]. All guarantees below consider the standard notation of regret (w. r. t. the best constant strategy), given above.

It might seem somewhat counterintuitive that there is no further assumption on the sequence of cost functions. If these are arbitrary (or chosen by an adversary) it seems impossible to determine a strategy $\boldsymbol{w}_{1:n}$ that has low cost $C(\boldsymbol{w}_{1:n})$. In such a setting no fixed competitor $\boldsymbol{w}$ will perform well, so C^* will also be large. (Hence, the competitor suffers from the same deficit.) However, the idea in OCP is to minimize the regret $C(\boldsymbol{w}_{1:n}) - C^*$ (and in the setting we have just sketched the regret might still be small). The more powerful the notation of regret is, the smaller C^* (and $C(\boldsymbol{w}_{1:n})$, if the strategy $\boldsymbol{w}_{1:n}$ is "good") will be.

Algorithms for OCP are typically based on standard offline-optimization techniques, the most prominent techniques are Online Gradient Descent (OGD) [108] and Online Newton Step (ONS) [37]. (See also [36] for more details and [38] for extensions.) If the euclidean norm of the gradients $|\nabla c_1(\boldsymbol{w})|$, $|\nabla c_2(\boldsymbol{w})|, \dots$ is bounded, then OGD has regret $O(\sqrt{n})$. The regret may further be improved to $O(\log n)$, if all cost functions are strongly convex. In the case of exp-concave cost functions, ONS also has regret $O(\log n)$. OGD requires time $O(T_{\text{proj}}^{\text{OGD}} \cdot n)$ and space $O(m)$, whereas ONS requires time $O(T_{\text{proj}}^{\text{ONS}} \cdot n)$ and space $O(m^2)$. The terms $T_{\text{proj}}^{\text{OGD}}$ and $T_{\text{proj}}^{\text{ONS}}$ account for the time required to project a point $\boldsymbol{w} \in \mathbb{R}^m$ to the point (in some sense) closest to $\boldsymbol{w}$ within the feasible set $\mathcal{S}$. These running times are usually independent of n, but dependent of $\mathcal{S}$ and unfortunately these may be rather large, e. g. ONS needs to solve a quadratic program in every step for that purpose.

Mixing in statistical data compression almost directly translates into the OCP framework. For simplicity let us consider an example based on a Linear Mixture Distribution (see Section 4.4.3 for a discussion of such mixture distributions). A Linear Mixture Distribution translates a p-vector $\boldsymbol{p} = (p_1, \dots, p_m)^\mathsf{T}$, given a weight vector $\boldsymbol{w}$ drawn from the unit simplex, into a prediction p s. t. $p(x) = w_1 \cdot p_1(x) + \cdots + w_m \cdot p_m(x)$, for all letters x. This scheme induces (coding) cost $c(\boldsymbol{w}) = -\log(w_1 \cdot p_1(x) + \cdots + w_m \cdot p_m(x))$, for a letter x and a p-vector $\boldsymbol{p}$. Similarly, when a mixer processes a sequence, the t-th step induces the cost function $c_t(\boldsymbol{w})$ that corresponds to the letter x_t and the p-vector $\boldsymbol{p}_t$. Obviously, it is straightforward to adopt an OCP algorithm that determines a sequence of weights for mixing. In the end all that needs to be satisfied to cast mixing into the OCP framework is that the weight space is compact and convex (e. g. weights are drawn from the unit simplex) and that $c_t(\boldsymbol{w})$ is convex and possibly differentiable, if we weight estimation relies on gradients. (As we can see a Linear Mixture Distribution obviously satisfies these criteria.)

If we apply OCP to estimate the weights of a mixer (recall the example above), then guarantees on regret are equivalent to guarantees on redundancy. For instance, regret $O(\sqrt{n})$ for linear mixing cast to OCP implies redundancy $O(\sqrt{n})$ w. r. t. a SWM with partition $\{1{:}n\}$. Later, in Section 4.5, we will use an OCP scheme to estimate weights for a class of mixers and generalize redundancy (regret) guarantees to arbitrary SWMs.

4.3 Our Contribution

Let us step back to the mixing methods directly related to statistical data compression in the survey we gave in Section 4.2. The Switching Algorithm enjoys strong code length guarantees that imply low redundancy w. r. t. arbitrary SWMs, but it is out of practical scope due to its running time and space requirements; the Snake Algorithm and the Reduced Complexity Algorithm do not offer code length guarantees, but cut down time and space complexity. Furthermore, these algorithms are rather restrictive, since we may only mix two distributions. Next, Beta-Weighting may be implemented efficiently, but only satisfies rather weak code length guarantees, since the partition of the competing SWM only consists of a single segment. Finally, LM is of high practical relevance, it works well in practice and may be implemented efficiently, but lacks a theoretical basis. Again, similarly to elementary modeling, a gap between theory and practice stands out.

In the remainder of this chapter we seek to brdige this gap. Most of the results from this chapter have previously been published in [58, 59]. In this chapter we present a greatly polished version thereof. We proceed as follows:

Section 4.4. We develop a framework that attacks the problem of mixing in the probabilistic view. Based on slightly different ideas, this framework induces two divergence minimization problems. We propose, discuss and solve the first problem in Section 4.4.2, the solution to this problem yields the Geometric Mixture Distribution for the weighted combination of probability distributions. Moreover, this mixture distribution generalizes PAQ's LM to a non-binary alphabet, so we introduce an underlying theoretical basis to LM. Next, in Section 4.4.3 we proceed similarly with the second divergence minimization problem. This leads to the Linear Mixture Distribution.

Section 4.5. All of the aforementioned mixture distribution rely on a given set of weights. In general these weights are unknown. To overcome this problem, we couple OGD and a given mixture distribution to obtain an Online Gradient Descent Mixer (OGDM). We prove that under mild assumptions on the mixture distribution such a scheme enjoys good code length guarantees w. r. t. a SWM.

Section 4.6. We show that the results of Section 4.5 apply to the Linear Mixture Distribution (Section 4.6.1) and to the Geometric Mixture Distribution (Section 4.6.2), which yields the Linear- and Geometric OGDM.

Section 4.7. Finally, we underpin our findings with experiments.

4.4 A Probabilistic View on Mixing

4.4.1 Source and Models

The Setting. In Section 4.4 we assume that a sequence $x_{1:n}$ we attempt to compress is generated by a probabilistic mechanism, a so-called *source*. We may view the generation mechanism as a stepwise process, where in step t the source draws the t-th letter x_t at random according to its *source distribution*, which may vary from step to step. (An alternate view is to assume that the source draws the whole sequence $x_{1:n}$ at random according to its source distribution on sequences of length n.) To encode the current letter in step t we are given m distributions, the *model distributions*, wrapped into a p-vector $\boldsymbol{p} = (p_1, p_2, \dots, p_m)^\mathsf{T}$, and a vector $\boldsymbol{w}$ of non-negative weights that sum to one. We may view the model distributions as the predictions of m models, where a weight w_i quantifies how well the model (distribution) i fits the unknown source distribution. At this point we assume that the weights are based on prior knowledge or have been learned from previous observations. (In Section 4.5 we will see how to actually determine the weights.) Based on the present information, that is $\boldsymbol{p}$ and $\boldsymbol{w}$, our task is to determine a single distribution that will be a good choice for encoding the upcoming letter x_t. Below we discuss two approaches to the present problem.

Encoding a Source Letter. Now suppose that we are given a model distribution p. The source emits the next letter x according to its source distribution q and we obtain expected code length

$$\mathrm{E}[\ell(x; p)] = \sum_{x \in \mathcal{X}} q(x) \log \frac{1}{p(x)} = H(q) + D(q \parallel p) \tag{4.6}$$

for the next letter x. By (4.6) we may split the expected code length into two parts, that is, the *source entropy* $H(q)$ and the KL-Divergence $D(q \parallel p)$ (or expected redundancy) of the model distribution p w. r. t. the source distribution q. A good strategy to choose the model distribution p will be to minimize the expected code length (4.6). Since the term $H(q)$ is fixed by the source, the distribution p should ideally minimize $D(q \parallel p)$. We have $D(q \parallel p) \geqslant 0$, which is zero if and only if $p = q$, i. e. the best model distribution is the source distribution itself.

4.4.2 Geometric Mixture Distribution

The Distribution. As the basis for the upcoming considerations we define:

Definition 4.4.1 *For a p-vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_m)^\mathsf{T}$ and a weight vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_m)^\mathsf{T} \in \Delta$, the* Geometric Mixture Distribution $p_{\boldsymbol{w}}(\boldsymbol{p})$ *is given by*

$$p_{\boldsymbol{w}}(x; \boldsymbol{p}) = \frac{p_1(x)^{w_1} \cdot \ldots \cdot p_m(x)^{w_m}}{\sum_{y \in \mathcal{X}} p_1(y)^{w_1} \cdot \ldots \cdot p_m(y)^{w_m}}. \tag{4.7}$$

The Geometric Mixture Distribution assigns probabilities by a weighted geometric average, where the denominator acts as a normalization term, so that probabilities sum up to one. In the following we consider the idea behind this type of mixture distribution.

Divergence Minimization. As we explain now, (4.7) is the solution to a divergence minimization problem. Recall that *the best model distribution is the source distribution itself*. Unfortunately, the source distribution is unknown, so we seek for an approximate source distribution, which may be used as model distribution. Now if p_1 is a good model distribution (w_1 is high), then the source distribution will be similar to p_1 and hence p_1 will have low expected redundancy w. r. t. q, so $D(q \parallel p_1) \approx 0$. In this situation p_1 will be a good approximation of q. However, choosing the model distribution with the highest weight would be overzealous (e. g. because weights may be erroneous), moreover several distinct model distributions may have identical weights, so we would have no hint which model distribution to choose. To overcome this problem we pursue the following idea:

> *We use an approximate source distribution as model distribution. The approximate source distribution should be "close to" good model distributions and "far away" from bad model distributions.*

The weight associated to each model distribution captures whether a model is good or bad, and we measure proximity between source and model distributions via the expected redundancy. We may translate these ideas into the following divergence optimization problem,

$$\min_q w_1 D(q \parallel p_1) + w_2 D(q \parallel p_2) + \cdots + w_m D(q \parallel p_m). \tag{4.8}$$

The minimizer of this problem will be our approximate source distribution. We next observe that we already know the solution of (4.8).

Theorem 4.4.2 *The Geometric Mixture Distribution is the minimizer of (4.8).*

Proof. Let $\boldsymbol{p} = (p_1, p_2, \dots, p_m)^\mathsf{T}$. The idea for the proof is to show that

$$\min_q D(q \parallel p_{\boldsymbol{w}}(\boldsymbol{p})) = \log c + \min_q \sum_{1 \leqslant i \leqslant m} w_i D(q \parallel p_i), \text{ for } c = \sum_{y \in \mathcal{X}} \prod_{1 \leqslant i \leqslant m} p_i(y)^{w_i},$$

since both optimization problems have the same minimizer and it is easy to see that $q = p_{\boldsymbol{w}}(\boldsymbol{p})$ is the minimizer of the l. h. s.. Simple arithmetics yield

$$\begin{aligned} D(q \parallel p_{\boldsymbol{w}}(\boldsymbol{p})) &= \sum_{x \in \mathcal{X}} q(x) \cdot \left(\log \frac{c}{\prod_{x \in \mathcal{X}} p_i(x)^{w_i}} - \log \frac{1}{q(x)} \right) \\ &= \log c + \sum_{x \in \mathcal{X}} q(x) \sum_{1 \leqslant i \leqslant m} w_i \cdot \left(\log \frac{1}{p_i(x)} - \log \frac{1}{q(x)} \right) \\ &= \log c + \sum_{1 \leqslant i \leqslant m} w_i \sum_{x \in \mathcal{X}} q(x) \cdot \left(\log \frac{1}{p_i(x)} - \log \frac{1}{q(x)} \right) \\ &= \log c + \sum_{1 \leqslant i \leqslant m} w_i D(q \parallel p_i). \end{aligned}$$ □

In the following we discuss special properties of the Geometric Mixture Distribution.

Over-Attracting Probability Mass. The Geometric Mixture Distribution has a distinguishing property, namely in case of a non-binary alphabet it may relocate probability mass s. t. the mixed probability of a letter x exceeds the probability assigned by any model distribution. More precisely,

$$p_{\boldsymbol{w}}(x; \boldsymbol{p}) > \max_{1 \leqslant i \leqslant m} p_i(x), \text{ where } \boldsymbol{p} = (p_1, p_2, \dots, p_m)^\mathsf{T}.$$

If this situation occurs, we say that the letter x over-attracts probability mass. Interestingly, the converse can not happen; the mixed probability of any letter x is never smaller than the smallest probability assigned by either model distribution.

$$p_{\boldsymbol{w}}(x;\boldsymbol{p}) = \frac{\prod_{1\leqslant i\leqslant m} p_i(x)^{w_i}}{\sum_{y\in\mathcal{X}} \prod_{1\leqslant i\leqslant m} p_i(y)^{w_i}} \geqslant \prod_{1\leqslant i\leqslant m} p_i(x)^{w_i} \geqslant \min_{1\leqslant i\leqslant m} p_i(x).$$

(We have $\prod_{1\leqslant i\leqslant m} p_i(y)^{w_i} \leqslant \sum_{1\leqslant i\leqslant m} w_i p_i(y)$, by the Arithmetic-Geometric-Mean Inequality.) In Example 4.4.3 we describe a situation in which over-attraction takes place.

Example 4.4.3 *For an alphabet of cardinality $N > 2$, we consider weights $\boldsymbol{w} = (1/2, 1/2)^{\mathsf{T}}$ and two model distributions s. t. for $0 < \varepsilon, q < 1$ we have*

$$p_1(x) = \begin{cases} q, & \text{if } x = 0,\\ (1-q)(1-\varepsilon), & \text{if } x = 1,\\ \frac{(1-q)\cdot\varepsilon}{N-2}, & \text{otherwise,}\end{cases} \quad p_2(x) = \begin{cases} q, & \text{if } x = 0,\\ (1-q)(1-\varepsilon), & \text{if } x = 2,\\ \frac{(1-q)\cdot\varepsilon}{N-2}, & \text{otherwise.}\end{cases}$$

The mixture probability $p_{\boldsymbol{w}}(0)$ of letter 0 is

$$\frac{p_1(0)^{1/2}\cdot p_2(0)^{1/2}}{\sum_{y\in\mathcal{X}} p_1(y)^{1/2}\cdot p_2(y)^{1/2}} = q\Big/\left[q + (1-q)\underbrace{\left(2\sqrt{\frac{\varepsilon(1-\varepsilon)}{N-2}} + \frac{N-3}{N-2}\varepsilon\right)}_{=:f(\varepsilon,N)}\right].$$

We now show that for any q there is an ε s. t. $p_{\boldsymbol{w}}(0) > q$. Clearly, $p_{\boldsymbol{w}}(0) > q$ is implied by $f(\varepsilon, N) < 1$. To observe this we bound $f(\varepsilon, N)$ from above and give a possible choice for ε.

$$f(\varepsilon, N) \leqslant 2\sqrt{\frac{\varepsilon}{N-2}} + (N-3)\sqrt{\frac{\varepsilon}{N-2}} = \frac{N-1}{\sqrt{N-2}}\cdot\sqrt{\varepsilon}$$

If we choose $0 < \varepsilon < (N-2)/(N-1)^2$ it follows that $f(\varepsilon, N) < 1$ and $p_{\boldsymbol{w}}(0) > q$.

Logistic Mixing. In addition to the technical properties mentioned above a central observation is that in case of a binary alphabet, PAQ's LM (cf. (2.33)) is equivalent to (4.7). More precisely, for $\boldsymbol{p} = (p_1, p_2, \ldots, p_m)^{\mathsf{T}}$ we get

$$\begin{aligned} p_{\boldsymbol{w}}(x;\boldsymbol{p}) &= \frac{\prod_{1\leqslant i\leqslant m} p_i(x)^{w_i}}{\prod_{1\leqslant i\leqslant m} p_i(x)^{w_i} + \prod_{1\leqslant i\leqslant m} (1-p_i(x))^{w_i}} \\ &= \left[1 + \prod_{1\leqslant i\leqslant m} \left(\frac{p_i(x)}{1-p_i(x)}\right)^{-w_i}\right]^{-1} \end{aligned}$$

$$= \left[1 + \exp\left(-\textstyle\sum_{1\leqslant i\leqslant m} w_i \cdot \ln \frac{p_i(x)}{1-p_i(x)}\right)\right]^{-1}$$

$$= \operatorname{sq}\left(w_1 \cdot \operatorname{st}(p_1(x)) + \cdots + w_m \cdot \operatorname{st}(p_m(x))\right).$$

(For functions st and sq see (2.32) and (2.31).) In other words: LM is the closed-form solution to the optimization problem (4.8) for the special case of a binary alphabet. This insight adds a theoretical basis to LM and, furthermore, allows us to recognize (4.7) as the generalization of LM to a non-binary alphabet.

Other Links to Literature and Applications. Several researchers previously used a Geometric Mixture Distribution to combine multiple model distributions in various different contexts. In statistics and Machine Learning, Logarithmic Opinion Pooling[4] refers to a Geometric Mixture Distribution [35]. Logarithmic Opinion Pooling has successfully been applied in hyperspectral image classification [46], gene identification [72] and object location in robotics [45], just to name a few examples. The Product of Experts algorithm [40] couples a Geometric Mixture Distribution with a fixed weight vector $\boldsymbol{w} = (1, 1, \ldots, 1)^\mathsf{T}$ (for weights that do not lie in the unit simplex the Geometric Mixture Distribution still remains a valid probability distribution) and a procedure to train the parameters of the model distribution by a randomized training process named contrastive divergence minimization [41]. Applications include the recognition of handwritten digits [62] and face recognition [87].

The Deterministic View. Above we established the minimization problem (4.8) in the probabilistic view. Surprisingly, it turns out that there exists another minimization problem in the deterministic view that shares the same minimizer, as we will explain now. These optimization problems have a nice symmetry. To observe this, consider the weighted difference in coding cost of a distribution q w. r. t. each model distribution for some letter x,

$$w_1(\ell(x; q) - \ell(x; p_1)) + \cdots + w_m(\ell(x; q) - \ell(x; p_m)). \tag{4.9}$$

In the probabilistic view we take the expectation of (4.9) w. r. t. the probability space induced by the source distribution q to obtain the cost function to minimize, see (4.8). (We have $\mathrm{E}[\ell(x; q) - \ell(x; p)] = D(q \parallel p)$, for distributions p and q.) If we do not take the expectation of (4.9), but the maximum

[4]Opinion Pooling, in general, refers to the problem of combining several model distributions.

over all letters x to obtain a cost function for minimization, then we get the optimization problem

$$\min_q \left(\max_{x\in\mathcal{X}} w_1(\ell(x;q)-\ell(x;p_1)) + \cdots + w_m(\ell(x;q)-\ell(x;p_m)) \right), \qquad (4.10)$$

belonging to the deterministic view, since the cost function does not involve any kind of randomization. So, in summary, to measure the proximity of the distribution q w. r. t. the set of m model distributions (which we attempt to minimize) there are two ways: First, we may use the *expected weighted redundancy* in the probabilistic view, leading to the cost function (4.8); second, we may use the *maximal weighted redundancy* in the deterministic view, resulting in the cost function (4.10).

We now argue that the Geometric Mixture Distribution also minimizes (4.10). By adding the constant $c = \log \sum_{y\in\mathcal{X}} \prod_{1\leqslant i\leqslant m} p_i(y)^{w_i}$, which is independent of x and q, to (4.10) and by rearranging we obtain the minimization problem

$$\min_q \Bigg(\underbrace{\max_{x\in\mathcal{X}} \log\frac{1}{q(x)} - \log\frac{1}{p_{\boldsymbol{w}}(x)}}_{=:\, f(q)} \Bigg), \qquad (4.11)$$

where $p_{\boldsymbol{w}}$ is the Geometric Mixture Distribution. Both problems, (4.10) and (4.11), have the same minimizer. Since q and $p_{\boldsymbol{w}}$ are probability distributions, there must exist a letter y s. t. $q(y) \leqslant p_{\boldsymbol{w}}(y)$, so we have $f(q) \geqslant 0$. Since $f(p_{\boldsymbol{w}}) = 0$ the Geometric Mixture Distribution must be a minimizer of (4.11).

4.4.3 Linear Mixture Distribution

The Distribution. In this section we treat linear averaging of probabilities.

Definition 4.4.4 *For a p-vector* $\boldsymbol{p} = (p_1, p_2, \ldots, p_m)^\mathsf{T}$ *and a weight vector* $\boldsymbol{w} = (w_1, w_2, \ldots, w_m)^\mathsf{T} \in \Delta$, *the* Linear Mixture Distribution $p_{\boldsymbol{w}}(\boldsymbol{p})$ *is given by*

$$p_{\boldsymbol{w}}(x;\boldsymbol{p}) = w_1 p_1(x) + w_2 p_2(x) + \cdots + w_m p_m(x). \qquad (4.12)$$

Divergence Minimization. By a reasoning slightly different to that of Section 4.4.2 we can obtain a Linear Mixture Distribution as the solution of another divergence minimization problem. The idea is as follows: If the weight

w_1 of model distribution p_1 is high, then model distribution p_1 will be a good approximation to the unknown source distribution q. Hence, the expected redundancy $D(q \parallel p)$ of a model distribution p w. r. t. the unknown source distribution q will be approximately $D(q \parallel p) \approx D(p_1 \parallel p)$. By the arguments of Section 4.4.2 we must not rely on the single model distribution with highest weight, but on multiple model distributions, which leads to the following idea:

> *There are m candidates that approximate the source distribution. Some candidates approximate the source distribution well, others do not. We are to find a model distribution that is "close to" good source approximations and "far away" from bad source approximations.*

Here, the weights capture the quality of approximation and we use the expected redundancy $D(p_i \parallel p)$ to measure the proximity of a model distribution p w. r. t. the approximate source distribution p_i. This leads to the optimization problem

$$\min_p w_1 D(p_1 \parallel p) + w_2 D(p_2 \parallel p) + \cdots + w_m D(p_m \parallel p). \tag{4.13}$$

The solution to this problem is straightforward.

Theorem 4.4.5 *The Linear Mixture Distribution is the minimizer of (4.13).*

Proof. Let $\boldsymbol{p} = (p_1, p_2, \dots, p_m)^\mathsf{T}$ and

$$c = \sum_{x \in \mathcal{X}} p_{\boldsymbol{w}}(x; \boldsymbol{p}) \log \frac{1}{p_{\boldsymbol{w}}(x; \boldsymbol{p})} - \sum_{x \in \mathcal{X}} \sum_{1 \leqslant i \leqslant m} w_i p_i(x) \log \frac{1}{p_i(x)}.$$

For the proof we show

$$\min_p D(p_{\boldsymbol{w}}(\boldsymbol{p}) \parallel p) = c + \min_p \sum_{1 \leqslant i \leqslant m} w_i D(p_i \parallel p) \tag{4.14}$$

which implies that $p_{\boldsymbol{w}}(\boldsymbol{p})$ minimizes (4.13), since the l. h. s. and the r. h. s. share the same minimizer and $p_{\boldsymbol{w}}(\boldsymbol{p})$ is the minimizer of the l. h. s.. We have

$$D(p_{\boldsymbol{w}}(\boldsymbol{p}) \parallel p) = \sum_{x \in \mathcal{X}} p_{\boldsymbol{w}}(x; \boldsymbol{p}) \log \frac{1}{p_{\boldsymbol{w}}(x; \boldsymbol{p})} - \sum_{x \in \mathcal{X}} p_{\boldsymbol{w}}(x; \boldsymbol{p}) \log \frac{1}{p(x)},$$

$$\begin{aligned}\sum_{1 \leqslant i \leqslant m} w_i D(p_i \parallel p) &= \sum_{1 \leqslant i \leqslant m} w_i \left(\sum_{x \in \mathcal{X}} p_i(x) \log \frac{1}{p_i(x)} - \sum_{x \in \mathcal{X}} p_i(x) \log \frac{1}{p(x)} \right) \\ &= \sum_{x \in \mathcal{X}} \sum_{1 \leqslant i \leqslant m} w_i p_i(x) \log \frac{1}{p_i(x)} - \sum_{x \in \mathcal{X}} p_{\boldsymbol{w}}(x; \boldsymbol{p}) \log \frac{1}{p(x)},\end{aligned}$$

so we conclude $D(p_{\boldsymbol{w}}(\boldsymbol{p}) \parallel p) = c + \sum_{1 \leqslant i \leqslant m} w_i D(p_i \parallel p)$ and (4.14). □

The Switching Source. The Linear Mixture Distribution is not only the minimizer of (4.13), but it also is equivalent to an assumption on the actual structure of the source. More precisely, the Linear Mixture Distribution resembles a *Switching Source*. To explain a Switching Source, we now discuss how it draws a letter at random. A Switching Source consists of m sources with source distributions $p_1, p_2, \ldots, p_m$ and a probabilistic switching mechanism. The switching mechanism selects source i with probability w_i, in turn source i emits a letter x with probability $p_i(x)$. Consequently, a Switching Source emits letter x with probability (4.12). Since the minimizer of (4.8) is unique (the cost function is convex in p and the set of all probability distributions is compact and convex), this optimization problem also is equivalent to the assumption of a Switching Source. (Actually, the proof of Theorem 4.4.5 is based upon this argument, see (4.14).).

4.5 Online Gradient Descent Mixers

In this section we study how to use OGD to estimate weights for a mixture distribution. The resulting mixer is called an Online Gradient Descent Mixer (OGDM). (Later, in Section 4.6 we will apply these results to the Linear Mixture Distribution and to the Geometric Mixture Distribution.) First, in Section 4.5.1 we give the general outline of an OGDM, justify this particular approach and discuss requirements we impose on a given mixture distribution that allow to obtain code length guarantees. The main result is presented in Section 4.5.2, where we provide code length guarantees w. r. t. a SWM for non-increasing gradient descent step size sequences. Finally, in Section 4.5.3 we refine the results of Section 4.5.2 for specific step size sequences.

4.5.1 The Setting

Online Gradient Descent Mixture. Our derivation of the Geometric Mixture Distribution in Section 4.4.2 and the Linear Mixture Distribution in Section 4.4.3 have something in common: We rely on a given set of weights, that in the end determine the contribution of every model distribution to the mixture distribution. Yet, we did not address the question of determining these weights. A straightforward approach to this problem is to adopt a method from the literature that fits our running time and space requirements. To end this we have chosen OGD from Machine Learning and Online Optimization. This choice is based upon several thoughts:

- LM uses a form of OGD and has proven to work well in practice (see Section 2.4.4 and Section 4.2).
- OGD has low running time and space requirements, thus it is a practical approach, unlike, e. g. ONS. (Recall the discussion of OCP Techniques in Section 4.2.)
- We may use analysis methods tailored to OGD and other familiar methods to obtain code length guarantees. That is, most importantly, [108], which systematically generalizes earlier approaches such as [17, 16, 39] (also see [18]).

By combining weight estimation via OGD and a Mixture Distribution we obtain an OGDM.

Definition 4.5.1 *An Online Gradient Descent Mixer (OGDM)* $\mathsf{MIX}\langle \boldsymbol{w}_1, \alpha_{1:\infty}, \{p_{\boldsymbol{w}}\}_{\boldsymbol{w}\in\Delta}\rangle$ *is given by an initial weight vector* $\boldsymbol{w}_1 \in \Delta$*, a sequence* $\alpha_{1:\infty}$ *of positive real step sizes and a family* $\{p_{\boldsymbol{w}}\}_{\boldsymbol{w}\in\Delta}$ *of mixture distributions. The mixer predicts* $\mathsf{MIX}(x_{<t}, \boldsymbol{p}_{1:t}) = p_{\boldsymbol{w}_t}(\boldsymbol{p}_t)$*, where*

$$\boldsymbol{w}_{i+1} := \operatorname{proj}(\boldsymbol{w}_i - \alpha_i \boldsymbol{d}_i), \textit{ for } 0 < i < t \textit{ and } \boldsymbol{d}_i := \nabla_{\boldsymbol{w}} \ell(x_i; p_{\boldsymbol{w}}(\boldsymbol{p}_i))\big|_{\boldsymbol{w}=\boldsymbol{w}_i}.$$

Such a gradient-based mixer operates stepwise. In the t-th step the mixer determines a mixture distribution $p_{\boldsymbol{w}}$ out of the class $\{p_{\boldsymbol{w}}\}_{\boldsymbol{w}\in\Delta}$, by choosing $\boldsymbol{w} = \boldsymbol{w}_t$ and it predicts $p_{\boldsymbol{w}_t}(\boldsymbol{p}_t)$, based on the m present model distributions stored in $\boldsymbol{p}_t$. After the prediction is made it determines the weight vector for the next step by an adjustment towards the direction of steepest descent w. r. t. the coding cost $\ell(x_t; p_{\boldsymbol{w}_t}(\boldsymbol{p}))$ of the t-th letter, i. e.

$$\boldsymbol{u} = \boldsymbol{w}_t - \alpha_t \cdot \nabla_{\boldsymbol{w}} \ell(x_t; p_{\boldsymbol{w}}(\boldsymbol{p}_t))\big|_{\boldsymbol{w}=\boldsymbol{w}_t}.$$

However, the (intermediate) weight vector $\boldsymbol{u}$ might not lie within the unit simplex. A solution to this problem is to choose the next weight $\boldsymbol{w}_{t+1}$ to be the point within the unit simplex closest (in the euclidean sense) to $\boldsymbol{u}$. The operation proj (see Section 2.1) serves for this purpose, so in the end we set $\boldsymbol{w}_{t+1} = \operatorname{proj}(\boldsymbol{u})$. There exist two popular algorithms, described in [27], for projecting a vector $\boldsymbol{u} \in \mathbb{R}^m$ onto the unit simplex. One may take either a deterministic approach that runs in time $O(m \log m)$, or a randomized algorithm which has expected running time $O(m)$. Both algorithms are rather simple to implement. We will soon rely on an important property of the projection operation [108], that is

$$|\operatorname{proj}(\boldsymbol{u}) - \boldsymbol{v}| \leqslant |\boldsymbol{u} - \boldsymbol{v}|, \text{ for all } \boldsymbol{u} \in \mathbb{R}^m \text{ and } \boldsymbol{v} \in \Delta. \tag{4.15}$$

Regularization and Approximation. The above projection-based weight update is not an ad-hoc technique, rather it may be defined in terms of a carefully engineered optimization problem. To explain this consider the gradient update based on the t-th letter. Clearly, a weight update depending on the code length of x_t should only alter the weight estimate $\boldsymbol{w}_t$ slightly: The information we are given for the weight update is little — the coding cost of a single letter. Consequently, we should penalize dramatic changes to $\boldsymbol{w}_t$, this is called regularization. To do so, we may determine the updated weight estimate as the solution of the L2-regularized problem,

$$\min_{\boldsymbol{u}\in\Delta}\left(\ell(x_t;p_u(\boldsymbol{p}_t))+\frac{c\cdot|\boldsymbol{u}-\boldsymbol{w}_t|^2}{2}\right),$$

where $c>0$ determines a trade-off between minimizing $\ell(x_t;p_u(\boldsymbol{p}_t))$ and altering the weight vector. We may approximately solve the above problem by replacing $\ell(x_t;p_u(\boldsymbol{p}_t))$ with its first-order Taylor-expansion at $\boldsymbol{w}_t$, so we obtain the problem

$$\min_{\boldsymbol{u}\in\Delta}\left(\ell(x_t;p_{\boldsymbol{w}_t}(\boldsymbol{p}_t))+(\boldsymbol{u}-\boldsymbol{w}_t)^\mathsf{T}\cdot\nabla_{\boldsymbol{w}}\ell(x_t;p_{\boldsymbol{w}}(\boldsymbol{p}_t))\big|_{\boldsymbol{w}=\boldsymbol{w}_t}+\frac{c\cdot|\boldsymbol{u}-\boldsymbol{w}_t|^2}{2}\right).$$

The solution to this problem is [11]

$$\operatorname{proj}\left(\boldsymbol{w}_t-\frac{1}{c}\cdot\nabla_{\boldsymbol{w}}\ell(x_t;p_{\boldsymbol{w}}(\boldsymbol{p}_t))\big|_{\boldsymbol{w}=\boldsymbol{w}_t}\right),$$

the gradient-projection update.

The Parameters. Not every parameter configuration of an OGDM that exhibits good practical performance will have (good) theoretical guarantees. So we impose (mild) restrictions on the parameters. As we will see below the mixture distribution plays a central role. The assumptions we require in Section 4.5 to provide a meaningful code length analysis are as follows:

Assumption 4.5.2 *For any p-vector* $\boldsymbol{p}=(p_1,p_2,\dots,p_m)^\mathsf{T}$, *any weight vector* $\boldsymbol{w}\in\Delta$ *and any letter* x *the mixture distribution* $p_{\boldsymbol{w}}(\boldsymbol{p})$ *satisfies:*

(a) *if the* i*-th component of* $\boldsymbol{w}$ *has value* 1*, then* $p_{\boldsymbol{w}}(\boldsymbol{p})=p_i$*,*

(b) *the code length* $\ell(x;p_{\boldsymbol{w}}(\boldsymbol{p}))$ *is differentiable in* $\boldsymbol{w}$ *and*

(c) *the code length* $\ell(x;p_{\boldsymbol{w}}(\boldsymbol{p}))$ *is a convex function of* $\boldsymbol{w}$*.*

We remark that Assumption 4.5.2 (b) and Assumption 4.5.2 (c) resemble common assumptions of the Machine Learning and Online Optimization literature, see [108]. Even though the above requirements are rather mild, they

Table 4.1: Notation used for the code length analysis of an OGDM within Section 4.5.

Symbol	Definition	Symbol	Definition
$x_{1:n}$	input sequence	$\mathcal{P}$	partition of SW
$\boldsymbol{p}_{1:n}, \boldsymbol{p}_t, p_{i,t}$	sequence $\boldsymbol{p}_{1:n}$ of p-vectors s. t. $\boldsymbol{p}_t = (p_{1,t}, \dots, p_{m,t})^\mathsf{T}$	MIX	Online Gradient Descent Mixer, see Def. 4.5.1
$p_{\boldsymbol{w}}(x; \boldsymbol{p})$	pr. of letter x for mixture distr. $p_{\boldsymbol{w}}(\boldsymbol{p})$ of pr. vector $\boldsymbol{p}$	α_t	t-th step size, non-negative
		$\boldsymbol{d}_t$	t-th step direction, see Def. 4.5.1
SW	Switching Mixer, see Def. 4.1.1	$\boldsymbol{w}_t$	t-th weight vector, see Def. 4.5.1

have been chosen with the analysis of a Geometric OGDM and a Linear OGDM in mind. Thus, we may even relax these assumption further: The weight space does not need to be the unit simplex, any compact convex set suffices, in turn Assumption 4.5.2 (a) must hold for an arbitrary weight vector, not necessarily the i-th unit vector. Furthermore, the code length need not be differentiable by $\boldsymbol{w}$, as long as a subgradient of $\ell(x; p_{\boldsymbol{w}}(\boldsymbol{p}))$ exists that we may use as step direction $\boldsymbol{d}_t$.

4.5.2 Online Gradient Descent Mixers vs. Switching Mixers

The Plan. We will carry out a code length analysis that compares an OGDM (when Assumption 4.5.2 is satisfied) to an arbitrary Switching Mixer. As we already noted, there exists a rich set of helpful analysis techniques that originate in the Online Optimization and Machine Learning community. In the following we adopt these techniques, most notably the approach of [108], and provide an extension to allow for an arbitrary Switching Mixer as competitor. (In terms of regret, our results allow for adaptive and shifting regret bounds, moreover our results generalize to arbitrary convex loss functions.) We split the analysis into two parts: First, we argue on the code length $\ell(x_{a:b}; \text{MIX}, \boldsymbol{p}_{1:b})$ of a OGDM MIX for an arbitrary segment $a{:}b$. Second, we consider the code length bounds for every individual segment $a{:}b \in \mathcal{P}$ of a SWM with partition $\mathcal{P}$. Summing over all segments from $\mathcal{P}$ allows us to conclude a code length bound for MIX w. r. t. SWMs. In Table 4.1 we summarize the most important notation for our analysis throughout Section 4.5.

Analysis. We begin the analysis with a code length bound that considers an arbitrary input segment. The central idea is to obtain an invariant on the progress $|\boldsymbol{w}_t - \boldsymbol{w}|^2 - |\boldsymbol{w}_{t+1} - \boldsymbol{w}|^2$ (proximity measures other than the euclidean distance are possible, as well) of the sequence of weights w. r. t. a desirable, or even arbitrary, weight vector $\boldsymbol{w}$. This technique is well-known and has previously been used e. g. in [108, 37, 38, 17, 16, 39].

Lemma 4.5.3 *For any $1 \leqslant i \leqslant m$, all input sequences sequence $x_{1:t}$ and all mixer inputs $\boldsymbol{p}_{1:t} = \langle (p_{1,k}, p_{2,k}, \dots, p_{m,k})^\mathsf{T} \rangle_{1 \leqslant k \leqslant t}$ a Gradient Descent Mixer $\mathsf{MIX}\langle \boldsymbol{w}_1, \alpha_{1:\infty}, \{p_{\boldsymbol{w}}\}_{\boldsymbol{w} \in \Delta} \rangle$ with non-increasing step sizes $\alpha_{1:\infty}$ satisfies*

$$\ell(x_{a:b}; \mathsf{MIX}, \boldsymbol{p}_{1:b}) \leqslant \frac{1}{\alpha_b} + \sum_{a \leqslant t \leqslant b} \left(\ell(x_t; p_{i,t}) + \frac{\alpha_t |\boldsymbol{d}_t|^2}{2} \right), \text{ for } 1 \leqslant a \leqslant b \leqslant t. \tag{4.16}$$

Proof. For the proof we first provide some technical inequalities, then obtain an invariant which we finally sum to yield (4.16).

Technical Inequalities. Let $\boldsymbol{w}$ be the i-th unit vector in which all components have value 0, except the i-th component, which has value 1. Assumption 4.5.2 implies

$$\begin{aligned} \boldsymbol{d}_t^\mathsf{T}(\boldsymbol{w}_t - \boldsymbol{w}) &\overset{\text{A4.5.2 (c)}}{\geqslant} \ell(x_t; p_{\boldsymbol{w}_t}(\boldsymbol{p}_t)) - \ell(x_t; p_{\boldsymbol{w}}(\boldsymbol{p}_t)) \\ &\overset{\text{A4.5.2 (a)}}{=} \ell(x_t; p_{\boldsymbol{w}_t}(\boldsymbol{p}_t)) - \ell(x_t; p_{i,t}) \quad \text{and} \qquad (4.17) \\ |\boldsymbol{u} - \boldsymbol{v}|^2 &\leqslant 2, \quad \text{for} \quad \boldsymbol{u}, \boldsymbol{v} \in \Delta. \qquad (4.18) \end{aligned}$$

Invariant. Again, let $\boldsymbol{w}$ be the i-th unit vector. We obtain the following:

$$\begin{aligned} |\boldsymbol{w}_t - \boldsymbol{w}|^2 - |\boldsymbol{w}_{t+1} - \boldsymbol{w}|^2 &\overset{(4.15)}{\geqslant} |\boldsymbol{w}_t - \boldsymbol{w}|^2 - |\boldsymbol{w}_t - \boldsymbol{w} - \alpha_t \boldsymbol{d}_t|^2 \\ &= 2\alpha_t \boldsymbol{d}_t^\mathsf{T}(\boldsymbol{w}_t - \boldsymbol{w}) - (\alpha_t |\boldsymbol{d}_t|)^2 \\ &\overset{(4.17)}{\geqslant} 2\alpha_t \big(\ell(x_t, p_{\boldsymbol{w}_t}(\boldsymbol{p}_t)) - \ell(x_t; p_{i,t}) \big) - (\alpha_t |\boldsymbol{d}_t|)^2 \\ \implies \frac{|\boldsymbol{w}_t - \boldsymbol{w}|^2 - |\boldsymbol{w}_{t+1} - \boldsymbol{w}|^2}{2\alpha_t} &\geqslant \ell(x_t, p_{\boldsymbol{w}_t}(\boldsymbol{p}_t)) - \ell(x_t; p_{i,t}) - \frac{\alpha_t |\boldsymbol{d}_t|^2}{2}. \qquad (4.19) \end{aligned}$$

Summing. By summing (4.19) for $t \in a{:}b$ we obtain

$$\begin{aligned} \sum_{a \leqslant t \leqslant b} \frac{|\boldsymbol{w}_t - \boldsymbol{w}|^2 - |\boldsymbol{w}_{t+1} - \boldsymbol{w}|^2}{2\alpha_t} &= \sum_{a \leqslant t \leqslant b} \frac{|\boldsymbol{w}_t - \boldsymbol{w}|^2}{2\alpha_t} - \sum_{a < t \leqslant b+1} \frac{|\boldsymbol{w}_t - \boldsymbol{w}|^2}{2\alpha_{t-1}} \\ &\leqslant \frac{|\boldsymbol{w}_a - \boldsymbol{w}|^2}{2\alpha_a} + \sum_{a < t \leqslant b} \frac{|\boldsymbol{w}_t - \boldsymbol{w}|^2}{2} \left(\frac{1}{\alpha_t} - \frac{1}{\alpha_{t-1}} \right) \\ &\overset{\substack{\alpha_t \leqslant \alpha_{t-1}, \\ (4.18)}}{\leqslant} \frac{1}{\alpha_a} + \sum_{a < t \leqslant b} \left(\frac{1}{\alpha_t} - \frac{1}{\alpha_{t-1}} \right) = \frac{1}{\alpha_b}, \qquad (4.20) \end{aligned}$$

for the l. h. s. of (4.19) and

$$\ell(x_{a:b}; \mathsf{MIX}, \boldsymbol{p}_{1:b}) - \sum_{a \leqslant t \leqslant b} \left(\ell(x_t; p_{i,t}) + \frac{\alpha_t |\boldsymbol{d}_t|^2}{2} \right), \tag{4.21}$$

for the r. h. s. of (4.19), whereby $\sum_{a \leqslant t \leqslant b} \ell(x_t; p_{\boldsymbol{w}_t}(\boldsymbol{p}_t)) \overset{\text{D4.5.1}}{=} \ell(x_{a:b}; \mathsf{MIX}, \boldsymbol{p}_{1:b})$. To end the proof we combine (4.20) and (4.21) and rearrange. □

We now generalize the previous code length bound to SWM.

Theorem 4.5.4 *The code length of an OGDM* MIX *with a non-increasing step size sequence $\alpha_{1:\infty}$ is bounded from above w. r. t. a SWM* SW *with partition $\mathcal{P}$ by*

$$\ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n}) \leqslant \ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \sum_{a:b \in \mathcal{P}} \left(\frac{1}{\alpha_b} + \frac{1}{2} \sum_{1 \leqslant t \leqslant n} \alpha_t |\boldsymbol{d}_t|^2 \right).$$

Proof. Let the t-th p-vector be $\boldsymbol{p}_t = (p_{1,t}, p_{2,t}, \dots, p_{m,t})^\mathsf{T}$. For the proof we apply Lemma 4.5.3 to any segment $s \in \mathcal{P}$, so

$$\begin{aligned}
\ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n}) &= \sum_{a:b \in \mathcal{P}} \ell(x_{a:b}; \mathsf{MIX}, \boldsymbol{p}_{1:b}) \\
&\overset{\text{L4.5.3}}{\leqslant} \sum_{\substack{s=a:b,\\ s \in \mathcal{P}}} \frac{1}{\alpha_b} + \sum_{t \in s} \left(\ell(x_t; p_{m_s,t}) + \frac{\alpha_t |\boldsymbol{d}_t|^2}{2} \right) \\
&\overset{\text{D4.1.1}}{=} \ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \sum_{a:b \in \mathcal{P}} \left(\frac{1}{\alpha_b} + \frac{1}{2} \sum_{1 \leqslant t \leqslant n} \alpha_t |\boldsymbol{d}_t|^2 \right). \quad \square
\end{aligned}$$

Discussion. Theorem 4.5.4 relates the redundancy of an OGDM to the step size sequence $\alpha_{1:\infty}$, to the norm of step directions $|\boldsymbol{d}_t|$ and to the structure of the competing SWM, represented by the partition $\mathcal{P}$. The result is of very general nature, but will be of great use to analyze OGDM coupled with the Linear- and Geometric Mixture Distribution for various step size sequences. Note that minimizing the redundancy term is a balancing act: One one hand, the step sizes must be sufficiently small (or decrease sufficiently fast), so that $\sum_{1 \leqslant t \leqslant n} \alpha_t |\boldsymbol{d}_t|^2$ is not too large; on the other hand, the step sizes may not be too small (or decrease too fast), to keep the term $\sum_{a:b \in \mathcal{P}} \frac{1}{\alpha_b}$ small. Moreover, the more complex the competing SWM is, i. e. as the number of segments in $\mathcal{P}$ increases, the harder it will be for a OGDM to score well. (It is hard to go head to head with a sophisticated competitor.) When the step size sequence is decreasing, the redundancy term $\sum_{a:b\in\mathcal{P}} \frac{1}{\alpha_b}$ also penalizes the location of a segment $a{:}b$, rather than just the number of segments. The closer a segment endpoint b lies towards the end of the sequence, the bigger the penalization term $\frac{1}{\alpha_b}$ will be. Such a behavior is plausible, since a small step size makes it hard to adapt to the input.

4.5.3 Choice of Step Sizes

Code Length Bounds. We now refine our previous results by considering a fixed step size and two varying step size choices. For either result we must bound the squared norm $|\boldsymbol{d}_t|^2$ of a step direction. We will rely on the results from the following corollary, when we analyze OGDM coupled with the Linear- and Geometric Mixture Distribution in Section 4.6.

Corollary 4.5.5 *The code length of an OGDM* MIX *with step sizes $\alpha_{1:\infty}$ is bounded from above w. r. t. to that of a SWM* SW$\langle\mathcal{P}, \{m_s\}_{s\in\mathcal{P}}\rangle$ *by:*

(a) *If $|\boldsymbol{d}_t|^2 \leqslant D^2$, for $1 \leqslant t \leqslant n$, and $\alpha = \alpha_1 = \alpha_2 = \dots$, then*

$$\ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n}) \leqslant \ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \frac{|\mathcal{P}|}{\alpha} + \frac{\alpha D^2}{2} \cdot n.$$

(b) *If $|\boldsymbol{d}_t|^2 \leqslant D^2$, for $1 \leqslant t \leqslant n$, and $\alpha_t = t^{-1/2}$, then*

$$\ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n}) \leqslant \ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \big(|\mathcal{P}| + D^2\big) \cdot \sqrt{n}.$$

(c) *If $\alpha_t|\boldsymbol{d}_t|^2 \leqslant \frac{2\delta}{1+\delta} \cdot \ell(x_t; \mathsf{MIX}(x_{<t}, \boldsymbol{p}_{1:t}))$, for $1 \leqslant t \leqslant n$ and $1 \leqslant i \leqslant m$, then*

$$\ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n}) \leqslant (1+\delta) \cdot \ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \frac{|\mathcal{P}|(1+\delta)}{\alpha_n}.$$

Proof. In either situation, we first apply Theorem 4.5.4, then we simplify the expression

$$\sum_{a:b\in\mathcal{P}} \frac{1}{\alpha_b} + \frac{1}{2} \sum_{1\leqslant t\leqslant n} \alpha_t |\boldsymbol{d}_t|^2 \tag{4.22}$$

and finally rearrange.

(a) By the step size choice we have (4.22) $= \frac{|\mathcal{P}|}{\alpha} + \frac{\alpha D^2}{2} \cdot n$.

(b) We have (4.22) $\leqslant (|\mathcal{P}| + D^2) \cdot \sqrt{n}$, since $\sum_{a:b\in\mathcal{P}} \frac{1}{\alpha_b} \leqslant \frac{|\mathcal{P}|}{\alpha_n} = |\mathcal{P}|\sqrt{n}$,

$$\sum_{1\leqslant t\leqslant n} \frac{\alpha_t|\boldsymbol{d}_t|^2}{2} \leqslant \frac{D^2}{2} \sum_{1\leqslant t\leqslant n} \frac{1}{\sqrt{t}} \quad \text{and} \quad \sum_{1\leqslant t\leqslant n} \frac{1}{\sqrt{t}} \leqslant 1 + \int_1^n \frac{\mathrm{d}z}{\sqrt{z}} \leqslant 2\sqrt{n}.$$

(c) We have (4.22) $\leqslant \frac{|\mathcal{P}|}{\alpha_n} + \frac{\delta}{1+\delta} \cdot \ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n})$, since $\sum_{a:b\in\mathcal{P}} \frac{1}{\alpha_b} \leqslant \frac{|\mathcal{P}|}{\alpha_n}$ and

$$\sum_{1\leqslant t\leqslant n} \frac{\alpha_t |\boldsymbol{d}_t|^2}{2} \leqslant \frac{\delta}{1+\delta} \sum_{1\leqslant t\leqslant n} \ell(x_t; \mathsf{MIX}(x_{<t}, \boldsymbol{p}_{1:t})) \overset{\text{D4.5.1}}{=} \frac{\delta}{1+\delta} \cdot \ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n}).$$

□

Discussion. Let us now inspect the different step size choices from Corollary 4.5.5. From a practical perspective a fixed step size is appealing, since there is no need to store or maintain a step size sequence. Corollary 4.5.2 (a) allows us to tune α s. t. the corresponding code length bound becomes minimal. Unfortunately, we must know the sequence length n and a desirable number of switches $|\mathcal{P}| - 1$ of a competing SWM in advance. To overcome this limitation the varying step size choice in Corollary 4.5.2 (b) gives redundancy $O(|\mathcal{P}| \cdot \sqrt{n})$ w. r. t. any SWM with partition $\mathcal{P}$. As long as a competing SWM is not too complex (has $|\mathcal{P}| = o(\sqrt{n})$ segments), we get sublinear redundancy. This improvement is not for free, the price we have to pay is a slight increase in processing time to maintain the step size sequence. Finally, the step size choice from Corollary 4.5.2 (c) yields a weaker code length guarantee than the former two step size choices, since the code length $\ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n})$ might be off the code length $\ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n})$ not just by an additive redundancy term, but also by a multiplicative factor $1 + \delta$. However, this weaker guarantee might turn the tide and provide a better upper bound, when $\ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n})$ is small.

4.6 Applications

We now have assembled the tools we need to analyze OGD coupled with either a Linear Mixture Distribution (Section 4.6.1) or a Geometric Mixture Distribution (Section 4.6.2). In both cases we proceed similarly. First, we introduce the corresponding mixture scheme and obtain the gradient, then we check for the validity of Assumption 4.5.2, and finally we apply Corollary 4.5.5 to conclude code length bounds for different step size choices. In Table 4.2 we summarize the most important notation for the purposes of Section 4.6.

4.6.1 Linear Mixture Distribution

The Linear Online Gradient Descent Mixer. We first define the subject of interest for this section, that is, OGD with a Linear Mixture Distribution.

Table 4.2: Notation used for the code length analysis of an OGDM within Section 4.6.

Symbol	Definition	Symbol	Definition
$x_{1:n}$	input sequence	m	dimension of pr. and weight vectors
$\boldsymbol{p}_{1:n}, \boldsymbol{p}_t, p_{i,t}$	sequence $\boldsymbol{p}_{1:n}$ of p-vectors s. t. $\boldsymbol{p}_t = (p_{1,t}, \dots, p_{m,t})^\top$	SW	Switching Mixer, see Def. 4.1.1
		$\mathcal{P}$	partition of SW
$p_{\boldsymbol{w}}(x; \boldsymbol{p})$	pr. of letter x for mixture distr. $p_{\boldsymbol{w}}(\boldsymbol{p})$ of pr. vector $\boldsymbol{p}$	LIN	Linear Online Gradient Descent Mixer, see Def. 4.6.1
$\boldsymbol{p}(x)$	$(p_1(x), \dots, p_m(x))^\top$, for pr. vector $\boldsymbol{p} = (p_1, \dots, p_m)^\top$	GEO	Geometric Online Gradient Descent Mixer, see Def. 4.6.5
$\boldsymbol{\ell}(x)$	$(\log \frac{1}{p_1(x)}, \dots, \log \frac{1}{p_m(x)})^\top$, for pr. vector $\boldsymbol{p} = (p_1, \dots, p_m)^\top$	α_t	t-th step size, non-negative
		$\boldsymbol{d}_t$	t-th step direction, see (4.24), (4.29)
$\varepsilon, \varepsilon_t$	lower bound on pr. $p_{i,t}(x)$	$\boldsymbol{w}_t$	t-th weight vector, see Def. 4.5.1

Definition 4.6.1 *For the class $\mathcal{C} = \{p_{\boldsymbol{w}}\}_{\boldsymbol{w} \in \Delta}$ of linear mixture distributions the OGDM* $\mathsf{LIN}\langle \boldsymbol{w}_1, \alpha_{1:\infty}, \mathcal{C}\rangle$ *is called the* Linear OGDM.

For an actual application of a Linear OGDM we rely on the gradient of the coding cost $\ell(x; p_{\boldsymbol{w}}(\boldsymbol{p}))$ for a letter x based on a Linear Mixture Distribution $p_{\boldsymbol{w}}(\boldsymbol{p})$ of the p-vector $\boldsymbol{p}$. The gradient works out to

$$\frac{\partial \ell(x; p_{\boldsymbol{w}}(\boldsymbol{p}))}{\partial w_i} = -\frac{p_i(x) \cdot \log e}{p_{\boldsymbol{w}}(x; \boldsymbol{p})} \implies \nabla_{\boldsymbol{w}} \ell(x; p_{\boldsymbol{w}}(\boldsymbol{p})) = -\frac{\log(e) \cdot \boldsymbol{p}(x)}{p_{\boldsymbol{w}}(x; \boldsymbol{p})}, \quad (4.23)$$

which allows us to specify the step direction $\boldsymbol{d}_t$. To do so, we evaluate (4.23) for the letter $x = x_t$, weight vector $\boldsymbol{w} = \boldsymbol{w}_t$, p-vector $\boldsymbol{p} = \boldsymbol{p}_t$ and use the identity $\mathsf{LIN}(x; x_{<t}, \boldsymbol{p}_{1:t}) = p_{\boldsymbol{w}_t}(x; \boldsymbol{p}_t)$, this yields

$$\boldsymbol{d}_t = -\frac{\log e}{\mathsf{LIN}(x_t; x_{<t}, \boldsymbol{p}_{1:t})} \cdot \boldsymbol{p}_t(x_t). \quad (4.24)$$

Finally, before we proceed with the code length analysis, we make an interesting observation, that is, there is a link to Beta-Weighting: If we take the gradient update $\boldsymbol{w}_{t+1} = \operatorname{proj}(\boldsymbol{w}_t - \alpha_t \boldsymbol{d}_t)$ and replace the scalar step size α_t with a diagonal step size matrix $\boldsymbol{A}_t$, s. t. the i-th diagonal position is

$$\frac{\mathsf{LIN}(x_t; x_{<t}, \boldsymbol{p}_{1:t}) - p_{t,i}(x_t)}{p_{t,i}(x_t)} \frac{w_{t,i}}{\log e},$$

then we obtain Beta-Weighting. (The intermediate weight vector $\boldsymbol{w} = \boldsymbol{w}_t - \alpha_t \boldsymbol{d}_t$ lies within the unit simplex and $\operatorname{proj}(\boldsymbol{w}) = \boldsymbol{w}$.) So we may view Beta-Weighting as a variable metric method for Online Optimization.

Analysis. Before we provide code length bounds we must verify Assumption 4.5.2 and the requirements of Corollary 4.5.5 (in essence we must bound $|\boldsymbol{d}_t|^2$). To do so, we first parenthesize the following technical statement:

Lemma 4.6.2 *For* $0 < \varepsilon \leqslant z \leqslant 1-\varepsilon$ *and* $f(z) := z^2 \ln \frac{1}{z}$ *we have* $f(z) \geqslant f(\varepsilon)$.

Proof. For the proof we show the two inequalities

$$f(z) \geqslant \min\{f(\varepsilon), f(1-\varepsilon)\} \quad \text{and} \quad f(1-\varepsilon) \geqslant f(\varepsilon), \text{ for } 0 < \varepsilon \leqslant \frac{1}{2}, \tag{4.25}$$

(we have $\varepsilon \leqslant \frac{1}{2}$, by the bounds on z) which imply the claim.

Bounding $f(z)$ *by* $f(\varepsilon)$ *and* $f(1-\varepsilon)$. Notice that $z_0 = e^{-1/2}$ is the root of the derivative $f'(z) = -z(1+2\ln z)$ of f, by examining $f'(z)$ we conclude:

$$\begin{aligned} f'(z) \geqslant 0, \text{ for } 0 < z < z_0 &\implies f(z) \geqslant f(\varepsilon), \text{ for } \varepsilon < z < z_0 \quad \text{and} \\ f'(z) \leqslant 0, \text{ for } z_0 \leqslant z < 1 &\implies f(z) \geqslant f(1-\varepsilon), \text{ for } z_0 \leqslant z \leqslant 1-\varepsilon, \end{aligned}$$

which implies the first inequality in (4.25).

Bounding $f(1-\varepsilon)$ *by* $f(\varepsilon)$. Let $g(\varepsilon) := f(\varepsilon)/f(1-\varepsilon)$, again we obtain the derivative

$$g'(\varepsilon) = -\underbrace{\frac{\varepsilon(1-\varepsilon)\cdot \ln\frac{1}{\varepsilon} \cdot \ln\frac{1}{1-\varepsilon}}{\left((1-\varepsilon)^2 \ln \frac{1}{1-\varepsilon}\right)^2}}_{A} \cdot \underbrace{\left(\frac{\varepsilon}{\ln\frac{1}{1-\varepsilon}} + \frac{1-\varepsilon}{\ln\frac{1}{\varepsilon}} - 2\right)}_{B} = -A \cdot B.$$

Clearly, $A \geqslant 0$ and by the basic inequality $\ln\frac{1}{z} \geqslant 1-z$ we obtain $B \leqslant 0$, so $g'(\varepsilon) \geqslant 0$. Since g is increasing, we have $g(\varepsilon) \leqslant g(\frac{1}{2}) = 1$, and so the second inequality in (4.25) follows. □

To proceed, we now provide the formal basis to apply Corollary 4.5.5.

Lemma 4.6.3 *For a Linear OGDM it holds that:*

(a) *Assumption 4.5.2 is satisfied,*

(b) *if any probability distribution in the mixer input $\boldsymbol{p}_{1:n}$ assigns probability at least $\varepsilon > 0$ to any letter, then*

$$|\boldsymbol{d}_t|^2 \leqslant \frac{m \log^2 e}{\varepsilon^2},$$

(c) *we have $\alpha_t |\boldsymbol{d}_t|^2 \leqslant \frac{2\delta}{1+\delta} \cdot \ell(x_t; \mathsf{LIN}(x_{<t}, \boldsymbol{p}_{1:t}))$, if the probability distributions of the t-th p-vector assign probability at least $\varepsilon_t > 0$ to any letter and if we further choose*

$$\alpha_t = \frac{2\varepsilon_t^2 \log \frac{1}{\varepsilon_t}}{m \log^2 e} \cdot \frac{\delta}{1+\delta}.$$

Proof. **(a)** It is easy to see that both, Assumption 4.5.2 (a) and Assumption 4.5.2 (b) (cf. (4.23)), are satisfied. For Assumption 4.5.2 (c) we observe that the Hessian

$$\nabla^2_{\boldsymbol{w}} \ell(x; p_{\boldsymbol{w}}(\boldsymbol{p})) = \frac{\log e}{p_{\boldsymbol{w}}(x; \boldsymbol{p})^2} \cdot \boldsymbol{p}(x)\boldsymbol{p}(x)^\mathsf{T}$$

is positive semi-definite, since $\boldsymbol{z}^\mathsf{T}\boldsymbol{p}(x)\boldsymbol{p}(x)^\mathsf{T}\boldsymbol{z} = (\boldsymbol{z}^\mathsf{T}\boldsymbol{p}(x))^2 \geqslant 0$, for arbitrary real-valued vectors $\boldsymbol{p}(x)$ and $\boldsymbol{z}$.

(b) By $|\boldsymbol{p}_t(x)|^2 \leqslant m$ and $\mathsf{LIN}(x; x_{<t}, \boldsymbol{p}_{1:t}) \geqslant \varepsilon$ we obtain

$$|\boldsymbol{d}_t|^2 \overset{(4.24)}{=} \left(\frac{\log e}{\mathsf{LIN}(x_t; x_{<t}, \boldsymbol{p}_{1:t})} \cdot |\boldsymbol{p}_t(x)| \right)^2 \leqslant \frac{m \log^2 e}{\varepsilon^2}.$$

(c) Notice that we may bound $|\boldsymbol{d}_t|^2$ similarly to **(b)**. We proceed by writing

$$\frac{\alpha_t |\boldsymbol{d}_t|^2}{\ell(x_t; \mathsf{LIN}(x_{<t}, \boldsymbol{p}_{1:t}))} \leqslant \frac{\alpha_t m \log^2 e}{z^2 \log \frac{1}{z}} \overset{\text{L4.6.2}}{\leqslant} \frac{\alpha_t m \log^2 e}{\varepsilon_t^2 \log \frac{1}{\varepsilon_t}} = \frac{2\delta}{1+\delta},$$

where we set $z = \mathsf{LIN}(x_t; x_{<t}, \boldsymbol{p}_{1:t})$ and applied Lemma 4.6.2 for $\varepsilon = \varepsilon_t$. This proves the claim. □

A Linear OGDM meets all prerequisites for an application of Corollary 4.5.5, so we finally conclude code length guarantees.

Corollary 4.6.4 *A Linear OGDM satisfies the code length bounds of Table 4.3, given the listed step size choices and requirements.*

Proof. For the proof we combine Corollary 4.5.5 and Lemma 4.6.3. Note that in either case the step size sequence is non-increasing.

Table 4.3: Code length bounds for a Linear OGDM $\mathsf{LIN}\langle \boldsymbol{w}_1, \alpha_{1:\infty}\rangle$ w. r. t. a SWM $\mathsf{SW}\langle \mathcal{P}, \{m_s\}_{s\in\mathcal{P}}\rangle$. The t-th p-vector of the mixer input $\boldsymbol{p}_{1:n}$ is $\boldsymbol{p}_t = (p_{1,t}, p_{2,t}, \ldots, p_{m,t})^\mathsf{T}$ (dimension: m).

Step Size α_t	**Requirement(s)**	**Code Length Bound** $\ell(x_{1:n}; \mathsf{LIN}, \boldsymbol{p}_{1:n}) \leqslant \ldots$
α	$p_{i,t}(x) \geqslant \varepsilon > 0$	$\ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \frac{\lvert\mathcal{P}\rvert}{\alpha} + \frac{\alpha m \log^2 e}{2\varepsilon^2} \cdot n$
$\frac{1}{\sqrt{t}}$	$p_{i,t}(x) \geqslant \varepsilon > 0$	$\ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \left(\lvert\mathcal{P}\rvert + \frac{m \log^2 e}{\varepsilon^2}\right) \cdot \sqrt{n}$
$\frac{2\varepsilon_t^2 \log \frac{1}{\varepsilon_t}}{m \log^2 e} \frac{\delta}{1+\delta}$	$p_{i,t}(x) \geqslant \varepsilon_t > 0$, $\delta > 0,\ \varepsilon_t \leqslant \varepsilon_{t-1}$	$(1+\delta) \cdot \ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \frac{\lvert\mathcal{P}\rvert(1+\delta)^2 m \log^2 e}{2\delta\varepsilon_n^2 \log \frac{1}{\varepsilon_n}}$

4.6.2 Geometric Mixture Distribution

The Geometric Online Gradient Descent Mixer. We now consider the combination of a Geometric Mixture Distribution and an OGDM, more formally:

Definition 4.6.5 *For the class $\mathcal{C} = \{p_{\boldsymbol{w}}\}_{\boldsymbol{w}\in\Delta}$ of Geometric Mixture Distributions the OGDM $\mathsf{GEO}\langle \boldsymbol{w}_1, \alpha_{1:\infty}, \mathcal{C}\rangle$ is called the* Geometric OGDM.

It will be very helpful to work with the following representation of a Geometric Mixture Distribution,

$$p_{\boldsymbol{w}}(x; \boldsymbol{p}) = \frac{2^{-\boldsymbol{\ell}(x)^\mathsf{T}\boldsymbol{w}}}{\sum_{y\in\mathcal{X}} 2^{-\boldsymbol{\ell}(y)^\mathsf{T}\boldsymbol{w}}}, \text{ where } \boldsymbol{\ell}(x) = \left(\log \tfrac{1}{p_1(x)}, \log \tfrac{1}{p_2(x)}, \ldots, \log \tfrac{1}{p_m(x)}\right)^\mathsf{T}$$
$$\text{and} \qquad \boldsymbol{p} = (p_1, p_2, \ldots, p_m)^\mathsf{T}. \tag{4.26}$$

This alternate formulation equals the initial formulation we gave in Definition 4.4.1 and greatly simplifies upcoming calculations that involve gradients. We now obtain the gradient of the coding cost $\ell(x; p_{\boldsymbol{w}}(\boldsymbol{p}))$ of a letter x for the Geometric Mixture Distribution $p_{\boldsymbol{w}}$ of p-vector $\boldsymbol{p}$ by

$$\begin{aligned}
\frac{\partial \ell(x; p_{\boldsymbol{w}}(\boldsymbol{p}))}{\partial w_i} &= \frac{\partial}{\partial w_i}\left(\boldsymbol{\ell}(x)^\mathsf{T}\boldsymbol{w} + \log \sum_{y\in\mathcal{X}} 2^{-\boldsymbol{\ell}(y)^\mathsf{T}\boldsymbol{w}}\right) \\
&= \log \frac{1}{p_i(x)} - \sum_{y\in\mathcal{X}} \log \frac{1}{p_i(y)} \cdot \frac{2^{-\boldsymbol{\ell}(y)^\mathsf{T}\boldsymbol{w}}}{\sum_{z\in\mathcal{X}} 2^{-\boldsymbol{\ell}(z)^\mathsf{T}\boldsymbol{w}}} \\
\Longrightarrow \quad \nabla_{\boldsymbol{w}} \ell(x; p_{\boldsymbol{w}}(\boldsymbol{p})) &= \boldsymbol{\ell}(x) - \sum_{y\in\mathcal{X}} p_{\boldsymbol{w}}(y; \boldsymbol{p}) \cdot \boldsymbol{\ell}(y)
\end{aligned} \tag{4.27}$$

$$= \sum_{\substack{y\in\mathcal{X},\\ y\neq x}} p_{\boldsymbol{w}}(y;\boldsymbol{p})(\boldsymbol{\ell}(x)-\boldsymbol{\ell}(y)). \tag{4.28}$$

By evaluating the gradient $\nabla_{\boldsymbol{w}}\ell(x;p_{\boldsymbol{w}}(\boldsymbol{p}))$ for the letter $x = x_t$, the weight vector $\boldsymbol{w} = \boldsymbol{w}_t$ and the p-vector $\boldsymbol{p} = \boldsymbol{p}_t$ we obtain the step direction as

$$\boldsymbol{d}_t \overset{(4.27)}{=} \boldsymbol{\ell}_t(x_t) - \sum_{y\in\mathcal{X}} \mathsf{GEO}(y;x_{<t},\boldsymbol{p}_{1:t})\cdot\boldsymbol{\ell}_t(y), \quad \text{where}$$
$$\boldsymbol{\ell}_t(x) = \left(\log\frac{1}{p_{1,t}(x)}, \log\frac{1}{p_{2,t}(x)}, \dots, \log\frac{1}{p_{m,t}(x)}\right)^{\mathsf{T}}. \tag{4.29}$$

We end this section with an interesting observation, namely that for the Geometric Mixture Distribution $p_{\boldsymbol{w}}(\boldsymbol{p})$ of p-vector $\boldsymbol{p} = (p_1, p_2, \dots, p_m)^{\mathsf{T}}$ we may express the gradient of the code length $\ell(x;p_{\boldsymbol{w}}(\boldsymbol{p}))$ in terms of information theoretic quantities. To see this, consider the i-th component of $\ell(x;p_{\boldsymbol{w}}(\boldsymbol{p}))$, which works out to

$$\log\frac{1}{p_i(x)} - \sum_{y\in\mathcal{X}} p_{\boldsymbol{w}}(y;\boldsymbol{p})\log\frac{1}{p_i(y)} = \log\frac{1}{p_i(x)} - H(p_{\boldsymbol{w}}(\boldsymbol{p})) - D(p_{\boldsymbol{w}}(\boldsymbol{p}) \parallel p_i). \tag{4.30}$$

Equation (4.30) allows for an interesting conclusion on the Geometric Mixture Distribution $p_{\boldsymbol{w}^*}$ that minimizes the coding cost of a given letter x. Suppose that we want to minimize the coding cost of a letter x, where we allow for arbitrary weight vectors that do not need to lie within the unit simplex. (Even then a Geometric Mixture Distribution is well defined, as long as all probability distributions in $\boldsymbol{p}$ assign positive probabilities, cf. Definition 4.4.1.) Since for the Geometric Mixture Distribution $p_{\boldsymbol{w}}(\boldsymbol{p})$ the code length $\ell(x;p_{\boldsymbol{w}}(\boldsymbol{p}))$ is convex in $\boldsymbol{w}$ (Assumption 4.5.2 (c) and Lemma 4.6.6 (a), see below), we may simply set (4.30) to 0, for all $1 \leqslant i \leqslant m$, to obtain the minimizer $\boldsymbol{w}^*$. In this situation there is an equilibrium: The weight vector $\boldsymbol{w}^*$ is chosen s. t. for every model distribution the code length $\log\frac{1}{p_i(x)}$ of letter x equals the expected code length $H(p_{\boldsymbol{w}^*}(\boldsymbol{p})) - D(p_{\boldsymbol{w}^*}(\boldsymbol{p}) \parallel p_i)$ of coding a letter drawn according to source distribution $p_{\boldsymbol{w}^*}(\boldsymbol{p})$ with model distribution p_i.

Analysis. Again, we must check the preconditions, so that we can draw benefit from Corollary 4.5.5, which is straightforward:

Lemma 4.6.6 *For a Geometric OGDM we have:*

(a) *Assumption 4.5.2 is satisfied,*

(b) *if any probability distribution in the mixer input $\boldsymbol{p}_{1:n}$ assigns probability at least $\varepsilon > 0$ to any letter, then*

$$|\boldsymbol{d}_t|^2 \leqslant m \log^2 \frac{1}{\varepsilon},$$

(c) *we have $\alpha_t |\boldsymbol{d}_t|^2 \leqslant \frac{2\delta}{1+\delta} \cdot \ell(x_t; \mathsf{GEO}(x_{<t}, \boldsymbol{p}_{1:t}))$, if the probability distributions of the t-th p-vector assign probability at least $\varepsilon_t > 0$ to any letter and if we further choose*

$$\alpha_t = \frac{2 \log e}{m \log^2 \frac{1}{\varepsilon_t}} \cdot \frac{\delta}{1+\delta}.$$

Proof. **(a)** It is easy to see that Assumption 4.5.2 (a) (see Definition 4.4.1) and Assumption 4.5.2 (b) (see (4.27)) are satisfied. For Assumption 4.5.2 (c) we first introduce some notation: Fix an arbitrary letter x, an arbitrary p-vector $\boldsymbol{p}$, adopt the notation given in (4.26) and define $\ell(\boldsymbol{w}) := -\log p_{\boldsymbol{w}}(x; \boldsymbol{p})$. It remains to show that $\ell(\boldsymbol{w})$ is convex. To do so, fix two weight vectors $\boldsymbol{u}, \boldsymbol{v} \in \Delta$, and observe that

$$\begin{aligned}
\ell(\boldsymbol{v}) - \ell(\boldsymbol{u}) &\overset{(4.26)}{=} \boldsymbol{\ell}(x)^{\mathsf{T}}(\boldsymbol{v}-\boldsymbol{u}) + \log \frac{\sum_{y\in\mathcal{X}} 2^{-\boldsymbol{\ell}(y)^{\mathsf{T}}\boldsymbol{v}}}{\sum_{z\in\mathcal{X}} 2^{-\boldsymbol{\ell}(z)^{\mathsf{T}}\boldsymbol{u}}} \\
&= \boldsymbol{\ell}(x)^{\mathsf{T}}(\boldsymbol{v}-\boldsymbol{u}) + \log \sum_{y\in\mathcal{X}} \frac{2^{-\boldsymbol{\ell}(y)\boldsymbol{u}}}{\sum_{z\in\mathcal{X}} 2^{-\boldsymbol{\ell}(z)\boldsymbol{u}}} \cdot 2^{-\boldsymbol{\ell}(y)^{\mathsf{T}}(\boldsymbol{v}-\boldsymbol{u})} \\
&= \boldsymbol{\ell}(x)^{\mathsf{T}}(\boldsymbol{v}-\boldsymbol{u}) + \log \sum_{y\in\mathcal{X}} p_{\boldsymbol{u}}(y;\boldsymbol{p}) \cdot 2^{-\boldsymbol{\ell}(y)^{\mathsf{T}}(\boldsymbol{v}-\boldsymbol{u})} \\
&\overset{\substack{\text{Jensen's}\\ \text{Ineq.}}}{\geqslant} \left(\boldsymbol{\ell}(x)^{\mathsf{T}} - \sum_{y\in\mathcal{X}} p_{\boldsymbol{u}}(y;\boldsymbol{p}) \cdot \boldsymbol{\ell}(y)^{\mathsf{T}} \right) \cdot (\boldsymbol{v}-\boldsymbol{u}) \\
&\overset{(4.27)}{=} \nabla \ell(\boldsymbol{u})^{\mathsf{T}} \cdot (\boldsymbol{v}-\boldsymbol{u}).
\end{aligned}$$

(b) Any letter probability is at least ε, so $\left|\log \frac{p_{t,i}(x)}{p_{t,i}(y)}\right| \leqslant \log \frac{1}{\varepsilon}$ and we conclude

$$\begin{aligned}
|\boldsymbol{d}_t|^2 &\overset{(4.29)}{\leqslant} \sum_{\substack{y\in\mathcal{X},\\ y\neq x_t}} \mathsf{GEO}(y; x_{<t}, \boldsymbol{p}_{1:t}) \sum_{1\leqslant i\leqslant m} \log^2 \frac{p_{t,i}(y)}{p_{t,i}(x_t)} \\
&\leqslant (1 - \mathsf{GEO}(x_t; x_{<t}, \boldsymbol{p}_{1:t})) m \log^2 \frac{1}{\varepsilon} \leqslant m \log^2 \frac{1}{\varepsilon}.
\end{aligned}$$

Table 4.4: Code length bounds for a Geometric OGDM $\mathsf{GEO}\langle \boldsymbol{w}_1, \alpha_{1:\infty}\rangle$ w. r. t. a SWM $\mathsf{SW}\langle \mathcal{P}, \{m_s\}_{s\in\mathcal{P}}\rangle$. The t-th p-vector of the mixer input $\boldsymbol{p}_{1:n}$ is $\boldsymbol{p}_t = (p_{1,t}, p_{2,t}, \dots, p_{m,t})^\mathsf{T}$ (dimension: m).

Step Size α_t	Requirement(s)	Code Length Bound $\ell(x_{1:n}; \mathsf{GEO}, \boldsymbol{p}_{1:n}) \leqslant \dots$
α	$p_{i,t}(x) \geqslant \varepsilon > 0$	$\ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \dfrac{\lvert\mathcal{P}\rvert}{\alpha} + \dfrac{\alpha m \log^2 \frac{1}{\varepsilon}}{2} \cdot n$
$\dfrac{1}{\sqrt{t}}$	$p_{i,t}(x) \geqslant \varepsilon > 0$	$\ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \left(\lvert\mathcal{P}\rvert + m \log^2 \dfrac{1}{\varepsilon}\right) \cdot \sqrt{n}$
$\dfrac{2\log e}{m \log^2 \frac{1}{\varepsilon_n}} \dfrac{\delta}{1+\delta}$	$p_{i,t}(x) \geqslant \varepsilon_t > 0$, $\delta > 0,\ \varepsilon_t \leqslant \varepsilon_{t-1}$	$(1+\delta) \cdot \ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) + \dfrac{\lvert\mathcal{P}\rvert(1+\delta)^2 m \log^2 \frac{1}{\varepsilon_n}}{2\delta \log e}$

(c) Notice that we may bound $|\boldsymbol{d}_t|^2$ similarly to **(b)**, we proceed by writing

$$\frac{\alpha_t |\boldsymbol{d}_t|^2}{\ell(x_t; \mathsf{GEO}(x_{<t}, \boldsymbol{p}_{1:t}))} \leqslant \frac{\alpha_t (1-z) m \log^2 \frac{1}{\varepsilon_t}}{\log \frac{1}{z}} \leqslant \frac{\alpha_t m \log^2 \frac{1}{\varepsilon_t}}{\log e} = \frac{2\delta}{1+\delta},$$

where we set $z = \mathsf{GEO}(x_t; x_{<t}, \boldsymbol{p}_{1:t})$ and applied $\ln \frac{1}{z} \geqslant 1 - z$, to obtain the last inequality. □

Since all requirements are met, we may now conclude code length bounds.

Corollary 4.6.7 *A Geometric OGDM satisfies the code length bounds of Table 4.4, given the listed step size choices and requirements.*

Proof. For the proof we combine Corollary 4.5.5 Lemma 4.6.6. Note that in either case the step size sequence is non-increasing.

4.7 Experiments

We now complement the theoretic results from the previous sections with an experimental study. Our main goal is to judge on the tightness of our code length bounds for a Linear- and a Geometric OGDM w. r. t. (optimal) SWMs. In our experiments we consider bit sequences and all step size choices proposed in the previous sections (see Table 4.3 and Table 4.4). Table 4.5 summarizes all mixers of consideration. For the mixers LIN3 and GEO3 we have chosen a step size that does not depend on the sequence length, in both cases this corresponds to the choice of $\delta = \frac{1}{10}$. For an OGDM with fixed step size choice we operate as follows: First, we measure the redundancy in a worst-case setting (since all of our bounds are worst-case bounds), then we compare our measurements to the corresponding redundancy bounds (implied a code length bound). As a bonus, we may compare the measured worst-case redundancy for Linear- and Geometric OGDMs to each other.

The present experiments are only based on artificial data. We will report on experiments on real-world data in Chapter 5, where we study modeling and mixing within a CTW-based model in its entirety. Our motivation for this way matches that of the experiments on elementary modeling in Chapter 3: In a statistical data compressor mixing (just as elementary modeling) on its own will not be fully decisive for the empirical performance on real-world data, rather the interaction of mixing and elementary modeling determines the compression performance. Hence, experiments on real-world data should consider a statistical data compressor as a whole, not only its single components.

In the following we first describe the experimental setup in Section 4.7.1 and provide experimental results and a discussion in Section 4.7.2.

4.7.1 Experimental Setup

The Worst-Case. As we already noted, the code length bounds we gave in Section 4.6 are worst-case bounds, since they hold for arbitrary sequences $x_{1:n}$ and arbitrary sequences $\boldsymbol{p}_{1:n}$ of p-vectors, whose distributions have probability at least $\varepsilon > 0$ on any letter. Hence, we should evaluate our results in a worst-case setting. To do so, we consider the worst-case redundancy,

$$r(n) := \max_{x_{1:n}, \boldsymbol{p}_{1:n}} \Big(\ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n}) - (1+\delta) \cdot \min_{\mathsf{SW}} \ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}) \Big), \quad (4.31)$$

for $n \in 1{:}T$. For a meaningful graphical representation of r as a function of n, we must restrict our attention to SWMs that have a fixed partition $\mathcal{P}$ of $1{:}T$, rather than allowing for arbitrary SWMs. Hence, from now on let $\mathcal{P}$ be fixed (see (4.34) for our final choice of $\mathcal{P}$). Moreover, we only consider inputs $x_{1:n}, \boldsymbol{p}_{1:n}$ where for a SWM it is worthwhile to switch back and forth between the components of its mixer input $\boldsymbol{p}_{1:n}$, in order to minimize the code length of $x_{1:n}$. Therefore it will be useful to introduce the notation of generating parameters: We say a partition $\mathcal{P}$ and distributions $p_1, p_2, \dots, p_{|\mathcal{P}|}$ *generate* an input $x_{1:n}, \boldsymbol{p}_{1:n}$, if for the i-th segment $a{:}b \in \mathcal{P}$ every letter within $x_{a:b}$ is drawn at random according to p_i and $\boldsymbol{p}_{1:n} = \langle (p_1, p_2, \dots, p_{|\mathcal{P}|})^{\mathsf{T}} \rangle_{1 \leqslant t \leqslant n}$ (the components do not vary over time).

A direct computation of (4.31) is intractable, since there are exponentially many sequences $x_{1:t}$ and, even worse, there is an uncountably infinite

Table 4.5: Parameter configuration, redundancy bounds w. r. t. SWMs whose partition has s segments within $1{:}n$ (there are $s-1$ switches in $1{:}n$) for all mixers we consider in our experiments. The bounds LIN1, LIN2, LIN3 correspond to the redundancy bounds in rows 1, 2 and 3 of Table 4.3, similarly for GEO1 to GEO3 and Table 4.4. In either case we have $m=4$ and set $\boldsymbol{w}_1 = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})^\mathsf{T}$.

Mixer	Parameters	Redundancy Bound
LIN1	$\varepsilon = \frac{1}{10}, \alpha = \frac{1}{\sqrt{n}}$	$(s + 200\log^2 e)\cdot\sqrt{n}$
LIN2	$\varepsilon = \frac{1}{10}, \alpha_t = \frac{1}{\sqrt{t}}$	$(s + 400\log^2 e)\cdot\sqrt{n}$
LIN3	$\varepsilon_t = \frac{1}{10}, \alpha = \frac{\log 10}{2200\log^2 e}, \delta = \frac{1}{10}$	$\frac{2420\log^2 e}{\log 10}\cdot s$
GEO1	$\varepsilon = \frac{1}{10}, \alpha = \frac{1}{\sqrt{n}}$	$(s + 2\log^2 10)\cdot\sqrt{n}$
GEO2	$\varepsilon = \frac{1}{10}, \alpha_t = \frac{1}{\sqrt{t}}$	$(s + 4\log^2 10)\cdot\sqrt{n}$
GEO3	$\varepsilon_t = \frac{1}{10}, \alpha = \frac{\log e}{22\log^2 10}, \delta = \frac{1}{10}$	$\frac{24.2\log^2 10}{\log e}\cdot s$

number of p-vectors to consider. Hence, we require an approximation, which we explain below.

A Single Input. Suppose that we are given a partition $\mathcal{P}$, the distributions $p_1, p_2, \ldots, p_{|\mathcal{P}|}$ and a generated input $x_{1:n}, \boldsymbol{p}_{1:n}$. We now explain how we approximate the innermost term in (4.31),

$$\ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n}) - (1+\delta)\cdot\min_{\mathsf{SW}} \ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n}), \tag{4.32}$$

in the current situation. It is obvious that a good approximation to the SWM that minimizes the code length for $x_{1:n}$ will predict the distribution p_i for any letter within the i-th segment of $\mathcal{P}$. More formally, $\mathsf{SW} = \mathsf{SW}'$, for $\mathsf{SW}'\langle\mathcal{P}, \{m_s\}_{s\in\mathcal{P}}\rangle$, where $m_s = i$ for the i-th segment s of $\mathcal{P}$, will serve as the approximate minimizer of $\ell(x_{1:n}; \mathsf{SW}, \boldsymbol{p}_{1:n})$. For $n \in 1{:}T$ we may now approximate (4.32) as

$$\ell(x_{1:n}; \mathsf{MIX}, \boldsymbol{p}_{1:n}) - (1+\delta)\cdot\ell(x_{1:n}; \mathsf{SW}', \boldsymbol{p}_{1:n}). \tag{4.33}$$

All Inputs. It remains to generalize our approximation to the set of all possible inputs. To approximately recover (4.31), we should ideally take the maximum of (4.33) over any input $x_{1:n}, \boldsymbol{p}_{1:n}$ that may be generated by $\mathcal{P}$, p_1, p_2, …, $p_{|\mathcal{P}|}$, for all distributions $p_1, p_2, \ldots, p_{|\mathcal{P}|}$ and $n \in 1{:}T$. As an approximate solution, we take the maximum over a finite subset of inputs samples across the whole input space. We generate the subset of inputs we consider as follows: For every $p_1, p_2, \ldots, p_{|\mathcal{P}|} \in \{p \mid p(0) \in \{\varepsilon, 2\varepsilon, \ldots, 1\}\}$ and $n \in \{1, 50, 100, \ldots, T\}$ we generate R inputs. Our parameter setup is

$$\begin{aligned} &\mathcal{P} = \{1{:}500, 501{:}1500, 1501{:}3500, 3501{:}T\},\\ &T = 5000,\\ &R = 50 \quad \text{and}\\ &\varepsilon = 0.1. \end{aligned} \tag{4.34}$$

By the choice of parameters we take the maximum in (4.31) over $R/\varepsilon^{|\mathcal{P}|} = 500{,}000$ inputs of length n for every $n \in \{1, 50, 100, \ldots, T\}$.

4.7.2 Results

Evaluation — Tightness of Bounds for LIN and GEO. The overall impression of a peek at Figure 4.1 may be summarized as follows: First of all, the bounds are rather loose (notice the logarithmic scale), they exceed the measured worst-case redundancy roughly by a factor of 100 for either LIN-variant and by a factor of about 10 for all GEO-variants. This implies that the analysis provides tighter bounds in the case of GEO. In either case our bounds are more useful in the asymptotic regime and do not provide a tight estimate on the redundancy for relatively short inputs. Furthermore, the redundancy bounds for LIN1, LIN2, GEO1 and GEO2 do not increase notably at segment boundaries (although such a transition occurs). The reason is simple: The number of segments is drastically smaller then the sequence length (e. g. see the bound for LIN1 in Table 4.5, the bound is about $(s + 200\log^2 e) \cdot \sqrt{n}$, so the constant term in brackets dominates, since $s \leqslant 4$, but $200\log^2 e \approx 416.3$); in case of LIN3 and GEO3 these transitions are more pronounced. Surprisingly, for LIN3 and GEO3 the measured worst-case redundancy *decreases* after a sufficient distance to a transition, whereas our bound is non-decreasing (also see Figure 4.2 for a plot in linear scale). We further discuss this effect below.

Evaluation — Worst-Case Redundancy across LIN and GEO. We now shift our attention towards the measured worst-case redundancy. Figure 4.2 depicts our worst-case redundancy measurements (no log-scale).

We first consider LIN1 and GEO1, where the step size is fixed at $\alpha = \frac{1}{\sqrt{n}}$. In either case the measured worst-case redundancy increases steadily and we observe small jumps at segment boundaries. For GEO1 the rate of increase and the magnitude of redundancy jumps is lower, compared to LIN1.

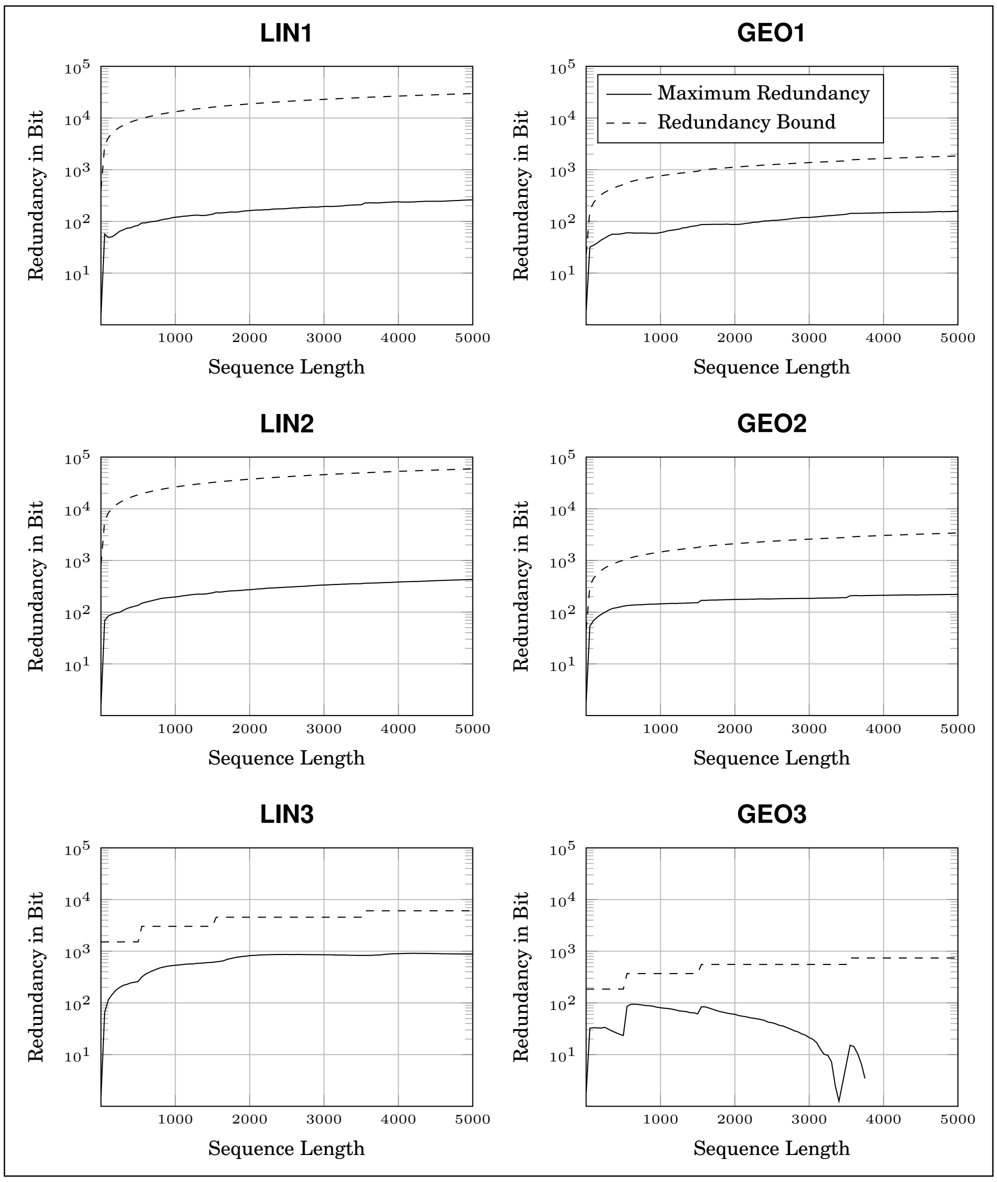


Figure 4.1: Redundancy bounds and approximate worst-case redundancy $r(n)$ for mixers from Table 4.5. The measured worst-case redundancy for GEO3 is discontinued, since it drops below 0.

With increasing sequence length this advantage adds up. For a sequence of length 5000 LIN1 has a worst-case redundancy $66\,\%$ *above* that of GEO1.

Let us now step towards LIN2 and GEO2, where the step size slowly decreases, that is $\alpha_t = \frac{1}{\sqrt{t}}$. For GEO2, to a more limited amount also for LIN2, the measured worst-case redundancy steadily grows, at a decreasing

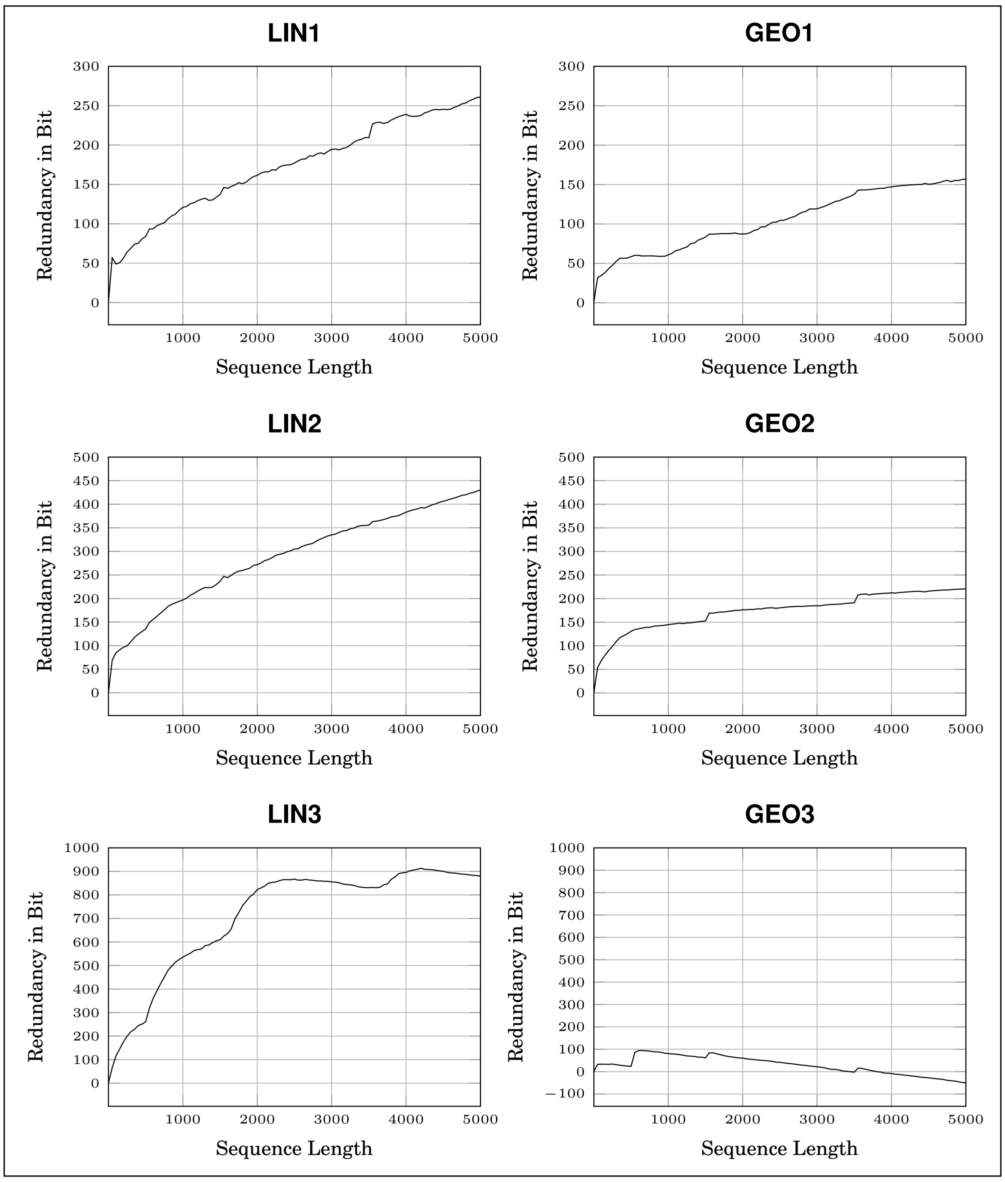


Figure 4.2: Approximate worst-case redundancy $r(n)$ for mixers from Table 4.5.

rate; furthermore, the segment boundaries only induce a very small abrupt increase in redundancy, when the sequence is long enough, see the plots for $n = 1500$ and $n = 3500$, the transition at $n = 500$ is barely visible. We believe that the varying step size causes this pattern: When the step size is large (the sequence length is small), then it is easier to track perturbations in the input – so the redundancy within a segment of $\mathcal{P}$ increases quickly –, but also it is easier to track changing statistics – so the redundancy across a segment boundary is rather low. The converse holds for small step sizes (the sequence length is large), i. e. it is harder to track perturbations in the input – the redundancy within a segment increases slowly – and it is harder to track changing statistics – the redundancy across a segment boundary may be high. Again, LIN2 accumulates significantly more redundancy than GEO2, nearly twice as much, for a sequence length of 5000.

It remains to examine the plots for LIN3 and GEO3, where the step size is fixed and independent of n. LIN3 and GEO3 show a surprising behavior: Right after a transition the redundancy increases, as one would expect, then, the rate of increase slows down and reverts – *the worst-case redundancy decreases*. This effect is far more pronounced for GEO3, the measured worst-case redundancy even *drops below* 0. We retraced this effect to the conservative estimation of probabilities for bits that appear rarely. To explain this behavior we consider the competitor SW' and GEO3 on a single input $x_{1:n}$, $\boldsymbol{p}_{1:n}$ generated by $\mathcal{P}, p_1, p_2, p_3, p_4$ (for $\mathcal{P}$ see (4.34)), where

$$p_1(0) = 0.7,\ p_2(0) = 0.1,\ p_3(0) = 0.5 \text{ and } p_4(0) = 0.9.$$

(For the construction of the input see "A Single Input" in Section 4.7.1.) In Figure 4.3 we depict the per-bit redundancy

$$\ell(x_t; \mathsf{GEO}(x_{<t}, \boldsymbol{p}_{1:t})) - (1+\delta) \cdot \ell(x_t; \mathsf{SW}'(x_{<t}, \boldsymbol{p}_{1:t})). \tag{4.35}$$

For instance, consider the segment $501{:}1500$. A 0-bit appears rarely, since all bits are drawn according to p_2 (we will roughly have $10\,\%$ of 0-bits). By inspecting Figure 4.3 we see that the per-bit redundancy for 1-bits is above zero, so GEO3 accumulates redundancy, if we observe a frequent bit (1-bit). Contrary, the per-bit redundancy for 0-bits is negative and has a relatively high absolute value, hence GEO3 improves over the competing SWM, if we observe an infrequent bit (0-bit). In order to achieve such redundancy characteristics, GEO3 must overestimate the probability of infrequent bits (these have negative redundancy) and underestimate the probability of frequent bits (these have positive redundancy). For LIN3 we observe the same effects, only the figures change (so we do not provide a plot): If we compare the per-bit redundancy of LIN3 to that of GEO3, we see that frequent bits receive even longer codewords and infrequent bits receive even shorter codewords. So LIN3 does more extreme probability over- and underestimation. Since this effect is neither specific for LIN3 or GEO3, we believe that it is caused by a similarity LIN3 and GEO3 share, that is, OGD for weight estimation.

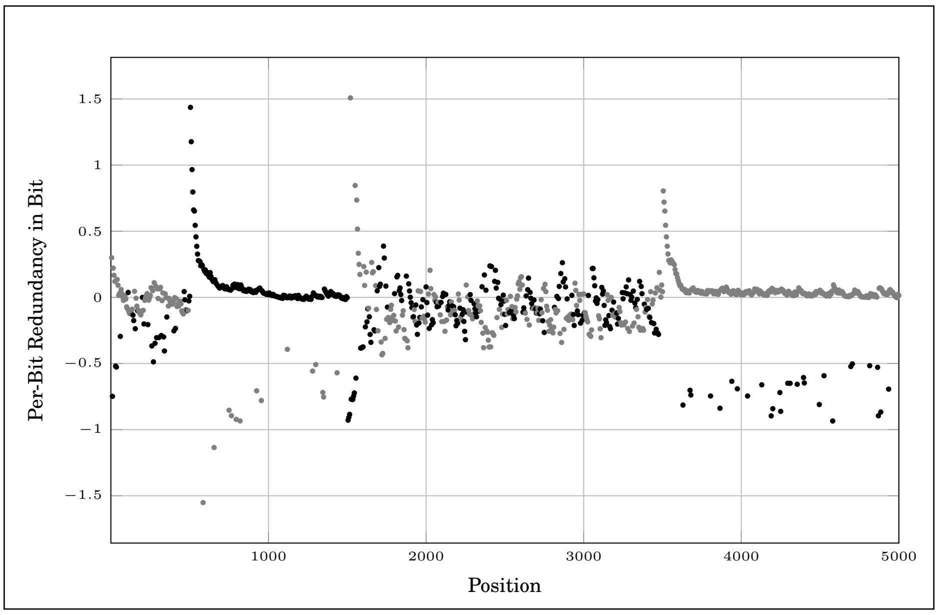


Figure 4.3: Measured per-bit redundancy (4.35) for GEO3. A gray dot represents the redundancy for a 0-bit, a black dot represents the redundancy for a 1-bit.

4.8 Summary

In addition to elementary modeling, mixing is another fundamental building block in statistical data compression. Viewed from a general perspective, the problem of mixing reduces to other more general problems in other fields of science, for instance Prediction with Expert Advice or Online Convex Programming. In the field of statistical data compression there are few approaches to mixing that meet practical running time and space constraints. Out of these especially PAQ's LM stands out, since it provides good empirical performance and allows for a low-complexity implementation. Unlike other approaches proposed by theoreticians, such as the Switching Method and Beta-Weighting, LM lacks a theoretical basis to support its good empirical performance.

As we have seen in this chapter, we may iron out the the lack of knowledge on LM. For this purpose we have realized three things (see Section 4.4.2 and Section 4.6.2): First, the distribution LM computes is a special case of a more general type of mixture distribution, that is, a Geometric Mixture Distribution; Second, the Geometric Mixture Distribution may be derived from an optimization problem that attempts to identify a data-generating source distribution in the probabilistic view; Third, by adopting analysis techniques from Online Convex Programming we obtain code length guarantees for a mixer based on a Geometric Mixture Distribution with weights estimated by OGD. In our theoretic results an arbitrary SWM serves as competitor. Moreover, with a slightly different idea, we may also motivate the Linear Mixture Distribution (Section 4.4.3) by an optimization problem in the probabilistic view and provide code length guarantees (Section 4.6.1). Both, the analysis of a Linear OGDM and a Geometric OGDM, is based upon a more general analysis technique that applies to a wide class of mixture distributions (see Section 4.5) and allows for code length bounds w. r. t. an arbitrary competing SWM.

We finally support our theoretic results with an experimental study. Our results indicate that the code length bounds we obtain are rather loose, so they only allow to judge on the asymptotics of code length and redundancy. (Bounds for Linear OGDM seem to be more loose than those for Geometric OGDM.) In every experiment we observed that Geometric OGDM has a lower worst-case redundancy w. r. t. a fixed SWM than Linear OGDM, which suggests that Geometric OGDM should be preferred over Linear OGDM. In the next chapter we will collect even more empirical evidence for this conjecture on non-synthetic data.

CHAPTER 5

Context Tree Mixing

CHAPTER OVERVIEW

In this chapter we introduce Context Tree Mixing (CTM), a generalization of CTW. Unlike CTW, CTM allows for arbitrary elementary models and mixers. In a code length analysis we show that if these models and mixers are drawn from a sufficiently powerful class, then CTM has low redundancy w. r. t. an arbitrary sequence of PCTs (not just w. r. t. a single PCT, as in case of CTW). We further equip CTM with PAQ-style elementary modeling and mixing to obtain the PAQ CTM statistical compressor. We show that this new algorithm satisfies the aforementioned code length guarantee. In addition to theoretical guarantees an experimental study shows that PAQ CTM outperforms traditional CTW.

5.1 Motivation

In statistical data compression practitioners aim for highest compression rates on real world data. Astonishingly enough, we may support popular approaches from the "practitioner's claim" for elementary modeling and mixing with powerful code length guarantees. As we saw in Chapter 3 and Chapter 4, these guarantees imply low redundancy w. r. t. a competitor that may adapt to the input. However, the result is still not quite satisfactory — can a statistical data compression algorithm also be supported by good code length guarantees w. r. t. an adaptive competitor? In this chapter we will investigate this question. (It turns out that we may answer this question in the affirmative.)

We attack the problem by simple means, that is, we combine CTW, the theoretician's approach to statistical data compression, with mixing and elementary modeling techniques that support adaptivity by code length guarantees. We may use relevant elementary models and mixers developed by practitioners, hence our procedure is a further step towards bridging the gap between theory and practice. Typically, the "practitioner's approaches" enjoy good empirical performance, so we may not only hope for powerful theoretic code length guarantees, but also for improvements in compression on real world data over CTW.

Let us step back to CTW for a second to give a more precise outline of our idea. In Chapter 2 we have seen that CTW computes a recursively defined mixture distribution. The way CTW obtains this prediction can be stated in two equivalent ways, either as a block probability recursive mixture or as a sequential recursive mixture. The former variant is well known, but makes it hard to cleanly spot the components “elementary model” and “mixer”, as defined in Section 2.2, whereas the latter method is less well known, but does not suffer from this deficit. We will exclusively rely on the sequential recursive mixture, since it allows us to easily identify and replace elementary models and mixers, just as intended.

5.2 Our Contribution

In this chapter we continue to narrow the gap between theory and practice. This time we do not just study a single component of a model, but the whole model. The results presented in this chapter have not been published previously. Our contributions are the following.

Section 5.3 We propose and analyze Context Tree Mixing (CTM), a generalization of CTW. Structurally CTM and CTW are the same, but CTM is more flexible, since it may employ arbitrary elementary models and mixers. In Section 5.3.1 we describe and discuss CTM and provide pseudocode. We introduce a class of models and mixers by assumptions on their code length guarantees. Based on these assumptions we provide a general code length analysis in Section 5.3.2. In the analysis we compare the code length of CTM with that of an arbitrary sequence of competing PCTs. No former result covered such a rich class of competitors. In Section 5.3.3 we demonstrate that our general result is powerful enough to obtain several classical results quite easily, e. g. [8, 101] (also see Section 2.4.3).

Section 5.4 In this section we couple CTM with PAQ approaches to elementary modeling and mixing and conclude code length guarantees. For this purpose Section 5.4.1 introduces a variation of PS for elementary modeling. Taking this and Geometric OGDM as a basis we introduce PAQ CTM. In Section 5.4.2 we first verify the technical prerequisites for the application of the results on CTM (from Section 5.3). Finally we apply the analysis techniques from Section 5.3 to derive a code length guarantee that compares PAQ CTM and a sequence of competing PCTs. Our results show that it is possible to (asymptotically) perform not much worse than this powerful class of competitors using practical approaches.

Section 5.5 We provide experimental results on CTM variants, including PAQ CTM, in a realistic setting. In Section 5.5.1 we first explain and justify which elementary models and mixers we consider in our experiments. We furthermore introduce the data set for experiments and we discuss implementation details. Section 5.5.2 provides experimental results and a discussion.

5.3 Context Tree Mixing

We now describe, discuss and analyze Context Tree Mixing (CTM). In Section 5.3.1 we describe the CTM model, discuss a typical implementation and consider the code length guarantees that spans the classes of elementary models and mixers applicable for a code length analysis. Subsequently, we provide a code length analysis in Section 5.3.2 and discuss the results. Our analysis machinery allows us to easily restore classical results on CTW for a binary and non-binary alphabet, as we show in Section 5.3.3.

5.3.1 The Setting

Algorithm CTM. Matching the sequential recursive formalization of CTW (see "Sequential Recursive CTW Mixture" in Section 2.4.3), we may state the formal definition of CTM in a very compact way (as usual, c denotes contexts):

Definition 5.3.1 *For an integer $D \geqslant 0$, a family $\{\mathsf{MDL}^c\}_{|c|\leqslant D}$ of elementary models and a family $\{\mathsf{MIX}^c\}_{|c|<D}$ of mixers, the* Context Tree Mixing *model is defined by $\mathsf{CTM}\langle D, \{\mathsf{MDL}^c\}_{|c|\leqslant D}, \{\mathsf{MIX}^c\}_{|c|<D}\rangle$. The prediction $\mathsf{CTM}(x_{<t})$ is given by the probability assignment $\mathsf{CTM} = \mathsf{CTM}_0$, defined recursively,*

$$\mathsf{CTM}_d(x_{<t}) := \begin{cases} \mathsf{MDL}^c(x^c_{<t}), & \text{if } d = D, \\ \mathsf{MIX}^c(x^c_{<t}, \mathrm{in}^c(x_{<t})), & \text{if } 0 \leqslant d < D, \end{cases} \tag{5.1}$$

where $c = x_{t-d:t-1}$ is the length-d context of x_t and the mixer input is

$$\mathrm{in}^c(x_{<t}) := \langle(\mathsf{MDL}^c(x^c_{<i}), \mathsf{CTM}_{d+1}(x_{<i})\rangle_{i\in\mathcal{T}_c(x_{1:t})}. \tag{5.2}$$

The above definition might be somewhat overwhelming, so let us discuss the structure of CTM and the way it operates in detail. Any context c of length at most D owns an elementary model MDL^c to map the context history to a prediction. In addition, every context c of length less than D has a mixer MIX^c which combines the prediction of MDL^c and that of child contexts xc (where x is a letter) of c. (One may think of models and mixer tied to contexts in a context tree with N^d contexts on levels $d = 0, 1, \dots, D$. However, we want to avoid this view, so that the reader does not mix up the context tree of a competing PCT with CTM's context tree.) By examining (5.1) and (2.30) we see that the CTM operates just like CTW, but replaces the KT elementary model with an arbitrary elementary model. Similarly, Beta-Weighting is replaced by an arbitrary mixer.

We now describe how CTM predicts the distribution $\mathsf{CTM}_d(x_{<t})$ on the t-th letter, given the sequence $x_{<t}$ (this suffices to understand the overall operation, since $\mathsf{CTM} = \mathsf{CTM}_0$.) Let c be the current length-d context of x_t with context history $x^c_{<t} = y_{1:k}$ (context c occurred k times in the past). If $|c| = D$, we are done by setting $\mathsf{CTM}_d(x_{<t}) = \mathsf{MDL}^c(y_{1:k})$. If $|c| < D$, there is more work to do. First, the elementary model predicts $u = \mathsf{MDL}^c(y_{1:k})$ and we recursively obtain the prediction $v = \mathsf{CTM}_{d+1}(x_{<t})$. Next, the mixer MIX^c combines these predictions by considering the mixer input $\text{in}^c(x_{<t}) = \boldsymbol{p}_{1:k+1}$, where $\boldsymbol{p}_{k+1} = (u, v)^\mathsf{T}$, so we are done by setting $\mathsf{CTM}_d(x_{<t}) = \mathsf{MIX}^c(y_{1:k}, \boldsymbol{p}_{1:k+1})$.

An Implementation. As we already explained in Section 2.4.3 (see Figure 2.4), we may unravel the recursion to obtain CTM_{d+1} when computing CTM_d. Figure 5.1 shows a non-recursive implementation. For the pseudocode we made the following natural assumptions:

- The implementation `mdl` of an elementary model MDL has the functions `Init()`, `Predict()` and `Update(`x`)`, where for any sequence $x_{1:\infty}$ the operations

  $$\texttt{mdl.Init()},\ p_1 \leftarrow \texttt{mdl.Predict()},\ \texttt{mdl.Update}(x_1),\\ p_2 \leftarrow \texttt{mdl.Predict()},\ \texttt{mdl.Update}(x_2), \dots$$

  produce the distributions $p_1, p_2, \dots$ s. t. $p_t = \mathsf{MDL}(x_{<t})$, for $t \geqslant 1$.

- Similarly, the implementation `mix` of a mixer MIX supports the functions `Init()`, `Mix(`u, v`)` and `Update(`u, v, x`)`. For any sequence $x_{1:\infty}$ and any sequence $\boldsymbol{p}_{1:\infty} = (u_t, v_t)^\mathsf{T}_{t\geqslant 1}$ of p-vectors, the operations

  $$\texttt{mix.Init()},\ p_1 \leftarrow \texttt{mix.Predict}(u_1, v_1),\ \texttt{mix.Update}(x_1, u_1, v_1),\\ p_2 \leftarrow \texttt{mix.Predict}(u_2, v_2),\ \texttt{mix.Update}(x_2, u_2, v_2), \dots$$

  produce the distributions $p_1, p_2, \dots$ s. t. $p_t = \mathsf{MIX}(x_{<t}, \boldsymbol{p}_{1:t})$, for $t \geqslant 1$.

```
Input   : Segment $x_{-D+1:0}$; uncompressed data, when encoding, or compressed
          data, when decoding.
Output  : Compressed data, when encoding, or decompressed data, when
          decoding.
Remark  : ctxMap maps contexts (strings) to tuples of elementary models and
          mixers, the arrays p$[0..D]$ and q$[0..D-1]$ hold distributions and the
          array c$[0..D-1]$ holds current context.

c[0..D-1] $\leftarrow x_{-D+1:0}$;
while not end-of-input do
    // Retrieve elementary models and mixers, c[$0..-1$] represents the empty
       context $\phi$
    for d $= 0, 1, \ldots, D$ do
        (mdl[d],mix[d]) $\leftarrow$ ctxMap.Lookup(c[$0..$d$-1$]);
        if mdl[d] = null then
            (mdl[d],mix[d]) $\leftarrow$ create new elementary model and mixer;
            mdl[d].Init(), mix[d].Init();
            ctxMap.Insert(c[$0..$d$-1$],(mdl[d],mix[d]));
        end
    end
    // Compute CTM$_0$ and store in p$[0]$ (tail recursion resolved)
    p[$D$] $\leftarrow$ mdl[$D$].Predict();
    for d $= D-1, \ldots, 0$ do
        q[d] $\leftarrow$ mdl[d].Predict();
        p[d] $\leftarrow$ mix[d].Predict(p[d$+1$],q[d]);
    end
    // Encode or decode a letter
    Read letter $x$ from input and encode $x$ using prediction p[0] or
    decode letter $x$ using prediction p[0] and write $x$ to output;
    // Update context, elementary models and mixers.
    c[$2..D$] $\leftarrow$ c[$1..D-1$],c[1] $\leftarrow x$;
    mdl[$D$].Update($x$);
    for d $= D-1, \ldots, 0$ do
        mdl[d].Update($x$);
        mix[d].Update($x$,p[d$+1$],q[d]);
    end
end
```

Figure 5.1: Pseudocode for compression and decompression using CTM.

The Class of Models and Mixers. Our upcoming analysis of CTM is based on assumptions on the redundancy of elementary models and mixers.

Before we give a formal specification of these assumptions, let us explain the general idea based on an elementary model. To do so, consider an elementary model MDL and an arbitrary segment $y_{i:j}$ of some sequence $y_{1:m}$. We assume that we may bound the code length the elementary model MDL assigns to the segment $y_{i:j}$ by

$$\ell(y_{i:j}; \mathsf{MDL}) \leqslant \alpha \cdot h(y_{i:j}) + f(y_{1:j}), \tag{5.3}$$

where the function f maps sequences to positive reals and $\alpha \geqslant 1$. Such a bound acts as a template, hence, we will leave α and f unspecified in the first place. Choosing a concrete elementary model allows us to specify α and f. (We take on that task in Section 5.4.) For instance, the code length PS assigns to a sequence may be bounded just as above (see (3.27) and Lemma 3.5.7). Moreover, in CTM we use context models MDL^c and we require a bound like (5.3) for every context c CTM maintains. Consequently, the sequence $y_{1:m}$ corresponds to a context history $x^c_{1:n}$ and the segment $y_{i:j}$ corresponds to a context history segment $x^c_{a:b}$ induced by some segment $x_{a:b}$ of the input. We allow the additive function f to depend on the context, hence we replace f with f^c. All of the above thoughts translate to mixers, as well.

Throughout our analysis in Section 5.3.2 we assume the following:

Assumption 5.3.2 *Fix a set $\mathcal{S} \subseteq \{a{:}b \mid 1 \leqslant a \leqslant b\}$ of segments and a CTM instance $\mathsf{CTM}\langle D, \{\mathsf{MDL}^c\}_{|c|\leqslant D}, \{\mathsf{MIX}^c\}_{|c|<D}\rangle$. For all sequences $x_{1:n}$ and all segments $a{:}b \in \mathcal{S}$, where $b \leqslant n$, there exist constants $\alpha, \beta \geqslant 1$ s. t.:*

(a) *Let c be a context of length $|c| \leqslant D$ and let $y_{i:j} = x^c_{a:b}$ be the segment of context history $y_{1:m} = x^c_{1:n}$, induced by $a{:}b$. The model MDL^c satisfies*

$$\ell(y_{i:j}; \mathsf{MDL}^c) \leqslant \alpha \cdot h(y_{i:j}) + f^c(y_{1:j}),$$

where the function f^c maps a sequence to positive reals.

(b) *Let c be a context of length $|c| < D$, let $y_{i:j} = x^c_{a:b}$ be the segment of context history $y_{1:m} = x^c_{1:n}$ and let $\boldsymbol{p}_{1:j}$ be the segment of mixer input $\mathrm{in}^c(x_{1:n}) = \boldsymbol{p}_{1:m} = \langle (u_t, v_t)^\mathsf{T} \rangle_{1\leqslant t\leqslant m}$, both induced by $a{:}b$. The mixer MIX^c satisfies*

$$\ell(y_{i:j}; \mathsf{MIX}^c, \boldsymbol{p}_{1:j}) \leqslant \beta \cdot \ell(y_{i:j}; p_{i:j}) + g^c(y_{1:j}, \boldsymbol{p}_{1:j}), \text{ for } p_{i:j} \in \{u_{i:j}, v_{i:j}\},$$

where the function g^c maps a string and a sequence of 2-dimensional p-vectors to positive reals.

Note that we must specify a segment $y_{i:j}$ of $y_{1:m}$ in order to see whether or not Assumption 5.3.2 is satisfied. If we carefully examine previous elementary models (and mixers) and check them against Assumption 5.3.2, we see that for arbitrary segments $y_{i:j}$ it may not always hold. For instance, consider the elementary model KT, which guarantees

$$\ell(y_{1:j}; \mathsf{KT}) \leqslant h(y_{1:j}) + \frac{N-1}{2}\log n + \log N,$$

so Assumption 5.3.2 is *only* satisfied when $i = 1$ (and we may read off $\alpha = 1$ and $f^c(y_{1:j}) = \frac{N-1}{2}\log j + \log N$). (In Section 5.4 we will propose variations of PS and GEO for our purposes which satisfy Assumption 5.3.2 for arbitrary segments $y_{i:j}$.)

Discussion. To capitalize on Assumption 5.3.2 we must understand its implications on models and mixers tied to some context c in CTM's context tree.

In the course of applying CTM on an input sequence $x_{1:n}$ we observe the context history $x^c_{1:n} = y_{1:m}$ at context c. Consequently, a segment $x_{a:b}$ of $x_{1:n}$ induces a context history segment $x^c_{a:b} = y_{i:j}$ at context c. Assumption 5.3.2 (a) says that the code length a model assigns to a *context history segment* $x^c_{a:b}$ is close to the empirical entropy of this particular segment. We express this proximity in code length by a multiplicative factor α and an additive redundancy term $f^c(y_{1:j})$.

Similarly, at the context c we observe the mixer input $\mathrm{in}^c(x_{1:n}) = \boldsymbol{p}_{1:m}$ and a corresponding segment $\boldsymbol{p}_{i:j} = \langle (u_t, v_t)\rangle_{i\leqslant t\leqslant j}$, induced by the segment $x_{a:b}$ of $x_{1:n}$. Assumption 5.3.2 (b) says that the code length a mixer assigns to a *context history segment* will be close to the code lengths $\ell(x^c_{a:b}; u_{i:j})$ or $\ell(x^c_{a:b}; v_{i:j})$. Consequently, the mixer's coding performance *simultaneously* is close to the coding performance of the elementary model at context c (the predictions $u_{i:j}$) and to that of child contexts of c (the predictions $v_{i:j}$) for the particular segment $x_{a:b}$.

More Notation. We now define some abbreviations for important redundancy and code length expressions.

Definition 5.3.3 *Fix a context c of length $|c| \leqslant D$, the input sequence $x_{1:n}$ and a segment $a{:}b$ of $1{:}n$. The context history at c is $x^c_{1:n} = y_{1:m}$ and the segment $x_{a:b}$ induce a context history segment $x^c_{a:b} = y_{i:j}$; if $|c| < D$, then we observed the mixer input $\mathrm{in}^c(x_{1:b}) = \boldsymbol{p}_{1:j}$ up until time b at context c. Based on this we define (see Assumption 5.3.2 for f^c and g^c)*

$$\ell^c_{\mathsf{CTM}}(a{:}b) := \sum_{t\in\mathcal{T}_c(x_{a:b})} \ell(x_t; \mathsf{CTM}_{|c|}(x_{<t})),\ r^c_{\mathrm{mdl}}(a{:}b) := f^c(y_{1:j}),\ r^c_{\mathrm{mix}}(a{:}b) := g^c(y_{1:j}, \boldsymbol{p}_{1:j}).$$

Table 5.1: Notation used for the code length analysis of CTM in Section 5.3.2.

Symbol	Definition	Symbol	Definition
D	maximum context length	$\ell^c_{\mathsf{CTM}}(a{:}b)$	code len. of $x^c_{a:b}$ for model $\mathsf{CTM}_{\lvert c\rvert}$
$\mathcal{P}, s$	partition of $1{:}n$, segment from $\mathcal{P}$	$r^c_{\mathrm{mdl}}(a{:}b)$	red. of MDL^c for $x^c_{a:b}$ w. r. t. h
$\alpha, \beta, f^c, g^c, \mathcal{S}$	see Asm. 5.3.2	$r^c_{\mathrm{mix}}(a{:}b)$	red. of MIX^c for $x^c_{a:b}$ w. r. t. SWM
$\mathsf{MDL}^c, \mathsf{MIX}^c$	model, mixer tied to context c	PCT (PCT_s)	PCT (associated to segment s)
$h(x_{1:n})$	$\sum_{x\in\mathcal{X}} c(x)\log({}^{n}/{}_{c(x)})$	$\mathcal{C}_L$ ($\mathcal{C}_{L,s}$)	leaf context set of PCT (PCT_s)
$\mathcal{T}_c(x_{a:b})$	all $t \in a{:}b$ s. t. x_t has context c	$\mathcal{C}_I$ ($\mathcal{C}_{I,s}$)	non-leaf context set of PCT (PCT_s)
$x^c_{a:b}$	context history seg. $\langle x_t\rangle_{t\in\mathcal{T}_c(x_{a:b})}$	$\mathcal{C}$ ($\mathcal{C}_s$)	set of all contexts of PCT (PCT_s)

Informally, the code length $\ell^c_{\mathsf{CTM}}(a{:}b)$ accounts for the number of bits we would have to pay if we encoded the context history $x^c_{a:b}$ based on the distribution $\mathsf{CTM}_{\lvert c\rvert}$ (see Definition 5.3.1). The terms r^c_{mdl} and r^c_{mix} capture the redundancy of modeling and mixing in the sense of Assumption 5.3.2 for a context history segment $x^c_{a:b}$. Note that the functions f^c and g^c disappear from the discussion. These functions are encapsulated by the terms r^c_{mdl} and r^c_{mix}.

5.3.2 Analysis

The Plan. To obtain code length guarantees on CTM we proceed in two steps. First, we show that the code length $\ell(x_{a:b};\mathsf{CTM})$ some CTM instance assigns to an arbitrary segment $x_{a:b}$ of the input is close to an intermediate context tree model. This model is similar to a PCT. However, unlike a PCT, the intermediate model does not predict a fixed distribution per context, rather its prediction is defined in terms of CTM (this is why we call it "intermediate", it lies in between CTM and a PCT): It has an underlying context tree with leaf context set $\mathcal{C}_L$, however it predicts the distribution $\mathsf{CTM}_{\lvert c\rvert}(x_t; x_{<t})$, for the context $c \in \mathcal{C}_L$ of x_t, in the t-th step. We will explain the details right after the corresponding result (see Lemma 5.3.5 and the subsequent discussion). Second, we prove that this intermediate model assigns a code length to $x_{a:b}$ that is close to that of an arbitrary PCT.

The Effect of Modeling and Mixing. As we have seen in Chapter 3 the code length of a carefully chosen elementary model is close to the empirical entropy. Further, mixing brings the code length $\ell^c_{\mathsf{CTM}}(a{:}b)$ close to that of the elementary model at context c and, at the same time, close to the total code length $\sum_{x\in\mathcal{X}} \ell^{xc}_{\mathsf{CTM}}(a{:}b)$ of child contexts of c. Hence, we expect that $\ell^c_{\mathsf{CTM}}(a{:}b)$ simultaneously is close to the empirical entropy $h(x^c_{a:b})$ of the context history at c and to $\sum_{x\in\mathcal{X}} \ell^{xc}_{\mathsf{CTM}}(a{:}b)$. The following lemma confirms this anticipation. Table 5.1 summarizes our notation for Section 5.3.2.

Lemma 5.3.4 *For any context c and any segment $a{:}b \in \mathcal{S}$ it holds that:*

(a) *If $|c| = D$, then $\ell^c_{\mathsf{CTM}}(a{:}b) \leqslant \alpha \cdot h(x^c_{a:b}) + r^c_{\mathrm{mdl}}(a{:}b)$.*

(b) *If $|c| < D$, then $\ell^c_{\mathsf{CTM}}(a{:}b) \leqslant \beta \cdot \sum_{x\in\mathcal{X}} \ell^{xc}_{\mathsf{CTM}}(a{:}b) + r^c_{\mathrm{mix}}(a{:}b)$ and*

(c) $\ell^c_{\mathsf{CTM}}(a{:}b) \leqslant \alpha\beta \cdot h(x^c_{a:b}) + \beta \cdot r_{\mathrm{mdl}}(a{:}b) + r^c_{\mathrm{mix}}(a{:}b)$.

Proof. For brevity we omit the dependence of expressions like $\ell^c_{\mathsf{CTM}}(a{:}b)$ on $a{:}b$. We distinguish two cases:

Case 1: $|c| = D$. The segment $a{:}b$ induces a segment $x^c_{a:b} = y_{i:j}$ of the context history $x^c_{1:n} = y_{1:m}$. By Definition 5.3.1 we have $\mathsf{CTM}_{|c|}(x_{<t}) = \mathsf{MDL}^c(x^c_{<t})$, so

$$\begin{aligned}\ell^c_{\mathsf{CTM}} \overset{\text{D5.3.1}}{=} \sum_{t\in\mathcal{T}_c(x_{a:b})} \ell(x_t; \mathsf{MDL}^c(;x^c_{<t})) = \ell(y_{i:j}; \mathsf{MDL}^c) &\overset{\substack{\text{A5.3.2 (a),}\\ \text{D5.3.3}}}{\leqslant} \alpha \cdot h(y_{i:j}) + r^c_{\mathrm{mdl}} \\ &= \alpha \cdot h(x^c_{a:b}) + r^c_{\mathrm{mdl}}.\end{aligned}$$

Case 2: $|c| < D$. Context c owns a mixer with mixer input $\mathrm{in}^c(x_{1:n}) = \boldsymbol{p}_{1:m} = \langle (u_i, v_i)^{\mathsf{T}} \rangle_{1\leqslant i\leqslant m}$ s. t.

$$u_{1:m} = \langle \mathsf{MDL}^c(x^c_{<t}) \rangle_{t\in\mathcal{T}_c(x_{a:b})} \quad \text{and} \quad v_{1:m} = \langle \mathsf{CTM}_{|c|+1}(x_{<t}) \rangle_{t\in\mathcal{T}_c(x_{a:b})}. \tag{5.4}$$

The segment $a{:}b$ induces a segment $x^c_{a:b} = y_{i:j}$ of the context history $x^c_{1:n} = y_{1:m}$ and a segment $\boldsymbol{p}_{i:j}$ of the mixer input $\boldsymbol{p}_{1:m}$. By Definition 5.3.1 we have $\mathsf{CTM}_{|c|}(x_{<t}) = \mathsf{MIX}^c(x^c_{<t}, \mathrm{in}^c(x_{<t}))$, so

$$\ell^c_{\mathsf{CTM}} \overset{\text{D5.3.1}}{=} \sum_{t\in\mathcal{T}_c(x_{a:b})} \ell(x_t; \mathsf{MIX}^c(x^c_{<t}, \mathrm{in}^c(x_{<t})) = \ell(y_{i:j}; \mathsf{MIX}^c, \boldsymbol{p}_{1:j}), \tag{5.5}$$

$$\begin{aligned}\ell(y_{i:j}; \mathsf{MIX}^c, \boldsymbol{p}_{1:j}) &\overset{\substack{\text{A5.3.2 (b),}\\ \text{D5.3.3}}}{\leqslant} \beta \cdot \ell(y_{i:j}; v_{i:j}) + r^c_{\mathrm{mix}} \\ &\overset{(5.4)}{=} \beta \cdot \sum_{t\in\mathcal{T}_c(x_{a:b})} \ell(x_t; \mathsf{CTM}_{|c|+1}(x_{<t})) + r^c_{\mathrm{mix}} \\ &= \beta \cdot \sum_{y\in\mathcal{X}} \sum_{t\in\mathcal{T}_{yc}(x_{a:b})} \ell(x_t; \mathsf{CTM}_{|yc|}(x_{<t})) + r^c_{\mathrm{mix}} \\ &\overset{\text{D5.3.3}}{=} \beta \cdot \sum_{y\in\mathcal{X}} \ell^{yc}_{\mathsf{CTM}} + r^c_{\mathrm{mix}} \quad \text{and} \end{aligned} \tag{5.6}$$

$$\begin{aligned}\ell(y_{i:j}; \mathsf{MIX}^c, \boldsymbol{p}_{1:j}) &\overset{\substack{\text{A5.3.2 (b),}\\ \text{D5.3.3}}}{\leqslant} \beta \cdot \ell(y_{i:j}; u_{i:j}) + r^c_{\mathrm{mix}} \\ &\overset{(5.4)}{=} \beta \cdot \sum_{t\in\mathcal{T}_c(x_{a:b})} \ell(x_t; \mathsf{MDL}^c(x^c_{<t})) + r^c_{\mathrm{mix}} \\ &= \beta \cdot \ell(y_{i:j}; \mathsf{MDL}^c) + r^c_{\mathrm{mix}}\end{aligned}$$

$$\begin{aligned}
&\overset{\text{A5.3.2 (a), D5.3.3}}{\leqslant} \alpha\beta \cdot h(y_{i:j}) + \beta \cdot r^c_{\text{mdl}} + r^c_{\text{mix}} \\
&= \alpha\beta \cdot h(x^c_{a:b}) + \beta \cdot r^c_{\text{mdl}} + r^c_{\text{mix}}. \qquad (5.7)
\end{aligned}$$

We combine (5.5) with (5.6) and (5.5) with (5.7) to end the proof. □

By Lemma 5.3.4 (b) the code length $\ell^c_{\mathsf{CTM}}(a{:}b)$ of an arbitrary context c is close the code length $\sum_{x\in\mathcal{X}} \ell^{xc}_{\mathsf{CTM}}(a{:}b)$ of child contexts xc of c. (Here, we measure the proximity in terms of a multiplicative factor β and an additive redundancy term $r^c_{\text{mix}}(a{:}b)$.) For the previous lemma we argued on a single context. Next, we generalize this argument to multiple contexts of an arbitrary context tree. We do this by applying Lemma 5.3.4 (b) to the root contexts, its child contexts, etc.

Lemma 5.3.5 *For an arbitrary context tree of depth at most D with set $\mathcal{C}_I$ of non-leaf contexts, set $\mathcal{C}_L$ of leaf contexts and any segment $a{:}b \in \mathcal{S}$ we have*

$$\ell(x_{a:b}; \mathsf{CTM}) \leqslant \sum_{c\in\mathcal{C}_L} \beta^{|c|} \cdot \ell^c_{\mathsf{CTM}}(a{:}b) + \sum_{c\in\mathcal{C}_I} \beta^{|c|} \cdot r^c_{\text{mix}}(a{:}b). \qquad (5.8)$$

Proof. For brevity we omit the dependence of expressions like $\ell^c_{\mathsf{CTM}}(a{:}b)$ on $a{:}b$. We now prove the claim by induction on the number of internal contexts in the context tree.

Base: $|\mathcal{C}_I| = 0$. We have $\mathcal{C}_I = \emptyset$ and $\mathcal{C}_L = \{\phi\}$, so the r. h. s. of (5.8) becomes

$$\sum_{c\in\{\phi\}} \beta^{|c|} \cdot \ell^c_{\mathsf{CTM}} + \sum_{c\in\emptyset} \beta^{|c|} r^c_{\text{mix}} = \ell^\phi_{\mathsf{CTM}} \overset{\text{D5.3.3}}{=} \ell(x_{a:b}; \mathsf{CTM}).$$

Step: $|\mathcal{C}_I| > 0$. Consider a context tree T of depth at most D, non-leaf contexts $\mathcal{C}_I$ and leaf contexts $\mathcal{C}_L$. Since $|\mathcal{C}_I| > 0$ there must exist a non-leaf context, whose child contexts are leaf contexts, let z be such a context. We construct (see Figure 5.2) a context tree T' with non-leaf contexts $\mathcal{C}'_I$ and leaf contexts $\mathcal{C}'_L$ by removing the child contexts of z, so z becomes a leaf context in T', i. e.

$$\mathcal{C}'_I := \mathcal{C}_I \setminus \{z\} \quad \text{and} \quad \mathcal{C}'_L := \mathcal{C}_L \setminus \{xz \mid x \in \mathcal{X}\} \cup \{z\}. \qquad (5.9)$$

We may now apply the induction hypothesis to T' and get

$$\ell(x_{a:b}; \mathsf{CTM}) \overset{\text{I. H.}}{\leqslant} \sum_{c\in\mathcal{C}'_L} \beta^{|c|} \cdot \ell^c_{\mathsf{CTM}} + \sum_{c\in\mathcal{C}'_I} \beta^{|c|} \cdot r^c_{\text{mix}}. \qquad (5.10)$$

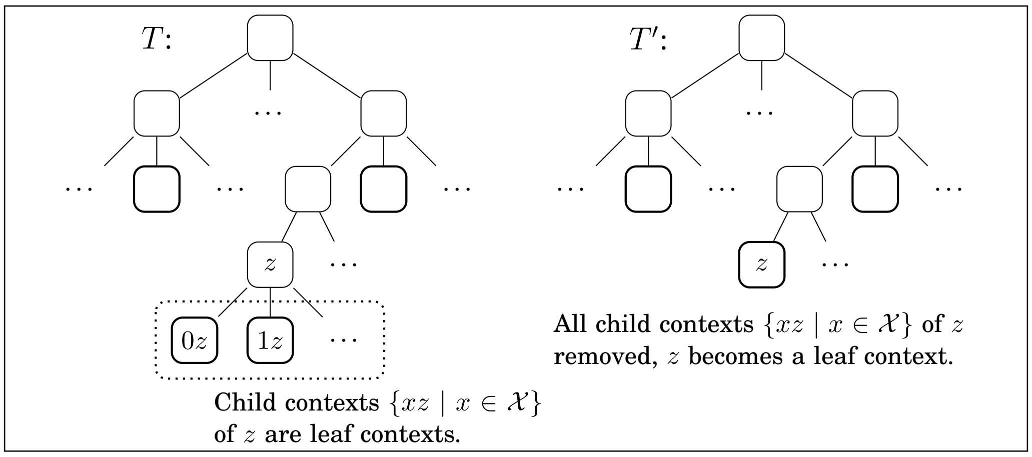

T:
T':
z
$0z$
$1z$
z
Child contexts $\{xz \mid x \in \mathcal{X}\}$ of z are leaf contexts.
All child contexts $\{xz \mid x \in \mathcal{X}\}$ of z removed, z becomes a leaf context.

Figure 5.2: Construction of the context tree T' with leaf context set $\mathcal{C}'_L$ and set $\mathcal{C}'_I$ of non-leaf contexts. In either context tree leaf contexts are drawn bold (non-leaf contexts non-bold).

Since z is a non-leaf context in T and T has depth at most D, we must have $|z| < D$, so we may apply Lemma 5.3.4 (b) to the context z,

$$\beta^{|z|}\ell^z_{\mathsf{CTM}} \overset{\text{L5.3.4 (b)}}{\leqslant} \beta^{|z|+1}\sum_{x\in\mathcal{X}}\ell^{xz}_{\mathsf{CTM}} + \beta^{|z|}r^z_{\text{mix}} \overset{(5.9)}{=} \sum_{c\in\mathcal{C}_L\setminus(\mathcal{C}'_L\setminus\{z\})}\beta^{|c|}\ell^c_{\mathsf{CTM}} + \sum_{c\in\mathcal{C}_I\setminus\mathcal{C}'_I}\beta^{|c|}r^c_{\text{mix}}, \tag{5.11}$$

where we used $\{xz \mid x \in \mathcal{X}\} = \mathcal{C}_L \setminus (\mathcal{C}'_L \setminus \{z\})$ and $\{z\} = \mathcal{C}_I \setminus \mathcal{C}'_I$, by (5.9). To conclude the proof, we plug (5.11) into the r. h. s. of (5.10), so

$$\begin{aligned}\ell(x_{a:b};\mathsf{CTM}) &\overset{(5.11)}{\leqslant} \sum_{c\in\mathcal{C}'_L\setminus\{z\}}\beta^{|c|}\ell^c_{\mathsf{CTM}} + \sum_{c\in\mathcal{C}'_I}\beta^{|c|}r^c_{\text{mix}} + \sum_{c\in\mathcal{C}_L\setminus(\mathcal{C}'_L\setminus\{z\})}\beta^{|c|}\ell^c_{\mathsf{CTM}} + \sum_{c\in\mathcal{C}_I\setminus\mathcal{C}'_I}\beta^{|c|}r^c_{\text{mix}}\\ &= \sum_{c\in\mathcal{C}_L}\beta^{|c|}\ell^c_{\mathsf{CTM}} + \sum_{c\in\mathcal{C}_I}\beta^{|c|}r^c_{\text{mix}}. \qquad\square\end{aligned}$$

Discussion. Let us discuss this last important technical result. In essence Lemma 5.3.5 states that the coding performance of CTM is close to an "intermediate" context tree model. We now elaborate on this. Fix a CTM instance CTM with depth parameter D and an arbitrary context tree with depth at most D, non-leaf contexts $\mathcal{C}_I$ and leaf contexts $\mathcal{C}_L$. Suppose that in the t-th step we predict the distribution $\mathsf{CTM}_{|c|}(x_{<t})$, where x_t has context $c \in \mathcal{C}_L$. By this prediction rule we define the intermediate model $\mathsf{M}(x_{<t}) := \mathsf{CTM}_{|c|}(x_{<t})$. (Such a model represents intermediate predictions that CTM computes, but we never use these predictions for coding.)

On the one hand, if we used the model M to encode a fragment $x_{a:b}$, a single leaf context $c \in \mathcal{C}_L$ would add $\ell^c_{\mathsf{CTM}}(a{:}b)$ bits to the total code length (see Definition 5.3.3); on the other hand, for CTM the context c charges at most $\beta^{|c|}\ell^c_{\mathsf{CTM}}(a{:}b)$ bits. For this proximity we have to pay a price, that is, $r^c_{\mathrm{mix}}(a{:}b)$, for *every* non-leaf context $c \in \mathcal{C}_I$. In other words, CTM is off M by a multiplicative factor plus some additive redundancy terms. So by Lemma 5.3.5 we essentially state that for a segment $x_{a:b}$ the coding performance of CTM is close to that of the fictitious model M. The overall redundancy balance of CTM w. r. t. M might seem somewhat pessimistic, since β and/or $r^c_{\mathrm{mix}}(a{:}b)$ might be large. But there is good news, for mixtures that we use in practice β can be brought arbitrarily close to 1, or even equals 1 (recall our results on OGDMs in Section 4.5.2). Moreover, $r^c_{\mathrm{mix}}(a{:}b)$ typically grows at rate $o(b)$; for Beta-Weighting we even have $r^c_{\mathrm{mix}}(a{:}b) = 1$, as we will see in Section 5.3.3. By the above discussion CTM with parameter D is close to (in terms of code length) the intermediate model M induced by an arbitrary context tree of depth at most D. As we will see below, an intermediate model M with some underlying context tree is also close to any PCT with the same underlying context tree.

Putting it all Together. We may now argue similarly to the discussion on mixing of the previous section. For short, code length $\ell^c_{\mathsf{CTM}}(a{:}b)$, for any context c, is close to the ideal code length $h(x^c_{a:b})$ in hindsight, cf. Lemma 5.3.4 (a) and Lemma 5.3.4 (c). Based on these results, we now extend Lemma 5.3.5 and obtain the following.

Lemma 5.3.6 *Consider the function $d(c) := \min\{|c|+1, D\}$ and an arbitrary context tree of depth at most D with set $\mathcal{C}_L$ of leaf contexts and set $\mathcal{C}$ of all contexts. For any segment $a{:}b \in \mathcal{S}$ we have*

$$\ell(x_{a:b}; \mathsf{CTM}) \leqslant \sum_{c\in\mathcal{C}_L} \alpha\beta^{d(c)} h(x_{a:b}) + \sum_{c\in\mathcal{C}_L} \beta^{d(c)} r^c_{\mathrm{mdl}}(a{:}b) + \sum_{\substack{c\in\mathcal{C},\\ |c|<D}} \beta^{|c|} r^c_{\mathrm{mix}}(a{:}b).$$

Proof. Again, we omit the dependence of expressions like $r^c_{\mathrm{mix}}(a{:}b)$ on $a{:}b$. First, Lemma 5.3.4 (c) implies

$$\sum_{\substack{c\in\mathcal{C}_L,\\ |c|<D}} \beta^{|c|}\ell^c_{\mathsf{CTM}} \overset{\text{L5.3.4 (c)}}{\leqslant} \sum_{\substack{c\in\mathcal{C}_L,\\ |c|<D}} \beta^{|c|}\cdot\left(\alpha\beta h(x^c_{a:b}) + \beta r^c_{\mathrm{mdl}} + r^c_{\mathrm{mix}}\right), \tag{5.12}$$

moreover, Lemma 5.3.4 (a) implies

$$\sum_{\substack{c\in\mathcal{C}_L,\\|c|=D}} \beta^{|c|}\ell^c_{\mathsf{CTM}} \overset{\text{L5.3.4 (a)}}{\leqslant} \sum_{\substack{c\in\mathcal{C}_L,\\|c|=D}} \beta^{|c|}\cdot(\alpha h(x^c_{a:b})+r^c_{\text{mdl}})\,. \tag{5.13}$$

By adding (5.12) and (5.13) we get

$$\sum_{c\in\mathcal{C}_L} \beta^{|c|}\ell^c_{\mathsf{CTM}} \leqslant \sum_{c\in\mathcal{C}_L}\left(\alpha\beta^{d(c)}h(x^c_{a:b})+\beta^{d(c)}r^c_{\text{mdl}}\right)+\sum_{\substack{c\in\mathcal{C}_L,\\|c|<D}}\beta^{|c|}r^c_{\text{mix}}. \tag{5.14}$$

To conclude the proof we combine (5.14) with Lemma 5.3.5 and rearrange. □

At this point most of the technical work is done. By applying Lemma 5.3.6 to the segments (induced by some partition) of a given an input sequence, it is now easy to obtain our first main result of this chapter.

Theorem 5.3.7 *Suppose that Assumption 5.3.2 holds. Consider a partition $\mathcal{P}$ of $1{:}n$, where $\mathcal{P}\subseteq\mathcal{S}$, and a family $\{\mathsf{PCT}_s\}_{s\in\mathcal{P}}$ of PCTs, where PCT_s has depth at most D, contexts $\mathcal{C}_s$ and leaf-contexts $\mathcal{C}_{L,s}$. It holds that*

$$\ell(x_{1:n};\mathsf{CTM})\leqslant\alpha\beta^D\cdot\sum_{s=a:b\in\mathcal{P}}\ell(x_{a:b};\mathsf{PCT}_s)+\beta^D\cdot\sum_{\substack{c\in\mathcal{C}_{L,s},\\s\in\mathcal{P}}}r^c_{\text{mdl}}(a{:}b)+\beta^{D-1}\cdot\sum_{\substack{c\in\mathcal{C}_s,|c|<D,\\s\in\mathcal{P}}}r^c_{\text{mix}}(a{:}b).$$

Proof. We fix a segment $s=a{:}b$ from $\mathcal{P}$ and omit the dependence of expression like $r^c_{\text{mix}}(a{:}b)$ on that segment. Next, we apply Lemma 5.3.6 to the context tree of $\mathsf{PCT}_s\langle\mathcal{C}_{L,s},\{p^c_s\}_{s\in\mathcal{C}_{L,s}}\rangle$ and substitute $h(x^c_{a:b})\leqslant\ell(x^c_{a:b};p^c_s)$, so

$$\begin{aligned}\ell(x_{a:b};\mathsf{CTM}) &\overset{\text{L5.3.6}}{\leqslant}\sum_{c\in\mathcal{C}_{L,s}}\alpha\beta^{d(c)}\ell(x^c_{a:b};p^c_s)+\sum_{c\in\mathcal{C}_{L,s}}\beta^{d(c)}r^c_{\text{mdl}}+\sum_{\substack{c\in\mathcal{C}_s,\\|c|<D}}\beta^{|c|}r^c_{\text{mix}}\\ &\overset{\substack{d(c)\leqslant D,\\ \text{D2.4.4}}}{\leqslant}\alpha\beta^D\ell(x_{a:b};\mathsf{PCT}_s)+\beta^D\sum_{c\in\mathcal{C}_{L,s}}r^c_{\text{mdl}}+\beta^{D-1}\sum_{\substack{c\in\mathcal{C}_s,\\|c|<D}}r^c_{\text{mix}}.\end{aligned} \tag{5.15}$$

To end the proof we sum (5.15) over all segments from $\mathcal{P}$. □

Discussion. All in all, Theorem 5.3.7 states that CTM is off the coding performance of an arbitrary sequence of PCTs by at most a multiplicative factor $\alpha\beta^D$ plus additive terms,

$$\beta^D \cdot \sum_{\substack{c\in\mathcal{C}_{L,s},\\ s\in\mathcal{P}}} r^c_{\text{mdl}}(a{:}b) \quad \text{and} \quad \beta^{D-1} \cdot \sum_{\substack{c\in\mathcal{C}_s, |c|<D,\\ s\in\mathcal{P}}} r^c_{\text{mix}}(a{:}b).$$

For every segment, the left term charges the cost of estimating the distributions p^c_s at every context $c \in C_{L,s}$ online and the right term measures the cost of estimating the context tree structure of PCT_s online. Keep in mind that the above bound holds for arbitrary sequences of PCTs, hence also for the optimal one. By a careful choice of elementary models and mixers, the multiplicative factor $\alpha\beta^D$ can be made small (even equal to 1) and the additive redundancy terms can be made $o(n)$ (in Section 5.4 we verify this claim). So we guarantee a vanishing per-letter redundancy w. r. t. a very strong class of competitors. No previous work treated such a rich class of competing schemes.

5.3.3 Example: Context Tree Weighting

Verifying Assumption 5.3.2. As we learned in Section 2.4.3 CTW employs the KT elementary model and Beta-Weighting for mixing. Hence, CTW is given by the CTM instance $\mathsf{CTW} = \mathsf{CTM}\langle D, \{\mathsf{MDL}^c\}_{|c|\leqslant D}, \{\mathsf{MIX}^c\}_{|c|=D}\rangle$, where we have $\mathsf{MDL}^c = \mathsf{KT}$ (see (2.16)), for all contexts c of length at most D, and $\mathsf{MIX}^c = \mathsf{BETA}$ (see (4.3)), for all contexts c of length less than D. For the current section we fix the set

$$\mathcal{S} = \{1{:}b \mid b \geqslant 1\} \tag{5.16}$$

of segments.

First, let us consider Assumption 5.3.2 (a). In Section 2.4.3 we have seen that for any sequence $y_{1:j}$ the model KT satisfies

$$\ell(y_{1:j}; \mathsf{KT}) \leqslant h(y_{1:j}) + \frac{N-1}{2}\log j + \log N \leqslant h(y_{1:j}) + \gamma(j), \quad \text{where}$$
$$\gamma(z) = \begin{cases} z, & \text{if } 0 \leqslant z \leqslant 1 \\ \frac{N-1}{2}\log z + \log N, & \text{otherwise} \end{cases},$$

hence Assumption 5.3.2 (a) is satisfied, whenever we consider a segment $y_{1:j} = x^c_{1:b}$ of the context history $y_{1:m} = x^c_{1:n}$. Hence, for the set $\mathcal{S}$ as given in (5.16) we have

$$\alpha = 1 \text{ and } f^c(y_{1:j}) = \gamma(j). \tag{5.17}$$

For a context history $y_{1:j} = x^c_{1:b}$ we subsequently obtain

$$r^c_{\mathrm{mdl}}(1{:}b) = \gamma(|x^c_{1:b}|). \tag{5.18}$$

Next, we check Assumption 5.3.2 (b). For any sequence $y_{1:j}$ and the mixer input $\boldsymbol{p}_{1:j} = \langle(u_t, v_t)\rangle_{1\leqslant t\leqslant j}$, induced by the sequences $u_{1:j}$ and $v_{1:j}$ of probability distributions, Beta-Weighting satisfies (see (4.5) and recall that $m = 2$)

$$\ell(y_{1:j}; \mathsf{BETA}, \boldsymbol{p}_{1:j}) \leqslant \ell(y_{1:j}; p_{1:j}) + 1, \text{ where } p_{1:j} \in \{u_{1:j}, v_{1:j}\},$$

hence Assumption 5.3.2 (b) is satisfied, whenever we consider a segment $y_{1:j} = x^c_{1:b}$ of the context history $y_{1:m} = x^c_{1:n}$ and a segment $\boldsymbol{p}_{1:j} = \mathrm{in}^c(x_{1:b})$ of the mixer input $\boldsymbol{p}_{1:m} = \mathrm{in}^c(x_{1:n})$. In turn, for the set $\mathcal{S}$ as given in (5.16) we have

$$\beta = 1 \text{ and } f^c(y_{1:j}, \boldsymbol{p}_{1:j}) = 1. \tag{5.19}$$

For a context history $y_{1:j} = x^c_{1:b}$ and a mixer input $\boldsymbol{p}_{1:j} = \mathrm{in}^c(x_{1:b})$ we get

$$r^c_{\mathrm{mix}}(1{:}b) = 1. \tag{5.20}$$

Applying Theorem 5.3.7. The premises of Theorem 5.3.7 are satisfied and we may choose $\mathcal{P} = \{1{:}n\}$ (in fact, this is the only partition of $1{:}n$ to satisfy $\mathcal{P} \subseteq S$) and fix a single PCT whose context tree has set $\mathcal{C}_I$ of non-leaf contexts, set $\mathcal{C}_L$ of leaf contexts and set $\mathcal{C} = \mathcal{C}_I \cup \mathcal{C}_L$ of all contexts. We conclude

$$\begin{aligned}
\ell(x_{1:n}; \mathsf{CTW}) &\overset{\substack{(5.17),(5.19)\\ \mathrm{T5.3.7}}}{\leqslant} \ell(x_{1:n}; \mathsf{PCT}) + \sum_{c\in\mathcal{C}_L} r^c_{\mathrm{mdl}}(1{:}n) + \sum_{\substack{c\in\mathcal{C},\\ |c|<D}} r^c_{\mathrm{mdl}}(1{:}n)\\
&\overset{\substack{(5.18),\\ (5.20)}}{=} \ell(x_{1:n}; \mathsf{PCT}) + \sum_{c\in\mathcal{C}_L} \gamma(|x^c_{1:n}|) + \underbrace{|\{c \in \mathcal{C} \mid |c| < D\}|}_{=\Gamma_D(\mathcal{C}_L),\text{ see (2.21)}}\\
&\leqslant \ell(x_{1:n}; \mathsf{PCT}) + |\mathcal{C}_L|\gamma\left(\frac{n}{|\mathcal{C}_L|}\right) + \Gamma_D(\mathcal{C}_L).
\end{aligned} \tag{5.21}$$

The last inequality results from Jensen's Inequality, since γ is convex, and from $\sum_{c\in\mathcal{C}_L} |x^c_{1:n}| = n$. Notice that we have just recovered the classical results on binary CTW given in [101, Theorem 2] and a slightly improved version of the results on CTW for a non-binary alphabet in [8]. Moreover, by the estimate

$$\Gamma_D(C_L) \leqslant |C_I| + |C_L| = \frac{N|C_L| - 1}{N - 1}$$

(complete N-ary trees satisfy $|\mathcal{C}_I| = \frac{|C_L|-1}{N-1}$) bound (5.21) becomes

$$\ell(x_{1:n}; \mathsf{CTW}) \leqslant \ell(x_{1:n}; \mathsf{PCT}) + |\mathcal{C}_L|\gamma\left(\frac{n}{|\mathcal{C}_L|}\right) + \frac{N|C_L|-1}{N-1},$$

so we precisely recover the result [8, Theorem 9].

5.4 PAQ meets Context Tree Mixing

We now equip CTM with a variation of the PAQ approaches to elementary modeling and mixing. The roadmap for this section is as follows: In Section 5.4.1 we describe and analyze a variation of PS that bounds letter probabilities away from 0. We similarly proceed with a Geometric OGDM that assumes p-vectors with distributions that have probabilities bounded away from 0. Based on this choice for elementary models and mixers we define PAQ CTM. In Section 5.4.2 we provide a code length analysis of PAQ CTM, split into two parts: First, we verify Assumption 5.3.2; based on this we apply Theorem 5.3.7 to obtain a code length bound for PAQ CTM. All results we present in this chapter hold for a binary alphabet only.

5.4.1 Elementary Modeling and Mixing

Elementary Modeling. For elementary modeling we consider a variant of PS that bounds any probability estimate of the t-th step from below by $\varepsilon_t > 0$. (Note that the sequence $\varepsilon_1, \varepsilon_2, \dots$ may still approach zero, we will rely on this later.)

Definition 5.4.1 *Let $\varepsilon_{1:\infty}$ be a sequence of reals s. t. $0 < \varepsilon_1, \varepsilon_2, \dots \leqslant \frac{1}{2}$ and let p be a distribution on a binary alphabet, where $p(0), p(1) \geqslant \varepsilon_1$. The Bounded Probability Smoothing (BPS) model $\mathsf{BPS}\langle\varepsilon_{1:\infty}, p\rangle$ predicts*

$$\mathsf{BPS}(x; x_{<t}) = \min\{1-\varepsilon_t, \max\{\varepsilon_t, \mathsf{PS}(x; x_{<t})\}\},$$

where $\mathsf{PS}\langle\alpha_{1:\infty}, p\rangle$ is an instance of PS s. t. $\alpha_t = e^{-\pi/\sqrt{12(t+1)}}$, for $t \geqslant 1$.

Note that the way BPS clamps probabilities always results in a valid distribution. To observe this consider a distribution p on a binary alphabet and a distribution p' defined by $p'(x) := \min\{1-\varepsilon, \max\{\varepsilon, p(x)\}\}$, where $0 < \varepsilon \leqslant \frac{1}{2}$. If $p(0), p(1) \in [\varepsilon, 1-\varepsilon]$, clamping has no effect and we have $p' = p$; if $p(0) < \varepsilon$, then the clamped distribution is given by $p'(0) = \varepsilon$ and $p'(1) = 1 - \varepsilon$ (the situation $p(1) < \varepsilon$ is symmetric).

In Definition 5.4.1 we use a smoothing rate sequence that we previously proposed in Section 3.5.4. By this smoothing rate choice BPS inherits code length guarantees similar to that of PS.

Lemma 5.4.2 *For a segment $x_{a:b}$ of $x_{1:n}$ the model* $\mathsf{BPS}\langle\varepsilon_{1:\infty}, p\rangle$ *satisfies*

$$\ell(x_{a:b}; \mathsf{BPS}) \leqslant h(x_{a:b}) + \frac{2\pi \log e}{\sqrt{3}} \cdot \sqrt{b} + \log \frac{1}{\varepsilon_1} + \sum_{a \leqslant t \leqslant b} \log \frac{1}{1 - \varepsilon_t}.$$

Proof. We split the proof into two parts. First we argue on the redundancy of the induced model PS w. r. t. the empirical entropy, second on the redundancy of BPS w. r. t. PS, finally we join the results.

Redundancy of PS *w. r. t. h.* By Lemma 3.5.7 we have

$$\ell(x_{a:b}; \mathsf{PS}) \leqslant h(x_{a:b}) + \log \frac{1}{p(0)\beta_{b-1}} + \sum_{a \leqslant t < b} \log \frac{1}{1 - \beta_t/\beta_{a-1}}$$

and analogously to the proof of Corollary 3.5.12 we may bound

$$\begin{aligned} \log \frac{1}{p(0)\beta_{b-1}} &\leqslant \log \frac{1}{\varepsilon_1} + \frac{\pi \log e}{\sqrt{3}} \cdot \sqrt{b} \quad \text{and} \\ \sum_{a \leqslant t \leqslant b} \log \frac{1}{1 - \beta_t/\beta_a} &\leqslant \frac{\pi \log e}{\sqrt{3}} \cdot \sqrt{b} \\ \implies \ell(x_{a:b}; \mathsf{PS}) &\leqslant h(x_{a:b}) + \log \frac{1}{\varepsilon_1} + \frac{2\pi \log e}{\sqrt{3}} \cdot \sqrt{b}. \end{aligned} \tag{5.22}$$

Redundancy of BPS *w. r. t.* PS. We first show that for $0 \leqslant u \leqslant 1$ and $0 < \varepsilon \leqslant \frac{1}{2}$ we have

$$v := \min\{1 - \varepsilon, \max\{\varepsilon, u\}\} \implies v \geqslant (1 - \varepsilon)u. \tag{5.23}$$

If $u < \varepsilon$, then $v = \varepsilon \geqslant u$, if $u > (1 - \varepsilon)$, then $v = 1 - \varepsilon \geqslant (1 - \varepsilon)u$, otherwise $v = u$; in either case we have $v \geqslant (1 - \varepsilon)u$. This implies $\mathsf{BPS}(x; x_{<t}) \geqslant (1 - \varepsilon_t)\mathsf{PS}(x; x_{<t})$, so

$$\ell(x_{a:b}; \mathsf{BPS}) \overset{(5.23)}{\leqslant} \sum_{a \leqslant t \leqslant b} \log \frac{1}{(1 - \varepsilon_t)\mathsf{PS}(x_t; x_{<t})} = \ell(x_{a:b}; \mathsf{PS}) + \sum_{a \leqslant t \leqslant b} \log \frac{1}{1 - \varepsilon_t}.$$

To end the proof we plug (5.22) into the above inequality. □

Mixing. For mixing we adopt Geometric OGDM. As we already announced, for meaningful code length guarantees we must assume that the mixer input only contains distributions with probabilities bounded away from 0.

Lemma 5.4.3 *For a segment $x_{a:b}$ of a sequence $x_{1:n}$ and for the mixer input $\boldsymbol{p}_{1:b} = \langle (u_t, v_t) \rangle_{1 \leqslant t \leqslant b}$, given by the sequences of distributions $u_{1:b}$ and $v_{1:b}$, where $u_t(x), v_t(x) \geqslant \varepsilon > 0$, the Geometric OGDM $\mathsf{GEO}\langle \alpha_{1:\infty}, \boldsymbol{w}_1 \rangle$ with step size $\alpha_t = t^{-1/2}$ satisfies*

$$\ell(x_{a:b}; \mathsf{GEO}, \boldsymbol{p}_{1:b}) \leqslant \ell(x_{a:b}; p_{a:b}) + (1 + 2\log^2 \varepsilon) \cdot \sqrt{b}, \ \textit{for} \ \ p_{a:b} \in \{u_{a:b}, v_{a:b}\}.$$

Proof. By Lemma 4.5.3 and Lemma 4.6.6 we have

$$\begin{aligned}\ell(x_{a:b}; \mathsf{GEO}, \boldsymbol{p}_{1:b}) - \ell(x_{a:b}; p_{a:b}) &\overset{\substack{\text{L4.6.6 (a),}\\ \text{L4.5.3}}}{\leqslant} \frac{1}{\alpha_b} + \sum_{a \leqslant t \leqslant b} \frac{\alpha_t |\boldsymbol{d}_t|^2}{2} \overset{\substack{\text{L4.6.6 (b),}\\ a \geqslant 1}}{\leqslant} \sqrt{b} + (\log^2 \varepsilon) \sum_{1 \leqslant t \leqslant b} \frac{1}{\sqrt{t}} \\ &\leqslant (1 + 2\log^2 \varepsilon) \cdot \sqrt{b},\end{aligned}$$

the last inequality is due to bounding the sum by an integral. □

Model PCTM. We now equip CTM with the BPS elementary model and the Geometric OGDM.

Definition 5.4.4 *Given some input sequence $x_{1:n}$ we call a CTM instance $\mathsf{PCTM}\langle D \rangle = \mathsf{CTM}\langle D, \{\mathsf{BPS}^c\}_{|c| \leqslant D}, \{\mathsf{GEO}^c\}_{|c| < D} \rangle$ PAQ Context Tree Mixing, if:*

(a) *$\mathsf{BPS}^c\langle \varepsilon^c_{1:\infty}, p^c \rangle$ is a BPS model s. t. if x_t has context c, then $\varepsilon^c_j = \frac{1}{t+1}$ for $|x^c_{<t}| = j - 1$ and $p^c(0), p^c(1) \geqslant \varepsilon^c_1$, if the context history $x^c_{1:n}$ is non-empty (otherwise the parameters of BPS^c are arbitrary).*

(b) *$\mathsf{GEO}^c\langle \alpha^c_{1:\infty}, \boldsymbol{w}^c_1 \rangle$ is a Geometric OGDM s. t. $\alpha^c_t = t^{-1/2}$, for $t \geqslant 1$ and $\boldsymbol{w}^c_1$ is arbitrary.*

The choice of $\varepsilon^c_{1:\infty}$ in Definition 5.4.4 (a) will be of central importance. Let us now take a closer look. Consider a context c with non-empty context history and a time instant t s. t. x_t has context c. At context c we already observed the context history $x^c_{<t} = y_{<j}$, so we have $|x^c_{<t}| = j - 1$ and thus $\varepsilon^c_j = \frac{1}{t+1}$. Consequently, when x_t has context c the model BPS^c predicts

$$\begin{aligned}\mathsf{BPS}^c(x; x^c_{<t}) &\overset{\text{D5.4.1}}{=} \min\{1 - \varepsilon^c_j, \max\{\varepsilon^c_j, \mathsf{PS}(x; y_{<j})\}\} \\ &\overset{\text{D5.4.4 (a)}}{=} \min\left\{\tfrac{t}{t+1}, \max\left\{\tfrac{1}{t+1}, \mathsf{PS}(x; x^c_{<t})\right\}\right\} \geqslant \tfrac{1}{t+1},\end{aligned} \tag{5.24}$$

where we used $x^c_{<t} = y_{<j}$. Informally we may summarize this as follows. If we predict a distribution on the letter x_t, which has context c, then we first get the intermediate prediction $p = \mathsf{PS}(x^c_{<t})$. Next, we obtain the distribution p' by clamping the probabilities, $p'(x) = \min\left\{\frac{t}{t+1}, \max\left\{\frac{1}{t+1}, p(x)\right\}\right\}$. Finally BPS^c predicts the clamped distribution, $\mathsf{BPS}^c(x^c_{<t}) = p'$.

5.4.2 PAQ Context Tree Mixing vs. a Sequence of Prediction Context Trees

Verifying Assumption 5.3.2. Our first step for the code length analysis of PAQ CTM is to show that Assumption 5.3.2 is satisfied. Based on this, we can adopt the analysis machinery for CTM.

By the choice of the elementary model to predict probabilities bounded away from 0, all intermediate predictions of PAQ CTM have probabilities bounded away from 0. The following lemma certifies this property.

Lemma 5.4.5 *PAQ CTM satisfies* $\mathsf{PCTM}_d(x; x_{<t}) \geqslant \frac{1}{t+1}$.

Proof. Define $\varepsilon := 1/(t+1)$ and let c be the length-d context of x_t. We use induction on $d = D, D-1, \dots, 0$ to prove the claim.

Base: $d = D$. We have $\mathsf{PCTM}_d(x; x_{<t}) \overset{(5.1)}{=} \mathsf{BPS}^c(x; x^c_{<t}) \overset{(5.24)}{\geqslant} \varepsilon$.

Step: $d < D$. The context c owns a mixer GEO^c with mixer input $\mathrm{in}^c(x_{<t}) = \boldsymbol{p}_{1:j}$ The p-vector $\boldsymbol{p}_j = (u, v)^\mathsf{T}$ has the components $u = \mathsf{BPS}^c(x^c_{<t})$ and $v = \mathsf{PCTM}_{d+1}(x_{<t})$. By (5.24) and by the induction hypothesis we get

$$u(x) \overset{(5.24)}{\geqslant} \varepsilon \text{ and } \quad v(x) \overset{\text{I. H.}}{\geqslant} \varepsilon. \tag{5.25}$$

We have $\mathsf{PCTM}_d(x_{<t}) \overset{(5.1)}{=} \mathsf{GEO}^c(y_{<j}, \boldsymbol{p}_{1:j})$ and for some $0 \leqslant w \leqslant 1$ the mixer GEO^c induces the probability assignment

$$\mathsf{GEO}^c(x; \boldsymbol{p}_{1:j}, y_{<j}) = \frac{u(x)^w v(x)^{1-w}}{\sum_{y\in\mathcal{X}} u(y)^w v(y)^{1-w}} \geqslant u(x)^w v(x)^{1-w} \overset{(5.25)}{\geqslant} \varepsilon.$$

To obtain the first inequality we bounded the denominator from above by $u(y)^w v(y)^{1-w} \leqslant wu(y) + (1-w)v(y)$ (Arithmetic-Geometric-Mean inequality) and we used $\sum_{y\in\mathcal{X}}(wu(y) + (1-w)v(y)) = 1$ (u and v are distributions). ☐

Based on the boundedness of the intermediate probabilities away from 0, we are able to show that Assumption 5.3.2 is satisfied and we may determine the redundancy expressions r^c_{mdl} and r^c_{mix}.

Lemma 5.4.6 *For the set $\mathcal{S} = \{a{:}b \mid 1 \leqslant a \leqslant b\}$ of segments the PAQ CTM model* PCTM *satisfies Assumption 5.3.2 and we have*
(a) $\alpha = 1$ *and* $r^c_{\mathrm{mdl}}(a{:}b) \leqslant \frac{2\pi \log e}{\sqrt{3}} \cdot \sqrt{b} + \log(b+1) + \sum_{t \in \mathcal{T}_c(x_{a:b})} \log \frac{t+1}{t}$ *and*
(b) $\beta = 1$ *and* $r^c_{\mathrm{mix}}(a{:}b) \leqslant (1 + 2\log^2(b+1)) \cdot \sqrt{b}$.

Proof. For the proof we first show Assumption 5.3.2 (a) by identifying α and f^c and then determine r^c_{mdl}. Assumption 5.3.2 (b) follows similarly. Now consider an input sequence $x_{1:n}$ and fix an arbitrary segment $a{:}b \in \mathcal{S}$ s. t. $b \leqslant n$. At some context c the segment $x_{a:b}$ of $x_{1:n}$ induces a context history segment $x^c_{a:b} = y_{i:j}$ of the whole context history $x^c_{1:n} = y_{1:m}$. If $|c| < D$, we similarly observe the segment $\mathrm{in}^c(x_{1:b}) = \boldsymbol{p}_{1:j}$ of the whole mixer input $\mathrm{in}^c(x_{1:n}) = \boldsymbol{p}_{1:m}$. If the context history $x^c_{a:b}$ is empty, Assumption 5.3.2 is trivially satisfied, otherwise we argue as follows:

Assumption 5.3.2 (a). By Lemma 5.4.2 the elementary model BPS^c satisfies

$$\ell(y_{i:j}; \mathsf{BPS}) \overset{\text{L5.4.2}}{\leqslant} h(y_{i:j}) + \underbrace{\frac{2\pi \log e}{\sqrt{3}} \cdot \sqrt{j} + \log \frac{1}{\varepsilon^c_1} + \sum_{i \leqslant k \leqslant j} \log \frac{1}{1 - \varepsilon^c_k}}_{= f^c(y_{1:j})},$$

so we can read off $\alpha = 1$ and f^c. We have

$$j \leqslant b,\ \varepsilon^c_1 \overset{\text{D5.4.4 (a)}}{\geqslant} \tfrac{1}{b+1} \text{ and } \{\varepsilon^c_k \mid i \leqslant k \leqslant j\} \overset{\text{D5.4.4 (a)}}{=} \left\{\tfrac{1}{t+1} \mid t \in \mathcal{T}_c(x_{a:b})\right\}$$
$$\Longrightarrow r^c_{\mathrm{mdl}}(a{:}b) = f^c(y_{1:j}) \leqslant \frac{2\pi \log e}{\sqrt{3}} \cdot \sqrt{b} + \log(b+1) + \sum_{t \in \mathcal{T}_c(x_{a:b})} \log \frac{t+1}{t}.$$

Assumption 5.3.2 (b). For $\mathcal{T} = \mathcal{T}_c(x_{1:b})$, $u_{1:j} = \langle \mathsf{BPS}^c(x^c_{<t})\rangle_{t \in \mathcal{T}}$ and $v_{1:j} = \langle \mathsf{PCTM}_{|c|+1}(x_{<t})\rangle_{t \in \mathcal{T}}$ the segment $\boldsymbol{p}_{1:j}$ of the mixer input is given by $\boldsymbol{p}_{1:j} = \langle (u_k, v_k)\rangle_{1 \leqslant k \leqslant j}$. Any distribution in the sequence $\boldsymbol{p}_{1:j}$ of p-vectors assigns at least probability $\varepsilon = \frac{1}{b+1}$ to any letter, since by (5.24) and by Lemma 5.4.5 we have $\mathsf{PCTM}_d(x; x_{<t}), \mathsf{BPS}^c(x; x^c_{<t}) \geqslant \frac{1}{t+1}$. Based on this Lemma 5.4.3 implies

$$\ell(y_{i:j}; \mathsf{GEO}^c, \boldsymbol{p}_{1:j}) \overset{\text{L5.4.3}}{\leqslant} \ell(y_{i:j}; p_{i:j}) + \underbrace{(1 + 2\log^2(b+1)) \cdot \sqrt{j}}_{= g^c(y_{1:j}, \boldsymbol{p}_{1:j})}, \text{ where } p \in \{u, v\},$$

so $\beta = 1$. By $j \leqslant b$ we finally conclude

$$r^c_{\mathrm{mix}}(a{:}b) = g^c(y_{1:j}, \boldsymbol{p}_{1:j}) \leqslant (1 + 2\log^2(b+1)) \cdot \sqrt{b}. \qquad \square$$

Applying Theorem 5.3.7. At this point all prerequisites for the analysis of PAQ CTM are satisfied. We now apply Theorem 5.3.7 to the PAQ CTM model to obtain a code length guarantee.

Theorem 5.4.7 *Let $\mathcal{P}$ be a partition of $1{:}n$ so that for every segment $s \in \mathcal{P}$ there is an associated PCT model $\mathsf{PCT}_s\langle \mathcal{C}_{L,s}, \{p_s^c\}_{c\in\mathcal{C}_{L,s}}\rangle$ with context tree depth at most D. The PAQ CTM model $\mathsf{PCTM}\langle D\rangle$ satisfies*

$$\ell(x_{1:n}; \mathsf{PCTM}) \leqslant \sum_{s=a:b\in\mathcal{P}} \Bigg(\ell(x_{a:b}; \mathsf{PCT}_s) + \Gamma_D(\mathcal{C}_{L,s}) \cdot (1 + 2\log^2(b+1))\sqrt{b} + |\mathcal{C}_{L,s}| \cdot \left(\frac{2\pi \log e}{\sqrt{3}}\sqrt{b} + \log(b+1)\right)\Bigg) + \log(n+1),$$

where Γ_D is given by (2.21).

Proof. We fix a single segment $s = a{:}b \in \mathcal{P}$, let the context tree of PCT_s have the set $\mathcal{C}_s$ of leaf- and non-leaf contexts and let $\mathcal{S}$ be the set of segments given in Lemma 5.4.6. Since $s \in \mathcal{S}$ we now use Lemma 5.4.6 to simplify some expressions

$$\sum_{\substack{t\in\mathcal{T}_c(x_{a:b}),\\ c\in\mathcal{C}_{L,s}}} \log\frac{t+1}{t} = \sum_{a\leqslant t\leqslant b} \log\frac{t+1}{t} = \log\frac{b+1}{a} \tag{5.26}$$

$$\sum_{\substack{c\in\mathcal{C}_s,\\ |c|<D}} r^c_{\mathrm{mix}}(a{:}b) \overset{\text{L5.4.6 (b)}}{=} \sum_{\substack{c\in\mathcal{C}_s,\\ |c|<D}} (1+2\log^2(b+1))\cdot\sqrt{b}$$
$$\overset{(2.21)}{=} \Gamma_D(C_{L,s})\cdot(1+2\log^2(b+1))\sqrt{b}, \tag{5.27}$$

$$\sum_{\substack{c\in\mathcal{C}_{L,s},\\ |c|<D}} r^c_{\mathrm{mdl}}(a{:}b) \overset{\text{L5.4.6 (a)}}{=} \sum_{c\in\mathcal{C}_{L,s}} \left(\frac{2\pi\log e}{\sqrt{3}}\cdot\sqrt{b} + \log(b+1) + \sum_{t\in\mathcal{T}_c(x_{a:b})} \log\frac{t+1}{t}\right)$$
$$\overset{(5.26)}{=} |\mathcal{C}_{L,s}|\cdot\left(\frac{2\pi\log e}{\sqrt{3}}\sqrt{b} + \log(b+1)\right) + \log\frac{b+1}{a}. \tag{5.28}$$

Clearly, any partition $\mathcal{P}$ satisfies $\mathcal{P} \subseteq \mathcal{S}$, so we apply Theorem 5.3.7 and plug in the above equalities,

$$\ell(x_{1:n}; \mathsf{CTM}) \overset{\substack{(5.27),\\(5.28)}}{\leqslant} \sum_{s=a:b\in\mathcal{P}} \Bigg(\ell(x_{a:b}; \mathsf{PCT}_s) + \Gamma_D(\mathcal{C}_{L,s})\cdot(1+2\log^2(b+1))\sqrt{b} + |\mathcal{C}_{L,s}|\cdot\left(\frac{2\pi\log e}{\sqrt{3}}\sqrt{b} + \log(b+1)\right) + \log\frac{b+1}{a}\Bigg).$$

To end the proof, we note $\sum_{a:b\in\mathcal{P}} \log\frac{b+1}{a} = \log(n+1)$ (a telescope sum). □

Discussion. The code length bound in Theorem 5.4.7 essentially states

$$\ell(x_{1:n}; \mathsf{PCTM}) = \sum_{s=a:b\in\mathcal{P}} \ell(x_{a:b}; \mathsf{PCT}_s) + O\left(\sqrt{n} \cdot \sum_{s\in\mathcal{P}} |\mathcal{C}_{L,s}| + \sqrt{n}\log^2 n \cdot \sum_{s\in\mathcal{P}} \Gamma_D(\mathcal{C}_{L,s})\right).$$

So the redundancy of PAQ CTM w. r. t. a sequence of PCTs associated to segments from $\mathcal{P}$ with depth at most D grows at rate $O(|\mathcal{P}|\sqrt{n}\log^2 n)$. (The constants in the big-Oh notation depend on the structure of the PCTs). The redundancy for the PCT associated to some segment s from $\mathcal{P}$ is composed as follows: First, for every leaf context $c \in \mathcal{C}_{L,s}$ the corresponding BPS elementary model pays $O(\sqrt{n})$ bits to estimate the distribution p^c_s. Second, for every leaf- and non-leaf context of length less than D (there are $\Gamma_D(C_{L,s})$ of them) the corresponding Geometric OGDM charges $O(\sqrt{n}\log^2 n)$ bits. This is the price the mixer associated to c has to pay in order to decide whether or not it is worthwhile to predict using the elementary model BPS at context c or using the models associated to child contexts of c. The average redundancy of PAQ CTM w. r. t. a single PCT vanishes at rate $O(\log^2(n)/\sqrt{n})$, whereas standard CTW achieves a rate of $O(\log(n)/n)$, smaller by a factor of $\sqrt{n}\log n$. So if we just consider a single PCT as competitor the theoretical guarantees of CTW clearly are superior. However, the result on PAQ CTM holds for a strictly larger and far more powerful class of competitors, which comes at its price. It is likely for PAQ CTM to outperform CTW. The experimental results in the next section shall confirm our mindset.

5.5 Experiments

Following our theoretic analysis, we conduct some experiments. Our experiments evaluate the compression performance of CTM equipped with various model-mixer-combinations on the Calgary Corpus. We direct our attention to the characteristics of PAQ components. Our main concern is to demonstrate that modeling and mixing from PAQ — without all of PAQ's heuristics and tricks — is on par with traditional approaches and shows better performance in case of heterogeneous files. Consequently, the experiments are of theoretical interest. In the following we first sketch the experimental setup and implementation details, subsequently we present and discuss our results on the Calgary Corpus for various CTM variants.

5.5.1 Experimental Setup

Elementary Models and Mixers. For our experiments we consider various combinations of elementary models and mixers. We concentrate on elementary models and mixers from PAQ that perform well without any prior knowledge of the input. For this reason we only consider elementary models and mixers that neither fix their parameters independent of the input length (e. g. an OGDM that chooses the step size $\alpha = 0.3$, independent of the input) nor that tune their parameters depending of the input length n (e. g. an OGDM that chooses the step size $n^{-1/2}$).

The former parameter choice is troublesome, since good parameters may drastically vary from file to file and even from context to context [85], for a single file. Hence we should determine different parameters by file type or by (class of) context and store the parameters or perform some kind of online parameter estimation. Clever methods for this purpose are object of research [85] and beyond the scope of this work.

The latter parameter choice disqualifies, since the length of every context history is only available after processing the whole input sequence. Hence, it is unrealistic to assume this kind of prior knowledge, especially for every context history. We may not easily store the context history lengths for decompression in a few bits, since there are millions of involved context histories. Furthermore, we give no results for Linear OGDM. In all our experiments on Linear OGDM we found it to perform much worse than Beta-Weighting and Geometric OGDM. Linear OGDM does not seem to be well suited for CTM on a binary alphabet.

To end this, we consider

- the LP elementary model (RF from (2.3) with $f_0 = 1$),
- the KT elementary model (RF from (2.3) with $f_0 = \frac{1}{N}$),
- the ZR elementary model [88], given by the probability assignment rule[1] $\mathsf{ZR}(x; x_{1:t}) = P(x_1 \dots x_t x)/P(x_1 \dots x_t)$, where

$$P(x_{1:n}) = \frac{\prod_{1\leqslant t\leqslant n} \mathsf{KT}(x_t; x_{<t}) + P_{\text{bias}}(x_{1:n})}{2} \quad \text{and}$$

$$P_{\text{bias}}(x_{1:n}) = \begin{cases} \frac{1}{2}, & \text{if } x_{1:n} \text{ is deterministic} \\ 0, & \text{otherwise,} \end{cases}$$

- the BPS elementary model initialized s. t. $p(0) = p(1) = \frac{1}{2}$ and
- BPS*, a variation of BPS that utilizes a heuristic to choose p.

[1]By the definition of $P(x_{1:n})$ and by (4.4) it can be seen that the ZR elementary model essentially combines the KT elementary model and a model that is highly bias towards deterministic sequences via Beta-Weighting.

For BPS* we use a simple heuristic related to information-inheritance [84]: Fix some time step s. t. the current length-d context $c_{1:d}$ has an empty context history. The BPS* model at this context predicts the uniform initial distribution p, viz. $p(0) = p(1) = \frac{1}{2}$. Immediately before the BPS* model at context $c_{1:d}$ is updated, we reset its initial distribution to match the BPS*-prediction of parent context $c_{1:d-1}$ *after* the BPS* elementary model of the parent context was updated. (Probabilities clamped to $[1/(t+1), t/(t+1)]$; this procedure does not invalidate our analysis since it does not depend on the actual value of the initial distribution.) This heuristic demonstrates an advantage of PS-based models over traditional models such as KT and LP — We may use heuristics without breaking the theoretical guarantees.

For mixing we consider

- the Geometric OGDM with step size $\alpha_t = \frac{1}{\sqrt{t}}$ and
- Beta-Weighting (see Chapter 4).

We tried to find heuristics to determine the initial weight of the Geometric OGDM. Surprisingly, we were not able to find a good heuristic that improves compression. Notice that the combination of Geometric OGDM with either BPS or BPS* is equivalent to PAQ CTM from Section 5.4. Other CTM instances we consider in our experiments correspond to already known CTW variants. The combination of Beta-Weighting and the KT elementary model refers to the basic CTW variant (coupled with techniques to handle a non-binary alphabet) as introduced in [101]. If we substitute the KT estimator by the ZR estimator, then we obtain the CTW variant proposed in [88]. The compression rates we measure for those variants are in line with the results reported in [103].

The Data. We use the most common form of the Calgary Corpus which consists of 14 files that can roughly be divided into three categories: executable binaries (`obj1`, `obj2`), digitized analogue data (`pic`, `geo`) and ASCII text in various formats (the remaining files, see Table 5.2). The figures we present here do not include redundancy due to Arithmetic Coding.

We first compress the Calgary Corpus file by file for all combinations of elementary models and mixers for the parameter $D = 6$ (a context consists of 6 letters, sized 8 bit each). Other context length parameters yield different figures, but a similar picture overall. To explore to which extent BPS and a Geometric OGDM handle inhomogeneous data we concatenate all files of the Calgary Corpus into a single file before compression and vary parameter $D \in \{1, 2, \ldots, 10\}$. (The concatenated file interleaves different data types, for instance a portion of ASCII text followed by executable code, thus can be considered to have heterogeneous statistics.)

Implementation Details. Our implementation operates on a binary alphabet. We treat a file as a sequence of 8-bit letters, thus we apply the following standard-implementation techniques: A flat binary alphabet decomposition, weighting only at letter boundaries and no root weighting [88]. In essence, a letter (over alphabet of size 2^8!) with length-D context $c_{1:D}$ receives the binary code word $y_{1:8}$ and is processed in 8 binary steps. We will now explain the i-th coding step in terms of the pseudocode Figure 5.1. We obtain the model-mixer-pair $(\mathsf{MDL}_0, \mathsf{MIX}_0)$ for the context $(\phi, y_{1:i})$, the pair $(\mathsf{MDL}_1, \mathsf{MIX}_1)$ for context $(c_{D:D}, y_{1:i})$, $\dots$, $(\mathsf{MDL}_D, *)$ for context $(c_{1:D}, y_{1:i})$ (a length D-context only requires a model); MIX_{D-1} combines predictions of MDL_D and MDL_{D-1} resulting in mixed prediction p, MIX_{D-2} combines mixed prediction p and prediction of MDL_{D-2}, etc. (just as in Figure 5.1).

We use 64-bit floating point numbers to store probabilities and weights, and we did neither make any attempt to speed up costly computations (e. g. square roots, logarithms, exponentials) with lookup tables, nor did we introduce specifically tailored data structures (such as the hash tables in [103] or the techniques used in PAQ [52]). (A more careful implementation should take advantage of these improvements and use fixed-point integer arithmetic.) Recall that our experiments are not intended to evaluate a practical compression algorithm, rather we want to compare the performance of traditional approaches to modeling and mixing to that of the PAQ approaches. (For efficient and empirically well-performing algorithms PAQ has demonstrated that one should adopt other implementation techniques and further heuristics.) Consequently, we don't include timings and only provide compression rates.

5.5.2 Evaluation

Single files, parameter $D = 6$. Across Geometric OGDM and Beta-Weighting, all elementary models align similarly, see Table 5.2 and Table 5.3. Models LP and KT show worst overall results and LP is significantly worse than any other elementary model, e. g. on average $5.66\ \%$ worse than BPS* for Beta-Weighting (or $4.52\ \%$ for Geometric OGDM). For textual files the model ZR shows best overall performance, which is not surprising, since it was designed for that purpose. However, on average, it outperforms the second-best model BPS* only by a small margin, $1.08\ \%$ for Beta-Weighting and $0.77\ \%$ for Geometric OGDM, respectively. Models BPS and BPS* show good performance in case of non-textual files, BPS* leaves all other elementary models

Table 5.2: CTM compression rates (in bpc) for various combinations of elementary models and mixers on the Calgary Corpus for $D = 6$. Best compression rates are typeset boldface, averages are weighted according to file sizes.

	Beta-Weighting					Geometric OGDM				
File	LP	KT	ZR	BPS	BPS*	LP	KT	ZR	BPS	BPS*
`bib`	2.047	1.924	**1.836**	1.917	1.860	2.007	1.904	**1.858**	1.944	1.902
`book1`	2.247	2.199	**2.183**	2.255	2.256	2.217	2.176	**2.166**	2.227	2.224
`book2`	2.034	1.955	**1.909**	1.956	1.947	1.978	1.911	**1.878**	1.915	1.903
`geo`	**4.579**	4.579	4.602	4.598	4.605	4.531	4.523	4.528	4.517	**4.505**
`news`	2.595	2.471	**2.379**	2.481	2.424	2.503	2.397	**2.344**	2.429	2.392
`obj1`	4.091	3.930	3.827	3.850	**3.768**	3.895	3.757	3.699	3.661	**3.599**
`obj2`	2.841	2.664	2.523	2.484	**2.412**	2.644	2.495	2.403	2.321	**2.281**
`paper1`	2.614	2.449	**2.321**	2.428	2.356	2.504	2.364	**2.283**	2.370	2.324
`paper2`	2.449	2.330	**2.249**	2.330	2.285	2.375	2.274	**2.223**	2.297	2.265
`pic`	0.803	0.799	0.803	**0.794**	0.798	0.767	0.765	0.771	**0.757**	0.767
`progc`	2.694	2.513	2.376	2.450	**2.371**	2.570	2.417	2.330	2.381	**2.330**
`progl`	1.926	1.781	1.666	1.703	**1.654**	1.849	1.728	1.655	1.676	**1.635**
`progp`	2.023	1.852	1.708	1.732	**1.665**	1.919	1.766	1.660	1.669	**1.621**
`trans`	1.845	1.641	1.450	1.547	**1.432**	1.766	1.587	1.450	1.535	**1.445**
Average	2.134	2.052	**1.998**	2.044	2.020	2.068	1.998	**1.963**	1.997	1.979

Table 5.3: BPS* vs. other elementary models; compression loss (in percent) of not using elementary model BPS*. (For instance, Beta-Weighting with model LP results in a 10.05 % larger compressed version of `bib` compared to Beta-Weighting with BPS*.) Averages are weighted according to file sizes.

	Beta-Weighting				Geometric OGDM			
File	LP	KT	ZR	BPS	LP	KT	ZR	BPS
`bib`	10.05	3.43	−1.30	3.08	5.54	0.10	−2.34	2.22
`book1`	−0.44	−2.53	−3.24	−0.05	−0.32	−2.14	−2.61	0.14
`book2`	4.47	0.42	−1.94	0.46	3.94	0.40	−1.33	0.61
`geo`	−0.56	−0.55	−0.05	−0.15	0.58	0.39	0.51	0.27
`news`	7.05	1.92	−1.86	2.34	4.65	0.21	−1.98	1.54
`obj1`	8.58	4.30	1.57	2.18	8.25	4.39	2.80	1.75
`obj2`	17.76	10.43	4.59	2.98	15.90	9.37	5.36	1.76
`paper1`	10.93	3.94	−1.47	3.05	7.73	1.73	−1.74	1.98
`paper2`	7.16	1.98	−1.58	1.99	4.85	0.41	−1.87	1.41
`pic`	0.61	0.05	0.55	−0.54	0.05	−0.28	0.52	−1.21
`progc`	13.62	6.01	0.22	3.31	10.31	3.74	0.03	2.22
`progl`	16.47	7.69	0.72	2.99	13.07	5.67	1.22	2.49
`progp`	21.54	11.27	2.57	4.02	18.40	8.97	2.41	3.00
`trans`	28.81	14.56	1.23	8.00	22.19	9.81	0.32	6.20
Average	5.66	1.59	−1.08	1.22	4.52	0.97	−0.77	0.91

Table 5.4: Beta-Weighting vs. Geometric OGDM; compression loss (in percent) of not using a Geometric OGDM. (For instance, for elementary model LP and Beta-Weighting result in a 1.97 % larger compressed version of `bib` compared to LP and a Geometric OGDM.) Averages are weighted according to file sizes.

File	LP	KT	ZR	BPS	BPS*
`bib`	1.97	1.05	−1.16	−1.39	−2.21
`book1`	1.35	1.06	0.80	1.27	1.46
`book2`	2.83	2.33	1.67	2.15	2.31
`geo`	1.05	1.25	1.63	1.77	2.21
`news`	3.67	3.07	1.46	2.13	1.34
`obj1`	5.02	4.60	3.45	5.15	4.70
`obj2`	7.44	6.77	4.97	7.01	5.74
`paper1`	4.39	3.58	1.66	2.44	1.38
`paper2`	3.10	2.45	1.16	1.45	0.87
`pic`	4.69	4.45	4.14	4.81	4.10
`progc`	4.84	4.00	1.97	2.87	1.78
`progl`	4.20	3.09	0.66	1.66	1.16
`progp`	5.46	4.90	2.89	3.75	2.73
`trans`	4.47	3.40	0.01	0.79	−0.89
Average	1.03	1.03	1.02	1.02	1.02

behind for files `obj1`, `obj2` (other elementary models produce larger files, between 1.57 % and 17.76 % for Beta-Weighting, and between 1.75 % and 15.9 % for Geometric OGDM, respectively). The model BPS scores over all other elementary models in case of file `pic`, nevertheless BPS* is close. Surprisingly, BPS* outperforms ZR for text files `progc`, `progp`, `progl` and `trans`, the compressed output of ZR is between 0.22 % and 2.57 % larger for Beta-Weighting, and between 0.03 % and 2.41 % larger for Geometric OGDM, respectively. For BPS this is not the case. We contribute this to our initialization heuristic, which seems to have a big impact for small files (for these files sizes range from 21 kB to 90 kB): “Small” files have short context histories, thus a good initialization of probability estimates may have a big impact on compression. In summary, the elementary model BPS* shows compression capability close to ZR, which is substantially more complex, since ZR can be considered the fusion of Beta-Weighting and KT-modeling.

Let us now compare Beta-Weighting and Geometric OGDM across all elementary models, see Table 5.2 and Table 5.4. In terms of average compression ratio Geometric OGDM consistently outperforms Beta-Weighting by roughly 1 %. Similarly in the vast majority of cases the compressed output of Beta-Weighting is larger than the compressed output of Geometric OGDM, by a margin of up to 7.44 %. Only for the file `bib` and elemen-

Table 5.5: CTM compression rates (in bpc) for various combinations of elementary models and mixers on the concatenation of all Calgary Corpus files. Best compression rates are typeset boldface.

	Beta-Weighting					Geometric OGDM				
D	LP	KT	ZR	BPS	BPS*	LP	KT	ZR	BPS	BPS*
1	3.761	3.761	3.767	**3.313**	3.316	3.704	3.702	3.704	**3.276**	3.278
2	2.956	2.935	2.936	**2.695**	2.707	2.880	2.864	2.860	**2.615**	2.623
3	2.498	2.456	2.443	**2.309**	2.321	2.401	2.368	2.355	**2.207**	2.216
4	2.297	2.236	2.206	**2.137**	2.140	2.186	2.139	2.119	**2.034**	2.037
5	2.235	2.165	2.125	2.084	**2.079**	2.116	2.063	2.039	1.979	**1.978**
6	2.214	2.139	2.095	2.066	**2.057**	2.091	2.035	2.011	1.961	**1.957**
7	2.206	2.129	2.081	2.058	**2.045**	2.081	2.024	1.999	1.955	**1.948**
8	2.202	2.123	2.073	2.054	**2.038**	2.076	2.017	1.992	1.951	**1.944**
9	2.200	2.120	2.069	2.053	**2.033**	2.073	2.015	1.989	1.949	**1.942**
10	2.199	2.119	2.067	2.051	**2.031**	2.072	2.013	1.988	1.949	**1.941**

tary models ZR, BPS and BPS* and for the file `trans` and elementary model BPS*, the output of Beta-Weighting is between $0.89\ \%$ and $2.21\ \%$ smaller compared to that of Geometric OGDM. For non-textual files `obj1`, `obj2` and `pic` Geometric OGDM considerably boosts compression, Beta-Weighting produces between $3.45\ \%$ and $7.44\ \%$ worse compression compared to Geometric OGDM. The elementary models LP and KT (which can be considered worst among all given elementary models) draw most profit from Geometric OGDM. In summary, a Geometric OGDM consistently outperforms Beta-Weighting by circa $1\ \%$ on average, whereas the per-file improvement may be drastically larger. Only in very few case we observe a compression loss, when we use Geometric OGDM instead of Beta-Weighting.

Concatenation, parameter $D \in \{1, 2, \dots, 10\}$. Theoretic results on BPS and Geometric OGDM suggest that these techniques may perform well in a non-stationary environment. We now want to experimentally undergird this observation by compressing a concatenation of all files from the Calgary Corpus.

We first evaluate PS-based elementary models across Beta-Weighting and Geometric OGDM, cf. Table 5.5 and Table 5.6. Apparently models BPS and BPS* outclass all other elementary models for both mixtures. For context lengths up to 4 letters BPS is in the lead in terms of compression, afterwards BPS* takes the lead, for either mixture. (The BPS*-initialization heuristic seems to slightly hurt compression for $D \leqslant 4$.) Nevertheless, the difference between BPS and BPS* is small, so we concentrate on BPS* from now on. The advantage of BPS* over other elementary models degrades as

Table 5.6: BPS* vs. other elementary models; compression loss (in percent) of not using elementary model BPS*. (For instance, Beta-Weighting with model LP results in a 13.44 % larger compressed version of `bib` compared to Beta-Weighting with BPS*.) Averages are weighted according to file sizes.

	Beta-Weighting				Geometric OGDM			
D	LP	KT	ZR	BPS	LP	KT	ZR	BPS
1	13.44	13.42	13.62	−0.08	13.00	12.94	13.00	−0.05
2	9.18	8.40	8.43	−0.47	9.78	9.16	9.02	−0.32
3	7.62	5.80	5.25	−0.53	8.34	6.85	6.29	−0.39
4	7.34	4.47	3.06	−0.14	7.30	5.00	4.02	−0.19
5	7.53	4.14	2.21	0.25	6.99	4.30	3.11	0.07
6	7.66	4.02	1.85	0.46	6.85	4.00	2.75	0.22
7	7.88	4.10	1.76	0.64	6.80	3.86	2.58	0.32
8	8.06	4.19	1.72	0.82	6.78	3.78	2.49	0.37
9	8.20	4.28	1.76	0.95	6.78	3.75	2.45	0.40
10	8.28	4.33	1.78	1.01	6.78	3.74	2.43	0.42

D increases. Still, for parameter $D \geqslant 7$ and Beta-Weighting, model LP produces roughly 8 %, model KT roughly 4 % and model ZR circa 1.7 % worse compression than BPS*; for Geometric OGDM these figures are 6.7 %, 3.7 % and 2.4 %. So far we can tell, that in the current setting the models LP, KT and ZR are inferior to BPS* (and BPS), for Beta-Weighting and Geometric OGDM.

We conclude the evaluation of our experiments by judging on the compression loss (across all elementary models) of not using a Geometric OGDM. A glance at Table 5.5 and Table 5.7 clearly shows that Beta-Weighting compresses worse than Geometric OGDM in every single situation. The compression loss ranges from 1.11 % for $D = 1$ and model BPS up to 6.12 % for $D = 10$ and model LP. For $D \geqslant 4$ Beta-Weighting compresses 4 % to 6 % worse than a Geometric OGDM. Similarly to single file compression, Geometric OGDM turns out to be superior to Beta-Weighting.

Overall results. In summary our experiments consistently give empirical evidence for the superiority of PAQ-based components, especially considering inhomogeneous files. In the vast majority of cases, a Geometric OGDM surpasses Beta-Weighting across all elementary models in every experimental setting. Improvements are pronounced in the case of heterogeneous files. These experimental findings underpin the advantage of a Geometric OGDM observed in former experiments in a different setting [58].

Regarding elementary modeling, PS-based models easily outperform the KT- and LP-model for either mixture in almost all experiments. For (mostly)

Table 5.7: Beta-Weighting vs. Geometric OGDM; compression loss (in percent) of not using a Geometric OGDM. (For instance, for elementary model LP and Beta-Weighting result in a 1.55 % larger compressed version of `bib` compared to LP and a Geometric OGDM.)

D	LP	KT	ZR	BPS	BPS*
1	1.55	1.59	1.70	1.11	1.15
2	2.63	2.47	2.64	3.04	3.20
3	4.06	3.72	3.73	4.60	4.75
4	5.09	4.52	4.08	5.11	5.05
5	5.65	4.96	4.21	5.32	5.12
6	5.90	5.13	4.19	5.35	5.11
7	6.02	5.20	4.12	5.30	4.96
8	6.08	5.23	4.04	5.29	4.83
9	6.11	5.25	4.01	5.29	4.71
10	6.12	5.25	3.99	5.28	4.66

homogeneous files PS-based models perform slightly worse than the computationally more demanding ZR-model. (Recall that ZR combines Beta-Weighting and the KT elementary model, so a comparison to a plain elementary model is somewhat unfair.) However, the situation changes for inhomogeneous files, where BPS turns out to be superior. Altogether PS-based models are at least on par with other elementary models of consideration, and take the lead for inhomogeneous data.

5.6 Summary

In the present chapter we studied the interaction of modeling and mixing in CTM, a statistical data compressor which is based on the CTW algorithm. CTM generalizes CTW, since it does not rely on specific elementary models and mixers. Rather CTM admits for models and mixers that are drawn from classes of models and mixers. These classes are spanned by code length guarantees. We have analyzed CTM and we have drawn the conclusion that if these classes are sufficiently expressive, then CTM has low redundancy w. r. t. a sequence of PCTs, not just w. r. t. a single PCT. The code length guarantees we deduced for CTM are of a very general nature, for instance these allow us to restore previous results on CTW for a binary and non-binary alphabet with little effort.

One may wonder whether the models and mixers from classes that allow for low redundancy w. r. t. a sequence of PCTs are of practical interest (due to running time and space considerations), or even if such classes exist at all. To tackle this issue we have equipped CTM with PAQ style elementary modeling and mixing, which gave rise to the PAQ CTM algorithm. (More precisely, PAQ CTM utilizes a slightly refined version of PS and Geometric OGDM.) The PAQ CTM algorithm allows us to answer both questions in the affermative: As our code length analysis shows PAQ CTM has low redundancy w. r. t. a sequence of PCTs.

Since PAQ CTM enjoys code length guarantees far more powerful than those of CTW it is likely for PAQ CTM to outperform classical CTW variants in experiments. To evaluate this expectation we carried out a variety of experiments on CTM with different elementary models and mixers. The CTM configurations within the experiments include variations of CTW and PAQ CTM. In the vast majority of cases PAQ CTM outperforms CTW or is at least on par. When we consider non-stationary data PAQ CTM is superior to CTW. This behavior is evidence for the ability of PAQ CTM to adapt to varying statistics, even for relatively short input sequences. In turn, we may conclude that PAQ style elementary modeling and mixing is not only superior to traditional approaches in terms of code length guarantees, but also in terms of empirical performance. So the PAQ approaches with all heuristics and tricks cut off still outperforms traditional approaches.

CHAPTER 6

Conclusion

The family of statistical data compression algorithms is a particularly powerful approach to data compression. In statistical data compression there is an increasing gap between theory and practice: Ad-hoc techniques proposed by practitioners often show superior empirical performance, but lack a sound theoretical basis. In contrast, techniques proposed by theoreticians are often inferior in terms of empirical performance, but rest on a theoretical basis. PAQ, a modern approach to statistical data compression, is a shining example for this contrast.

In this work we made first attempts to close the aforementioned gap between theory and practice, concentrating on PAQ. In doing so we first identified a common design pattern that all major statistical data compression algorithms follow and fit these algorithms into our framework. The general recipe is to combine multiple simple predictions, produced by elementary models, using a mixer (or multiple mixers). Based on this we identify approaches to elementary modeling and mixing that are commonly used in practice but lack a theoretical basis. Our main contribution is to support these approaches by theoretical code length guarantees. In addition, our code length guarantees utilize competitors that have the ability to adapt to changing inputs. It is well-known that this property is beneficial for compression.

Our work added a theoretical basis to the elementary models RFD, RFS and several variations of PS. We may further support our theoretical results by experiments. Note that PS is the PAQ approach to elementary modeling, hence we add a theoretical justification to PAQ-style elementary modeling. For mixing we first proposed the Linear- and Geometric Mixture Distribution that allow for the weighted combination of multiple probability distributions. We obtained these techniques as the minimizer of two slightly different information theoretic minimization problems. Since these mixture distributions rely on a given set of weights we adopted OGD for weight estimation. This choice has two advantages: PAQ has proven that this technique works well in statistical data compression and it is known to have a theoretical basis. Consequently, we derived the class of OGDMs that, as we show, are supported by code length guarantees. These results translate to the combination of the Linear- and Geometric Mixture Distribution with OGD, so we add a theoretical basis to Linear and Geometric OGDMs. This basis is supported by experiments. Note that the Geometric OGDM is a generalization of LM from PAQ to an alphabet of arbitrary size. Hence, we provide a theoretical basis for the PAQ approach to mixing and even a generalization. Elementary modeling and mixing on their own do not make up a statistical data compressor, rather the way these components interact. A statistical data compressor may be equipped with PAQ-style elementary modeling and mixing to improve its empirical performance over classical methods and still retain theoretic guarantees. To demonstrate this we introduced CTM, a generalization of CTW that allows for arbitrary elementary models and mixers. If these components are drawn from a sufficiently rich class, then our results on CTM show that it enjoys code length guarantees stronger than those of CTW. These guarantees certify the ability to perform not much worse than a sequence of competing PCTs. We equip CTM with a variation of PS elementary model and Geometric OGDM for mixing, which gives birth to the PAQ CTM statistical compressor. (Note these techniques are the PAQ approaches to elementary modeling and mixing.) PAQ CTM outperforms variants of CTW and enjoys low redundancy w. r. t. a sequence of PCTs.

For all of the approaches from the “practitioner’s claim” we considered in this work we were able to determine a theoretical justification. In addition, experiments indicate that PAQ-style elementary modeling and mixing outclasses classical approaches, even with all of PAQ’s heuristics and bells and whistles cut off. Nevertheless, there is still work to take up:

- Our results on the family of PS and RFS elementary models only hold for a binary alphabet. A generalization of these results to an alphabet of arbitrary size should be part of future research.

- The code length analysis of PS and RFS is based on an input that maximizes the redundancy w. r. t. the empirical entropy. Since candidates for the worst-case input are known, it seems worthwhile to precisely characterize the worst-case redundancy in the Θ-sense.

- In general, neither code length analysis (for elementary models and mixers) takes the similarity of adjacent input segments into account: Suppose an elementary model (mixer) performs well w. r. t. its competitor for some input segment $x_{a:b}$, since it obtained a good estimate on the locally optimal distribution (weight vector). Consequently, the elementary model (mixer) should also perform well in the segment $x_{b+1:c}$, if the locally optimal distribution (weight vector) is close to that of the former segment $x_{a:b}$. Future research should enhance code length guarantees to take this effect into account.

- Based on further mathematical properties of code length, e. g. strong convexity, we might be able to enhance the code length guarantees for OGDM.

- As we have seen, PPM and CTM (or CTW) share great structural similarity. We conjecture that based on assumptions on the escape probability assignment of PPM we should be able to adopt our methods for the analysis of CTM to obtain code length guarantees for PPM. This is especially exciting, since in the deterministic setting there is no analysis of any PPM variant available.

- PAQ utilizes a variety of other ad-hoc techniques to improve its compression. Most notably SSE and FSMs for probability estimation. To further bridge the gap between theory and practice we consider a theoretical justification of these approaches for future research.

By the results of this thesis we made first steps towards closing the gap between theory and practice in statistical data compression. But still, there is a lot of work to do to carry on this process.

List of Abbreviations

AC	Arithmetic Coding
BPS	Bounded Probability Smoothing
CTM	Context Tree Mixing
CTS	Context Tree Switching
CTW	Context Tree Weighting
DEPLUMP	PLUMP abbreviates Power Law Unbounded Markov Prediction
DMC	Dynamic Markov Coding
FSM	Finite State Machine
ISW	Imaginary Sliding Window
KT	Krichevsky Trofimov (Estimator)
LM	Logistic Mixing
LP	Laplace (Estimator)
OCP	Online Convex Programming
OGD	Online Gradient Descent
OGDM	Online Gradient Descent Mixer
ONS	Online Newton Step
PAQ	“pack”
PCT	Prediction Context Tree
PPM	Prediction by Partial Matching
PS	Probability Smoothing
PTW	Partition Tree Weighting
PWS	Piecewise Stationary Model
RFD	Relative Frequencies with Discount
RFS	Relative Frequencies with Smoothing
SM	Sequence Memorizer
SWM	Switching Mixer
ZR	Zero-Redundancy (Estimator)

List of Figures

List of Tables

Bibliography

[1] "7zip Format," http://www.7-zip.org/7z.html, accessed: 2014-07-01.

[2] "WinZIP Additional Compression Methods Specification," http://www.winzip.com/comp_info.htm, accessed: 2014-07-01.

[3] "Algorithms for adaptive huffman codes," *Information Processing Letters*, vol. 18, no. 3, pp. 159–165, 1984.

[4] J. Åberg, "A universal source coding perspective on PPM," Ph.D. dissertation, 1999.

[5] J. Åberg, Y. M. Shtarkov, and B. Smeets, "Estimation of escape probabilities for PPM based on universal source coding theory," in *Proc. IEEE International Symposium on Information Theory*, 1997, pp. 65–65.

[6] S. Azhar, G. J. Badros, A. Glodjo, M.-Y. Kao, and J. H. Reif, "Data Compression Techniques for Stock Market Prediction." in *Proc. IEEE Data Compression Conference*, vol. 4, 1994, pp. 72–82.

[7] N. Bartlett and F. Wood, "Deplump for Streaming Data," in *Proc. IEEE Data Compression Conference*, vol. 21, 2011, pp. 363–372.

[8] R. Begleiter and R. El-Yaniv, "Superior guarantees for sequential prediction and lossless compression via alphabet decomposition," *The Journal of Machine Learning Research*, vol. 7, pp. 379–411, 2006.

[9] T. Bell and A. Moffat, "A Note on the DMC Data Compression Scheme," *The Computer Journal*, vol. 32, pp. 16–20, 1989.

[10] M. G. Bellemare, J. Veness, and E. Talvitie, "Skip Context Tree Switching," in *Proc. International Conference on Machine Learning*, vol. 31, 2014, pp. 1458–1466.

[11] D. P. Bertsekas, "Incremental gradient, subgradient, and proximal methods for convex optimization: A survey," *Optimization for Machine Learning*, vol. 2010, pp. 1–38.

[12] C. Bloom, "Solving the problems of context modeling," *informally published report, see http://www.cbloom.com/papers*, 1998.

[13] A. Bratko, B. Filipič, G. V. Cormack, T. R. Lynam, and B. Zupan, "Spam Filtering Using Statistical Data Compression Models," *Journal of Machine Learning Research*, vol. 7, pp. 2673–2698, 2006.

[14] S. Bunton, "The Structure of DMC," in *Proc. IEEE Data Compression Conference*, vol. 5, 1995, pp. 72–81.

[15] ——, "On-Line Stochastic Processes in Data Compression," Ph.D. dissertation, Washington, USA, 1996.

[16] N. Cesa-Bianchi, "Analysis of two gradient-based algorithms for on-line regression," in *Proc. Annual Conference on Computational Learning Theory*, vol. 10.

[17] N. Cesa-Bianchi, P. M. Long, and M. K. Warmuth, "Worst-case Quadratic Loss Bounds for On-line Prediction of Linear Functions by Gradient Descent," *IEEE Transactions on Neural Networks*, vol. 7, pp. 604–619, 1993.

[18] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*, 1st ed. Cambridge University Press, 2006.

[19] I.-C. K. Chen, J. T. Coffey, and T. N. Mudge, "Analysis of Branch Prediction via Data Compression," in *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*, vol. 7.

[20] R. Cilibrasi, P. Vitányi, and R. De Wolf, "Algorithmic Clustering of Music Based on String Compression," *Computer Music Journal*, vol. 28, no. 4, pp. 49–67, 2004.

[21] R. Cilibrasi and P. M. B. Vitányi, "Clustering by compression," *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1523–1545, 2005.

[22] J. G. Cleary, W. J. Teahan, and I. H. Witten, "Unbounded length contexts for ppm," in *Proc. IEEE Data Compression Conference*, vol. 5, 1995, pp. 52–61.

[23] J. G. Cleary and I. H. Witten, "Data Compression Using Adaptive Coding and Partial String Matching," *IEEE Transactions on Communications*, vol. 32, pp. 396–402, 1984.

[24] A. R. Cohen and P. M. B. Vitányi, "Normalized Compression Distance of Multisets with Applications," *CoRR*, vol. abs/1212.5711, 2013.

[25] G. V. Cormack and R. N. S. Horspool, "Data Compression Using Dynamic Markov Modelling," *The Computer Journal*, vol. 30, pp. 541–550, 1987.

[26] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Wiley-Interscience, 2006.

[27] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, "Efficient projections onto the l 1-ball for learning in high dimensions," in *Proc. of the International Conference Machine Learning*, vol. 25, 2008, pp. 272–279.

[28] J. Duda, "Optimal encoding on discrete lattice with translational invariant constrains using statistical algorithms," *CoRR*, vol. abs/0710.3861, 2007. [Online]. Available: http://arxiv.org/abs/0710.3861

[29] ——, “Asymmetric numeral systems,” *CoRR*, vol. abs/0902.0271, 2009. [Online]. Available: http://arxiv.org/abs/0902.0271

[30] ——, “Asymmetric numeral systems as close to capacity low state entropy coders,” *CoRR*, vol. abs/1311.2540, 2013. [Online]. Available: http://arxiv.org/abs/1311.2540

[31] N. Faller, “An adaptive system for data compression,” *Asilomar Conference on Circuits, Systems and Computers*, vol. 7, pp. 593–597, 1973.

[32] E. Frank, C. Chui, and I. H. Witten, “Text categorization using compression models,” in *Proc. IEEE Data Compression Conference*, vol. 10, 2000, pp. 200–209.

[33] R. G. Gallager, “Variations on a theme by Huffman,” *IEEE Transactions on Information Theory*, vol. 24, no. 6, pp. 668–674, 1978.

[34] J. Gasthaus, F. Wood, and Y. W. Teh, “Lossless compression based on the Sequence Memoizer,” in *Proc. IEEE Data Compression Conference*, vol. 20, 2010, pp. 337–345.

[35] C. Genest and J. V. Zidek, “Combining probability distributions: A critique and an annotated bibliography,” *Statistical Science*, pp. 114–135, 1986.

[36] E. Hazan, “Efficient algorithms for online convex optimization and their applications,” Ph.D. dissertation, 2006.

[37] E. Hazan, A. Kalai, S. Kale, and A. Agarwal, “Logarithmic Regret Algorithms for Online Convex Optimization,” in *Conference on Learning Theory*, vol. 19, 2006, pp. 499–513.

[38] E. Hazan and C. Seshadhri, “Efficient Learning Algorithms for Changing Environments,” in *Proc. of the International Conference Machine Learning*, vol. 26, 2009, pp. 393–400.

[39] D. P. Helmbold, J. Kivinen, and M. K. Warmuth, “Relative loss bounds for single neurons,” *IEEE Transactions on Neural Networks*, vol. 10, pp. 1291–1304, 1999.

[40] G. E. Hinton, “Products of experts,” in *International Conference on Artificial Neural Networks*, vol. 9, 1999, pp. 1–6.

[41] ——, “Training products of experts by minimizing contrastive divergence,” *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[42] P. G. Howard and J. S. Vitter, “Analysis of arithmetic coding for data compression,” in *Proc. IEEE Data Compression Conference*, vol. 1, 1991, pp. 3–12.

[43] ——, “Practical Implementations of Arithmetic Coding,” Brown University, Providence, USA, Tech. Rep., 1992.

[44] A. Ingber and M. Feder, “Prediction of individual sequences using universal deterministic finite state machines,” in *Proc. IEEE International Symposium on Information Theory*, 2006, pp. 421–425.

[45] D. Joho and W. Burgard, “Searching for objects: Combining multiple cues to object locations using a maximum entropy model,” in *IEEE International Conference on Robotics and Automation*, 2010, pp. 723–728.

[46] M. Khodadadzadeh, J. Li, A. Plaza, P. Gamba, J. A. Benediktsson, and J. M. Bioucas-Dias, “A new framework for hyperspectral image classification using multiple spectral and spatial features,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2014 IEEE International*, 2014, pp. 4628–4631.

[47] S. S. Kozat, A. C. Singer, and G. C. Zeitler, “Universal Piecewise Linear Prediction Via Context Trees,” *IEEE Transactions on Signal Processing*, vol. 55, no. 7, pp. 3730–3745, 2007.

[48] R. E. Krichevsky and V. K. Trofimov, “The performance of universal encoding,” *IEEE Transactions on Information Theory*, vol. 27, no. 2, pp. 199–207, 1981.

[49] M. Kufleitner, E. Binder, and A. Fries, “Combining Models in Data Compression,” in *Proc. Symposium on Information Theory in the Benelux*, vol. 30, 2009, pp. 135–142.

[50] G. G. Langdon, “An Introduction to Arithmetic Coding,” *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, 1984.

[51] M. Li, X. Chen, X. Li, B. Ma, and P. M. Vitányi, “The similarity metric,” *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 3250–3264, 2004.

[52] M. Mahoney, “Data Compression Explained,” http://mattmahoney.net/dc/dce.html, accessed: 2014-07-01.

[53] ——, “The PAQ Data Compression Programs,” http://mattmahoney.net/dc/paq.html, accessed: 2015-03-09.

[54] M. V. Mahoney, “The PAQ1 Data Compression Program,” http://mattmahoney.net/dc/paq1.pdf, accessed: 2014-07-01.

[55] ——, “Fast Text Compression with Neural Networks,” in *FLAIRS Conference*, 2000, pp. 230–234.

[56] ——, “Adaptive Weighing of Context Models for Lossless Data Compression,” Florida Tech., Melbourne, USA, Tech. Rep., 2005.

[57] G. Martin, “Range encoding: an algorithm for removing redundancy from a digitised message,” in *Data Recording Conference*, 1979.

[58] C. Mattern, “Mixing strategies in data compression,” in *Proc. IEEE Data Compression Conference*, vol. 22, 2012, pp. 337–346.

[59] ——, “Linear and Geometric Mixtures — Analysis,” in *Proc. IEEE Data Compression Conference*, vol. 23, 2013, pp. 301–310.

[60] ——, “On Probability Estimation by Exponential Smoothing,” in *Proc. IEEE Data Compression Conference*, vol. 25, 2015, p. 460.

[61] ——, “On Probability Estimation via Relative Frequencies and Discount,” in *Proc. IEEE Data Compression Conference*, vol. 25, 2015, pp. 183–192.

[62] G. Mayraz and G. E. Hinton, “Recognizing handwritten digits using hierarchical products of experts,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 189–197, 2002.

[63] N. Merhav, “On the minimum description length principle for sources with piecewise constant parameters,” *IEEE Transactions on Information Theory*, vol. 39, no. 6, pp. 1962–1967, 1993.

[64] N. Merhav and M. Feder, “Universal prediction,” *IEEE Transactions on Information Theory*, vol. 44, pp. 2124–2147, 1998.

[65] E. Meron and M. Feder, “Finite-memory universal prediction of individual sequences,” *IEEE Transactions on Information Theory*, vol. 50, no. 7, pp. 1506–1523, 2004.

[66] ——, “Optimal finite state universal coding of individual sequences,” in *Proc. IEEE Data Compression Conference*, vol. 14, 2004, pp. 332–341.

[67] A. Moffat, “Implementing the PPM data compression scheme,” vol. 38, no. 11, 1990, pp. 1917–1921.

[68] A. Moffat and A. Turpin, *Compression and Coding Algorithms*, 1st ed. Kluwer, 2002.

[69] A. O’Neill, M. Hutter, W. Shao, and P. Sunehag, “Adaptive context tree weighting,” in *Proc. IEEE Data Compression Conference*, vol. 22, 2012, pp. 317–326.

[70] R. C. Pasco, “Source Coding Algorithms for Fast Data Compression.” Ph.D. dissertation, Stanford, USA, 1976.

[71] D. Rajwan and M. Feder, “Universal finite memory machines for coding binary sequences,” in *Proc. IEEE Data Compression Conference*, vol. 10, 2000, pp. 113–122.

[72] R. Ranawana and V. Palade, “A neural network based multi-classifier system for gene identification in dna sequences,” *Neural Computing & Applications*, vol. 14, no. 2, pp. 122–131, 2005.

[73] J. Rissanen, “A universal data compression system,” *IEEE Transactions on Information Theory*, vol. 29, no. 5, pp. 656–664, 1983.

[74] J. J. Rissanen, “Generalized Kraft Inequality and Arithmetic Coding,” *IBM Journal of Research and Development*, vol. 20, no. 3, pp. 198–203, 1976.

[75] R. Rojas, *Neural Networks: A Systematic Introduction*. Springer Science & Business Media, 2013.

[76] B. Ryabko, “Compression-based Methods for Nonparametric Prediction and Estimation of Some Characteristics of Time Series,” *IEEE Transactions on Information Theory*, vol. 55, no. 9, pp. 4309–4315, 2009.

[77] B. Ryabko and A. Fionov, "Fast and Space-Efficient Adaptive Arithmetic Coding," in *Proc. IMA International Conference*, 1999, pp. 270–279.

[78] K. Sadakane, T. Okazaki, and H. Imai, "Implementing the Context Tree Weighting Method for Text Compression," in *Proc. IEEE Data Compression Conference*, vol. 10, 2000, pp. 123–132.

[79] A. Said, "Introduction to Arithmetic Coding - Theory and Practice," *HP Laboratories Report*, 2004.

[80] D. Salomon and G. Motta, *Handbook of Data Compression*, 1st ed. Springer, 2010.

[81] K. Sayood, *Introduction to Data Compression*, 4th ed. Elsevier, 2012.

[82] M. Schindler, "A Fast Renormalisation for Arithmetic Coding," in *Proc. IEEE Data Compression Conference*, vol. 8, p. 572.

[83] G. I. Shamir and N. Merhav, "Low-complexity sequential lossless coding for piecewise-stationary memoryless sources," *IEEE Trans. on Information Theory*, vol. 45, pp. 1498–1519, 1999.

[84] D. Shkarin, "PPM: one step to practicality," in *Proc. IEEE Data Compression Conference*, vol. 12, 2002, pp. 202–211.

[85] C. Steinruecken, "Lossless Data Compression," Ph.D. dissertation, London, England, 2015.

[86] P. Sunehag, W. Shao, and M. Hutter, "Coding of non-stationary sources as a foundation for detecting change points and outliers in binary time-series," in *Proce. Australasian Data Mining Conference*, vol. 10, 2012, pp. 79–84.

[87] Y. W. Teh and G. E. Hinton, "Rate-coded restricted boltzmann machines for face recognition," *Advances in neural information processing systems*, pp. 908–914, 2001.

[88] T. J. Tjalkens, P. A. Volf, and F. M. Willems, "A Context-Tree Weighting Method for Text Generating Sources," in *Proc. IEEE Data Compression Conference*, vol. 7, 1997, p. 472.

[89] J. Tuehola and T. Raita, "Application of a finite-state model to text compression," *The Computer Journal*, vol. 36, pp. 607–614, 1993.

[90] J. Veness, M. G. Bellemare, M. Hutter, A. Chua, and G. Desjardins, "Optimal encoding on discrete lattice with translational invariant constrains using statistical algorithms," *CoRR*, vol. abs/0710.3861, 2007. [Online]. Available: http://arxiv.org/abs/0710.3861

[91] J. Veness, K. S. Ng, M. Hutter, and M. H. Bowling, "Context Tree Switching," in *Proc. IEEE Data Compression Conference*, vol. 22, 2012, pp. 327–336.

[92] J. Veness, M. White, and M. Bowling, "Partition Tree Weighting," in *Proc. IEEE Data Compression Conference*, vol. 23, 2013, pp. 327–336.

[93] P. M. Vitányi, “Compression-based Similarity,” in *Proc. Data Compression, Communication and Processing*, vol. 1, 2011, pp. 111–118.

[94] P. A. Volf, “Weighting Techniques in Data Compression: Theory and Algorithms.” Ph.D. dissertation, Eindhoven, The Netherland, 2002.

[95] P. A. Volf and F. M. Willems, “Switching between two universal source coding algorithms,” in *Proc. IEEE Data Compression Conference*, vol. 8, 1998, pp. 491–500.

[96] ——, “The switching method: elaborations,” in *Proc. Symposium on Information Theory in the Benelux*, vol. 19, 1998, pp. 13–20.

[97] F. M. Willems, “Coding for a binary independent piecewise-identically-distributed source,” *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 2210–2217, 1996.

[98] ——, “The context-tree weighting method: extensions,” *IEEE Transactions on Information Theory*, vol. 44, pp. 792–798, 1998.

[99] F. M. Willems and M. Krom, “Live-and-die coding for binary piecewise i.i.d. sources,” in *Proc. IEEE International Symposium on Information Theory*, 1997, pp. 68–68.

[100] F. M. Willems, Y. M. Shtarkov, and T. J. Tjalkens, “Context Tree Weighting: Multi-alphabet Sources,” *IEEE Transactions on Information Theory*, vol. 14, pp. 128–135, 1993.

[101] ——, “The context-tree weighting method: basic properties,” *IEEE Transactions on Information Theory*, vol. 41, pp. 653–664, 1995.

[102] ——, “Reflections on “The Context Tree Weighting Method: Basic properties”,” *Newsletter of the IEEE Information Theory Society*, vol. 47, no. 1, 1997.

[103] F. M. Willems and T. J. Tjalkens, *Complexity Reduction of the Context-Tree Weighting Algorithm: A Study for KPN Research*. Euler Institute of Discrete Mathematics and its Applications, 1997.

[104] ——, “Reducing the complexity of the context-tree weighting method,” in *Proc. International Symposium on Information Theory*, 1998, p. 347.

[105] F. M. Willems, T. J. Tjalkens, and T. Ignatenko, “Context-tree weighting and maximizing: Processing betas,” in *Proc. Inaugural Workshop of the Information Theory and its Applications*, 2006.

[106] I. H. Witten and T. Bell, “The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression,” *IEEE Transactions on Information Theory*, vol. 37, no. 4, pp. 1085–1094, 1991.

[107] I. H. Witten, R. M. Neal, and J. G. Cleary, “Arithmetic Coding for Data Compression,” *Commununications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[108] M. Zinkevich, “Online Convex Programming and Generalized Infinitesimal Gradient Ascent,” in *Proc. of the International Conference Machine Learning*, vol. 20, 2003, pp. 928–936.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Bei der Auswahl und Auswertung folgenden Materials haben mir die nachstehend aufgeführten Personen in der jeweils beschriebenen Weise unentgeltlich geholfen:

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch bewertet wird und gemäß Paragraph 7 Abs. 10 der Promotionsordnung den Abbruch des Promotionsverfahrens zur Folge hat.

Ort, Datum Unterschrift