École polytechnique fédérale de Lausanne

Master thesis spring 2024

Master of Science in APPLIED MATHEMATICS

Neural Data Compression

Author:
Paul Bagourd

Supervisor:
Vassilis Papadopoulos

Professor in charge:
Clément Hongler



Table of contents

1 Introduction **1**

2 Common notation **2**

3 Information theory **3**
3.1 Probability theory . . . 3
3.2 Source coding . . . 4
3.3 Entropy coding . . . 5
3.4 Kolmogorov complexity . . . 11

4 Arithmetic Coding **14**
4.1 Encoding . . . 14
4.2 Decoding . . . 15
4.3 Optimality . . . 16
4.4 Practical implementation . . . 17
4.5 Fixed-precision arithmetic . . . 17
4.6 Running time . . . 18

5 ANS coding **19**
5.1 A matter of representation . . . 19
5.2 Symmetric Numeral Systems . . . 20
5.3 Encoding . . . 20
5.4 Decoding . . . 22
5.5 Practical implementation . . . 23
5.6 Running time . . . 23
5.7 Optimality . . . 23
5.8 Comparison with AC coding . . . 24

6 Source Modelling **27**
6.1 Mathematical Modeling . . . 27
6.2 Artificial Neural Networks . . . 27
6.2.1 Single Artificial Neuron . . . 27
6.2.2 Feedforward Neural Network (FNN) . . . 28
6.2.3 Learning a probability distribution . . . 29
6.3 Predictive models . . . 30
6.4 Variational Auto-Encoders . . . 30

7 Bits-back coding and chaining **35**
7.1 Bits-back coding . . . 35
7.2 The need for randomness . . . 38
7.3 Practical implementation . . . 39
7.3.1 Clipping everywhere . . . 40
7.3.2 Symbolisation . . . 41
7.3.3 Graph summary . . . 42

8 Implementation results **44**
8.1 AC implementation 44
8.1.1 Language comparison 44
8.1.2 Context comparison 46
8.1.3 Model size comparison 48
8.1.4 Image comparison 49
8.2 BB-ANS implementation 50

9 Conclusion **52**

1 Introduction

Data compression is quite recent compared to other information processing technologies. Cryptography, for instance, goes back to antiquity with the famous Caesar cipher. Of course, lossy data compression has always been implicitly used, from transcribing important concepts to establishing insightful maps of territories or even lossless compression when transmitting signals during navigation. However, it was only with the invention of the telegraph in 1837 that the questions about how to encode information succinctly began with the creation of Morse code. As communication and the amount of information have increased exponentially during the computer science revolution, it has become essential to reduce, simplify and store information suitably to spread it or receive it seamlessly. In this context, data compression grew as a major domain of research.

Although more ancient, data modelling also followed the same exponential path due to exponential growth computation power. Closely linked to artificial intelligence, it has made huge progress in the last decade. This is explained by using artificial neural networks that work surprisingly well to fit any function. Intuitively, we could say that modelling and compressing data are two faces of the same coin: if we know how to compress data, then we implicitly understand its underlying patterns. Conversely, if we smoothly model data, we have implicitly learned its underlying regularities and can consequently provide the essential information.

With this idea in mind, we call neural compression methods any methods that use neural networks to smoothly model and compress data. This thesis presents how two specific neural compression methods perform by considering theoretical and practical perspectives. We begin by setting the framework of information theory necessary to define information coding and quantify data compression. We then explain with practical considerations Arithmetic Coding (AC) and Asymmetric Numeral System coding (ANS), which are two functionally different optimal coding schemes that can work with different probabilistic models. For this purpose, we introduce several methods to model information sources, focusing on variational auto-encoders and a specific latent variable model. This model sets the ground to implement bits-back coding that goes back to the 80s [HVC93][Wal90]. This method encodes data using a latent variable model as an intermediate if the sender accepts sending random auxiliary bits that the receiver can decode and transmit back. Although not recent, this method got more attention in the realm of data compression as some attempts were made to use a continuous latent space and obtain promising results, namely the BB-ANS coding [TBB19]. From this perspective, we provide details of bits-back coding with a theoretical approach and a method for practical implementation. In the last section, we present the results of our implementations regarding AC and BB-ANS, hoping to give some experimental evidence supporting the theory.

2 Common notation

In this section, we give some common notations used in this work. We tried staying consistent across the different sections using an intuitive notation.

- $X^n \coloneqq X_1, \ldots, X_n$ stands for any sequence of symbols indexed from 1 to n, for $n \in \mathbb{N}$.
- For any probability distribution p_X with corresponding random variable X, we denote

$$p_X(x^n) = p_X(x_1, \ldots, x_n) = Pr[X_1 = x_1, \ldots, X_n = x_n]$$

  or simply $p(x^n)$ when the context is clear
- Unless specified otherwise, the binary logarithm $\log_2$ will be denoted by log.
- Throughout this work, we use the standard asymptotic notation to describe and compare the order of growth of functions. For any positive real-valued functions $f(n)$ and $g(n)$ we write:
  - $f = O(g)$: f grows no faster than g, i.e., there exists an n_0 and a constant $c > 0$ such that for all $n > n_0$, $f(n) \leq cg(n)$.
  - $f = o(g)$, if $\lim_{n\to\infty} f(n)/g(n) = 0$.
  - $f = \Omega(g)$, if $g = O(f)$.
  - $f = \omega(g)$, if $g = o(f)$.
  - $f = \Theta(g)$, if $f = O(g)$ and $g = O(f)$.
- For a set E, E^* denotes the set of all finite sequences of symbols in E.

3 Information theory

In this section, we present the important concepts of information theory that will be essential for quantifying compression. The birth of information theory is often attributed to Shannon with his 1948 seminal paper [Sha48]. Nevertheless, information theory is deeply rooted in the realm of probability theory, which we introduce here.

3.1 Probability theory

Let us begin with some basic definitions.

Definition 1 (σ-algebra) $\mathcal{B}$ is a σ-algebra of subsets of a set Ω if it is a nonempty collection of subsets of Ω closed under complement, countable unions, and countable intersections.

Definition 2 (Measurable space) The ordered pair $(\Omega, \mathcal{B})$ is called a measurable space.

Definition 3 (Probability measure) A probability measure Pr on $\mathcal{F}$ is a function $Pr : \mathcal{F} \to [0, 1]$ such that:

- if $\{A_i\}_{i=1}^{\infty} \subseteq \mathcal{F}$ is a countable collection of pairwise disjoint sets, then

$$Pr(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} Pr(A_i)$$

- the measure of the entire sample space is equal to one: $Pr(\Omega) = 1$

Remark 1 We also call Pr the probability distribution or probability function.

Definition 4 (Probability space) Let (Ω, $\mathcal{B}$) be a measurable space and Pr be a probability measure on some universe. The tuple (Ω, $\mathcal{B}$, Pr) is called a probability space

Definition 5 (Random variable) Let Ω be a set. A random variable X is defined as a function $X(\omega)$ on a probability space for any $\omega \in \Omega$.

Definition 6 (Probability of a random variable X) For any random variable X, we will denote by $p_X(x)$ the probability $Pr(X = x)$ associated to X for any event $\{X(\omega) = x\}$ where $\omega \in \Omega$.

Remark 2 When the context is clear, we will simply denote $p(x)$ for $p_X(x)$.

3.2 Source coding

We now get to specific definitions of information theory.

Definition 7 (Stochastic process) Let T be a set. A stochastic process $\{X_t\}_{t\in T}$ is an indexed sequence of random variables. T is called the *index set* and X_t can take values from a mathematical set $\mathcal{X}_t$ possibly depending on t and called the *state space*.

Definition 8 (Source of information) A source of information X is any stochastic process that generates a sequence of symbols called the *source symbols* according to some probability distribution.

Definition 9 (Source alphabet) The source alphabet $\mathcal{X}$ is the set of possible source symbols.

Example 1 In English, the source alphabet consists of letters and spaces, with certain letters appearing more than others.

Definition 10 (Discrete source) A source is discrete if its source alphabet is discrete

Definition 11 (Encoder) An encoder Enc is any function

$$Enc\colon \mathcal{X}^* \longrightarrow \mathcal{C}^*$$
$$s^n \mapsto c^{n'}$$

that maps any sequence from the source alphabet $\mathcal{X}$ to a corresponding sequence in the code alphabet $\mathcal{C}$

The code alphabet can be composed of arbitrary and conventional symbols. However, from now on, we will consider only the set of binary digits $\{0, 1\}$ called the bits, a term coined by statistician John Wilder Tukey in 1948.

Definition 12 (Decoder) A Decoder Dec is any function

$$Dec\colon \mathcal{C}^* \longrightarrow \mathcal{X}^*$$
$$c^{n'} \mapsto s^n$$

that maps any sequence from the code alphabet $\mathcal{C}$ code symbols to a corresponding sequence in the source alphabet $\mathcal{X}$

Definition 13 (Source coding) Source coding is the mapping of a source symbol sequence to an alphabet symbols sequence by some encoder such that the source symbols can be exactly recovered from the sequence (lossless source coding) or recovered within some distortion (lossy source coding) by some decoder.

Definition 14 (Code) If the pair (Enc, Dec) achieves source coding, it is called a code.

Definition 15 (Symbol code) A symbol code is a code that maps each symbol from the source alphabet individually, independently from the context in which they appear. An encoded symbol is then called a *codeword*.

Example 2 An important class of symbol codes is uniquely decodable codes. This means that every encoded string is the code one and only one string from the source symbol. In fact, even if codewords are unique, it could be possible that several sequences of source symbols lead to the same encoded string after concatenation.

Definition 16 (Stream code) Contrarily to symbol codes, a stream code encodes the next symbol depending on the previous symbols that appeared in the sequence, leveraging the fact that the symbols may be correlated.

Definition 17 (Compression ratio) Let $S \in \mathcal{X}^*$ and (Enc, Dec) a code. The compression ratio for S is defined as

$$R(S) = \frac{|S|}{|Enc(S)|}$$

3.3 Entropy coding

Entropy is a central notion in information theory. It dates back to the 19th century as a measure of the degree of disorder or randomness in a system, The second law of thermodynamics states that in an isolated system, entropy tends to increase over time, leading to the concept of irreversibility in natural processes. This physical concept inspired Claude Shannon to formalise the notion of information in the middle of the 20th century.

Definition 18 (Discrete Shannon entropy) Let X be a random variable with probability function p. The Shannon entropy of X is

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x) = \mathbb{E}_{X \sim p} [\log_2(X)]$$

Definition 19 (Differential Shannon entropy) If the domain $\mathcal{X}$ of X is continuous, we can generalise the definition of Shannon entropy to

$$H(X) = -\int_{x \in \mathcal{X}} p(x) \log_2 p(x) = \mathbb{E}_{X \sim p} [\log_2(X)]$$

Definition 20 (Convergence of random variables) Given a sequence of random variables, $X_1, X_2, \ldots$ we say that the sequence $X_1, X_2, \ldots$ converges to a random variable X:

1. In *probability* if for every $\epsilon > 0, lim_{n\to+\infty} Pr(|X_n - X| > \epsilon) = 0$
2. In *mean square* if $lim_{n\to+\infty} \mathbb{E}_{X\sim p}\left[(X_n - X)^2\right] = 0$
3. *Almost surely* if $Pr(lim_{n\to+\infty} X_n = X) = 1$

Definition 21 (Typical set) Let X be an i.i.d. source with alphabet $\mathcal{X}$ and probability distribution p. The ϵ-typical set up to time n is defined as follows:

$$A_n^\epsilon = \left\{(x_1, \ldots, x_n) \in \mathcal{X}^n : |-\frac{1}{n}\log p(x_1, \ldots, x_n) - H(X)| < \epsilon\right\}$$

Definition 22 (Discrete Memoryless Source (DMS)) A discrete Memoryless Source (DMS) is a source of information X such that $(X_i)_{i\in\mathbb{N}}$ are i.i.d.

Lemma 1 (Week Law of Large Numbers) *Let X be a DMS with $\mathbb{E}_{X\sim p}[X] = \mu < \infty$ then for any $\epsilon > 0$*

$$lim_{n\to+\infty} Pr\left(|\frac{X_1 + \cdots + X_n}{n} - \mu| > \epsilon\right) = 0$$

Proof Let us define $Y_n = \frac{X_1+\cdots+X_n}{n}$. Assuming the variance of X is finite, the proof follows from Chebyshev's inequality applied on the Y_n □

Lemma 2 (Asymptotic equipartition property (AEP)) *Let X be a DMS, then for any $\epsilon > 0$*

$$lim_{n\to+\infty} Pr((X_1, \ldots, X_n) \notin A_n^\epsilon) = 0$$

or equivalently

$$-\frac{1}{n}\log Pr(X_1, \ldots, X_n) \to H(X) \text{ in probability}$$

Proof Since

$$H(X) = \mathbb{E}_{X\sim p}[-\log_2 p(X)] < +\infty$$

AEP directly follows from the Weak Law of Large Numbers that we apply in the last equation :

$$\begin{aligned}
&lim_{n\to+\infty} Pr((X_1, \ldots, X_n) \notin A_n^\epsilon)\\
=&lim_{n\to+\infty} Pr\left(|-\frac{1}{n}\log_2 p(X^n) - H(X)| \geq \epsilon\right)\\
=&lim_{n\to+\infty} Pr\left(|\frac{-\log_2 p(X_1) + \cdots + (-\log_2 p(X_n))}{n} - H(X)| \geq \epsilon\right)\\
=&0
\end{aligned}$$

for all $\epsilon > 0$ □

Theorem 3 (Shannon's source coding theorem (1948)) *For any i.i.d. source X and lossless coding scheme (enc, dec), the expected length of an encoded string $enc(X^n)$ asymptotically satisfies*

$$lim_{n\to+\infty}\, \mathbb{E}_{X\sim p}\left[\frac{l(enc(X^n))}{n}\right] \geq H(X)$$

Moreover, this bound is achievable. For every $\epsilon > 0$, there exists $n_\epsilon \in \mathbb{N}$ sufficiently large and a lossless coding scheme $(enc, dec)_\epsilon$ such that for all $n \geq n_\epsilon$

$$\mathbb{E}_{X\sim p}\left[\frac{l(enc(X^n))}{n}\right] = H(X) + \epsilon$$

Remark 3 The original formulation of the source coding theorem given by Shannon in [Sha48] is not exactly the one we give in this paper, but the result is essentially the same.

Proof Let us first prove some properties of the typical set. We use the notation x^n to denote a sequence $x_1, x_2, \dots x_n$. Let $l(x^n)$ be the length of the codeword corresponding to x^n. Let us fix $\epsilon > 0$. By definition of A_n^ϵ, for any $x^n \in A_n^\epsilon$

$$2^{-n(H(X)+\epsilon)} \leq p(x^n) \leq 2^{-n(H(X)-\epsilon)} \tag{1}$$

We can show the following points

- $p((X_1, \dots, X_n) \in A_n^\epsilon) > 1-\epsilon$ for all $\epsilon > 0$ and sufficiently large n_ϵ, following from AEP property (Lemma 2)
- $|A_n^\epsilon| \leq 2^{n(H(X)+\epsilon)}$, which follows from lower bound in (1):

$$1 = \sum_{x\in\mathcal{X}^n} p(x) \geq \sum_{x\in A_n^\epsilon} p(x) \geq \sum_{x\in A_n^\epsilon} 2^{-n(H(X)+\epsilon)} = 2^{-n(H(X)+\epsilon)}|A_n^\epsilon|$$

- $|A_n^\epsilon| \geq (1-\epsilon)2^{n(H(X)-\epsilon)}$:

$$1 - \epsilon < Pr(A_n^\epsilon) \leq \sum_{x\in A_n^\epsilon} 2^{-n(H(X)-\epsilon)} = 2^{-n(H(X)-\epsilon)}|A_n^\epsilon|$$

For achievability, we follow the construction given in [Cov99] page 87 and let the reader check the end of the proof if interested. We order all elements in each set according to some arbitrary order. Then, we can represent each sequence of A_n^ϵ by giving the index of the sequence in the set. Since

$$|A_n^\epsilon| \leq 2^{n(H(X)+\epsilon)}$$

the indexing requires no more than $\lceil n(H+\epsilon)+1\rceil$ bits, the extra bit being necessary because $n(H+\epsilon)$ may not be an integer. We prefix all these sequences by a 0, giving a total length upper bounded by $n(H+\epsilon)+2$ bits to represent each sequence in A_n^ϵ Similarly, we can index each sequence not in A_n^ϵ by using not more than $n\log|\mathcal{X}|+1$ bits. Prefixing these indices by 1, we have a code for all the sequences in x^n. We have the following features:

- The code is one-to-one and easily decodable. The initial bit acts as a flag bit to indicate the length of the codeword that follows.
- We have used a brute-force enumeration of the atypical set $x^n \setminus A_n^\epsilon$ without taking into account the fact that it is actually smaller than A_n^ϵ
- Surprisingly, this is good enough to yield an efficient description.
- The typical sequences have short descriptions of length approximately nH.

We prove optimality for uniquely decodable codes. For general stream codes, the proof follows from the proof of Theorem 3 presented below. We use Kraft-McMillan inequality for uniquely decodable code in the case of a binary code alphabet. The inequality says:

$$\sum_{x \in \mathcal{X}} 2^{-l(Enc(x))} \leq 1$$

Since the source is memoryless, we also have

$$\mathbb{E}_{X \sim p}\left[\frac{l(enc(X^n))}{n}\right] = \sum_{x \in \mathcal{X}} p(x) l(enc(x))$$

Hence

$$H(X) - \mathbb{E}_{X \sim p}\left[\frac{l(enc(X^n))}{n}\right] = \sum_{x \in \mathcal{X}} p(x)(-\log p(x) - l(x))$$

$$= \sum_{x \in \mathcal{X}} p(x)(\log \frac{2^{-l(x)}}{p(x)}) \leq \log \sum_{x \in \mathcal{X}} p(x) \frac{2^{-l(x)}}{p(x)} \leq 0$$

where the first inequality follows using Jensen's inequality since $\sum_{x \in \mathcal{X}} p(x) = 1$. The last inequality follows from Kraft-McMillan inequality. □

Definition 23 (Entropy coding) An entropy coding is any lossless data compression method such that

$$lim_{l(S) \to +\infty} \mathbb{E}_{X \sim p}\left[\frac{l(enc(S))}{l(S)}\right] < H(X) + 1$$

Some famous historical examples of entropy codes for DMS are Huffman codes which we do not describe here. In fact, the DMS assumption that we have used so far for an introductory purpose is too simple: source symbols are not independent; they often depend on the previous symbols. Hence, a more realistic hypothesis is *ergodicity*. To present the general theorem of source coding for ergodic sources, we introduce several formal definitions related to ergodic theory. Those formal definitions are not necessary to understand the main work in this thesis. The reader can go directly to Theorem 6, the general source coding theorem we extensively use.

Definition 24 (Transformation) A transformation T on a probability space (Ω, $\mathcal{B}$, P) is any function that maps from Ω to Ω

Definition 25 (Measure-preserving transformation) A transformation T on a probability space (Ω, $\mathcal{B}$, P) is called measure-preserving if the measure of any measurable

set A in Ω is the same as the measure of its image under T. That is,

$$Pr(T^{-1}(A)) = Pr(A)$$

Definition 26 (Time-shift transformation) In the case of a stochastic process X, the transformation $T : X_i \mapsto X_{i+1}$ on the probability space is called the time-shift transformation.

Definition 27 (Time-invariant transformation) A time-shift transformation T is called time-invariant if it is a measure-preserving transformation.

Definition 28 (Stationary process) A stationary process is a stochastic process for which the time-shift transformation is invariant.

In simpler terms, a stationary process satisfies the equation

$$Pr(X_1 = x_1, \ldots, X_n = x_n) = Pr(X_{t+1} = x_1, \ldots, X_{t+n} = x_n)$$

for every n, every time shift $t \in \mathbb{Z}$ and every $x_i \in \mathcal{X}$.

Definition 29 (Invariance under a transformation) Let Pr be a measure on a measurable space $(\Omega, \mathcal{B}$. A measurable set A is called invariant under a transformation T if

$$Pr(T^{-1}(A)) = Pr(A)$$

Definition 30 (Ergodic transformation) A measure-preserving transformation T on a probability space $(\Omega, \mathcal{B}, Pr)$ is called ergodic if, for any measurable set A that is invariant under T, $Pr(A)$ is either 0 or 1.

Definition 31 (Ergodic process) An ergodic process is a stochastic process for which the time-shift transformation is ergodic.

Definition 32 (Stationary and ergodic source) A source of data produced by a stationary process is called a stationary source. Similarly, a source of data produced by an ergodic process is called an ergodic source

In our context of data compression, where we wish to compress texts and images, data symbols do not arise independently from each other but depend on past symbols. Hence, the source is not memoryless, so the Shannon source coding no longer applies. However, we can assume it is ergodic. Using this assumption, we can refine the source coding theorem that applies to any ergodic source: the Shannon–McMillan–Breiman theorem (Asymptotic equipartition property (AEP) for ergodic sources).

Definition 33 (Joint entropy) The joint entropy of random variables $X_1, \ldots, X_n$ on a same value-alphabet $\mathcal{X}$ is defined as

$$H(X_1, \ldots, X_n) = -\sum_{x^n \in \mathcal{X}^n} p(x^n) \log p(x^n) = -\mathbb{E}_{X \sim p} \left[\log p(X^n)\right]$$

Definition 34 (Conditional entropy) The conditional entropy of X_n given $X_{n-1}, \ldots, X_1$ is defined as

$$H(X_n|X^{n-1}) = -\sum_{x_n \in \mathcal{X}, x^{n-1} \in \mathcal{X}^{n-1}} p(x^n) \log p(x_n|x^{n-1}) = -\mathbb{E}_{X \sim p} \left[\log p(X_n|X^{n-1})\right]$$

Definition 35 (Entropy rate) For a stochastic process $\{X_i\}$, the entropy rate is defined as the asymptotic average of the joint entropy when the limit exists

$$H(X) = lim_{n \to +\infty} \frac{H(X_1, \ldots, X_n)}{n}$$

Example 3 (Typewriter) If a typewriter has m equally likely output letters, he can produce m^n sequences of length n, all of them equally likely. Hence $H(X_1, \ldots, X_n) = \log m^n$ and the entropy rate is $H(X) = \log m$ bits per symbol.

Definition 36 (Conditional entropy rate) We can also define a related quantity for entropy rate when the limit exists:

$$H'(X) = lim_{n \to +\infty} H(X_n|X_{n-1}, \ldots, X_1)$$

Theorem 4 *For a stationary stochastic process, both limits in Definition 35 and Definition 36 exist and are equal:*

$$H(X) = H'(X)$$

Proof See [Cov99][Section 4.2, p.75] □

Theorem 5 (Shannon–McMillan–Breiman theorem) *If H is the entropy rate of a finite-valued ergodic process $\{X_t\}_{t \in T}$, then*

$$-\frac{1}{n} \log Pr(X_1, \ldots, X_n) \to H(X) \text{ with probability } 1$$

Proof See [Cov99][Section 16.8, p.644] □

Hence, we can define the same notion of typical set as in Definition 21 for DMS and reapply the same reasoning we did in Theorem 3 to prove the following general theorem:

Theorem 6 *For any ergodic source X and lossless coding scheme (enc, dec), the expected length of an encoded string $enc(X^n)$ asymptotically satisfies*

$$lim_{n\to+\infty}\, \mathbb{E}_{X\sim p}\left[\frac{l(Enc(X^n))}{n}\right] \geq H(X)$$

Moreover, this bound is achievable. For every $\epsilon > 0$, there exists $n_\epsilon \in \mathbb{N}$ sufficiently large and a lossless coding scheme $(enc, dec)_\epsilon$ such that for all $n \geq n_\epsilon$

$$\mathbb{E}_{X\sim p}\left[\frac{l(Enc(X^n))}{n}\right] = H(X) + \epsilon$$

More precisely, since

$$\begin{aligned}\mathbb{E}_{X\sim p}\left[R(X^n)\right] &= \mathbb{E}_{X\sim p}\left[\frac{size(X^n)}{l(Enc(X^n))}\right] \\ &= \mathbb{E}_{X\sim p}\left[\frac{n\log|\mathcal{X}|}{l(Enc(X^n))}\right]\end{aligned}$$

where R is the compression ratio defined in Definition 17, the theorem says that we can find a lossless coding scheme such that

$$\frac{\log|\mathcal{X}|}{H(X)} > lim_{n\to+\infty}\, \mathbb{E}_{X\sim p}\left[R_C(X^n)\right] \geq \frac{\log|\mathcal{X}|}{H(X)} - \epsilon$$

with $\epsilon' = \epsilon\log|\mathcal{X}|$. We retrieve the fact that if by assumption the source emits uniformly distributed symbols, then $H(X) = \log|\mathcal{X}|$ and we cannot find any pattern, which is why $R_C(X^n)$ is upper bounded by 1 and the sequence cannot be compressed.

In summary, Shannon entropy says how difficult it will be to compress data from a source of information given the hypothetical probabilities claimed by some models. For instance, if the probabilistic model is not precise, its claimed entropy might be low but will not reflect the true entropy of the source. Hence, the use of entropy codings will be optimal relative to the model but result in discrepancies with the true realisation of the source. If the probabilistic model is precise, then its claimed entropy increases, but the entropy encoding has more chances to suit the source of information and reach its unknown true entropy without discrepancies.

3.4 Kolmogorov complexity

The Shannon entropy discussed just above is an approach to data compression where we assume that the source is random. If we can somehow learn how random it is, essentially by source modelling, then we can potentially compress any strings of symbols up to the maximum amount of randomness arising from the source, that is, the entropy times the number of symbols. Informally, an entropy coding is essentially a function that compresses randomness up to full uniform randomness.

Kolmogorov complexity is a dual approach where the focus is put on describing how much determinism there is in a sequence by looking at the programs that could have output that specific sequence on some universal Turing machine. Essentially, when Shannon entropy says how much one could compress, at best, a sequence from a source given a specific model, Kolmogorov complexity says how difficult, at least, the true model is given a specific infinite sequence from the source. Although hypothetical, it is a helpful way of thinking, and it can be a tool to quantify the compressibility and randomness of random sequences. We do not go into the details of Turing machines and models of computation, keeping a relatively informal approach to give some insights based on [Cov99].

Definition 37 (Kolmogorov complexity) The Kolmogorov complexity $K_{\mathcal{U}}(x)$ of a string x with respect to a universal computer $\mathcal{U}$ is defined as

$$K_{\mathcal{U}}(x) = \min_{p:\mathcal{U}(p)=x} l(p)$$

The conditional Kolmogorov complexity is similarly defined as

$$K_{\mathcal{U}}(x|l(x)) = \min_{p:\mathcal{U}(p,l(x))=x} l(p)$$

This is the shortest description length if the Turing machine $\mathcal{U}$ has the length of x.

Example 4 (The fractal) The Mandelbrot set is an example of an object that looks very complex but is essentially very simple if one knows the deterministic rules to compute it. Hence, its Kolmogorov complexity is essentially constant in the fractal's precision. We can transmit only the language with the program to compute it instead of compressing the whole set of pixels with an entropy code.

In the context of data compression, Kolmogorov complexity naturally comes with the need to describe the program to be implemented by the decoder. Hence, Kolmogorov complexity is only defined up to a description language, for instance, the language to describe the Turing machine. This language is constant in size since it is a finite set of characters, and the compression works by assuming the receiver understands this language. Nonetheless, the importance of Kolmogorov complexity is motivated by the following universal property:

Theorem 7 (Universality of Kolmogorov complexity) *If $\mathcal{U}$ is a universal Turing machine, for any other Turing machine A there exists a constant c_A such that*

$$K_{\mathcal{U}}(x) \leq K_A(x) + c_A$$

for all strings $x \in \{0,1\}^*$, and the constant c_A does not depend on x

Proof See [Cov99][Section 14.2, p.467] □

In the case of a discrete memoryless source, Kolmogorov complexity is closely linked to the entropy. An interesting theorem is stated as

Theorem 8 (Relationship of Kolmogorov complexity and entropy) *Let X be a discrete memoryless source, then*

$$\mathbb{E}_X \left[\frac{1}{n} K(X^n|n) \right] \to H(X)$$

Proof See [Cov99][Section 14.2, p.473] □

We can also mention some aspect directly related to incompressibility:

Definition 38 (Algorithmically incompressible sequence) A sequence x^n is said to be algorithmically random if $K(x^n|n) \geq n$.

Definition 39 An infinite string x is said to be incompressible if

$$lim_{n \to +\infty} \frac{1}{K(x_1 x_2 x_3 \dots x_n|n)} = 1$$

Theorem 9 *If a string x is incompressible, it satisfies the law of large numbers in the sense that*

$$\frac{1}{n} \sum_{i=1}^{n} x_i \to \frac{1}{2}$$

Proof See [Cov99][Chapter 14, p.478] □

Hence, the proportions of 0's and 1's in any incompressible string are almost equal. We can view this theorem as a dual version of Shannon's source coding theorem. If $X \sim_{i.i.i.} Ber(\theta)$, then its entropy is maximal for $\theta = 1/2$, so according to Theorem 3, it is the less compressible source among all $\theta \in [0, 1]$. More precisely, we have the following theorem:

Theorem 10 *Let $X \sim_{i.i.d.} Ber(\theta)$, then*

$$\frac{1}{n} K(X^n|n) \to H_0(\theta) = H(X)$$

since $H_0(x) \coloneqq x \log x - (1 - x) \log(1 - x)$

Proof See [Cov99][Chapter 14, p.478] □

4 Arithmetic Coding

Arithmetic Coding is an entropy coding that can be used in lossless data compression. It is a stream code. The whole process can be smoothly visualised in Figure 1, but let us explain the coding scheme and show how to implement it on a computer. From now on, we will use the notation in [Sai23] and represent an interval $[\alpha, \beta)$ by $[b, l\rangle$ where $b = \alpha$ is the base and $l = \beta - \alpha$ the length. Moreover, let us define the cumulative distribution up to symbol s excluded

$$c(s) = \sum_{k=0}^{m-1} p(s)$$

where $p(s)$ is a probability distribution.

4.1 Encoding

First, we start with an interval set to $[0, 1\rangle = [0, 1)$. Then, we update the interval as follows.

$$[b_k, l_k, 1\rangle = [b_{k-1} + c(s_k)l_{k-1}, p(s_k)l_{k-1}\rangle$$

As we encode the $k-th$ symbol s_k from a string S, the current interval gets entirely divided into sub-intervals. The number of sub-intervals corresponds to the alphabet size $|\mathcal{X}|$, and each sub-interval size is proportional to the symbol probability, keeping the same arbitrary order throughout the process. Then, the encoder updates the current interval with the sub-interval corresponding to the symbol being encoded, positioning the base at value $b_{k-1} + c(s_k)l_{k-1}$. Hence, at the end of the encoding, the interval size is reduced down to

$$l_S = \prod_{s \in S} p(s)$$

Finally, Arithmetic Coding proceeds in transmitting a binary sequence whose float value lies in the final interval such that the length of that sequence is

$$L_S = \lceil -\log_2(l_S) \rceil \tag{2}$$

It is indeed guaranteed that such a binary sequence exists and is unique. Of course, we could argue that we could even give the shortest binary sequence whose value lies in the final interval, which can be shorter than L_S. However, this would also require transmitting $l(S)$ since the decoder would not know when to stop decoding. In principle, with the former approach of length L_S, the decoder does not have to know $l(S)$ because he only needs to stop decoding when $\lceil -\log_2(l(I_n)) \rceil$ happens to be the length of the original transmitted value v. However, to simplify the implementation, we also transmit $l(S)$ anyway, but this overhead is negligible, as it is shown in the proof of optimality.

4.2 Decoding

AC decoding is straightforward. It is clear that the resulting binary sequence unambiguously maps to the sequence of intervals created during the encoding. Anyone with access to the probabilities and the symbol order used during encoding can decode the original string symbol by symbol. More precisely, the decoder proceeds as follows:

1. it starts with the encoded value $\hat{v}$
2. it divides the interval $I_0 = [0, 1\rangle$ into sub-intervals proportionally to symbol probabilities, similarly to the encoding process
3. it searches in which sub-interval I_{n+1} $\hat{v}$ lies and updates the current interval I_n into I_{n+1} for $n = 0$. The symbol s_0 corresponding to this sub-interval I_1 is the first encoded symbol.
4. it applies step 2&3 to the nested intervals I_n for $n \in [0, l(S) - 1]$ to decode the entire string.

In the image example below taken from Wikipedia, the source is such that the letters 'k' and 'w' appear with probability 1/4 and the letter 'i' with probability 1/2. We can visualise the different intervals depending on the symbol to encode.

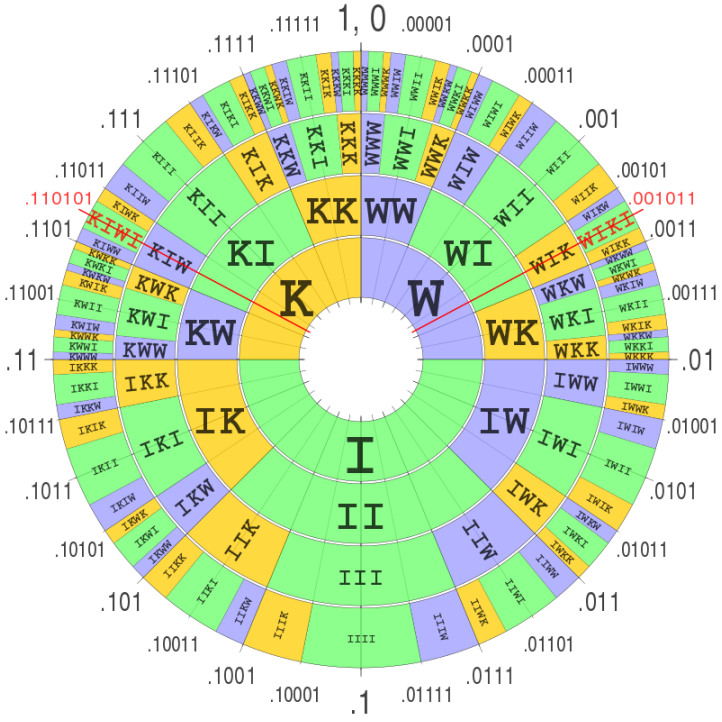


Fig. 1: Visualisation of an AC process for strings of length 4 (from Wikipedia)

4.3 Optimality

We now show the following theorem of optimality:

Theorem 11 *AC is an entropy coding*

Proof Let $l(S) = N$, we should prove that

$$lim_{N \to +\infty} \mathbb{E}_{X \sim p} \left[\frac{l(enc(S))}{N} \right] < H(X) + 1$$

for any discrete random variable. We simply follow the proof from [Sai23]. To begin with, we have to consider additional information that must be added to the encoded sequence for convenient transmission. This may include:

- information about the probabilities to decode the compressed sequence (if the receiver does not have access yet). This information is independent of the size of the file to compress and is referred to as the overhead $\sigma > 0$.
- $\lceil \log_2 N \rceil$ extra bits to transmit the number of symbols to decode. This is not necessary, but this simplifies the process.

We conclude from equation 2 that the number of bits per symbol used for coding a sequence S is bounded by

$$L_S \leq \frac{\sigma + \lceil \log_2 N \rceil - \log_2(l_S)}{N} \text{ bits/symbol}$$

Since

$$l_S = \prod_{k=1}^{N} p(s_k)$$

we have

$$L_S \leq \frac{\sigma - \lceil \log_2 N \rceil + \sum_{s \in S} \log_2(p(s))}{N} \text{ bits/symbol}$$

hence

$$\begin{aligned} \mathbb{E}[L_S] &\leq \frac{\sigma - \lceil \log_2 N \rceil + \mathbb{E}\left[\sum_{s \in S} \log_2 p(s)\right]}{N} \\ &= \frac{\sigma + \lceil \log_2 N \rceil}{N} - \frac{1}{N} \sum_{(s_1, \ldots, s_n) \in \mathcal{X}^N} \log_2 p(s_1, \ldots, s_n) \\ &= \frac{\sigma + \lceil \log_2 N \rceil}{N} + \frac{1}{N} H(X_1, \ldots, X_n) \leq \frac{\sigma + \lceil \log_2 N \rceil}{N} + H(X) \end{aligned}$$

By Shannon's theorem, the average number of bits per symbol cannot be smaller than the entropy, so we have

$$H(X) \leq \mathbb{E}[L_S] \leq H(X) + \frac{\sigma + \lceil \log_2 N \rceil}{N}$$

it follows that

$$lim_{N \to +\infty} \mathbb{E}[L_S] = H(X)$$

so Arithmetic Coding achieves an optimal compression performance. □

4.4 Practical implementation

Let us discuss the practical issues that need to be solved to correctly implement AC. We use the same ideas as in [Sai23] to overcome practical issues. The main issue is that the precision needed for exact arithmetic operation increases throughout the encoding. For instance, if we had $p(0) = 0.99$ and we repeatedly code symbol 0, we would have

$$l_0 = 1, l_1 = 0.99, l_2 = 0.9801, l_3 = 0.970299, l_4 = 0.96059601$$

However, CPU registers do not have infinite precision. More precisely, float numbers are represented as follows with binary base:

$$sign \cdot mantissa \cdot base^{exponent}$$

A computer typically applies norm IEEE 754 that specifies two formats for float numbers: simple and double precision using 32bits or 64bits. The norm is

$$(sign, mantissa, exponent) = (1, 23, 8)$$

for simple precision or

$$(sign, mantissa, exponent) = (1, 52, 11)$$

for double precision.

4.5 Fixed-precision arithmetic

Since the precision is finite, we should obviously make approximations at some point. Encoding and decoding only require additions and multiplications involving the probabilities of the symbols.
For multiplications, if $[a]_p$ means the approximation of a up to p bits precision, we can show in this case that for all l of precision less than $p//2$ bits in the mantissa, we have

$$[l \cdot a]_p = l \cdot [a]_p$$

Hence, we can first approximate the probability model to the finite precision level p to simulate exact multiplication up to precision $p//2$. It is shown in [Sai23] that the impact of such approximations on the compression rate is negligible.
For additions, an issue arises when two numbers do not have the same order of magnitude. In *simple precision* (4.4) for instance, additing $a = 0.1 \cdot 2^0$ with $b = 0.1 \cdot 2^{-23}$ will result in $0.1 \cdot 2^0 = a \neq a + b$ because there is no way to express $0.10000000000000000000001$ using the *simple precision* representation (4.4). In fact, in this case, we should at least use 24 bits for the mantissa.

To prevent such a situation, at the end of each encoding step, we can rescale the interval to keep the following properties

$$l \in (0.5, 1] \text{ and } b \in [0, 1) \tag{3}$$

While the implementation provided in [Sai23] can seem complicated, the philosophy behind is

- If the updated interval falls in the upper half of $[0, 1]$, we can just lower it by 0.5 by operation $b - 0.5$ and then rescale it by 2.
- If the updated interval falls in the upper half of $[0, 1]$, we can directly rescale it by 2.
- If the interval crosses value $1/2$, then we are not lucky. In this case, we have no choice but to continue the encoding with smaller and smaller length l values, hoping that at some point, the interval will fall either above or below $1/2$, which should happen with a very high probability.

In each case, we just have to keep in memory what we did in a buffer accordingly so that the decoder does the same operation synchronously. This includes shiftings, rescalings (by writing '1' or '0' in the buffer, respectively) and updating the buffer depending on how long we waited during the bad case, broadly speaking. Although this naive approach does not work if the interval crosses value $1/2$ with a width lower than 2^{-p} where p is the precision, we can still deal with this situation by lowering the base b so that it falls in the lower part and storing in memory this change with a carry propagation in the buffer.

4.6 Running time

- During the encoding process, since we work in finite precision, the running time is linear with the number of bit-operations. When encoding an arbitrary symbol, since each sub-function involves a constant number of operations, the running time is bounded by $\mathcal{O}(l(S))$
- Decoding is symmetric to encoding, except that we need extra computation to find the interval symbol containing the current value. This can be done efficiently with a binary search in logarithmic time with the alphabet size $|\mathcal{X}|$. Hence, decoding time is bounded by $\mathcal{O}(\log |\mathcal{X}| \cdot l(S))$

Overall, since $|\mathcal{X}|$ is fixed, the running time is linear in $l(S)$ as we will verify in the implementation results.

5 ANS coding

Asymmetric Numeral Systems (ANS) is a family of entropy coding methods introduced by Jarek Duda in 2009 and further developed in 2013 [Dud13]. It is based on a generalisation of the standard numeral system used to express natural numbers. In this section, we will present how ANS coding works. To this end, let us introduce some notions about numeral systems.

5.1 A matter of representation

Representations are ubiquitous in mathematics and everyday life. In everyday life, we construct representations about merely everything to tackle uncertainties and model future situations in our mental lives. Our decisions rely on both our intuition and our representation of the world. Therefore, it is crucial to develop representations that are suited to the emerging reality.

Definition 40 (Systems of representation) A system of representation is defined as any system which expresses a set of implicit or explicit information.

In mathematics, representations are used to represent concepts in the smoothest way in terms of manipulation and exchange of information. New mathematical theories often become more accessible via notations and framework improvements. A typical example is how civilisations used to represent numbers throughout history. In the information theory realm, we will highlight how data compression is a smooth interplay between the information to compress and its representation. We also provide an informal discussion on numeral systems comparison in Appendix **??** and on self-reference in Appendix **??**.

Definition 41 (Numeral system) A numeral system is a representation system for expressing numbers of a given set, consistently using digits or other symbols.

Example 5 The Egyptian numeral system consisted of a finite set of hieroglyphic symbols expressing the values in $\{10^i, i \in \{0, \dots, , 6\}\}$. Multiples of these values were expressed by repeating the symbol as many times as needed. For instance, a stone carving from Karnak's temple shows the number 3800 as:



Fig. 2: Hieroglyphs representing the number 3800

Example 6 In English, instead of writing as many suitable symbols as needed to express a quantity, we make use of symbols of smaller values to specify the number of repetitions. For instance, we do not say "hundred, hundred, hundred, two" but "three hundred and two" for 302.

Definition 42 (Positional system) A positional system is a numeral system in which the contribution of a symbol to the value of a number is the value of the symbol multiplied by a factor (base) determined by the position of the symbol.

Example 7 The Babylonian numeral system, base 60, was the first positional system to be developed. Its influence is present today in the way time and angles are counted in tallies related to 60, such as 60 minutes in an hour and 360 degrees in a circle.

5.2 Symmetric Numeral Systems

Let us take the example of the symmetric binary numeral system. It is called *symmetric* since it dedicates equal representation size for the even and odd numbers: an even (resp. odd) number ends up represented with a 0 (resp. 1). We do this because, without further information, we assume that the numbers are drawn from a uniform distribution; without other information, this is the best we can do.

Having defined the binary numeral system and explaining how it works, we can proceed as follows. We can add information contained in a bit symbol $\hat{s} \in \{0, 1\}$ to a previous numeral representation $\hat{x}$ by updating quantity x to $x' = 2x + s$ and then representing back this quantity into the binary numeral system. Since the quantity $x' = 2x + s$ grows approximately as $x' \sim 2x$, the representation size grows as $\log(x') \sim \log x + 1$. We conclude that we only used 1 bit to add information of symbol $\hat{s}$ onto representation $\hat{x}$. If the symbol distribution is uniform, we have $-log(p_{\hat{s}}) = 1$ whether $\hat{s} = 0, 1$, so according to Shannon's source theorem, we could not have done better in average. This shows that the symmetric positional numeral system can be seen as an entropy coding for the Bernoulli random variable of parameter $1/2$.

5.3 Encoding

Similarly to symmetric numeral systems, ANS coding works as encoding symbols into a single natural number x, but this time with a general probability distribution. We will denote by $E(x, s)$ the encoding function that, depending on the current natural number x storing previous information and a symbol s to encode, returns a greater natural number x' implicitly containing information about x and s. To decode s from x', we first search for the symbol s and then need to update x' to the previous value x to be able to decode some potential symbols stored in x. This is implemented by a decoding function $D(x') = (x, s)$. As it is explained in the original paper [Dud13], in standard numeral systems, different digits are treated as containing the same amount

of information: 1 bit in the binary system, or generally $\log(b)$ bits in a base b numeral system. However, an event of probability p contains $\log(p)$ bits of information. So, while standard numeral systems are optimal for uniform digit probability distributions, to optimally encode general distributions, we should try to asymmetrize this concept.

In fact, we would like to adapt the growth from x to x' to an arbitrary source of numbers by using a suitable trade-off. We could increase x a lot in the case of a rare symbol number to leave space for smaller growth in the case of a likely symbol. More precisely, if we want to achieve entropy coding, the information quantity is supposed to behave as $x' \sim x \cdot p_s^{-1}$.

Before explaining formally how to derive a suitable algorithm that fulfills the objective, the reader might want to get a visualisation of the formalisation in Figure 3 comparing AC and ANS and helping to understand the general process. Let us suppose that instead of probabilities, we have access to the symbol frequencies F_s and that they sum up to M. Suppose that for each symbol s, we create a set as follows:

$$E_s \coloneqq \bigcup_{j=0}^{+\infty} S_j^{(s)}$$

where

$$S_j^{(s)} \coloneqq \{M \cdot j + \sum_{i<s} F_i, \dots, M \cdot j + \sum_{i \leq s} F_i\}$$

We see that E_s is typically made of slots $S_j^{(s)}$ of equal size F_s and lying at equal distance in $\mathbb{N}$. Moreover, we have by construction

$$\bigcup_{s \in \mathcal{X}} E_s = \mathbb{N} \text{ and } \bigcap_{s \in \mathcal{X}} E_s = \emptyset$$

The rule that summarises the trade-off for the growth and generalises the binary numeral system explained in Subsection 5.2 is :

$$\textit{new } x' \textit{ takes the value of the } x\textit{-th element in } E_s. \tag{4}$$

Here, we take the convention that we start the information growth at $x_0 = 0$ to simplify notation, but any $x_0 \in \mathbb{N}^*$ works fine, inducing a $\log x_0$ overhead, as long as the sets E_s remain unchanged. Moreover, for ease of notation, we let $s = s_n$ denote the $n - th$ symbol to encode. Then the rule (4) for general positional systems translates into the following recursion formula:

$$x_n = E(x_{n-1}, s) = M \cdot \left\lfloor \frac{x_{n-1}}{F_s} \right\rfloor + x_{n-1} \% F_s + b_s$$

where

$$b_s = \sum_{i<s} F_i$$

Indeed, starting from 0 and going through the successive natural numbers, we know that at each increment by M, we will visit exactly F_s elements in E_s. Hence, to reach the x-th element in E_s, we can begin the growth up to $M \cdot \left\lfloor \frac{x}{F_s} \right\rfloor$ and be sure to visit exactly the first $F_s \cdot \left\lfloor \frac{x}{F_s} \right\rfloor$ elements of E_s. Then, we still need to visit the remaining $x\%F_s$ succeeding elements in E_s. As we constructed the sets, we know that we have to increment again $M \cdot \left\lfloor \frac{x}{F_s} \right\rfloor$ by $\sum_{i<s} F_i$ to reach the beginning of the next slot of E_s and then increment by $x\%F_s$ to finally reach its x-th element.

5.4 Decoding

What the decoder needs is the length N of the sequence to decode. For instance, stopping when the stack x_n equals 0 is insufficient since encoding s_0^N or just s_0 gives the same stack 0. Of course, the decoder also needs the knowledge of the convention x_0. Given the length N and x_0, the encoding process is deterministic and bijective so it is possible to go backwards and decode the information stored in x.

The first step is to decode the last encoded symbol s. This is done by using the invariance under the interval $[0, M-1]$, hence computing $x\%M$ and then searching for

$$s : b_s \leq x\%M < b_{s+1}$$

Since the function $s \mapsto b_s$ is increasing, this can be done by implementing a binary search in $\mathcal{O}(\log|\mathcal{X}|)$

The second step is to update the stack of information x back to the original stack just before s was encoded. This is done via the function

$$Dec(x, s) = F_s \cdot \left\lfloor \frac{x}{M} \right\rfloor + x\%F_s - b_s$$

Indeed, we know that x is the number corresponding to some x' appearance of x' in E_s. Hence, it suffices to count how many symbols in E_s we could have visited during a potential encoding, knowing that we reached x, which will give us x'. We know that E_s is made of slots of F_s elements. Moreover, we can go from the beginning of one slot to the beginning of the next one by a move of M units in the natural numbers. Since the first element in E_s starts at b_s location in the partition, going from $x_0 = 0$ up to x, we know that we visited exactly $\left\lfloor \frac{x}{M} \right\rfloor$ entire slots of E_s and therefore visited $F_s \cdot \left\lfloor \frac{x}{M} \right\rfloor$ elements of E_s. It then remains to count how many elements are left in the remaining slot of E_s to go up to corresponding x. Since we visited the first element of E_s at b_s, the last element that we visited in E_s up to now corresponds to $b_s + M \cdot \left\lfloor \frac{x}{M} \right\rfloor$ in the partition. Hence, it remains exactly $x - (b_s + M \cdot \left\lfloor \frac{x}{M} \right\rfloor)$ elements to visit in the remaining slot of E_s in order to reach the corresponding x in the natural numbers. Since $x - (b_s + M \cdot \left\lfloor \frac{x}{M} \right\rfloor) = x\%F_s - b_s$, we obtain the result.

5.5 Practical implementation

In our implementation, we made the following approximation: given a probability p and an integer precision, we find a frequency $\hat{f}$ approximating p up to *precision* bits:

$$\hat{f} \leq 2^{precision} \cdot p < \hat{f} + 1$$

However, we need to make sure not to have null frequencies. Otherwise, we could not apply ANS. Moreover, for running time efficiency, the frequencies should sum up to $2^{precision}$ to perform bitwise operations. Hence, we add or subtract 1 to the positive highest frequencies, as many times as needed to reach $2^{precision}$. Finally, we discard the null frequencies at the extremums of the Gaussian in order to apply ANS coding.

5.6 Running time

As we can see with the simplicity of the formulas, there is no major issue with implementing ANS once the approximations of the probabilities upstream are made and converted into integer frequencies. We can just store information in the growing stack x_n by applying integer operations. However, the value of the stack x_n can explode with n. For instance, in the case of a uniform distribution, $x_n \geq M^n$. Hence, the runtime cost of arithmetic operations at stage n thus scales linearly with the amount of data $\Omega(n)$ for a sequence of n symbols that has been compressed so far and stored in the single integer x_n, leading to an overall cost of $\Omega(n^2)$.
The solution is to split x_n into a "buffer" list and a "head". The buffer would typically contain integers of size less than $2^{precision}$ that we call words. The head would typically be an integer in $[2^{precision}, 2^{2 \cdot precision} - 1]$. When the head gets bigger than $2^{2 \cdot precision}$, the encoder flushes some data from head to bulk while the decoder refills data from bulk to head whenever the head gets lower than $2^{precision}$. Doing this stream variant, the running time becomes linear with the amount of data instead of quadratic.

5.7 Optimality

Let us show that ANS is an entropy coding scheme. We denote s_n for the n-th symbol to encode. Moreover, we use the convention $x_0 = 1$ to perform the recursion up to term $\log x_0$.

Theorem 12 *ANS is an entropy coding*

Proof Since

$$x_n = E(x_{n-1}, s_n) = M \cdot \left\lfloor \frac{x_{n-1}}{F_{s_n}} \right\rfloor + x_{n-1} \% F_{s_n} + b_{s_n}$$

and

$$x_{n-1} \% F_{s_n} + b_{s_n} \leq 2M$$

we have the following inequality for $n \geq 2$:

$$\begin{aligned}
\log x_n &= \log Enc(x_{n-1}, s_n) \\
&\leq \log\left(\frac{M x_{n-1}}{F_{s_n}}\left(1 + \frac{2F_{s_n}}{x_{n-1}}\right)\right) \\
&\leq \log x_{n-1} + \log\left(\frac{M}{F_{s_n}}\right) + o\left(\frac{2M}{x_{n-1}}\right) \\
&= \log x_{n-1} - \log \hat{p}_{s_n} + o\left(\frac{1}{n}\right)
\end{aligned}$$

where the last inequality comes from noticing that $x_i \geq i$. Hence

$$\begin{aligned}
&\mathbb{E}_{S_n \sim X}\left[\log Enc(x_{n-1}, S_n) | S_1, \ldots, S_{n-1}\right] \\
&= \log x_{n-1} - \sum_{s_n \in \mathcal{X}} p_{s_n} \log \hat{p}_{s_n} + o\left(\frac{1}{n}\right)
\end{aligned}$$

Using this recursive inequality, we deduce the average bits per symbol

$$\begin{aligned}
\frac{1}{n}\mathbb{E}_{\mathcal{X}}\left[\log x_n\right] &\leq \frac{1}{n}\sum_{S \in \mathcal{X}^n} p(S) \sum_{s_i \in S}\left(-\log\frac{\hat{p}_{s_i}}{p_{s_i}} p_{s_i} + o\left(\frac{1}{i}\right)\right) \\
&= \frac{1}{n}H(X^n) + \frac{1}{n}D_{KL}(p^n || \hat{p}^n) + o\left(\frac{\log n}{n}\right) \\
&\to H(X) + o\left(\frac{1}{Mm}\right)
\end{aligned}$$

since in the case of our frequency approximation

$$\begin{aligned}
\frac{1}{n}D_{KL}(p^n || \hat{p}^n) &= \frac{1}{n}\sum_{S \in \mathcal{X}^n} p(S) \sum_{s_i \in S} \log\frac{p_{s_i}}{\hat{p}_{s_i}} \\
&\leq \frac{1}{n}\sum_{S \in \mathcal{X}^n} p(S) \sum_{s_i \in S} \log\frac{M p_{s_i}}{M p_{s_i} - 1} \\
&= \frac{1}{n}\sum_{S \in \mathcal{X}^n} p(S) \sum_{s_i \in S} o\left(\frac{1}{Mm}\right) \\
&= o\left(\frac{1}{Mm}\right)
\end{aligned}$$

where the inequality is deduced from frequency approximations in subsection 5.5 and the two last equations are assuming that there exist $m > 0$ such that $m = \min_{i \in \mathbb{N}} p_{s_i}$ and with $l(S) = n$. Hence, if we choose $M = \Omega(m^{-2})$ then

$$\begin{aligned}
lim_{n \to +\infty}\frac{1}{n}\mathbb{E}_{S^n \sim X^n}\left[ANS_{ENC}(S^n)\right] &= lim_{n \to +\infty}\frac{1}{n}(\mathbb{E}_{S^n \sim X^n}\left[\log x_n\right] + \log n + \log(\text{"}x_0\text{"})) \\
&= H(X) + o\left(\frac{1}{Mm}\right) \qquad < A
\end{aligned}$$

so the coding achieves Shannon entropy. □

5.8 Comparison with AC coding

We can now compare AC and ANS coding. Let us give a visual summary of both codes :

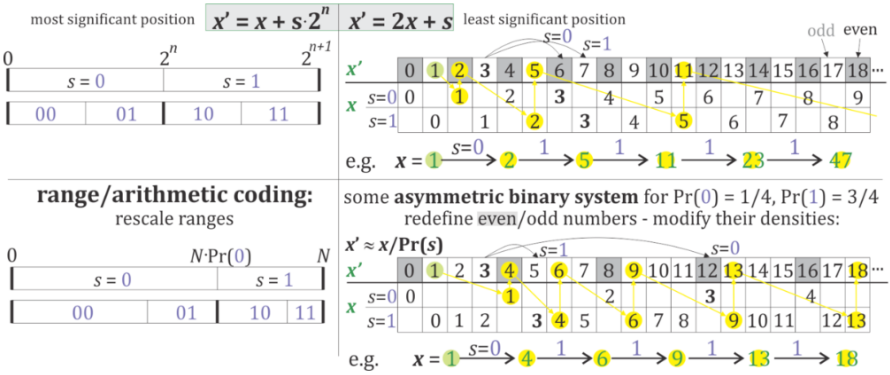


Fig. 3: Visualisation of the AC and ANS coding process

The upper part compares AC and ANS on a binary source with a probability model $p(0) = p(1) = 1/2$. The lower part is with probability model $p(0) = 1/4, p(1) = 3/4$. We can notice that the asymmetric binary system performs better to encode sequence "01111" ($18 < 47$). This is due to better agreement with frequencies of sequence to encode. However, since the stack starts at $x_0 = 1$, the final integer 47 is not the decimal representation of the binary number 01111
From what we have seen, both ANS and AC are entropy coding. However, ANS is faster if we use a buffer as shown by the experiments in [Bam22].

In terms of architecture, ANS and AC are both stream codes, but they are radically different in how they operate individual symbols and individual sequences. What first comes to sight is that AC is queue-like, working as FIFO (the first encoded symbol is the first being decoded). Meanwhile, ANS is stack-like, working as LIFO (the last encoded symbol is the first being decoded). In fact, AC corresponds to adding new information in the most significant position in updating the current interval with a magnitude at the current precision level. For instance, encoding the sequence $S =$ "365" using AC is done this way :

$$0 \rightarrow_{+3\times10^2} 300 \rightarrow_{+6\times10^1} 360 \rightarrow_{+5\times10^0} 365$$

On the other hand, ANS generalizes symmetric numeral systems by adding information in the least significant position. For instance, encoding the sequence $S =$ "365" in the symmetric numeral system is done this way according to the algorithm we gave:

$$0 \rightarrow_{\times10+3} 3 \rightarrow_{\times10+6} 36 \rightarrow_{\times10+5} 365$$

where an error at stage k would only incur a difference of at most $10^{(l(S)-k+1)}$. Another remark is that AC reaches Shannon entropy per symbol; each symbol s added to the stream directly incurs a growth of $-\log p_s$ bits due to rescaling

the current interval length proportionally to the probability of the new symbol. However, the resulting growth when encoding a symbol in an ANS stack depends not only on the probability symbol but also on its index position in the arbitrary ordering we chose, coming from the b_s term in the encoding rule. For instance, in the decimal system, symbol $'9'$ is at the ninth position, so encoding 9 as the first symbol would require $\lceil \log 9 \rceil = 4$ bits if the code alphabet is binary, since $b_s = \sum_{s=0}^{8} F_s = 9$. Similarly, encoding the symbol "3" would require only 2 bits. However, on average, ANS still achieves Shannon entropy. Its LIFO architecture makes it relevant for bits-back coding that we study in Section 7.

6 Source Modelling

Source coding introduced in Subsection 3.2 is the task of converting symbols from a source to symbols from a code. If we want to achieve this task efficiently, we need to learn how the source behaves. This task of modelling the source is mainly done by developing mathematical models.

6.1 Mathematical Modeling

Mathematical modelling is an important research domain in machine learning and, more specifically, probabilistic modelling. Let us give here a brief definition:

Definition 43 (Probabilistic model) Let X be a dataset. A probabilistic model for X is any probability distribution p_X on X

6.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are models of computation inspired by the structure of neural networks in the brain. They are a good choice architecture for probabilistic models where the parameters of the probability distributions that describe a dataset are obtained via the neural network's output. Let us derive a short explanation based on [Mac03].

6.2.1 Single Artificial Neuron

The Single Artificial Neuron is a feed-forward device: it has a number I of inputs x_i from which we give the data information and one output y that collects modified information. In fact, each input is associated with a weight w_i. With this specific architecture, the single neuron processes information from the inputs to the output the following way. First, we define the activation by the quantity

$$a = \sum_i w_i x_i$$

Then, the output y also called the activity of the neuron, is set as a function of the activation that can be deterministic (Linear, Sigmoid, etc) or stochastic

$$y = f(a)$$

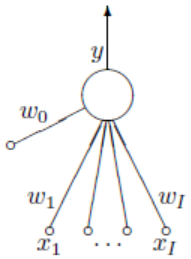


Fig. 4: A single artificial neuron in [Mac03]

6.2.2 Feedforward Neural Network (FNN)

A Feedforward Neural Network consists of several single neurons linked together through their inputs and outputs, similar to the brain. For that reason, we can describe it mathematically as a graph where the nodes are the neurons, and the edges indicate the links between the neurons. An FNN is then an oriented graph in which the neurons are placed in layers, and the information flows from the inputs to the outputs indicated by the direction of the arrows. The layer at the left, that is, where we input the original data, is called the input layer. Similarly, the layer at the right where we collect the final outputs is called the output layer. The layers in between, possibly many for a deep neural network (DNN), are called the hidden layers. For FNNs, edges are uni-directional, and information only flows from left to right.

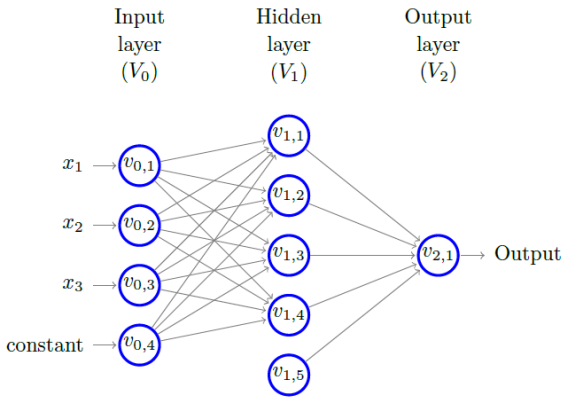


Fig. 5: A FNN presented in [SSBD14]

In contrast, if we allow information to flow backwards, that is, arrows pointing from layer V_i to V_{i-1}, then the neural is called a recurrent neural network (RNN). However, for probabilistic modelling, we will only use DNNs that are feed-forward.

6.2.3 Learning a probability distribution

DNN are surprisingly useful for probabilistic models. If we make the assumption that the probabilistic model of some data X follows a family distribution $\mathcal{F}_{\mathbf{\Theta}}$ with a set of parameters $\mathbf{\Theta}$, then DNNs can search for some good parameters θ (possibly a vector) with associated probability function $p_\theta(x)$. Combining different ANNs in various architectures leads to models that can model the data structure from different perspectives. We will present some of them thereafter, but let us recall how a DNN learns a probability distribution p.

Definition 44 (Input space) The input space $\mathcal{X}$ is the set of possible data points $x = (x_1, \ldots, x_n)$ that the FNN can take as input

Definition 45 (Evidence) Let p be the probability measure of some random variable $x \in \mathcal{X}$. We call $p(x)$ the evidence of x.

Definition 46 (The likelihood) The likelihood is the function over $\mathcal{X} \times \Theta$ given by

$$L(x, \theta) = p(\theta|x)$$

The likelihood describes how likely the parameters are to be θ given that we observed data x. Intuitively, it is reasonable to think that the observed data are, on average, likely to be observed. For this reason, an ideal probabilistic model should give the highest possible probabilities given those observed data. Hence, we search for a probabilistic model that fits the observed data by maximising the likelihood. Therefore, we define our loss function as

$$\mathcal{L}_X(\theta) = -\frac{1}{|X|} \sum_{x \in X} \log p(\theta|x)$$

where $X \subset \mathcal{X}$. We say the training is online if X consists of only one data point x. Else, we call it "batch training". The logarithm of the likelihood is popular because it is well-suited for probabilities, turning joint probabilities into a sum with Bayes theorem. In fact, we are rather interested in the argument

$$\theta_{optimal} = arg \max_{\theta \in \Theta} \mathcal{L}_X(\theta)$$

This is where the architecture of ANN is relevant. Training the FNN to output a good vector parameter θ^* is done by back-propagating the error loss function between the desired output indicated by the data point in the training data set and the actual output of the FNN for this data point. The error is communicated backward layer by layer, and the weights of each layer are updated according to the opposite gradient of the error with the hope it will

compensate for the error in the next step if the objective function is differentiable. We do not go into the details and formulas for back-propagation, but the reader can have a look at book [SSBD14], chapter 20.

6.3 Predictive models

Predictive models are probabilistic models used to predict future outcomes given past data. Given $\{X_k\}$ a stationary process, the model attempts to describe the set of conditional probabilities given by

$$\begin{aligned} &Pr(X_k = x_k | X_{k-1} = x_{k-1}) \\ &Pr(X_k = x_k | X_{k-1} = x_{k-1}, X_{k-2} = x_{k-2}) \\ &\vdots \\ &Pr(X_k = x_k | X_{k-1} = x_{k-1}, \dots, X_{k-c} = x_{k-c}) \end{aligned}$$

where c is the context size. Over recent years, the Transformer architecture has been particularly adapted for such a task. It is a specific neural network architecture that relies on the self-attention mechanism, which allows the model to weigh the importance of different tokens (X_i) in a sequence relative to each other [VSP+17]. This is crucial for capturing dependencies between tokens regardless of their distance from each other in the input sequence X^n.
The efficiency of such an architecture is such that it is now widely used for natural language processing tasks. We notice that such a probability model directly fits the AC queue-like (FIFO) architecture. Indeed, decoding sequences from the first to last tokens according to the previous tokens allows the decoder to access the relevant conditional probabilities given by the model he agrees on with the encoder. For the exact same reason, this model does not smoothly suit the ANS stack-like (LIFO) architecture. Hence, another probabilistic model paradigm is needed to make use of such an entropy code.

6.4 Variational Auto-Encoders

Obviously, predictive models are good for predictive tasks but can also be used for generative tasks. However, other types of models exist that allow for such a task, among them diffusion models and latent variable models. A latent variable model is a statistical model that relates a set of observable variables x to a set of arbitrary unobserved variables z called latent variables. In some broad sense, z is a set of data that the model uses to better explain the behaviour of x. Among others, famous examples are hidden Markov models and mixture models. A latent variable model explains a dataset X since it reveals hidden structures.
The recent latent variable model we focus on is the Variational Auto-Encoders (VAE). We base our short explanation on the detailed Medium article by Joseph Rocca [Roc19]. In order to describe a VAE, let us briefly describe an

auto-encoder. An auto-encoder consists of a pair $(e(x), d(x))$ of encoder/decoder. The encoder maps the initial data $x \in \mathbb{R}^n$ in a latent space $\mathbb{R}^m$ of smaller dimension using function $e(\cdot)$. The decoder $d(z)$ conversely maps encoded data in the latent space to data in the initial space. A good auto-encoder should be able to reconstruct a maximum of information from the original data while having a low latent space dimension. This is what happens in principal component analysis (PCA), and an auto-encoder can be seen as a generalisation. For auto-encoders, though, we use neural networks as functions for the encoder/decoder functions so that they learn, after some iterative optimisation of a specific loss function, what the best encoding/decoding functions are. The training loss depends on the objective but is generally the mean square error

$$\mathcal{L} = \|x - d(e(x))\|^2$$

that quantifies the reconstruction error. The minimisation is simply done by gradient descent over the parameters of the networks.
The problem is that auto-encoders perform badly in the context of content generation. Indeed, we only asked the auto-encoder to reconstruct an encoded data point among a training data set, and there is no reason to think a random data point in the latent space will give a meaningful output when decoded. In fact, to minimise the loss, the model could overfit the training data at the cost of some continuity for $d(\cdot)$. More precisely, the model needs to be regularised by changing its architecture a bit. Instead of returning data points z in the latent space, the encoder outputs parameters (μ_x, σ_x) of a Gaussian distribution

$$p(\cdot|x)) = \mathcal{N}(\mu_x, \sigma_x)$$

which depends on the input x. Then, the decoder samples some z from $p(\cdot|x)$ and outputs a reconstructed data point as usual. This way, the auto-encoder cannot focus on reconstructing points by points in the training dataset; he has to find a more general reconstruction method, consequently allowing content generation when giving a general z in the latent space.

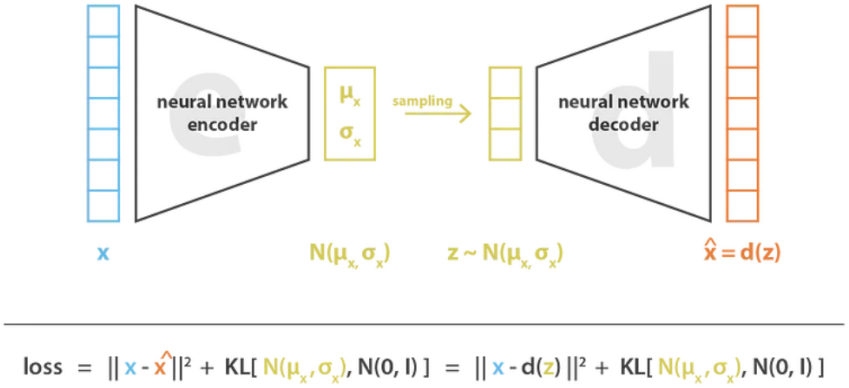


Fig. 6: Scheme of a VAE explained in [Roc19]

Of course, we have to control (μ_x, σ_x) by enforcing σ_x not to drop to 0 during training. Indeed, an extremely narrow Gaussian distribution would only output a sample equal to μ_x, a single data point in the latent space. Hence, we would be in a similar case to an auto-encoder with our continuity problem unsolved. Moreover, even with this caution, the means μ_x cannot be too far from one data point x to another x' since otherwise, this would be relatively the same as allowing near 0 variances for σ_x and σ'_x by simply rescaling the latent space. In order to prevent such a situation, we enforce that $p(\cdot|x)$ stays close to $\mathcal{N}(0,1)$. This is conveniently done by adding a Kullback-Leibler divergence term to the loss

$$\mathcal{L} = \|x - d(z))\|^2 + KL[\mathcal{N}(\mu_x, \sigma_x^2), \mathcal{N}(0,1)]$$

In summary, changing the auto-encoder as we did by setting a "gradient" over the information encoded in the latent space is what defines a variational auto-encoder (VAE). However, we need to explain some mathematical details in order to use neural networks and find good parameters. The theorem of Bayes gives us

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x|u)p(u)du} \tag{5}$$

Let us introduce some useful terminology and call $p(z)$ the prior, $p(z|x)$ the posterior and $p(x|z)$ the likelihood. In theory, from Bayes's theorem, a VAE is completely defined by $p(x|z)$ and $p(z)$. We have already set $p(z) \sim \mathcal{N}(0,1)$. Moreover, we also restrict $p(x|z)$ to be some specific Gaussian distribution

$$p(x|z) \sim \mathcal{N}(f(z), v(z))$$

where $f, v \in F$, the set of convolutional neural network functions and $c > 0$. However, in general, the integral in the denominator for Equation (5) is cost

computing, and we have to make an approximation. For this, we assume the approximation $q(z|x)$ follows a Gaussian distribution

$$q(z|x) \sim \mathcal{N}(g(x), h(x))$$

where $g, h \in F$ are parameters to be found with convolutional neural networks. We need to approximate $q(z|x)$ to the target $p(z|x)$ by minimising the Kullback-Leibler pseudo-distance, which we incorporate in the global loss. We can show that

$$\begin{aligned}&\arg\min_{g,h\in F} KL[\mathcal{N}((g(x), h(x))), p(z|x)] \\ &= \arg\max_{g,h\in F} \mathbb{E}_{z\sim q(\cdot|x)}\left[-\frac{\|x - f(z)\|_2^2}{2v(z)} - \frac{1}{2}\log v(z)\right] - KL[q(z|x), p(z)]\end{aligned}$$

We now took the 2-norm because, using a probabilistic model, we take the loss to be the probability loss

$$\log p(x|z) = \log\exp\frac{(x - f(z))^2}{2v(z)} + \frac{1}{2}\log v(s) = \frac{(x - f(z))^2}{2v(z)} + \frac{1}{2}\log v(s)$$

We retrieve the trade-off we mentioned between minimising the reconstruction error and staying close to $p(z)$ with the KL term. For now, we have assumed f and v were known to find g^* and h^* during training. However, we also need to incorporate f and v in the maximisation. Assuming we have access to optimal $q^*(z|x)$, the maximisation problem should be

$$\max_{f,v\in F} \mathbb{E}_{z\sim q^*(\cdot|x)}\left[\log p(x|z)\right] = \max_{f,v\in F} \mathbb{E}_{z\sim q^*(\cdot|x)}\left[\frac{\|x - f(z)\|_2^2}{2v(z)} - \frac{1}{2}\log v(z)\right]$$

so it is natural to gather the objectives in the same maximisation problem

$$\max_{f,v,g,h\in F} \mathbb{E}_{z\sim q(\cdot|x)}\left[-\frac{\|x - f(z)\|_2^2}{2v(z)} - \frac{1}{2}\log v(z)\right] - KL[q(z|x), p(z)] \tag{6}$$

Hence, we can find the appropriate weights for the FNN by backpropagation and gradient descent on the loss. Since the distributions are Gaussian, we can derive an explicit form for Equation (6). But the important fact we notice is that the objective function to be maximised can be written as

$$L(f, g, h) = \mathbb{E}_{z\sim q^*(\cdot|x)}\left[\log\frac{p(x, z)}{q(z|x)}\right] \tag{7}$$

which is the Evidence Lower Bound. In fact, we can show using Jensen's inequality that it is a lower bound on the logarithm of the evidence $\log p(x)$. Therefore, the ELBO certifies how close we have approximated a set of data

X. For this reason, the ELBO is well-studied in variational Bayesian methods, and various methods exist to maximise it.

7 Bits-back coding and chaining

Bits-back coding was originally developed by Wallace, Hinton and Camp in the 90s [Wal90],[HVC93]. The former paper was for classification tasks using inference, and the latter was to reduce the description length of neuron weights. Bits-back coding is a technique that involves variational inference and can be applied to any latent variable model. The first application using a continuous latent space is explained in [TBB19]. We also mention the article [Bam22] that clearly explains bits-back from the basics with a toy example, although the latent space is discrete and the prior distribution uniform. We will try to give a detailed explanation based on [TBB19] and then give some comments.

7.1 Bits-back coding

Suppose we have a latent variable model constituted of a prior, posterior and likelihood $\{p(z), p(x|z), p(z|x)\}$. If we want to encode and send a data point x_0, we could draw a sample z_0 from whatever distribution $q(z)$, encode both z_0 and x_0 according to the forward models, $p(z)$ and $p(x|z_0)$ respectively, using any entropy code. Since we use an entropy code, this would asymptotically result in a message length of

$$-\log_2 p(z_0) - \log_2 p(x_0|z_0) = -\log_2 p(x_0, z_0)$$

The receiver could then decode x_0 by first decoding z_0 according to $p(z)$ and then decoding x_0 according to $p(x|z_0)$. The decoder would be then left with the important information x_0 and a collateral information z_0

This seems consistent, but in practice, there are two catches :

- How can this be a good coding scheme given that we also transmitted z_0?
- how could we draw a sample z_0 from $q(z)$?

For the first point, the drawbacks of transmitting z_0 are obvious:

$$-\log_2 p(x_0, z_0) > -\log_2 p(x_0)$$

so the message length is greater than the information content, and the method is not optimal according to Shannon's source coding theorem. For the second point, there are various methods, but a crucial point is noticing this lemma

Lemma 13 *Sampling a random variable Z from a specific distribution q is equivalent to decoding a uniform random string of the encoding space with an entropy coding of that same distribution q.*

Proof We do not prove the lemma for general entropy codings, only for ANS and AC codings. Encoding any symbol z using ANS gives an integer $x \leq M$. Hence, to

decode any z, we should sample an integer u from the uniform discrete distribution $\mathcal{U}(1, M)$. We then have:

$$\begin{aligned} Pr(ANS_{dec}(U) = z) &= \sum_{u=1}^{M} Pr(ANS_{dec}(U) = z | U = u) Pr(U = u) \\ &= \frac{1}{M} \sum_{u=1}^{M} \mathbb{1}_{\{dec(u)=z\}} = \frac{F_z}{M} = Pr(Z = z) \end{aligned}$$

Using AC, encoding any symbol s gives a rational $x \in [0, 1)$. Hence, to decode any s, we should sample an integer u from the uniform continuous distribution $\mathcal{U}(0, 1)$. We then have:

$$\begin{aligned} Pr(AC_{dec}(U) = z) &= \int_0^1 Pr(c(z-1) \leq U < c(z) | U = u) Pr(U = u) du \\ &= \int_0^1 \mathbb{1}_{\{c(z-1) \leq u < c(z)\}} du = Pr(Z = z) \end{aligned}$$

□

Using this lemma, if we want to sample a latent z_0, it suffices to uniformly sample in the encoding space and then decode this sample with an entropy coding. Since, in practice, the encoding space is the binary space, this random element should consist of uniform random bits. Decoding a specific z_0 asymptotically requires $-\log q(z_0)$ bits by Shannon's theorem, but since we decode only for one step, one should rather take

$$R \coloneqq max_{z \in \mathcal{Z}} l(Enc(z)) \geq max_{z \in \mathcal{Z}} \lceil -\log_2 q(z) \rceil \text{ bits}$$

where Enc is any entropy encoder. We notice we have equality for AC but not for ANS.

Hence, if we let $q(z|x_0)$ denote the distribution that we used, possibly depending on x_0, what if the receiver wanted to virtually discard our cost of communicating z_0 by encoding it back according to the same distribution? Of course, it is too late, and the decoder cannot return the information as if he had never received it. But in principle, the net expected message length given x_0 would decrease to

$$\begin{aligned} &\mathbb{E}_{z_0 \sim q(z|x_0)} \left[(-\log p(z_0)) + (-\log p(x_0|z_0)) - (-\log q(z_0|x_0)) \right] \\ &\qquad = -\mathbb{E}_{z_0 \sim q(z|x_0)} \left[\log \frac{p(x_0, z_0)}{q(z_0|x_0)} \right] \end{aligned}$$

Remarkably, this happens to be the negative of the well-studied Evidence Lower-Bound (ELBO) that we mentioned to be the loss of any VAE model in Equation 7. It is direct to show using Gibb's inequality that the optimal distribution q that maximises the ELBO is the posterior $p(\cdot|x_0)$, which achieves an optimal expected message length equals to the information content $-\log p(x_0)$. This simply reflects the fact that $(-\log p(z_0|x_0)))$ is the

information content of z_0 knowing x_0. Here, the decoder would just be maximising the information he can give back, just as during encoding, we should have used $p(z|x_0)$ to minimise the number of random bits upstream when sampling z_0. However, as we explained in Subsection 6.4, the posterior is not available in practice since the integral is cost computing, and we need to use the approximate posterior.

Returning to the original problem, even if the decoder cannot remedy our over transmission by z_0, his idea to give the information back is still interesting. However, the work is essentially on our side because, like in any lossless compression scheme, the decoder should only perform the exact reverse operations that occurred during the encoding.
In fact, our sampling method to solve the second point can be relevant to solving the first point because uniform randomness is conserved throughout encoding (z_0, x_0), and we could take advantage of this. Because we have introduced uniform randomness upstream, the binary string resulting from encoding (z_0, x_0) that we end up with is just as random as the original uniform random bits, that is, uniformly distributed. Hence, we could use those easy-accessed uniform random bits to start the same encoding procedure on the next symbol x_1. Just as for symbol x_0, we could draw a latent z_1 from $q(z|x_1)$ decoding from those bits and then encode x_1 according to $p(x|z_1)$, completing a second encoding step. In fact, we could continue this process to encode data symbols x_i in a chain-like procedure. By doing so, we do not just use easy-accessed uniform random bits at each step time, we take advantage of randomness conservation to sample a latent z_i at each step from the current inherent random encoded string, decreasing its current length via sampling but then increasing it for transmitting z_i. In fact, spending the cost of transmitting the current sampled latent in the prevision to sampling the next one, it is as if we had virtually transmitted only one latent z during the whole encoding process, which is of negligible overhead.

Since we have

$$\begin{aligned}\mathbb{E}_{z_0\sim q(z|x_0)}\left[\log\frac{p(x_0,z_0)}{q(z_0|x_0)}\right] &= \mathbb{E}_{z_0\sim q(z|x_0)}\left[\left(\log\frac{p(x_0,z_0)}{p(z_0|x_0)}\cdot\frac{p(z_0|x_0)}{q(z_0|x_0)}\right)\right]\\ &= \log p(x_0) - D_{KL}(q(\cdot|x_0)||p(\cdot|x_0))\end{aligned}$$

Hence, if we apply the chain-like procedure, the total expected length in bits per symbol would be

$$\frac{1}{n}\,\mathbb{E}_{x^n\sim X^n}\left[l(Enc(x^n))\right] = \frac{1}{n}\left(R + \sum_{i=0}^{n} - \mathbb{E}_{z_i\sim q(z|x_i)}\left[\log\frac{p(x_i,z_i)}{q(z_i|x_i)}\right]\right)$$

$$= \frac{1}{n}\left(R + \sum_{i=0}^{n} -\log p(x_i) + D_{KL}(q(\cdot|x_i)||p(\cdot|x_i))\right)$$
$$= H^n(X) + \frac{R}{n} + \frac{1}{n}\sum_{i=0}^{n} D_{KL}(q(\cdot|x_i)||p(\cdot|x_i))$$

If

$$D \coloneqq lim_{n\to+\infty}\frac{1}{n}\sum_{i=0}^{n} D_{KL}(q(\cdot|x_i)||p(\cdot|x_i)) < +\infty$$

then

$$lim_{n\to+\infty}\frac{1}{n}\,\mathbb{E}_{x^n\sim X^n}\left[l(Enc(x^n))\right] = H(X) + D \text{ bits/symbol}$$

where we made the assumption that the data points x_i are i.i.d. and applied Shannon's source coding theorem. Hence, if $D < 1$, which should be the case if the approximation is good enough, this chain-like procedure is asymptotically optimal. Some works have been done in [RUS+21] to reduce D asymptotically, but we do not go into the details here. Hence, we have explained how the two catches stated above are linked to each other and how applying the lemma for the bits-back trick resolves them at once and allows us to achieve optimality. In section 5, we explained ANS and noticed that it is a stream code with a LIFO architecture. A stack-like entropy code is essential for the bits-back trick since the decoder needs access to the most recent symbol encoded (latent z) to get the information on the distribution parameters of $p(x|z)$ and decode the next symbol. We now refer to this bits-back chaining procedure as BB-ANS according to the terminology introduced in [TBB19].

7.2 The need for randomness

The terminology "bits-back" or "chaining with bits-back" can be confusing since it emphasizes giving the bits back. However, this is only a sub-consequence of the encoder decoding random bits from the stack. In fact, the decoder is only doing the opposite of what the encoder did, and this manifests in giving some bits back, but this might not be where our attention should focus. Maybe we should instead look at the encoding and wonder why something as meaningless as uniform randomness allows us to engage in an information process and reach the entropy rate. Of course, one short conclusion would be that randomness is better than nothing. But, surprisingly, we can engage a compression scheme from disinformation. It is as if, relative to information, disinformation was somehow helpful. How could we explain this? The VAE attempts to establish links between the data and some hidden structures by minimising a loss function with stochastic gradients. Hence, one possibility would be that the hidden structure only exists on average over the data, and the only way to reveal it is by sampling from the distribution $p(z|x)$, revealing the hidden structure on average. This is not the case for

AC with an auto-regressive model, where we access unique probabilities up to a specific context that has effectively happened in the past. But in some sense, AC and BB-ANS could be two faces of the same coin if we consider the latent variable z to be some "imaginary" context, explaining why we have to work on average using randomness. Just as for ANS coding, where a concept is updated onto its representation (*"new x' goes to x-th appearance"*), in BB-ANS, there are updates between the input space and the latent space via $p(X|Z)$ and $p(Z|X)$ so that the latent space can communicate its structure to the input space and vice-versa. Sampling z_0 during BB-ANS encoding upon x_0, we might obtain a skewed z_0 at first, but this skewness participates in revealing the hidden structure on average. As we sample more and more latent points z, we may reveal the hidden structure emerging from the dataset X and leverage a relevant insight on the source $p(x)$ of symbol to encode.

"J'ai pétri de la boue et j'en ai fait de l'or." (Charles Baudelaire)

7.3 Practical implementation

Let us go through the main steps to implement BB-ANS. A summary graph is shown below in Figure 8. Since we already explained ANS implementation in Subsection 5.5, we assume we have access to encoding and decoding functions $\text{ANS_ENC}(\mu_z, \sigma_z)(S)(s)$ and $\text{ANS_DEC}(\mu_z, \sigma_z)(S)$ where S and s are arbitrary stack and symbol respectively. We mind the reader that in Section 5 describing ANS, we rather used notation x_n and s_i to speak about encoding symbol s_i from stack x_n, but the notation introduced here is more adapted to distinguish latent z_i from input value x_i. We also assume that we have already implemented the function that converts probabilities into frequencies as explained in 5.5. For this implementation, we decided to use a VAE model, a latent variable model for which, as we said earlier in Equation 7, the loss is directly related to the compression ratio. Therefore, we only use Gaussian distributions. Moreover, since we typically compress images, the input space $\mathcal{X}$ is typically pixel values. For each pixel x_i and latent variables z_i comes associated probabilities

$$p(z_i), q(z_i|x_i), p(x_i|z_i)$$

from the prior, approximate posterior, and likelihood Gaussian distribution.

$$\begin{aligned} p(\cdot) &\sim \mathcal{N}(0, \sigma_p^2) \\ p(\cdot|x_i) &\sim \mathcal{N}(\mu_{z_i}, \sigma_{z_i}^2) \\ p(\cdot|z_i) &\sim \mathcal{N}(\mu_{x_i}, \sigma_{x_i}^2)) \end{aligned}$$

where σ_p is the sigma prior to being found with some heuristics, and the other parameters are given by the trained VAE. To avoid any confusion, it is important to have in mind the three distinct types of encodings and decodings that are going to be mixed up in the process :

- BB-ANS encoding/decoding functions (the general coding)

- ANS encoding/decoding functions to code any input and latent value
- VAE encoding/decoding to access the parameters of the Gaussian likelihood and approximate posterior

We also mention that pixel values X, as for latent values Z, must be encoded and decoded together in one shot in the stack. In fact, the VAE can only output Gaussian parameters if it is given the whole set of pixel values X or latent value Z. This is precisely how he may respectively deduce or decode the hidden structure of the image that we leverage for compression, similar to AC, where the associated auto-regressive or transformer-based model leverages the structure of a token neighbourhood using a self-attention mechanism. From now on, the notation $f(X)$ or $f(Z)$ will be used to indicate that we apply the function to the whole set X or Z, f typically being the ANS functions and the clipping functions.

7.3.1 Clipping everywhere

The main workload in the implementation of BB-ANS is the discretisation of the latent space. ANS and AC deal with symbols in a finite alphabet. However, Gaussian distributions given by the VAE inherently describe continuous random variables. If we wish to encode a continuous variable, we must restrict it to such a finite alphabet. We first introduce the notion of clipping and define a clip as a function that sets a real value to the nearest one in its associated discretisation:

Definition 47 ($clip(v, d)$) Let $v \in \mathbb{R}$. then the clip according to v with distance d is the function

$$\begin{aligned} clip(v,d)\colon \mathbb{R} &\longrightarrow \{v + i \cdot d, i \in \mathbb{Z}\} =: V \\ x &\mapsto arg \min_{v_i \in V} |x - v_i| \end{aligned}$$

What we can do now is for any value v_i in the clip $\{v + i \cdot d, i \in \mathbb{Z}\}$, we associate a bin $b_i = [v_i, v_i + d]$ and a probability mass $P(i) = p(v_i) \cdot d$ where p is a given probability distribution. However, in the framework of ANS and AC, the discretized space also needs to be finite. Since the distributions are Gaussian, we simply need to discard negligible probabilities. From the frequency approximations, we know that

$$2^{precision} \cdot P(i) < \hat{f} + 1$$

Hence, it is sufficient to solely keep probabilities satisfying

$$p(v_i) < 2^{-precision} \cdot d$$

Otherwise, the approximated frequency would be null, and we would discard it anyway.

We need to specify where we should apply those clips. A first clip needs to be applied directly on the pixel values x. Even if the pixel values are discrete up to the precision format of the original image, float32 or float64, we will clip them according to the lower or equal resolution we want to compress the image with. But crucially, this is because we want, after decoding the first set of variables X_0 during the decoding process and getting the distribution parameters (μ_Z, σ_Z) from VAE(X_0), the decoder to retrieve the exact same distribution parameters that we got during the very first step of encoding. Therefore, the decoder can encode back the latent Z_0 according to the original one, giving the original bits back. For such a discretisation of X, we introduce a new resolution parameter $prec \in \mathbb{N}^*$ and apply

$$X \mapsto clip(0, 2^{-prec})(X) =: \hat{X}$$

Of course, if there is no X_1 to decode next, the decoder does not care in giving the true bits-back, but in case of a next data point X_1 to decode, the decoder needs to rightfully encode back the latent using the same distribution that we used during encoding. As we already insisted in 7.1, this precaution is essential for him to obtain the relevant stack value and to decode the next z_1 and x_1. Since the VAE that we use is deterministic, it is possible to satisfy such a constraint.

Moreover, we also need to clip the mean parameter of the Gaussian distributions each time we get it from the VAE. In fact, given a Gaussian distribution, the way we proceeded to compute the bins and their associated probability was to take equally spaced value w_i on the right and left of μ starting from value μ up to $w_{\pm i} \coloneqq \mu \pm d \cdot i$ such that

$$w_{\pm i} < 2^{-precision} \cdot d$$

However, since we are going to encode values from clipped input $\hat{X}$, we want the bins in the discretisation of the Gaussian to correspond to the clip of $\hat{X}$ to keep the values unchanged through encoding and decoding. Hence, we should take the same distance $d = 2^{-prec}$ and clip μ to the same clip of $\hat{X}$, that is taking

$$\hat{\mu} \coloneqq clip(0, 2^{-prec})(X)$$

Finally, we conclude that we simply applied a universal $clip(0, 2^{-prec})$ on all the possible values in the input and latent space. If we think about it, we could not have done otherwise because, as we said, we cannot allow the decoder to lose information that we used during encoding, and so anyway, all clips must be clipped together into one universal clip.

7.3.2 Symbolisation

In the coding framework, the last step to encode the discretized values is to index the non-discarded B bins from 0 to $B - 1$. We can choose an arbitrary

ordering among symbols, but the increasing-values ordering is the simplest. This way, a bin with the lowest associated value is indexed by 0 and the bin with the highest associated value by $B - 1$.
We can implement this procedure by calling a function SYMB(v, x) that takes in arguments a finite ordered set of increasing real values v and a real value x to return the index i of the nearest value v_i in v. This way, from whatever real value x, we get the symbol of the nearest bin to it. Similarly, we can define a function DESYMB(v, i) which, given a symbol index i and a finite array of increasing real values v, returns v_i.
Those two functions are useful sub-functions for the encoding function ANS_ENC$(\mu, \sigma)(S)(\cdot)$ and the decoding function ANS_DEC$(\mu, \sigma)(S)$, where S stands for the current stack. For ANS_ENC$(\mu, \sigma)(S)(\cdot)$, we can then simply pass in argument a real value x corresponding to whatever pixel value or latent value and the function will automatically encode the value on the stack. Similarly, for ANS_DEC$(\mu, \sigma)(S)$, the function will automatically return the relevant clipped values so that we can treat them on the spot. Hence, we now write ANS_ENC$(\mu, \sigma)(S)(X)$ for the encoding function.

7.3.3 Graph summary

We can summarise the whole process of bits-back coding in the two graphs below. The dotted arrows indicate that a universal clipping has been applied to the variable. The equality $Z = \hat{Z}$ is because we apply the desymbolisation function inside ANS_DEC hence the output value is naturally clipped. A last error depicted by the red triangles could occur due to the different distribution when sampling and encoding Z with ANS. In fact, during encoding with BB-ANS, we have decoded Z according to $(\hat{\mu}_Z, \hat{\sigma}_Z)$ but we encode it according to $(0, \hat{\sigma}_p)$. If a decoded $\hat{z} \in \hat{Z}$ happens to be unlikely relative to the prior Gaussian distribution $\mathcal{N}(0, \hat{\sigma}_p)$, since we had to discard the extreme bins b_i with $p(v_i) < 2^{-precision-prec}$, it is possible that the sampled $\hat{z}$ will be clipped again to a more probable but wrong bin in the discretisation of $\mathcal{N}(0, \hat{\sigma}_p)$, losing complete information on the original $\hat{z}$ and making the chaining process incorrect. A similar case could happen when decoding with BB-ANS if the prior and the posterior are too far apart. In principle though, since the training of the VAE should minimise the second term in the ELBO

$$ELBO = \mathbb{E}_{z \sim q(\cdot|x)} \left[\ln p(x|z)\right] - KL[q(z|x), p(z)]$$

this should not happen for reasonable hyper-parameters *precision* and *prec*.

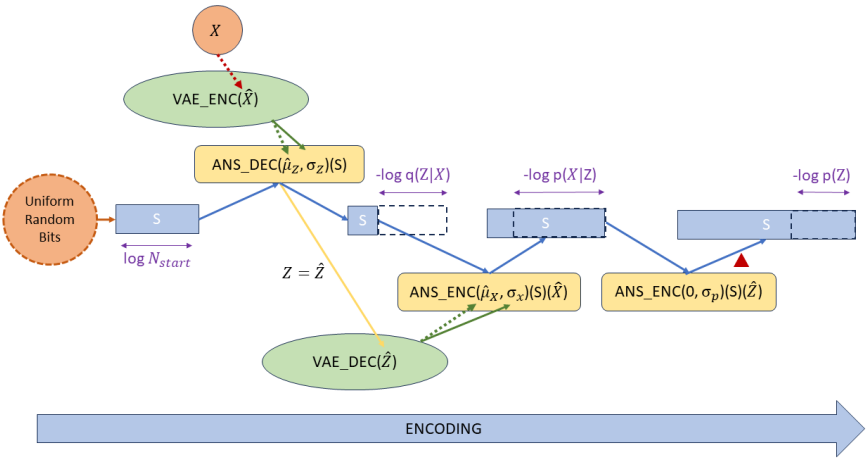


Fig. 7: Encoding according to BB-ANS

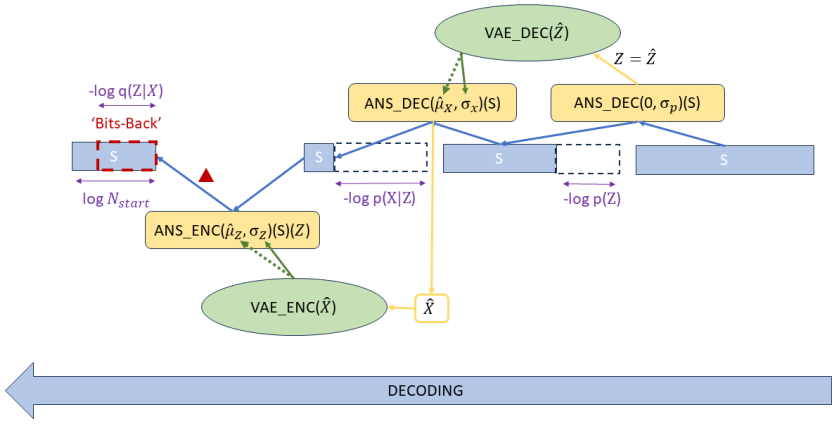


Fig. 8: Decoding according to BB-ANS

8 Implementation results

Let us describe the results we obtained implementing both AC and BB-ANS.

8.1 AC implementation

The typical choice architecture for AC is the transformer on auto-regressive models since it is well-suited to FIFO coding. We tested our implementation on texts and images.

- For text experiments, we took the piece *'A Midsummer Night's Dream'* by William Shakespeare and a biography of Ernest Dupuy titled *'Victor Hugo, son oeuvre poétique'* that can both be found in Project Gutenberg webpage. For the trained model, we used OpenAI GPT-2 models *gpt2* (124 million parameters) and *gpt2-medium* (354 million parameters) that can be found on the Hugging Face platform. GPT-2 is a large transformer-based language model trained on a dataset of eight million web pages.
- For image experiments, we took images of Swiss mountains. For the sake of the experiment, we rescaled the images to $n \times n$ pixels where n is a multiple of 8. We used the ImageGPT-small model from OpenAI, a GPT-2-like model with 76 million parameters trained to predict the next pixel value in raster order. ImageGPT models are also available on the Hugging Face platform. ImageGPT-small was trained on ImageNet Dataset, a widely used dataset in computer vision.

We mention the fact that we first have to give the first token to the model because gpt2, as a predictive model, needs a starting context to predict the probabilities of the next token. Only after we can encode the second token, which turns out to be the first encoded symbol. Therefore, the decoder should also be given this first token with the model oracle to get the probabilities of the second token and deduce it with the decoding algorithm. However, this overhead is negligible since the information of a token is $\log |\mathcal{X}| = \log 50257 \approx 2$ bytes.
We need to describe the context that we use to model probabilities. The context $(speed, window)$ signifies that at the $k-th$ token to encode, the probabilities for the next symbol are computed conditionally on the $k\%window + speed$ preceding tokens, and the window is moved forward with $speed$ units every time k reaches $window$. If unspecified, the context is by default $(speed, window) = (512, 1024)$, the biggest one possible with gpt2.

8.1.1 Language comparison

Below are the results for language comparison.

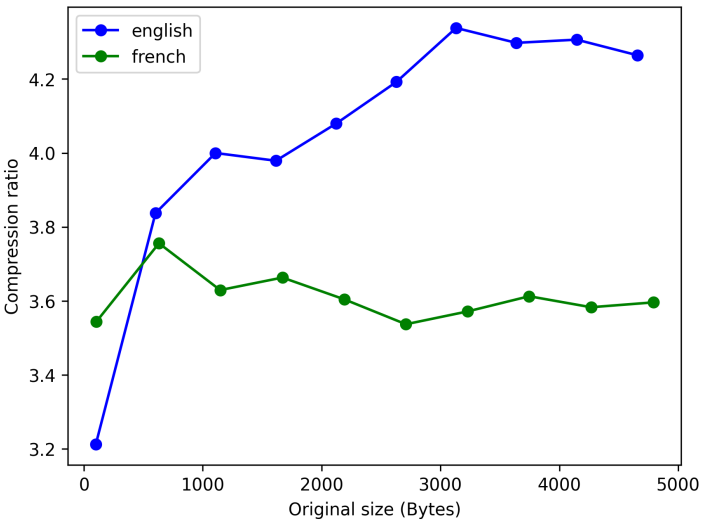


Fig. 9: Compression between English and French text

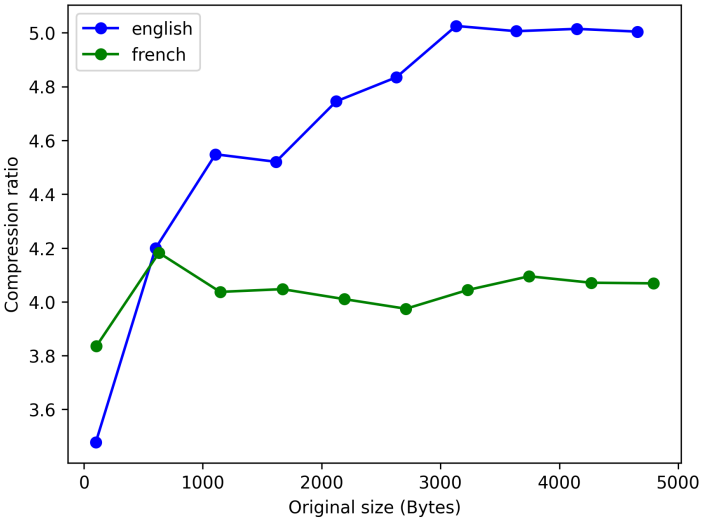
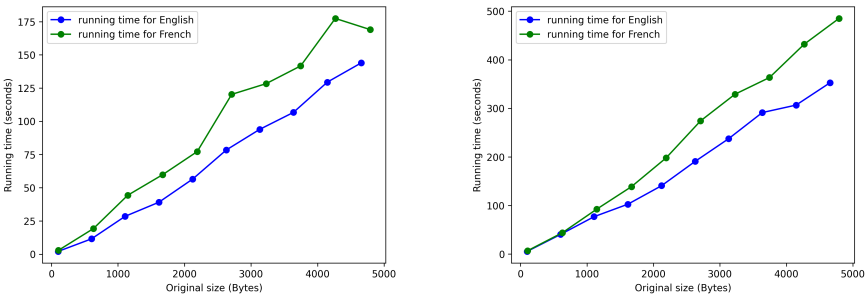


Fig. 10: Compression between English and French text

From what we see, AC performs better in English than in French. This is because gpt2 and gpt2-medium were mainly trained on English web pages. Hence, the model better knows what is more likely to come next if it is an English text, having a better knowledge of this language. The compression

ratio from gpt2 to gpt2-medium increases for both languages as expected. The increase is stronger for English (roughly 0.7) than for French (roughly 0.4), though. Intuitively, this is not surprising: the better we train a model onto a language and poorly onto another one, the stronger the impact will be for the preferred language compared to the last language when increasing the number of parameters, which act as an amplifier.
We wonder why the performance is reversed for the first data point. This is because the French text that we took starts with a short introduction in prose, whereas the English text is only made of verses. As prose is more frequent than versification in web pages, the model predicts better for prose. Shortly after, however, the French text continues with verses, and English catches up with French.



(a) Running time with gpt2

(b) Running time with gpt2-medium

Fig. 11: Running time for language experiment

As expected for AC, the running time is linear with the original size.

8.1.2 Context comparison

To compare the impact of context on compression, we made 4 different experiments with the context of $(speed, window) = (window/2, window)$. The $k-th$ symbol to encode, the probability of the next symbol is computed conditionally on the $k\%speed+speed$ preceding symbols (tokens). Using gpt2, we obtain those results.

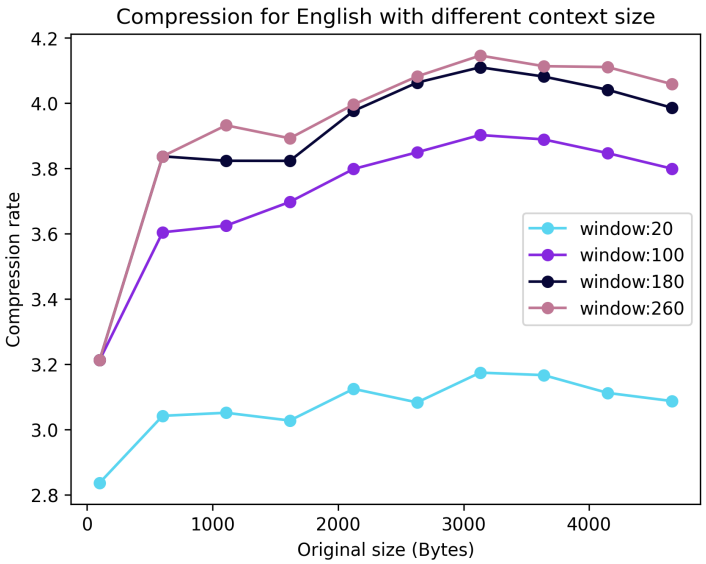


Fig. 12: Compression for English with different context

The compression rate increases with the size of the window, but this increase is weaker and weaker as the size increases linearly. Intuitively, this reflects that words in a neighbourhood are firmly connected, so the probability of saying the next word can be very accurate, given whether we know the past words or not. However, it is reasonable to think that the importance of past words fades as we move away in a text. Hence, there will not be huge differences between having a context of one paragraph or one paragraph and a sentence, for instance. We refine this observation in the graph below. We should also remember that considering how gpt2 tokenizes words, a context of, say, $window = 200$, is equivalent to a small paragraph.

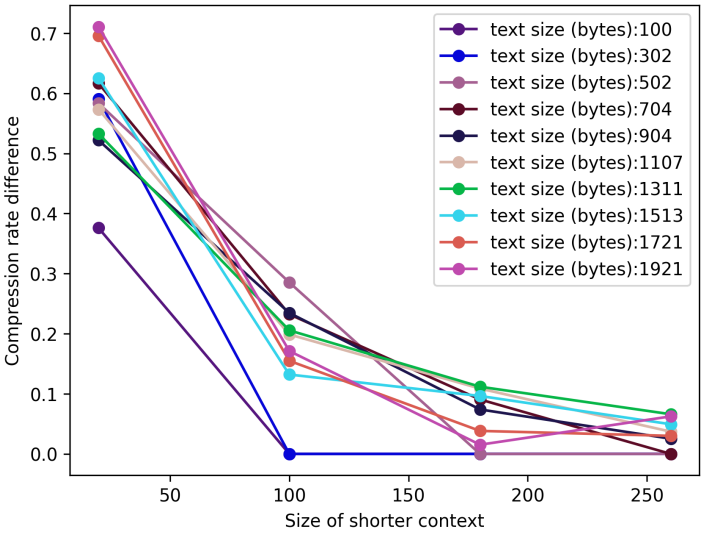


Fig. 13: Decay of the importance of context for English

The above figure shows how the context size becomes less critical for compression. On the ordinate axis are the differences in the compression ratio between a context experiment of size s (shorter context) and a context experiment of size $s + 80$ for s, going from 20 up to 260 with step 80.

8.1.3 Model size comparison

We give the plots with gpt2 and gpt2-medium for English. The impact of the number of parameters of the Transformer on the compression rate seems to increase until 1000 bytes of text and then stabilises.

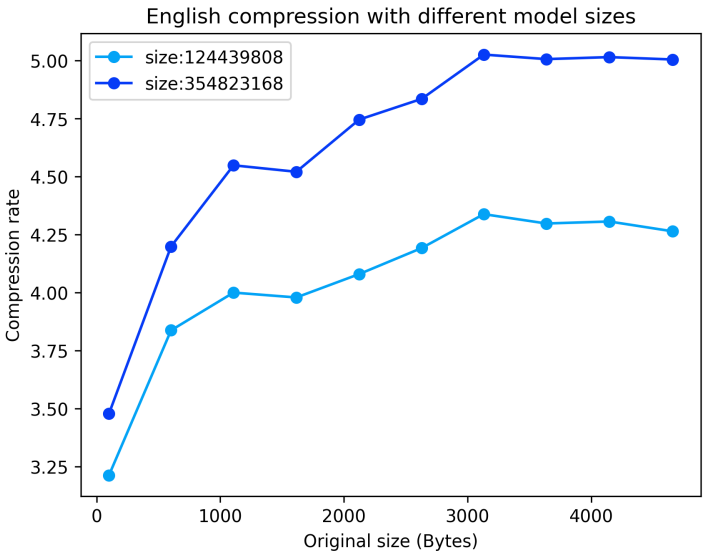


Fig. 14: Compression for English with gpt2 and gpt2-medium

8.1.4 Image comparison

We now study image compression using the ImageGPT model. We experimented on four 32×32 images RGB with 512 colours per pixel. We obtained the following compression ratio :

	Image 1	Image 2	Image 3	Image 4
Compression ratio	14.1	13.9	13.1	18.4
Runtime (seconds)	155	160	164	166

Fig. 15: ImageGPT experiment

Image 3 has a better compression rate because it is actually the image of a cat, and the model was instead trained on visages or animals. The other pictures are mountain landscapes, so their compression is approximately the same, structurally being the same. We could not change the pixel resolution since ImageGPT was trained on 32×32 images, and unlike texts, the context of tokens in a sequence does not behave the same in raster order. For instance, because positionally encoded from left to right, in a 16×16 image, pixel number 16 should be interpreted as far from pixel 17, but for 32×32, they are interpreted as neighbours.

8.2 BB-ANS implementation

For BB-ANS implementation, we first choose the VAE from [RBL+22] provided by the Hugging Face platform. Our implementation was successful on images and seems to run faster than AC with ImageGPT. However, the graphs below show that the compression ratios are unexpectedly low. This is because the loss during training the model is actually not appropriate for minimising the NELBO (near an auto-encoder).

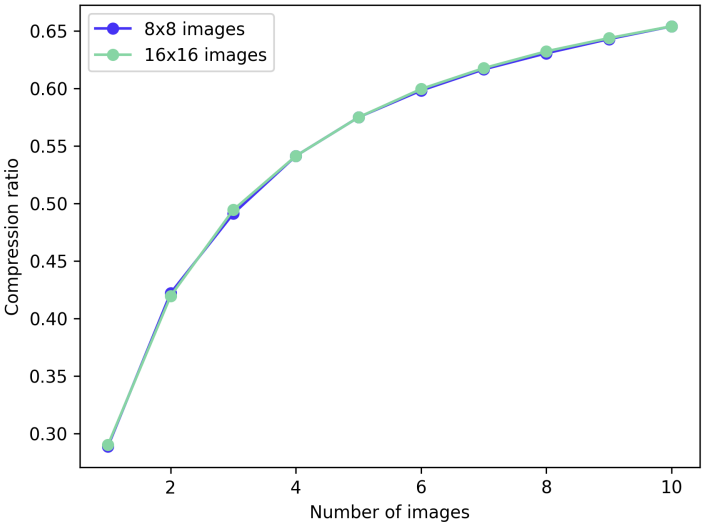


Fig. 16: Average cumulative compression ratio for BB-ANS

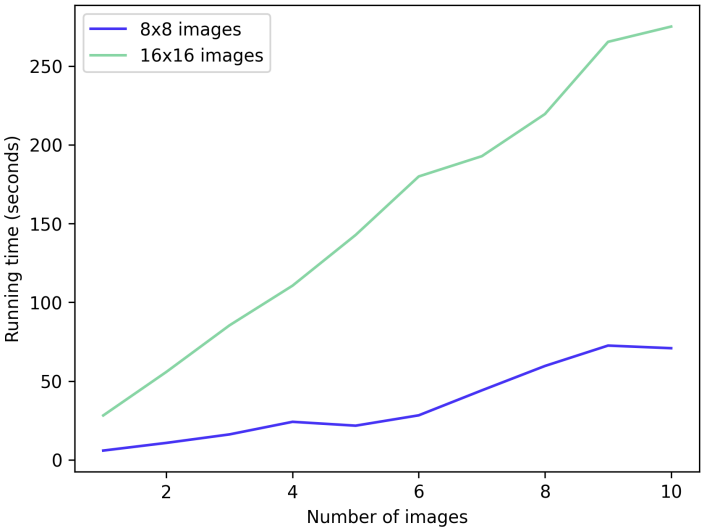


Fig. 17: Running time for BB-ANS

Hence, we had to train our own VAE models on the MNIST dataset. The two models we trained have latent shape "medium" $(4, 16, 16)$ and "small" $(1, 16, 16)$ and both input shape $(1, 32, 32)$. Our results averaged over 3 different digit images "i" in the MNIST dataset are shown below. We retrieve the same compression ratio (roughtly 5) claimed by Townsend [TBB19].

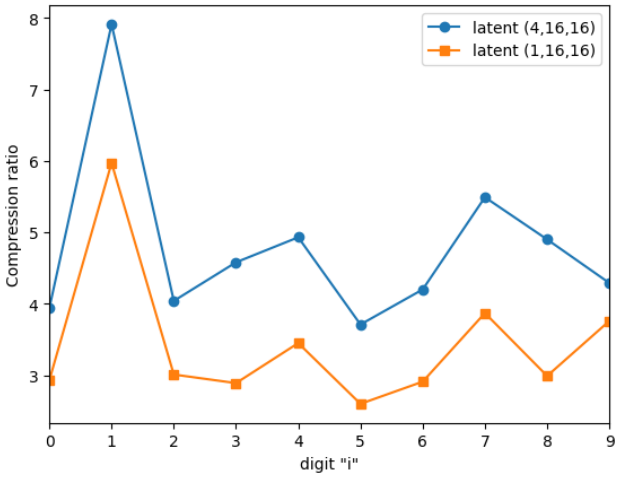


Fig. 18: Visual comparison

9 Conclusion

In this work, we discussed neural data compression, focusing on two optimal methods. We began by setting the theoretical landscape necessary to correctly understand AC and ANS coding. We saw that two dual approaches could be envisioned to quantify the complexity of a source of information: Shannon entropy and Kolmogorov complexity, the former being more practical and the latter more conceptual. On that basis, we delved into the precise functioning of AC and ANS, showing their optimality and analysing their different requirements for efficient implementation.
This analysis led us to study various neural network models that could fit with the corresponding optimal coding schemes. If the explicit probabilities are not accessible, we saw how we could suitably use a latent variable model with ANS to reach optimality still. In particular, we explained the BB-ANS coding in-depth, trying to shed some light and generalisations on a counter-intuitive process. Although quite similar to the one in [TBB19], we proposed an implementation method that mainly differs from the discretisation of the latent space.
In the last section, we presented our AC implementation results, proving that this method can significantly compress texts and images relative to an accessible predictive model. Implementation results for BB-ANS coding show that the coding method works. However, further work needs to be done to better understand the effect of the latent shape and the type of image on the compression ratio and a comparison with the AC method could also be investigated.

References

[Bam22] Robert Bamler. Understanding entropy coding with asymmetric numeral systems (ans): a statistician's perspective. *arXiv preprint arXiv:2201.01741*, 2022.

[Cov99] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[Dud13] Jarek Duda. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2013.

[HVC93] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.

[Mac03] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[RBL+22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[Roc19] Joseph Rocca. Understanding variational autoencoders (vaes), 2019.

[RUS+21] Yangjun Ruan, Karen Ullrich, Daniel S Severo, James Townsend, Ashish Khisti, Arnaud Doucet, Alireza Makhzani, and Chris Maddison. Improving lossless compression rates via monte carlo bits-back coding. In *International Conference on Machine Learning*, pages 9136–9147. PMLR, 2021.

[Sai23] Amir Said. Introduction to arithmetic coding–theory and practice. *arXiv preprint arXiv:2302.00819*, 2023.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[TBB19] James Townsend, Tom Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. *arXiv*

preprint arXiv:1901.04866, 2019.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Wal90] Chris S Wallace. Classification by minimum-message-length inference. In *Advances in Computing and Information—ICCI'90: International Conference on Computing and Information Niagara Falls, Canada, May 23–26, 1990 Proceedings*, pages 72–81. Springer, 1990.