Universidade de Aveiro
2021

Milton Duarte Teixeira da Silva

Efficient biosequence compression using neural networks

Compressão eficiente de sequências biológicas usando uma rede neuronal

Universidade de Aveiro
2021

Milton Duarte Teixeira da Silva

Efficient biosequence compression using neural networks

Compressão eficiente de sequências biológicas usando uma rede neuronal

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Diogo Rodrigues Marques Pratas, Investigador Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Armando José Formoso de Pinho, Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	**Professor Doutor Joaquim João Estrela Ribeiro Silvestre Madeira** Professor Auxiliar da Universidade de Aveiro
vogais / examiners committee	**Professor Doutor Luís Filipe Barbosa de Almeida Alexandre** Professor Catedrático da Universidade da Beira Interior (Arguente Principal)
	Doutor Diogo Rodrigo Marques Pratas Investigador Auxiliar da Universidade de Aveiro (Orientador)

agradecimentos / acknowledgements

Muito obrigado aos meus orientadores, Diogo Pratas e Armando Pinho pela disponibilidade, ideias e motivação que me deram durante todo este longo período. É visível a paixão que têm por esta área e esse sentimento motivou-me muito.
Aos meus pais e amigos, obrigado pelo apoio. Sem eles este trabalho não seria possível.

Resumo

Contexto: O aumento da produção de dados genómicos levou a uma maior necessidade de modelos que possam lidar de forma eficiente com a compressão sem perdas de biosequências. Aplicações importantes incluem armazenamento de longo prazo e análise de dados baseada em compressão. Na literatura, apenas alguns artigos recentes propõem o uso de uma rede neuronal para compressão de biosequências. No entanto, os resultados ficam aquém quando comparados com ferramentas de compressão de ADN específicas, como o GeCo2. Essa limitação deve-se à ausência de modelos específicos para sequências de ADN. Neste trabalho, combinamos o poder de uma rede neuronal com modelos específicos de ADN e aminoácidos. Para isso, criámos o GeCo3 e o AC2, dois novos compressores de biosequências. Ambos usam uma rede neuronal para combinar as opiniões de vários modelos específicos.
Resultados: Comparamos o GeCo3 como um compressor de ADN sem referência em cinco conjuntos de dados, incluindo um conjunto de dados balanceado de sequências de ADN, o cromossoma Y e o mitogenoma humano, duas compilações de genomas de arqueas e vírus, quatro genomas inteiros e duas coleções de dados FASTQ de um viroma humano e ADN antigo. O GeCo3 atinge uma melhoria sólida na compressão em relação à versão anterior (GeCo2) de 2,4%, 7,1%, 6,1%, 5,8% e 6,0%, respectivamente. Como um compressor de ADN baseado em referência, comparamos o GeCo3 em quatro conjuntos de dados constituídos pela compressão aos pares dos cromossomas dos genomas de vários primatas. O GeCo3 melhora a compressão em 12,4%, 11,7%, 10,8% e 10,1% em relação ao estado da arte. O custo desta melhoria de compressão é algum tempo computacional adicional (1,7 $\times$ a 3,0 $\times$ mais lento do que GeCo2). A RAM é constante e a ferramenta escala de forma eficiente, independentemente do tamanho da sequência. De forma geral, os rácios de compressão superam o estado da arte. Para o AC2, as melhorias e custos em relação ao AC são semelhantes, o que permite que a ferramenta também supere o estado da arte.
Conclusões: O GeCo3 e o AC2 são compressores de sequências biológicas com uma abordagem de mistura baseada numa rede neuronal, que fornece ganhos adicionais em relação aos biocompressores específicos de topo. O método de mistura proposto é portátil, exigindo apenas as probabilidades dos modelos como entradas, proporcionando uma fácil adaptação a outros compressores de dados ou ferramentas de análise baseadas em compressão. O GeCo3 e o AC2 são distribuídos sob GPLv3 e estão disponíveis para download gratuito em `https://github.com/cobilab/geco3` e `https://github.com/cobilab/ac2`.

Abstract

Background: The increasing production of genomic data has led to an intensified need for models that can cope efficiently with the lossless compression of biosequences. Important applications include long-term storage and compression-based data analysis. In the literature, only a few recent articles propose the use of neural networks for biosequence compression. However, they fall short when compared with specific DNA compression tools, such as GeCo2. This limitation is due to the absence of models specifically designed for DNA sequences. In this work, we combine the power of neural networks with specific DNA and amino acids models. For this purpose, we created GeCo3 and AC2, two new biosequence compressors. Both use a neural network for mixing the opinions of multiple specific models.

Findings: We benchmark GeCo3 as a reference-free DNA compressor in five datasets, including a balanced and comprehensive dataset of DNA sequences, the Y-chromosome and human mitogenome, two compilations of archaeal and virus genomes, four whole genomes, and two collections of FASTQ data of a human virome and ancient DNA. GeCo3 achieves a solid improvement in compression over the previous version (GeCo2) of 2.4%, 7.1%, 6.1%, 5.8%, and 6.0%, respectively. As a reference-based DNA compressor, we benchmark GeCo3 in four datasets constituted by the pairwise compression of the chromosomes of the genomes of several primates. GeCo3 improves the compression in 12.4%, 11.7%, 10.8% and 10.1% over the state-of-the-art. The cost of this compression improvement is some additional computational time ($1.7\times$ to $3.0\times$ slower than GeCo2). The RAM is constant, and the tool scales efficiently, independently from the sequence size. Overall, these values outperform the state-of-the-art. For AC2 the improvements and costs over AC are similar, which allows the tool to also outperform the state-of-the-art.

Conclusions: The GeCo3 and AC2 are biosequence compressors with a neural network mixing approach, that provides additional gains over top specific biocompressors. The proposed mixing method is portable, requiring only the probabilities of the models as inputs, providing easy adaptation to other data compressors or compression-based data analysis tools. GeCo3 and AC2 are released under GPLv3 and are available for free download at `https://github.com/cobilab/geco3` and `https://github.com/cobilab/ac2`.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

But I want to impress on you the notion that two hundred years from now, we will talk about Watson and Crick the same way that people talk about Isaac Newton in terms of physics. And that will be so, because we are only beginning to perceive the ramifications of this enormous revolution that was triggered by their discovery. That is the field of molecular biology and genetics and biochemistry which has totally changed our perceptions of how life on Earth is actually organized.

—Prof. Robert A. Weinberg

This chapter provides a description of the main problem this dissertation addresses, the key idea of our solution and the main results and contributions of our work.

The problem that we will focus on is the lossless compression of biological sequences. In other words, reducing the number of bits needed to represent deoxyribonucleic acid (DNA) and amino acid sequences without loss of information. DNA encodes amino acid sequences (proteins), so given a sequence of DNA, the corresponding amino acids can be obtained, as seen in Fig. 1.1.

DNA and amino acids have a 3D structure, as seen in Fig. 1.3, however the result of sequencing them is a linear string of characters, that is normally stored in "FASTQ" or "FASTA" formats, which are uncompressed text files [1]. Examples of "FASTA" file contents can be seen in Fig. 1.2.

The problem of biosequence compression is important, because of the exponential increase in sequencing that has been occurring, and is projected to continue in the foreseeable future [2]. Discarding the sequenced data is not an alternative, given its high importance in many contexts, specifically in biomedical (e.g., personalized medicine) and anthropological fields. In this context, compression has three main usages. First, the reduction of storage costs. Reducing these might seem unjustified given the large memory capacities available today, but not only is storage currently one of the primary costs of sequencing [3], it is also projected that by the year 2025 between 2 and 40 exabytes of sequences will be pro-

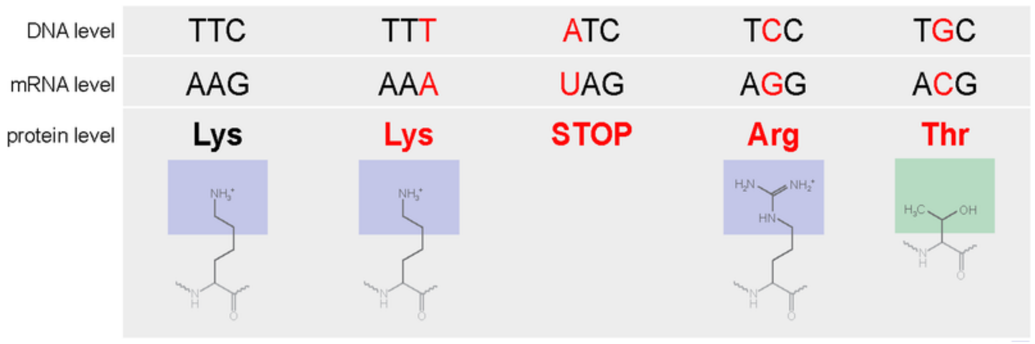


Figure 1.1: Translation from DNA to amino acids. Adapted from Jonsta247, licensed under CC BY-SA 4.0.

```
>NC_000011.10:c5227071-5225464 Homo sapiens chromosome 11, GRCh38.p13 Primary Assembly
ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAGACACCATGGTGCATCTGACTCCTGA
GGAGAAGTCTGCCGTTACTGCCCTGTGGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGC
AGGTTGGTATCAAGGTTACAAGACAGGTTTAAGGAGACCAATAGAAACTGGGCATGTGGAGACAGAGAAG
ACTCTTGGGTTTCTGATAGGCACTGACTCTCTCTGCCTATTGGTCTATTTTCCCACCCTTAGGCTGCTGG
TGGTCTACCCTTGGACCCAGAGGTTCTTTGAGTCCTTTGGGGATCTGTCCACTCCTGATGCTGTTATGGG
CAACCCTAAGGTGAAGGCTCATGGCAAGAAAGTGCTCGGTGCCTTTAGTGATGGCCTGGCTCACCTGGAC
AACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCACTGTGACAAGCTGCACGTGGATCCTGAGAACT
TCAGGGTGAGTCTATGGGACGCTTGATGTTTTCTTTCCCCTTCTTTTCTATGGTTAAGTTCATGTCATAG
GAAGGGGATAAGTAACAGGGTACAGTTTAGAATGGGAAACAGACGAATGATTGCATCAGTGTGGAAGTCT
CAGGATCGTTTTAGTTTCTTTTATTTGCTGTTCATAACAATTGTTTTCTTTTGTTTAATTCTTGCTTTCT
TTTTTTTTCTTCTCCGCAATTTTTACTATTATACTTAATGCCTTAACATTGTGTATAACAAAAGGAAATA
TCTCTGAGATACATTAAGTAACTTAAAAAAAAACTTTACACAGTCTGCCTAGTACATTACTATTTGGAAT
ATATGTGTGCTTATTTGCATATTCATAATCTCCCTACTTTATTTTCTTTTATTTTTAATTGATACATAAT
CATTATACATATTTATGGGTTAAAGTGTAATGTTTTAATATGTGTACACATATTGACCAAATCAGGGTAA
TTTTGCATTTGTAATTTTAAAAAATGCTTTCTTCTTTTAATATACTTTTTTGTTTATCTTATTTCTAATA
CTTTCCCTAATCTCTTTCTTTCAGGGCAATAATGATACAATGTATCATGCCTCTTTGCACCATTCTAAAG
AATAACAGTGATAATTTCTGGGTTAAGGCAATAGCAATATCTCTGCATATAAATATTTCTGCATATAAAT
TGTAACTGATGTAAGAGGTTTCATATTGCTAATAGCAGCTACAATCCAGCTACCATTCTGCTTTTATTTT
ATGGTTGGGATAAGGCTGGATTATTCTGAGTCCAAGCTAGGCCCTTTTGCTAATCATGTTCATACCTCTT
ATCTTCCTCCCACAGCTCCTGGGCAACGTGCTGGTCTGTGTGCTGGCCCATCACTTTGGCAAAGAATTCA
CCCCACCAGTGCAGGCTGCCTATCAGAAAGTGGTGGCTGGTGTGGCTAATGCCCTGGCCCACAAGTATCA
CTAAGCTCGCTTTCTTGCTGTCCAATTTCTATTAAAGGTTCCTTTGTTCCCTAAGTCCAACTACTAAACT
GGGGGATATTATGAAGGGCCTTGAGCATCTGGATTCTGCCTAATAAAAAACATTTATTTTCATTGCAA

>sp|P68871|HBB_HUMAN Hemoglobin subunit beta OS=Homo sapiens OX=9606 GN=HBB PE=1 SV=2
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPK
VKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFG
KEFTPPVQAAYQKVVAGVANALAHKYH
```

Figure 1.2: Example FASTA sequences of DNA (top) and amino acid (bottom) for human hemoglobin beta. Both begin with a header that contains information of the sequence. The following lines contain the letters that encode it.

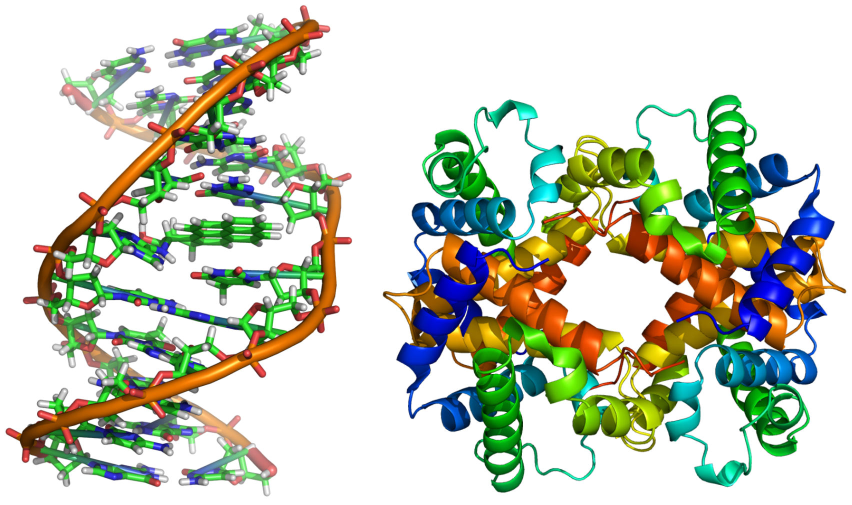

(a) DNA (b) Hemoglobin beta

Figure 1.3: An illustration of the 3D structure of DNA (Fig. 1.3a) and amino acid sequence (hemoglobin beta) (Fig. 1.3b). By Zephyris, and Emw licensed under CC BY-SA 3.0.

duced every year. These estimates, put genomics on par with the most demanding domains (e.g., astronomy) [2]. Researchers often need to download many sequences and this can be time-consuming and difficult especially, with large files. By using data compression, the transmission costs can be reduced [4]. Finally, data compression is used for the analysis of biological sequences [5]. This is something that might not be immediately obvious, but it was actually one of the first use cases of compression applied to DNA sequences [6, 7] and remains one of the most important [8]. Compression can be used to identify related species, segment DNA into homogeneous sequences, find important mutations, among many others [8, 9].

Biosequence compression can be solved either with general or special purpose compressors. By exploiting the unique characteristics of the sequences, specialized compressors achieve results that are better in time, memory and resulting size than their general purpose counterparts [10, 11]. Furthermore, in the case of protein compression, using the sequence (primary structure) along the secondary structure information is more efficient than compressing just the sequence [12].

Compression can be separated into two main phases: modeling and coding [13, 14]. During modeling, the next symbol of the sequence is predicted. This prediction is then feed into the encoder along with the actual symbol. A higher probability given to the symbol, result in fewer bits needed to encode it [15]. Coding is considered a solved problem, but modeling is not [13]. For coding there are methods that are proved to output the most efficient representations. This is the case with Huffman coding, in the case of encoding one symbol at a time with known probability distributions [16]. In the other cases, arithmetic coding achieves close to optimal coding with a small overhead [17]. On the other hand, modeling is provably not solvable [13]. The problem of modeling can be reduced to finding a program that outputs the sequence we want to compress [18]. We can search only for programs that are smaller than the sequence, but the issue is that some programs don't halt and we have no way of knowing which do and which don't [19].

One way to predict the next symbol is to combine the predictions of multiple experts. If the mixing is adequate, then the final prediction will be better than any individual expert over the entire sequence. Deliberate combination of experts' opinions dates back to at least 753 BC in the Senate of the Roman Kingdom [20]. This was a gathering of elders who voted on laws and provided advice for the king among other things. Even now, many institutions use majority voting as a solution for the mixing of opinions. This scheme can be improved by noticing that, given certain contexts, some experts are more likely to be correct, and so we assign more importance to their opinion. This is called weighted majority voting [21]. There is a field of artificial intelligence that studies how to create and combine appropriate experts. It is called ensemble learning or mixture of experts [22].

The GeCo2 [23] and AC [24] are state-of-the-art compressors for DNA and amino acids, respectively. These data compressors use a number of configurable Markov models along with substitution tolerant context model [25]. To combine the predictions of these models, both compressors use an algebraic combiner, where weights are attributed to each model and updated based on the model performance, with a specific forgetting factor for each.

The key idea of this dissertation is to augment the mixing of GeCo2 and AC with a stacked generalization approach [26]. This consists of using a learner that receives the outputs of the models and is trained to give the correct probabilities for the symbols. To learn the mapping between model outputs and the correct symbol we use a multilayer perceptron trained online with stochastic gradient descent.

We produce two new tools (GeCo3 and AC2) that incorporate this new mixing method. To benchmark these, we compare them against a range of state-of-the-art compressors with balanced, fair and wide datasets. Our results show substantial improvements in the compressed size at the cost of higher processing time. In the very near future, this cost should get smaller, due to the inclusion of specialized processing for neural networks in general purpose CPUs [27, 28]. The mixer used in these tools only needs as inputs the probabilities of the models, so it can be easily exported to other compressors. To aid in this process, we also provide instructions on how to integrate the mixer into existing data compressors. The GeCo3 and AC2 can enable improved analysis, long term storage, and for the first time, shows the power and cost of advanced expert mixing with specialized models in the context of biosequence compression.

1.1 Main contributions

The main contributions of this master project include new software:

- The GeCo3 tool, for compression, decompression and analysis of DNA sequences. Available at `https://github.com/cobilab/geco3`.

- The AC2 tool, for compression, decompression and analysis of amino acid sequences. Available at `https://github.com/cobilab/ac2`.
- The portable mixer and instructions on how to integrate it into other projects. Available at `https://github.com/cobilab/nn-expert-mixer`

Additionally:

- A journal article [29] summarizing the key results of GeCo3.
- The necessary instructions and data for full reproducibility of the article results [30].

1.2 Dissertation overview

This dissertation is structured in five chapters. The first is the present one and contains an introduction to the problem, the motivation to tackle it, an outline of our approach to solve it and the qualitative results. The next chapters contain:

- Chapter 2, historical context for the three main threads of this work (data compression, mixture of experts and biosequences) and a summary of the state-of-the-art in biosequence compression;
- Chapter 3, a description of the mixer used, its properties and details of the implementation;
- Chapter 4, the benchmarks for DNA and amino acid sequences compression, with several datasets, compressors and analysis of these results;
- Chapter 5, some concluding remarks along with future directions.

Chapter 2

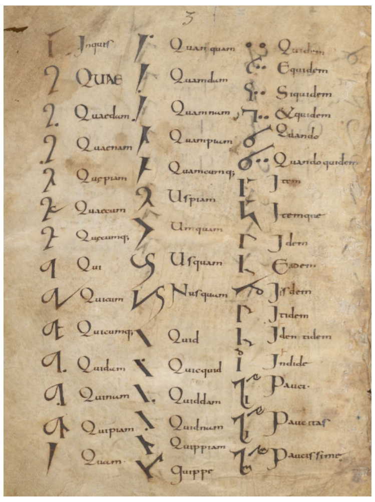
Background

This chapter serves to provide an overview and historical context of the necessary tools needed to achieve our goal of improving biosequence compression. We start with a summary of the history of data compression and the motivations for applying this technique. We then move to the problem of biosequence compression, exploring the unique features of biosequences and how the specialized compressors use them to achieve better results than the general purpose data compressors. Next we discuss how the mixture of experts can be done and show how neural networks can be organized and trained. Finally, we try to answer what type of network is more adequate to use for the mixture of experts.

2.1 Brief history of compression

Data compression is the process by which the number of bits required to represent the same information is reduced. This modern definition seems to imply that compression is inherently tied to computers, but actually one of the earliest evidence of deliberate compression comes from the Athenian Acropolis, fourth century BC, where a system of abbreviations was used for writing [31, pg. 1425]. Curiously, this is also the time and place of what can be considered the first computer (Antikythera mechanism), the first steam engine (aeolipile) among many others [32]. These early signs of tachygraphy are not considered a fully mature shorthand system. This emerged in the first century BC, where it was used to record the speeches of Cicero [33]. There are actually two forms of data compression happening in this instance. The first is lossless compression of the text by usage of shorthand, and the second is lossy compression of speech.

An important landmark in the history of compression occurs in 1837, when Samuel Morse created an early version of Morse code. It was used to encode text transmitted by telegraph [34]. Like many modern encoding schemes, Morse code uses fewer bits to represent more frequent symbols. This important insight began formalization with the inception of information theory in 1948, by Claude Shannon [15]. In 1952, Huffman presented a coding algorithm that is optimal, provided that the symbols are coded individually [16]. In the same year, one of the first application of compression to telecommunications was proposed


(a) Ancient shorthand

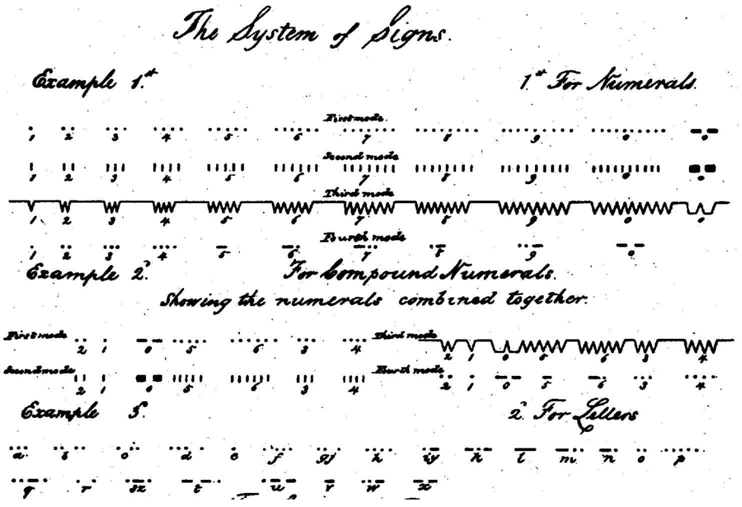

The System of Signs.
Example 1st
1st For Numerals.
First mode
Second mode
Third mode
Fourth mode
Example 2.
For Compound Numerals.
Showing the numerals combined together.
Example 3.
2. For Letters
(b) Morse code

Figure 2.1: Fragment of Ancient Roman shorthand encoding scheme, and an excerpt of the original Morse code patent showing the number of dots and dashes to be used for numbers and letters.

[35]. By 1967, prototypes existed to compress television broadcast using run-length encoding [36]. Run-length coding consists of noticing that long sequences of repeated symbols can be substituted by the repeated pattern and a number corresponding to the length of the repetition [37]. For example, if we have the sequence 0000000000 we can encode it by $(0, 10)$, which means: repeat the symbol zero ten times.

In 1977 and 1978 a new class of lossless compressors was proposed by Abraham Lempel and Jacob Ziv [38, 39]. The main idea of the algorithms is to notice that strings of symbols may occur several times in the sequence to be compressed. For example in English text, the word "that" is very common. This entire word can be coded by an identifier such as the relative position of the last occurrence and the length of the word. This forms an implicit dictionary, but an explicit dictionary can also be used, in that case, the identifier can be the index of the word. The descendants of these dictionary coders are in widespread use in image compression (PNG and GIF), in the ubiquitous ZIP format and in many others.

An important coding improvement was introduced with the generalized arithmetic coder in 1979 [40]. Unlike Huffman coding, symbols are not coded one at a time. This allows arithmetic coding to get very close to the optimal coding [41].

In 1996, Schmidhuber and Heil introduced a data compressor based on neural networks. This showed superior compression to dictionary methods, but was thousands of times slower [42]. In 2000, Mahoney refined this method to use online training and predict one bit at a time (instead of a character). The resulting algorithm was five orders of magnitude faster. Two years later saw the birth of the PAQ family of compressors also by Mahoney. The first PAQ was based on neural networks, but using counts for predictions instead of weights. As of 2020, PAQ and its descendants are top ranked in various compression challenges. The

Cmix [43] is one of the derivatives introduced in 2014. It used thousands of independent models mixed with neural networks and currently holds the record for compression of the enwik9 corpus [44].

Data compression is pervasive in all types of data stored and transmitted by computers. Compression is used in websites [45], video games [46], images [47], audio [48] and video [49]. It is so pervasive that encoders and decoders are implemented in specialized hardware and included in almost every consumer CPU and GPU [50, 51].

2.2 The advantages of data compression

From the examples we mentioned, two advantages of data compression can be deduced: storage size and transmission speed. Now we touch upon a third advantage of data compression. We argue that the development of modern mathematical notation, is also an example of data compression. Before this, mathematical problems where reasoned about in natural language. As an example, consider the following excerpt from [52]:

> What is the square which when taken with ten of its roots will give a sum total of thirty nine? Now the roots in the problem before us are ten. Therefore take five, which multiplied by itself gives twenty five, an amount you add to thirty nine to give sixty four. Having taken the square root of this which is eight, subtract from this half the roots, five leaving three. The number three represents one root of this square, which itself, of course is nine. Nine therefore gives the square.

The modern formulation of this same problem would be:

$$x^2 + 10x = 39, \tag{2.1}$$

and the solution:

$$x = \left(\sqrt{\left(\frac{10}{2}\right)^2 + 39}\right) - \frac{10}{2} = 8 - 5 = 3 \implies x^2 = 9. \tag{2.2}$$

In this case, data compression is used not as a way to improve transmission or storage of data, but as tool of thought [53]. It reveals a new structure to mathematical problems that was previously difficult to see in natural language, and it enables easier manipulation of the problem and its solution [54].

Another example of compression as a tool of though, comes from programming. During this activity, compression can be used as a thinking tool to organize and build abstractions by noticing repetitions or patterns and then removing them, thus compressing the source code [55].

2.3 Data compression of biosequences

Compressors can be classified into two classes: general purpose and specialized. General purpose compressors address multiple natures of data being characterized by flexibility while consuming low computational resources, generally, attaining satisfactory compression rates. Specialized compressors have the advantage of being able to exploit the distinct characteristics of certain types of data, often these compressors can't deal with data for which they were not designed for.

Biosequences have several specific properties, namely high copy number, high heterogeneity, high level of substitution mutations, and multiple rearrangements, such as inverted repeats [56, 57]. Since these sequences are an output of biochemical and computational methods, they may have other alteration sources, for example environmental factors [58, 59], pathogen species [60, 61], and unknown sources [62]. Representing biosequences requires the ability to model heterogeneous, dynamic, incomplete, and imperfect information [63]. These specific characteristics led to the development of the field which studies and constructs specific DNA and amino acid compressors.

2.3.1 Biosequence properties

We now explain and give examples of some of the defining characteristics of biosequences.

- Single Nucleotide Polymorphism (SNP): A place where a single base is changed (substitution), deleted (deletion) or inserted (addition). E.g. TT**C** to TT**T**.
- Insertions and Deletions (Indels): Addition or deletion of multiple bases.
- Tandem Base Mutations: Substitution of multiple adjoining bases.
- Direct repeat: A sequence of bases that repeats, with or without other bases in-between. The latter is also called a tandem repeat. E.g. **TTACG**...**TTACG**.
- Mirror Repeat: A repetition of the reversed sequence, also called a palindrome. E.g. **TTACG**...**GCATT**.
- Pairing Repeat: A repetition of the sequence complement. E.g. **TTACG**...**AATGC**.
- Inverted repeat: A repetition of the mirror complement. E.g. **TTACG**...**CGTAA**.
- Fusion: Merging of two segments of DNA into a single new segment. For example, the human chromosome 2 is probably the result of the fusion between the chromosomes 2A and 2B of primates.
- Fission: Splitting of one DNA segment into two or more segments.

- Translocation: Transfer of a DNA segment from one chromosome to a nonhomologous chromosome or to a new site on the same chromosome.
- Mutation probabilities: Some mutations are more likely to occur than others. This is due to the fact that some substitutions either in DNA or amino acids do not drastically alter the function and others do. These probabilities are known and can be exploited to improve compression, especially in the case of amino acids [64].

2.3.2 DNA sequence compression

Compression of DNA sequences can be done by applying two main techniques. A sequence can be compressed without using additional sequences, this is called reference-free compression (horizontal mode). A sequence can also be compressed with respect to one or more reference sequences, in this case the method is reference-based (vertical mode). Reference-based methods achieve substantial compression ratios when the similarity between the reference sequence and the target sequence is high. This is the case between individuals of the same species, related species or in the case of DNA from the same individual that was re-sequenced. We now present the main developments of both techniques separately. There are some algorithms that have both compression types, in that case, they are mentioned in both sections if there are substantial differences between the horizontal and vertical mode.

Reference-free compression

The first specialized DNA compressor, Biocompress [65], was introduced in 1993. The Biocompress algorithm supports both reference-free (horizontal mode) and referential (vertical mode) compression. It is based on the universal lossless data compression algorithm proposed by Lempel and Ziv (LZ) [38], augmented by the detection of palindromes and repeats which are then coded by the position and size of the first occurrence. In the following year, Biocompress-2 was introduced. The main difference was the introduction of an arithmetic encoder of order two [66].

The Cfact algorithm [67] uses a suffix tree to obtain the longest exact repeats combined with dictionary coding. Two bits are used per symbol on the first occurrence or when no repeats are detected, otherwise an identifier is used for the repeat.

The CDNA [68] algorithm introduced the notion of approximate repeats, while using a mixture of experts. The weight of each model is learned by the expectation–maximization algorithm. ARM [69] models approximate repeats including reverse complement repeats. Like CDNA, a mixture of experts is also used with expectation–maximization (EM) for weight updates. A faster alternative to EM is also proposed along the ARM. At the time they where introduced, the CDNA and ARM algorithms vastly outperformed other algorithms in resulting compressed size.

GenCompress [70] and DNACompress [71] generalize the Lempel Ziv algorithm to handle approximate repeats.

NMLComp [72] uses the normalized maximum likelihood to model approximate repeats, and combines it with a first order Markov model. The combination is done by selecting the model that produces the shortest code-length. The GeNML algorithm [73] improves NMLComp both in speed and compression ratio by reducing the match length. This has the effect of reducing the cost of search and it is also used get smaller code lengths.

XM [74] uses a combination of experts, some specialized in statistical properties (Markov and context Markov models) and other in repetitions (copy and reverse models), and the result is encoded with an arithmetic encoder. The combination is done based on Bayesian averaging and the model weight's are updated based on the average code length of the model. In terms of compression ratio this is still one of the best DNA compressors, at the cost of higher computational resources.

DNASC [75] uses statistical and substitutional methods (based on the LZ algorithm). It first compresses the DNA horizontally and then vertically by dividing the horizontal result into blocks. DNACompact [76] converts the DNA sequence into a sequence of words and then uses Word-Based Tagged Code encoding. FCM-Mx [77] uses multiple finite-context models combined by mixing the predictions with a dynamically updated weight per model. The weight is updated according to the probability that the sequence was generated by that model and follows an exponentially decaying impulse response.

POMA [78] uses substitutional methods for vertical compression. It uses an hybrid approach based on particle swarm optimization to efficiently design the reference dictionary. The GenCodex [79] algorithm focuses on compression speed by using a parallel algorithm that can be executed in multi-core CPUs or GPUs. GenCodex encodes each of the four DNA symbols into two bit patterns, if there are consecutive byte repetitions then they are encoded into two bytes, one for the pattern and one for the count of repetitions.

DNA-COMPACT [80] is a vertical and horizontal DNA compressor based on two passes. The first pass is dedicated to finding repetitions while the second is dedicated to statistical properties. The latter uses multiple models and combines their predictions using logistic regression, the result is passed to an arithmetic encoder.

HighFCM [81] relies on pre-analysis of the DNA sequence to find low complexity regions and then use Markov models with deep contexts (up to 32). These models enable good compression of highly repetitive sequences while keeping the memory low. Multiple models with different context are used in a competitive fashion, that is, the predictions of best one is are selected to encode a given block of fixed size.

CoGI [82] treats the DNA sequence as a binary image and then partitions the image into black rectangles, encoding their positions and sizes with a static entropy coding scheme. It supports reference-based and reference-free encoding.

GeCo [83] uses multiple context and substitution tolerant context models with a mixing similar to FCM-Mx. Substitution tolerant context models assume that a number of point mutations can occur in the sequence before discarding the context. The OCW [84] algorithm focuses on the mixing of models and more specifically how to weight the context models and optimize these weights using the least-square algorithm.

OBComp [85] calculates the frequencies of the four DNA symbols and then encodes the two most frequent with one bit each. The other two are encoded by position in separate files.

Afterwards, it uses a modified run-length encoding followed by the Huffman algorithm.

GeCo2 [23] is an evolution of GeCo that allows more control over the compression parameters. Namely, each model can have a specific weight decay, cache-hash sizes can also be controlled and finally, context models can be made to run only with inverted repeats. The Jarvis [86] uses context models, substitution tolerant context models and stochastic repeat models. The mixing of context and repeat models is done with a soft blending algorithm similar to FCM-Mx/GeCo. The final prediction is chosen in a competitive fashion between the mixed context and mixed repeat models.

NAF [11] encodes sequences into four bits to be able to represent all possible FASTA characters and then passes the result to a general-purpose compressor (zstd).

Reference-based compression

DNAzip [4] was an algorithm proposed to allow the compression of the entire reference human genome down to a size appropriate to be sent as an email attachment. The motivation was to greatly facilitate the sharing of the genomes. The algorithm assumes that information about the DNA variations (i.e. point variants, insertions/deletions and inversions) is provided. It then compresses this information by using four techniques: variable integer sizes for the positions of the variations (VINT), relative instead of absolute positions (DELTA), SNP mapping (DBSNP) and K-mer partitioning (KMER).

RLZ [87] uses self-indexes, a data structure that provides efficient operations on compressed text, to store the reference sequence as a dictionary and then uses LZ encoding for all other sequences.

GRS [88] finds the longest common sequences, and encodes the difference using Huffman coding.

GDC [89] is similar to a variation of the RLZ algorithm, but it uses hashing instead of self-indexes. Compression is also based on the LZ algorithm with Huffman coding and is performed on blocks to allow random access.

COMRAD [90] is also based on a dictionary approach with Huffman encoding. The algorithm uses a general purpose technique that allows random access.

GReEn [91] uses a copy model and a frequency model based only on the properties of the target sequence. The prediction for the next symbol is selected either from the copy or the frequency model and feed into an arithmetic encoder.

FRESCO [92] constructs a compressed suffix tree for the reference sequence and uses it to searches for matches between the reference and target sequence. The matches are delta encoded. This algorithm is targeted at compressing collections of genomes.

GDC2 [93] is an evolution of GDC. It constructs an hash-table with linear probing for the reference sequence an uses two-level LZ factoring followed by an arithmetic encoder. Like FRESCO the main purpose is to compress collections of genomes.

iDoComp [94] consists of three steps: a mapping generator, post-processing of the mapping and encoding done with and arithmetic encoder. The mapping generator assumes a suffix array is pre-computed for the reference sequence and then uses greedy parsing to produce the matches. This is followed by an heuristic to look for consecutive matches than can be merged.

ERGC [95] is a multi-stage algorithm suitable for parallel execution. It does this by dividing the reference and target sequence into blocks and uses an hash-table with delta encoding.

HiRGC [96] is based on two bit encoding with greedy maximum matching on a hash-table followed by a PPMD encoder.

HRCM [97] first extracts reference sequence information, then it matches the sequences using a hash-table and finally delta encodes the information.

2.3.3 Amino acid sequence compression

In this section we discuss the specific amino acid compressors. Some of the DNA compressors also allow protein compression, therefore, we avoid to repeat them.

One of the first attempts at protein compression was done in 1999 [64]. The main conclusion of the paper, reflected by the title ("Protein is incompressible"), was that proteins, from a single organism, are incompressible. The proposed algorithm (CP) uses multiple models and mixes the predictions according to predetermined weight. Later, a paper with the opposite title ("Protein is compressible") presented the algorithm ProtComp [98]. ProtComp exploits approximate repeats and uses an hybrid method combining a substitution approach using Huffman coding and a first order Markov model with arithmetic coding. ProtCompSecS [12], adds to ProtComp a dictionary based method to encode the secondary information related to proteins.

The algorithm presented in [99] uses the Burrows–Wheeler transform and the sorted common prefix combined with substitutions to exploit the long range correlation in amino acid sequences.

CaBLASTP [100] fuses dictionary and sequence alignment methods for the compression of protein databases. The algorithm searches for good sequence alignments, and when one exists, stores an index instead of the sequence.

AC [24] uses an ensemble of Markov models (finite context and substitution tolerant) with adaptive weights per model and arithmetic encoder, the algorithm is similar to the GeCo2 DNA compressor.

2.3.4 Main ideas

With an overview of the landscape of biocompressors, we can start to distill the main trends for the compressors. There are three main approaches: dictionary, statistical and hybrid. Dictionary compressors are based on the LZ algorithm or some variation of it. The method consists of finding repeated subsequences and then encoding that information. Statistical methods use one or more context models (or some variation) that predict the next symbol. Unlike dictionary methods, the sequence is usually processed one symbol at a time. Hybrid approaches, as the name implies, use some combination of both methods. In hybrid compressors, the first stage is typically based on a dictionary method and the remaining symbols are compressed with statistical models. In Fig. 2.2, a diagram shows the main steps for dictionary and statistical compressors.

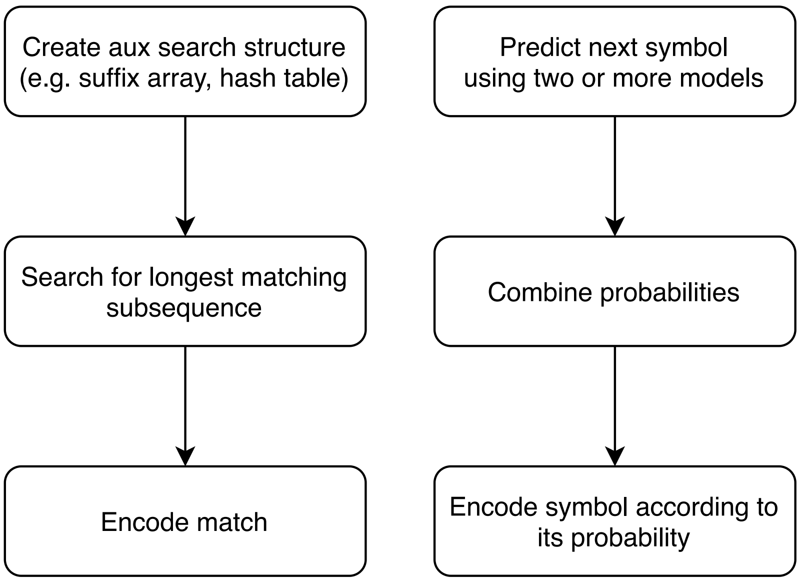
Due to the way dictionary compressors work, the compressor and decompressor are very different, both in implementation details and performance characteristics. The compressor needs to do the heavy lifting (search for the optimal subsequences), while the decompressor needs to copy data from the compressed file and the already partially decoded sequence. On the other hand, statistical compressors and decompressors are more symmetric. The work done in the compressors, updating the models and combining the predictions, needs to be repeated in the decompressor.

For reference-based compression, when the reference sequence and target sequence are very similar, the best results are often obtained with dictionary methods. With reference-free compression, statistical methods generally produce better results due to the high adaptability of the models, which allows them to cope with the high occurrence of mutations.

Crucial to all methods that use multiple models is the choice of how to combine the predictions. Some methods opt for a competition of models, where the model with best compression for a subsequence is chosen. This method needs to do the compression for all models and then choose the best, thus the information of the best model needs to be passed to the decompressor. To amortize this added cost, this method is generally used with blocks of a fixed size. This leaves room for improvements, as other models can be better in different parts of the block.

The other method of combining the experts is cooperation. In this case, all models contribute to the final prediction. The exact way the mixing is done will be discussed in the next section. Besides the initial parameters for the mixer, no side information is need for the decompressor, because it will do the same operations as the compressor.

Our objective is to improve the mixing of experts of two statistical compressors: GeCo2 and AC. In the next section we survey the available approaches to solve this problem.


Create aux search structure (e.g. suffix array, hash table)
Search for longest matching subsequence
Encode match
Predict next symbol using two or more models
Combine probabilities
Encode symbol according to its probability

(a) Dictionary compressor. (b) Statistical compressor.

Figure 2.2: The main steps for dictionary and statistical compressors.

2.4 Mixture of experts

Mixture of experts or ensemble learning is a subfield of artificial intelligence concerned with the creation of models (experts) and the merging of their opinions [101]. These are typically represented either by probabilities (continuous) or votes (discrete) assigned to each possible answer. For our purposes we are concerned with the merging of opinions. This combination can be done statically or dynamically. One of the problems to be solved in ensemble learning is how to deal with non-stationary (drifting) sources [102, 101]. In non-stationary sequences, the probability distribution changes over time, due to this dynamic nature, models and mixers that can adapt online have better performance [102]. This problem is of great importance to us, because DNA appears to be non-stationary [103]. The GeCo2 and AC already deal with this problem by using a passive approach. Passive methods do not explicitly detect concept drift, but nevertheless constantly adapt to the current performance of the models. In GeCo2 and AC, both experts and weights are updated. Unlike the method proposed in [104], no models are removed or added to the ensemble.

We now focus on the two main approaches to combination: weighting and meta-learning.

2.4.1 Weighting

With weighting, the expert opinions are combined by assigning weights to each. The weights can be updated according to the models' performance. The simplest weighting method is majority voting, that is, the answer with most votes is the final output. As a refinement, it can be noticed that some models are better than others, and so more weight is attributed to those models. This scheme is called weighted voting. These methods, and other variations, work for discrete answers (classes), but in the case of GeCo2 and AC, the models output continuous values. These algorithms use an algebraic combiner with weights for each model, updated to reflect their performance. In the literature, this is called weighted average [22]. Other algebraic combiners are possible, e.g., selecting the maximum, minimum or mean value for each class. In the case of GeCo2 there are four classes representing the DNA, in the case of AC the number of classes is around twenty.

Weights can be fixed or dynamic. The former can be determined automatically by training in a validation set, or manually after analysis of the expected model properties. Dynamic weights can be updated based on the likelihood of obtaining the model given the data (Bayesian Model Averaging [105]), the final output confidence [106] or using optimization techniques to minimize the prediction error [107].

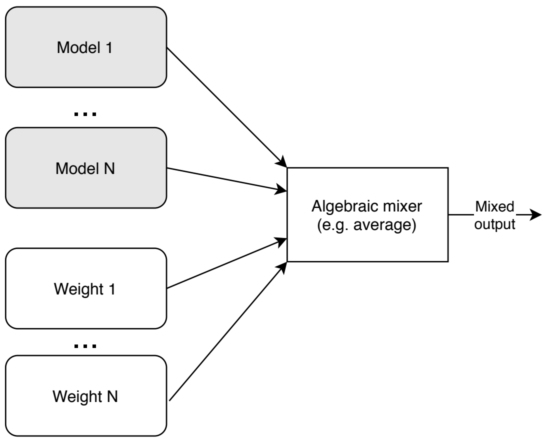


Figure 2.3: An abstract view of the weighting approach which has explicit weights for each model.

2.4.2 Meta-learning

Meta-learning or stacked generalization [26], uses a learner supplied with the models' outputs and trained to give the correct answer. At a high level, this is not very different from using optimization techniques to minimize the error, except that in this case the weights are not explicit. Not having explicit weights allows the learner to output probabilities outside the range of the models' predictions. That is, stacked generalization can be used even with a single model, in this case to correct the output of that model.

The learner used can range from a simple linear regressor, to a more complicated one, such as a neural network. Neural networks are used in some of the most advanced data compressors, both as mixers and models [43, 10, 108, 109]. Due to their good results in a wide range of tasks [110, 111, 112], we have elected to use them for the mixing approach in

GeCo3 and AC2. Like the previous versions of these algorithms, we will deal with concept drift in passive way by using online learning (training the network at every new symbol in the sequence). Another interesting effect of constant training is that besides the adaptation to concept drift, this might also lead to occasional overfitting [113] which has been reported to be beneficial in non-stationary sources [114].

Next, we provide an overview of neural networks so that we may choose an appropriate type for integration with GeCo2 and AC.

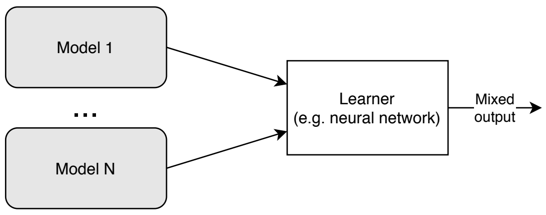


Figure 2.4: In the meta-learning approach, the learner can be viewed as a black box that takes the model outputs and is trained to output the correct class.

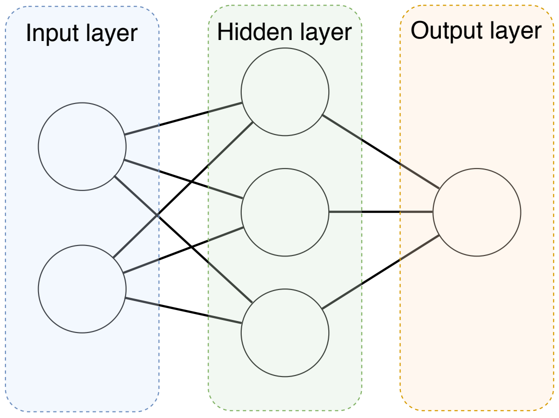
2.5 Neural Networks

The first published work related to artificial neural networks was by McCulloch and Pitts in 1943 [115]. This early work focused on creating a model of the nervous system, described by logical calculus based on propositional logic. Despite the initial interest in the topic, neural networks had two main shortcomings, they could not be trained to do general computing (e.g., the networks could not compute the XOR function) and the required resources to run the networks were too high. These problems lead them to falling out of fashion. To solve the XOR problem, Minsky and Papert [116], showed that multilayer neural networks were needed, but no training algorithm existed for them. At the time, only layerless networks could be trained with the algorithm developed in [117].

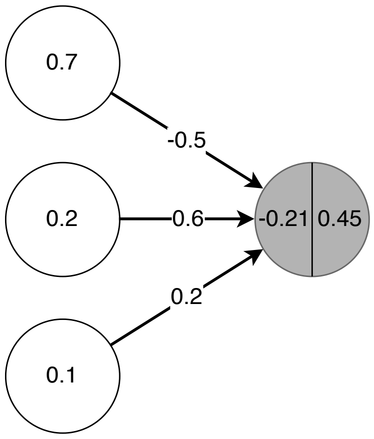
Backpropagation, introduced in 1974 [118], was the first algorithm to allow the training of multilayer networks, thus allowing the XOR problem to be solved. Meanwhile, due to advances in hardware manufacturing and architecture, the computational resources continued to improve. These compounding advances have allowed a new approach to be used, called deep learning. This uses many layers with different properties, stacked to achieve the best results in many tasks [119], including playing games such as Go [120] or Dota2 [121], language models [122], image analysis [123], protein structure prediction [124], and many others [125]. The advantage of these many layers is to remove the need for manual feature extraction. The extracted features are many times crucial to obtain good results with neural networks [119]. With deep learning, the raw data can be feed directly into the deep network, and the training, along with the different layers, automatically extracts the necessary features.

We now explain how some of most popular neural networks work. The techniques explained are the basic building blocks that can then be used to build more complex networks.

2.5.1 Multilayer perceptron



(a) Example multilayer perceptron architecture.



(b) Single perceptron result.

Figure 2.5: In (a), an example multilayer perceptron with two inputs nodes and one output. This network is organized into three fully connected (dense) layers, meaning that nodes in previous layers connect to all nodes in the next layer. In (b), an example of the forward pass calculation for a node (in grey). The left value (-0.21) is the value after the dot product but before the activation, the right value (0.45) is the value after the sigmoid activation.

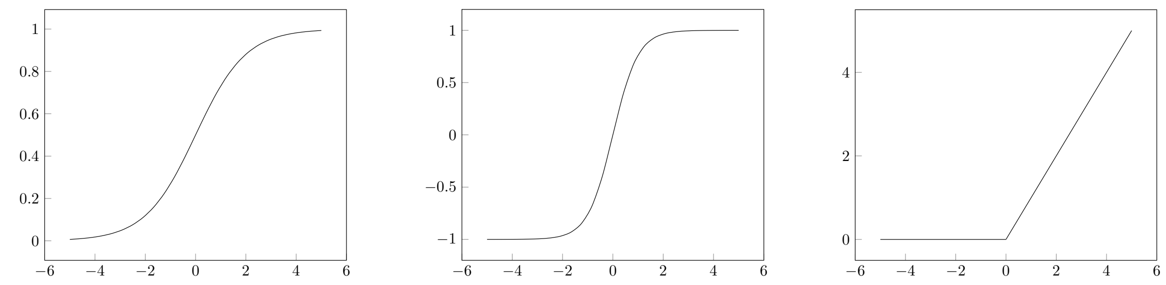
One of the simpler neural networks is the multilayer perceptron. This network is composed of two fundamental units, the connections and the neurons. The network can be seen also as a graph where the connections and neurons are the edges and nodes respectively. The edges and nodes are typically represented by a floating point number. In Fig. 2.5a, we show a representation of a multilayer perceptron where each node connects to all nodes in the next layer. This network has inputs and output nodes corresponding to the values of the problem to be solved. For example, this network could be used to model a binary XOR circuit. The network edges are initialized with random numbers.

To obtain the value of a node, the values of all nodes connected to it are multiplied by the edge values and summed together. Finally, the summation is passed to an activation function and that is the final node value, as show in Fig. 2.5b. This can be represented by the following equation:

$$node_{i,j} = f\left(\sum_{k=1}^{n} node_{i-1,k} \times weight_{k,j}\right), \tag{2.3}$$

where j and k are indices for nodes in layer i and $i-1$, respectively, and n are the number of nodes in layer $i-1$. The activation function is represented by f, some examples of these can be see in Fig. 2.6. The calculations are repeated for all nodes in the network. Organizing these into layers, we can obtain the value for all nodes in one layer by doing a matrix multiplication of weights by node values of the previous layer.

For this type of network, an important consideration to be taken into account, is the use of bias neurons. These are nodes that have a fixed value equal to one. They allow the network to output values different from zero even when all inputs are zero. Another consideration is the weight initialization, the weights are generally randomly initialized, but the type of distribution used can help improve the training time. Finally, the activation function is almost always non-linear to allow the network to model non-linear functions.



(a) Logistic activation function.

(b) Hyperbolic tangent activation function.

(c) Rectified linear unit activation function.

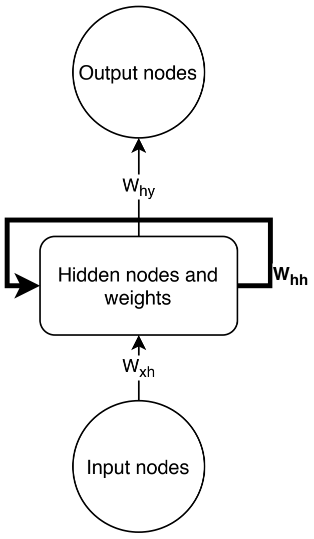
Figure 2.6: Three examples of non-linear activation functions that can be applied to the neural network nodes.

2.5.2 Recurrent neural networks

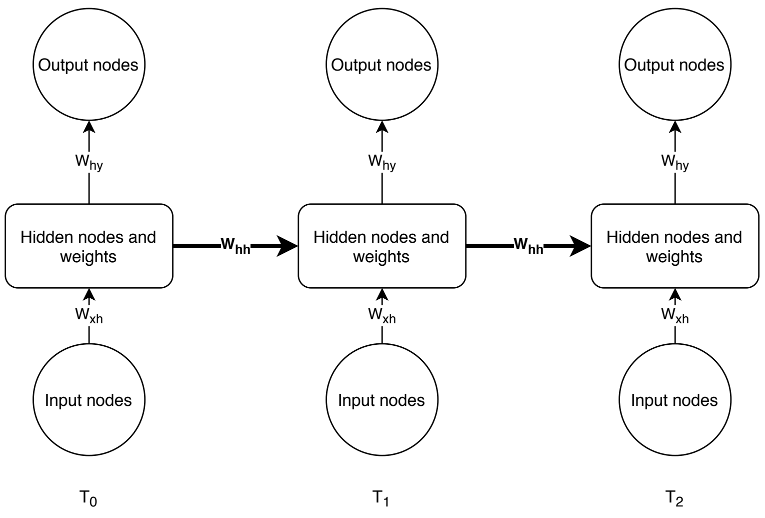
One of the first works about recurrent neural networks appeared in 1985 [126]. Recurrent neural networks (RNNs) are most useful to deal with sequential data [127]. These networks are equivalent to feedforward networks (such as the multilayer perceptron) where the feedforward network is duplicated at each time step. In other words, they have a similar architecture to the MLP, but with the main difference being that they propagate the hidden parameters to the next training step.

In Fig. 2.7, we show a diagram of a recurrent neural network and the same network unfolded for three time steps. These steps can correspond, for example, to the processing of three consecutive symbols in a DNA sequence. The hidden nodes in a RNN are connected to the themselves via weights.

Recurrent networks are difficult to train due to what is called the vanishing and exploding gradients [128]. To deal with exploding gradients, the gradient values can be clipped. The vanishing gradients can be solved in three ways: by choosing an appropriate activation function (such as the rectified linear unit show in Fig. 2.6c), by initializing the weights to the identity matrix and the biases to zero or by using gated cells. This last option leads to at least two subtypes of RNNs, the long short-term memory (LSTM) [129] and the gated



(a) Recurrent neural network.



(b) Recurrent neural network unfolded.

Figure 2.7: In (a), a diagram of a simple RNN. The hidden nodes connect to themselves via weights (W_{hh}). In (b), the same network expanded to three time steps. Each network at time T_i is a multilayer perceptron where the hidden nodes of the previous step are connected to the hidden nodes of the next.

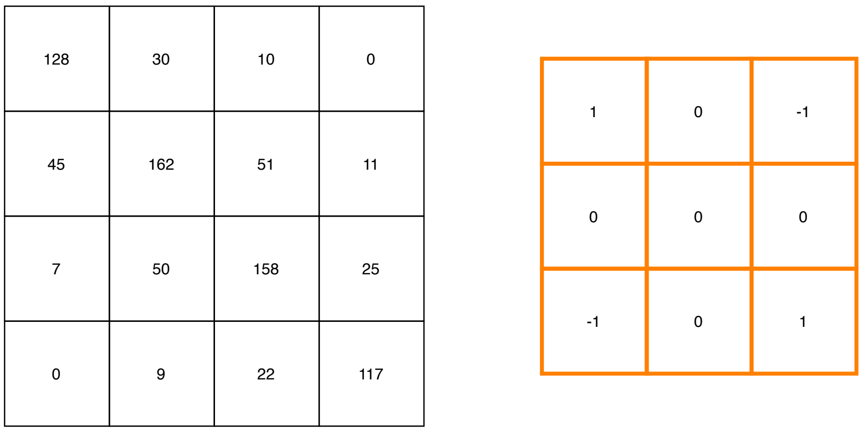
recurrent unit (GRU) [130]. The purpose of these gated cells is to control what information is passed to the next cell.

2.5.3 Convolutional neural networks

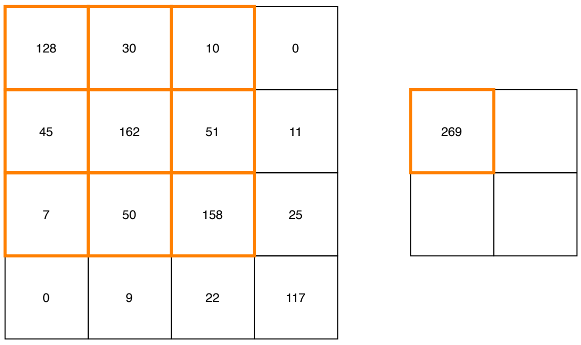
Convolutional neural networks (CNNs) are inspired by the visual cortex organization [131]. They where first introduced by Fukushima under the name neocognitron [132] for the purpose of image recognition. Later, these types of networks were used for speech recognition and trained using backpropagation [133]. The current architecture and training are mainly due to work in 1989 by LeCun [134].

One of the key aspects of these networks is the automatic extraction of features from raw data. This is specially important in images where the number of features and their possible transformations (scale, rotation, occlusion, deformation, light) is high. The defining feature of these networks is the convolutional layer. In Fig. 2.8, we see diagram of a 2D convolutional filter applied to a 2D grid (which can represent an image). This convolutional filter applies pointwise multiplication between the values of the filter and the grid, followed by summing all products. The filter slides across the grid until the entire grid is processed.

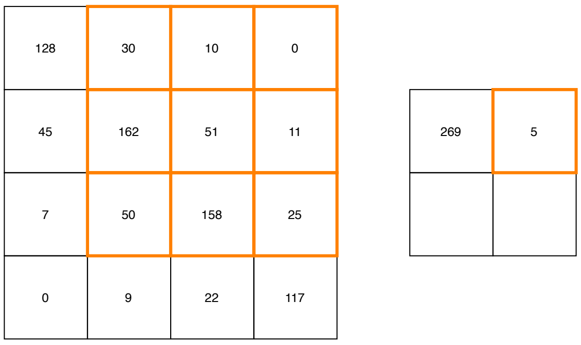
The convolutional layers can be combined with other layers such as the multilayer perceptron or pooling layers. When the stacks of layers become very deep, training is harder and so a strategy to overcome this is to repeat the output of a previous layer to a deeper layer, skipping the intervening ones [135]. This is another strategy inspired by the brain's organization and the resulting architecture is called residual neural network.



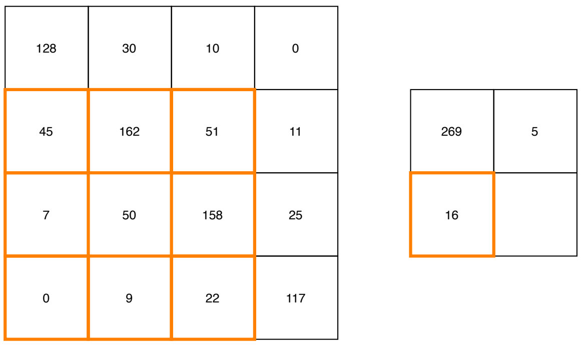
(a) Grid and convolution matrix.



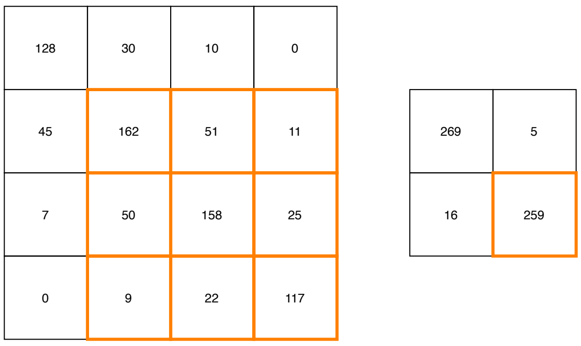
(b) Grid and convolution first step.



(c) Grid and convolution second step.



(d) Grid and convolution third step.



(e) Grid and convolution fourth step.

Figure 2.8: An example of the several steps necessary to calculate the aplication of the convolution matrix to a 2D grid. The convolution matrix moves with a stride of one in both axis.

2.5.4 Neural network learning

We just described three types of neural networks, but one key ingredient is missing from the description. By itself, the forward stage is not very useful, because the weights are randomly initialized. For these to encode useful information regarding a problem, the networks needs to be trained, i.e., the weights need to be updated. There are some alternatives to train them, such as neuroevolution and equilibrium propagation, but due to its efficiency, by far the most popular is gradient descent with backpropagation.

To train the network, we need to know how to adjust the weights, both the direction (positive or negative) and the magnitude. To do this, we need the correct outputs (expected) and a measure of how far the current outputs (actual) are from the expected. This distance is obtained with a loss function and is called the error of the network. The essential question we are trying to answer is "If this weight is changed, how is the error affected?". These types of questions can be answered with derivatives, in this case, the gradient is used because we are interested to know how the output is affected by the change in many variables (weights). We are also trying to get the output as close to the expected as possible, so we want to optimize the weights such that they fit the expected outputs. That is, we are using the gradients to go towards the minimum error. This optimization is called gradient descent. Backpropagation is used to calculate the gradients for all layers. It uses the chain rule to propagate the errors in one layer to the next in order to know how to update the weights.

An important parameter in most gradient descent algorithms is the learning rate. This parameters controls the magnitude of the weight adjustment. A small step might get the process stuck in a local minimum. A large step might lead to a divergent process, thus never finding the minimum. There are algorithms that automatically adjust the size of the step, but their benefits are largely dependent on the problem [136, 137].

2.5.5 Neural network learning example

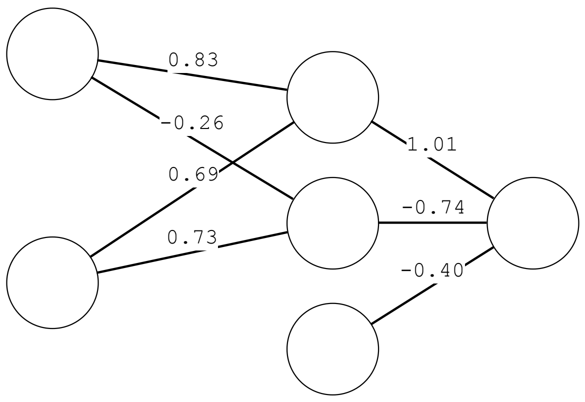
To better understand the training procedure, we now walk-through an example of one complete step of the multilayer perceptron. We will use a network with the topology and initial weights as shown in Fig. 2.9a. The activation and loss function are the logistic and squared error, respectively. The logistic is defined as

$$f(x) = \frac{1}{1+e^{-x}}, \tag{2.4}$$

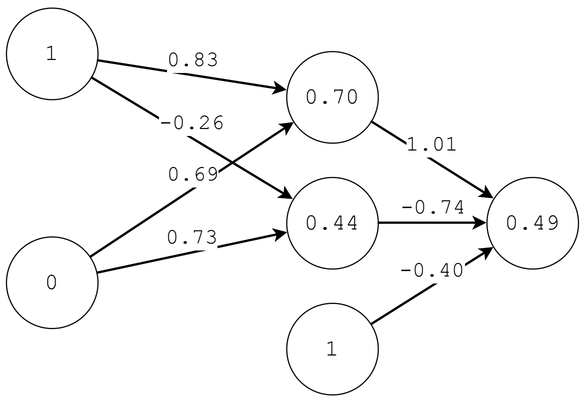
and the total error is given by

$$E = \sum_{i=1}^{n} \frac{1}{2}(\hat{y}_i - y_i)^2, \tag{2.5}$$

where y_i is the value of the output node i and $\hat{y}_i$ is the expected output of that node, also called the target.



(a) Multilayer perceptron topology.



(b) Multilayer perceptron output.

Figure 2.9: Example multilayer perceptron network topology and result of forward pass. The activation for all neurons is done using the logistic function.

In Fig. 2.9b, we show the result of the forward pass with the input vector $[1, 0]$. The actual output is 0.49. Assuming we wanted to train this network to calculate the XOR function, the expected output is one. This means the total error is $\frac{1}{2}(1 - 0.49)^2 = 0.13$. Now we need to know how to adjust each weight connected to the output node so as to minimize the error. For the weights between the output and hidden layer, we calculate the partial derivative of the error in relation to a weight, in other words:

$$\frac{\partial E}{\partial w_i}, \tag{2.6}$$

and by applying the chain rule we get

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i}, \tag{2.7}$$

where z is the output node value before the activation is applied. An example of z value can be seen in the left side of the grey node in Fig. 2.5b.

Expanding each of the terms of Eq. (2.7), for the first part we obtain

$$\frac{\partial E}{\partial y} = \hat{y} - y. \tag{2.8}$$

For the second part, we use the derivative of the logistic function and the result is

$$\frac{\partial y}{\partial z} = f(z)(1 - f(z)) = y(1 - y), \tag{2.9}$$

where f is the logistic function. The final part of the derivative is

$$\frac{\partial z}{\partial w_i} = \frac{\partial(w_i h_j + w_{i+1} h_{j+1} + w_{i+2} h_{j+2})}{\partial w_i} = h_j. \tag{2.10}$$

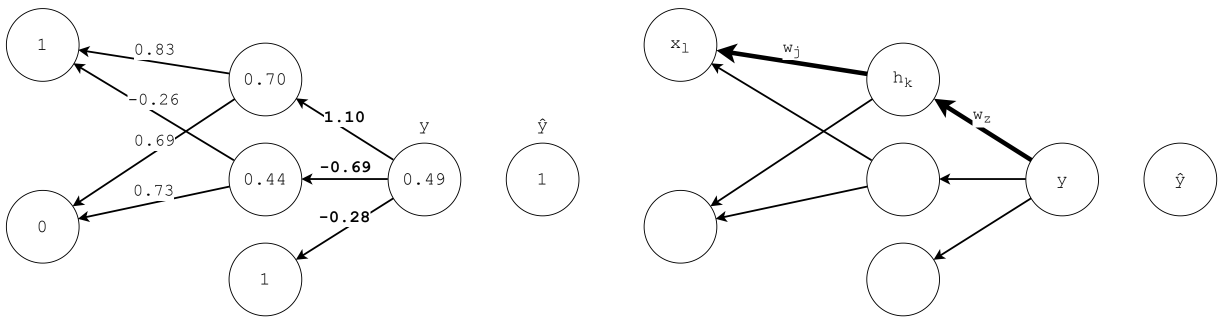
Putting the three parts together we get the equation for the weight update between the output and the hidden layer,

$$\frac{\partial E}{\partial w_i} = (\hat{y} - y)y(1-y)h_j, \tag{2.11}$$

where h_j is the value of the hidden node connected to the output node by the weight w_i. To update the w_i we sum the result of Eq. (2.11) multiplied by the learning rate to the previous value of w_i.

$$w_i \coloneqq w_i + \alpha(\hat{y} - y)y(1-y)h_j, \tag{2.12}$$

where α is the learning rate. We multiply each term of the total error by $\frac{1}{2}$ to simplify the derivative and this is valid because the result is multiplied by the learning rate, which is a input parameter. For our example we assume $\alpha = 1$ and update the weights as show in Fig. 2.10a.



(a) Weights updated (hidden, output).

(b) Node and weight labels.

Figure 2.10: In (a), the updated weights between the hidden and the output layer are shown. In (b), an example of the mapping of variable names in Eq. (2.17) to the network being trained.

The weight updates for the next layer can be calculated in a similar fashion,

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial h_k}\frac{\partial h_k}{\partial z_k}\frac{\partial z_k}{\partial w_j}, \tag{2.13}$$

where w_j is a weight connecting the input layer to the hidden node k and z_k is the value of hidden node k before activation. This function can be expanded further by applying the chain rule,

$$\frac{\partial E}{\partial h_k} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial h_k}, \tag{2.14}$$

and using the chain rule again together with the results from Eqs. (2.8) and (2.9) we have

$$\frac{\partial E}{\partial z} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z} = (\hat{y} - y)y(1-y). \tag{2.15}$$

Using this results we can rewrite Eq. (2.14) into,

$$\frac{\partial E}{\partial h_k} = (\hat{y} - y)y(1-y)\frac{\partial z}{\partial h_k} = (\hat{y} - y)y(1-y)w_j. \tag{2.16}$$

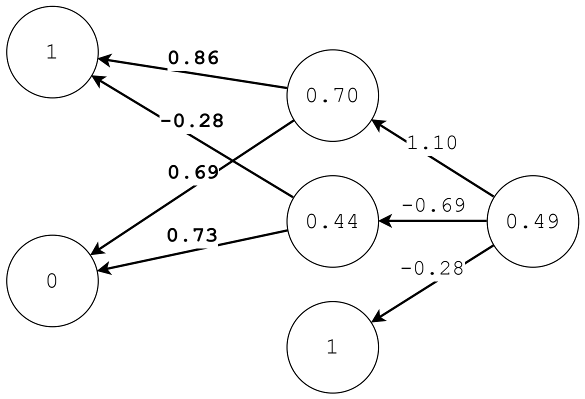
By using this result and following similar calculations as done in Eqs. (2.9) and (2.10) we get

$$\frac{\partial E}{\partial w_j} = (\hat{y} - y)y(1-y)w_z(1-h_k)h_k x_l, \tag{2.17}$$

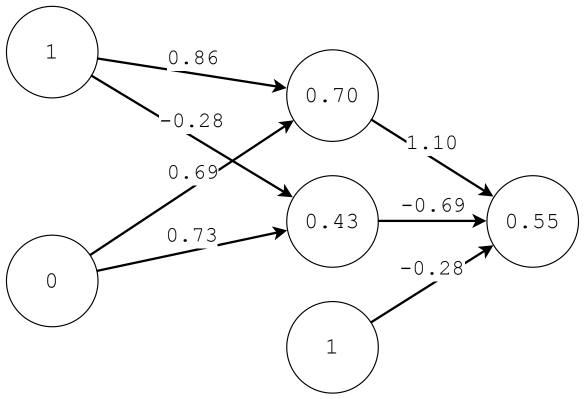
where x_l is the value of the input node connected to the hidden node h_k by weight w_j and w_z is the weight connecting h_k to the output. It can help to visualize the variables in the network graph with a diagram as seen in Fig. 2.10b. Using this result we can update the weights between the input and hidden layer using,

$$w_j := w_j + \alpha(\hat{y} - y)y(1-y)w_z(1-h_k)h_k x_l. \tag{2.18}$$

Applying this function to the remaining weights we get the result seen in Fig. 2.11a. Finally, we verify what effect this training step had on the error, by applying the forward step as seen in Fig. 2.11b and then we calculate the value of the loss function, $\frac{1}{2}(1-0.55)^2 = 0.10$. In this instance we see that the error has decreased for these inputs.



(a) Weights updated (input, hidden).



(b) Network output after training.

Figure 2.11: In (a), the updated weights between the input and the hidden layer are shown. In (b), the network output after updating all the weights.

2.5.6 What network to use?

This is a difficult question to answer because of the many factors involved. The most important factor, is that the mixer should provide better compression than the current approach. On the other hand, if the mixer provides better compression, but the computational resources are too high, then this will severely limit the application to larger sequences. Given the amount of biosequence data currently produced and the expected increases, this is an important concern. The sequences can easily reach gigabytes and as such the mixer should be efficient. We don't have a specific time or memory target, but we will keep this concern in mind, since high execution times or memory consumption will limit the applicability of the mixer. Finally, we want this mixer to be easily integrated into other compressors. This means that it is desirable to reduce both the hardware and software dependencies required by the mixer.

To make an informed decision, we searched for sources that compared multiple networks. We found a comparison specifically for DNA in [10]. This describes a compressor based on neural networks, and it compares two types of RNNs and a MLP. Overall the RNNs provides the best compression, however this depends on the dataset. Another important point is that the described method uses a top of the line GPU and it takes hours to get the results for sequences of 10MB. We also found benchmarks of neural networks to time series prediction problems. These fit the stochastic nature of the issue we are analyzing [138].

In terms of computational resources, intuitively, we deduce that the multilayer perceptron (MLP) will have the best performance as the elements are simpler. This is corroborated by [139], where we see that even with large networks, the MLP is the fastest network by approximately 50%, the convolutional is the second fastest and finally the recurrent networks. The memory consumption is dictated by the number of parameters the network has. More complex and larger networks should require more memory, however this is not shown in the paper.

In terms of accuracy, the performance appears very dependent on the dataset. In [140], a multilayer perceptron (MLP), a convolutional neural network (CNN), a recurrent neural network (RNN) and a long short term memory (LSTM) are used to predict the values of the stock market. The results favor the CNN followed by the MLP, with the LSTM and the RNN trailing behind. In many datasets, the MLP is superior to the CNN.

In [139], a MLP, a CNN and a LSTM are compared using several datasets. Overall the CNN preforms better followed by the LSTM and then the MLP. As in the previous paper, no network is the best in all the datasets. In [141], the authors compare an LSTM to a MLP and conclude that the MLP has equal or better performance than the LSTM.

In [142], a comparison with several neural networks and datasets is made. Among them, several types of CNNs and a MLP. Two types of CNNs, the residual neural network (ResNet) [135] and the fully convolutional neural network (FCN) [143], present the best overall accuracy, with the MLP placing fourth out of the nine evaluated networks.

In [144], an hybrid network, combining a CNN and a LSTM is used to predict power consumption, stock values and gas concentrations. In some datasets the CNN has better predictions than the LSTM. The proposed hybrid approach always presents better predictions in the three datasets.

Almost all methods described use very large networks and because of this, a GPU is used to train and evaluate them. The runtime for small sequences (tens of megabytes) takes hours to complete and achieve results worse than GeCo2 [10, 108]. We would like the mixer to be used for large (gigabytes) sequences, given these results, we tested one of the most accurate CNN, the FCN and the MLP. This allows us to better understand the trade-offs that can be made given our datasets.

Chapter 3

Methods

This chapter provides the procedures and some of the motivations behind our choices. We start by testing some of the most promising neural network architectures for the problem of expert mixing for biosequence compression. We then describe the mixer for GeCo3, including the neural network architecture, the training algorithm, the input transformation and the derived features. Finally, we describe the adaptations applied for the AC2 mixer when compared to the GeCo3.

3.1 Testing different network architectures

To help us determine what network we should implement, we tested two types of neural networks with DNA sequence data. We choose the fully convolutional neural network (FCN) because it has some of the best accuracy result for multivariate time-series. We also assess the multilayer perceptron (MLP) due to the good accuracy and less demanding computation resources.

We employ a stacked generalization approach with online training to help correct for concept drift. To use stack generalization, the neural networks take as inputs the outputs of the models and are trained to output the correct symbol. The training is done with every new symbol. We used GeCo2 as a base for the mixer experiments because compressing DNA is easier than amino acids, so it should help detect compression improvements, which we hope can be translated to the AC compressor.

For these experiments we are not concerned with absolute execution time, but rather with the relative time and the accuracy. Given these relaxed constrains, we choose to use the TensorFlow machine learning framework. This allows us to experiment with several architectures with reduced effort. We modified GeCo2 to output the model predictions (input data) along with the actual symbol (training data) to a CSV file. The data contained in this file is used to train and evaluate the performance of the different networks.

Our test network architectures are based on the work presented in [142]. All the trained networks share the same gradient descent procedure, the Adam [145]. This algorithm has an adaptive learning rate thus freeing us to experiment without the need to tune the learning rate. The networks also share the categorical crossentropy loss function, which is used when an input can only belong to one class out of many. The final layer is a softmax layer with four nodes, one for each of the possible DNA symbols. The softmax function is used to represent a probability distribution over many (in our case four) different outcomes. Unless stated otherwise, the activation function for the convolutional layers is the ReLU and for the others it is the logistic. The MLP inputs are stretched according to the procedure defined in [13] and we subtract the stretched mean probability, which, for the case of DNA, we assume to be 0.25. This has the effect of forcing the average close to zero which, as explained in [146], can help the network learn faster. The FCN inputs, consist of the model outputs for the last ten symbols.

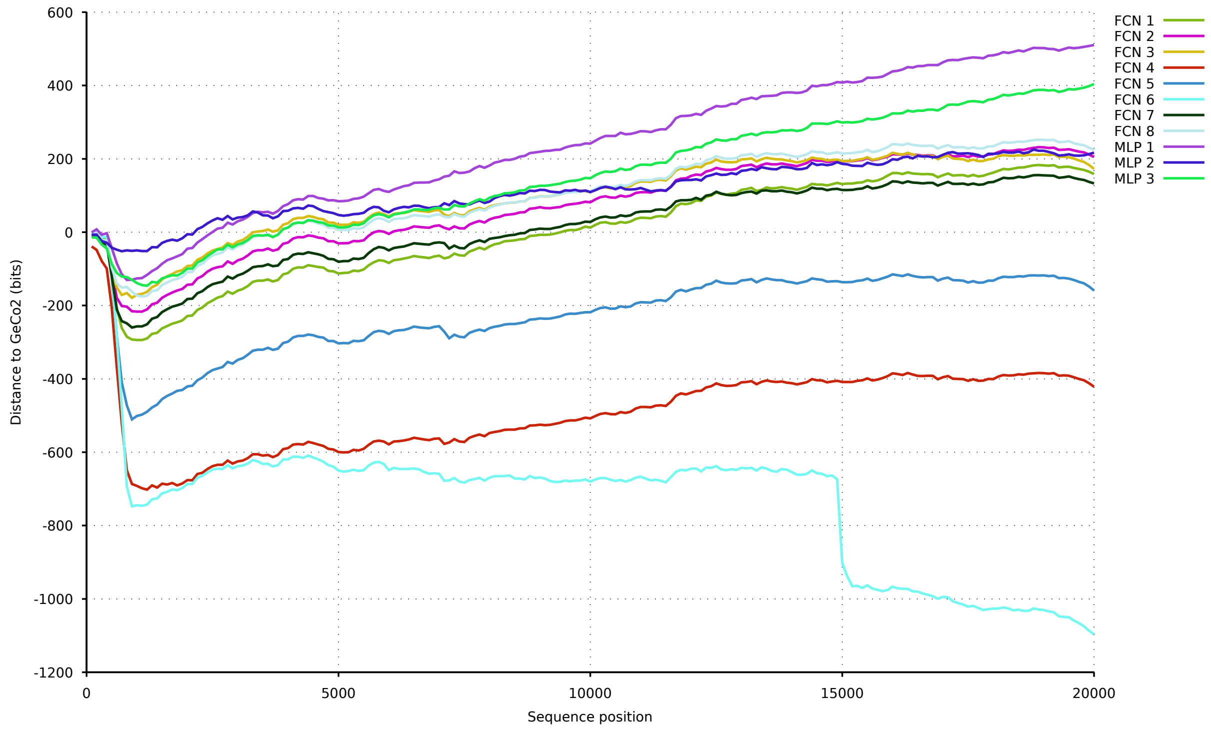


Figure 3.1: Comparison of different neural network architectures. The plot shows the distance to the GeCo2 for the first 20 thousand symbols of the EnIn DNA sequence. Positive distances indicate the network is outputting less bits than GeCo2.

In Fig. 3.1, we show the results of eight FCNs and three MLPs evaluated with the first 20,000 symbols for the DNA sequence of *Entamoeba invadens* (EnIn) proposed in [147]. The differences between these networks are the width of the layers (how many nodes or filters) and the depth of the network (how many layers). The graph plots the distance to the number of bits that GeCo2 would output, estimated by taking the $\log_2$ of the symbol probability. Positive values indicate the network is compressing better than GeCo2. All network plots exhibit an accuracy valley close to the beginning of the sequence. In the beginning all models are ignorant and will output the same values for all symbols

(0.25). Once the models start having opinions the networks will begin to adjust the random weights. We think this is the cause for this initial decrease in accuracy, because GeCo2 doesn't need to learn how to combine the probabilities.

The networks with worse accuracy are the FCN4, FCN5 and FCN6. The FCN5 and FCN6 are the deeper FCN networks with three convolutional layers, in addition, FCN6 has a batch normalization layer after each convolution. The FCN6 is the network evaluated in [142]. The FNC4 uses a single layer but instead of the ReLU activation uses the logistic. The FCN7 uses the past twenty model predictions (instead of the past ten) and the FCN8, which has the best accuracy, uses the last five.

The networks with best accuracy are the MLP1 and MLP3. The MLP1 is the smaller network with only one layer and 32 hidden nodes. The MLP3 is the wider and deeper MLP, with two layer each with 128 nodes.

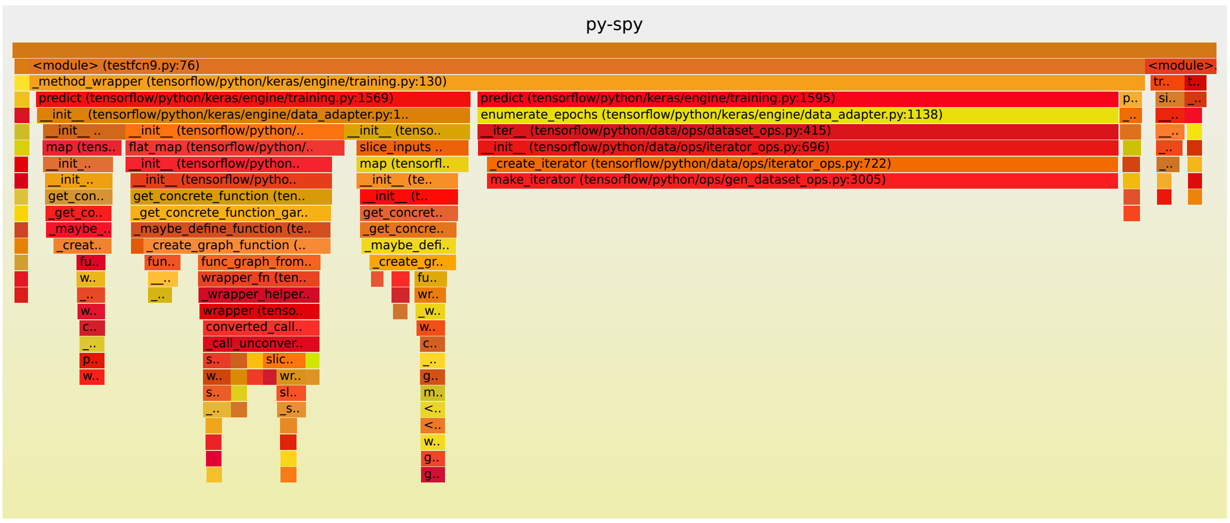


Figure 3.2: Results of profiling a FCN coded with the TensorFlow framework. The profiler used is py-spy, a sampling profiler. The flame graph shows that most of the time is spent on the __init__ procedures of training. For batch training this is not a big problem, but for online training it dominates the execution time.

All networks appear to take approximately one hour to process the data. This likely indicates that when using TensorFlow in this manner, online training instead of batch, there is a large overhead per symbol. We confirm this by profiling the python script, as shown in Fig. 3.2, almost all the time is spent on __init__ procedures.

These preliminary results point us towards an MLP implementation, but before proceeding we would like to gauge how the networks perform in many complete sequences. Given the sequence sizes and the long time the TensorFlow scripts take, we used Genann, an open-source neural network implementation in C available at `https://github.com/codeplea/genann`. We integrate this network directly into the GeCo2. It is an MLP with logistic activation function for all nodes, using stochastic gradient descent.

In Fig. 3.3, we show the distance to GeCo2 of MLP1, the network with best accuracy,

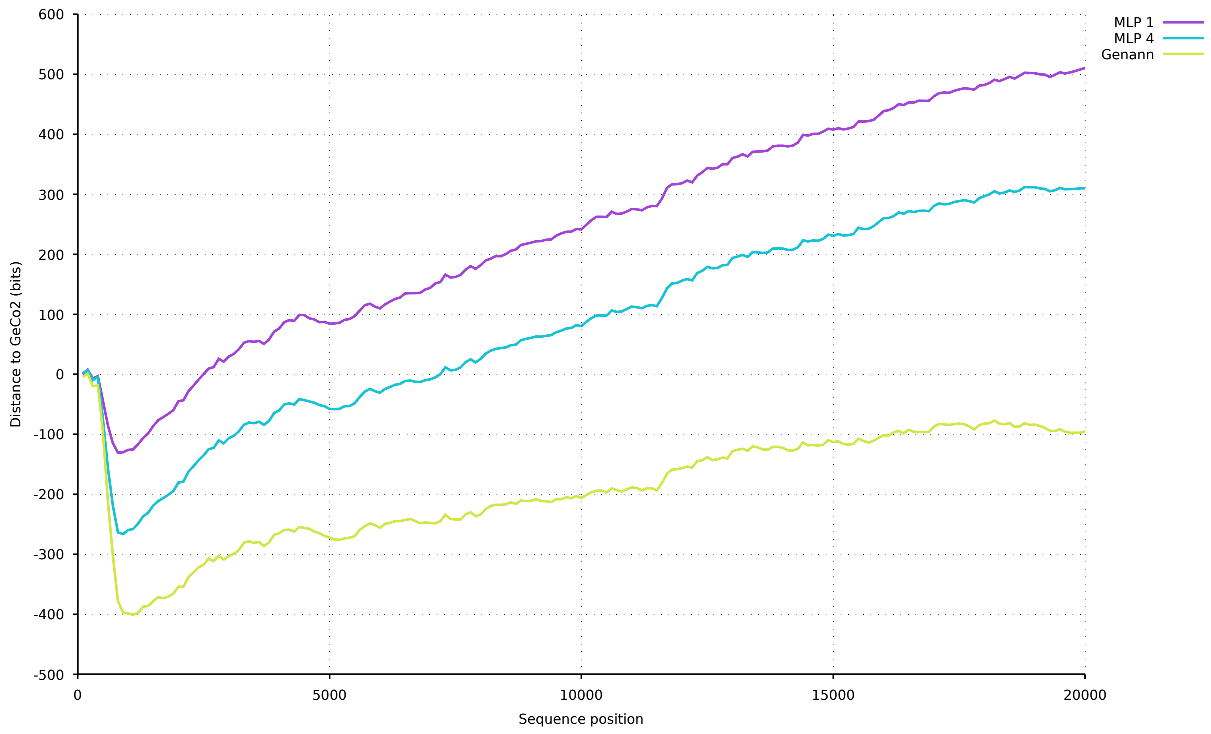


Figure 3.3: The distance of Genann and the two best neural network architectures from Fig. 3.1 to the GeCo2 for the first 20 thousand symbols of the EnIn DNA sequence. Positive distances indicate the network is outputting less bits than GeCo2. The Genann plot has a similar shape to the others, but stays consistently below zero.

---

compared to the implementation with Gennan, and because Gennan doesn't come with the Adam optimizer, the softmax activation or the categorical crossentropy, we include MLP4 which uses a similar configuration, but coded using TensorFlow. While the shape of the plots are similar, the absolute results are quite different, with the Gennan never rising into positive distances.

Genann integrated with the GeCo2 compresses the entire EnIn sequence, of 26,403,087 symbols, in seconds. Unfortunately the result in Fig. 3.4, show that not only the network stops improving, but actually gets much worse when evaluated over the entire sequence. Even if we determined why the absolute results are different between Genann and TensorFlow, the shape of both plots are similar so it would seem the same problems will occur.

Stacked generalization can be used in another way. With a single model it can learn to correct the model predictions [26]. So the next idea we tried, was to use as an additional input the GeCo2 mixing output. This finally gets us to a better result than GeCo2 as seen in Fig. 3.5. This shows the tremendous impact that the right inputs can have on a neural network performance. Curiously, the promise of deep learning doesn't seem to hold in the case of online training, at least with the tested architectures and inputs, but the idea of finding good features and/or transforming the inputs is still crucial. On the other hand, in the case of the MLP, the neural network architecture doesn't seem to have much effect, we tested with two layers of 8 and 16 hidden nodes each and one layer of 64 and 80 hidden nodes, meaning the number of features is approximately the same. The network with two layers of 8 hidden nodes (h8_8) performs worse than the others, but the other networks have similar accuracy, even in the case of the largest network, the h16_16, with a total number of 28,672 weights.

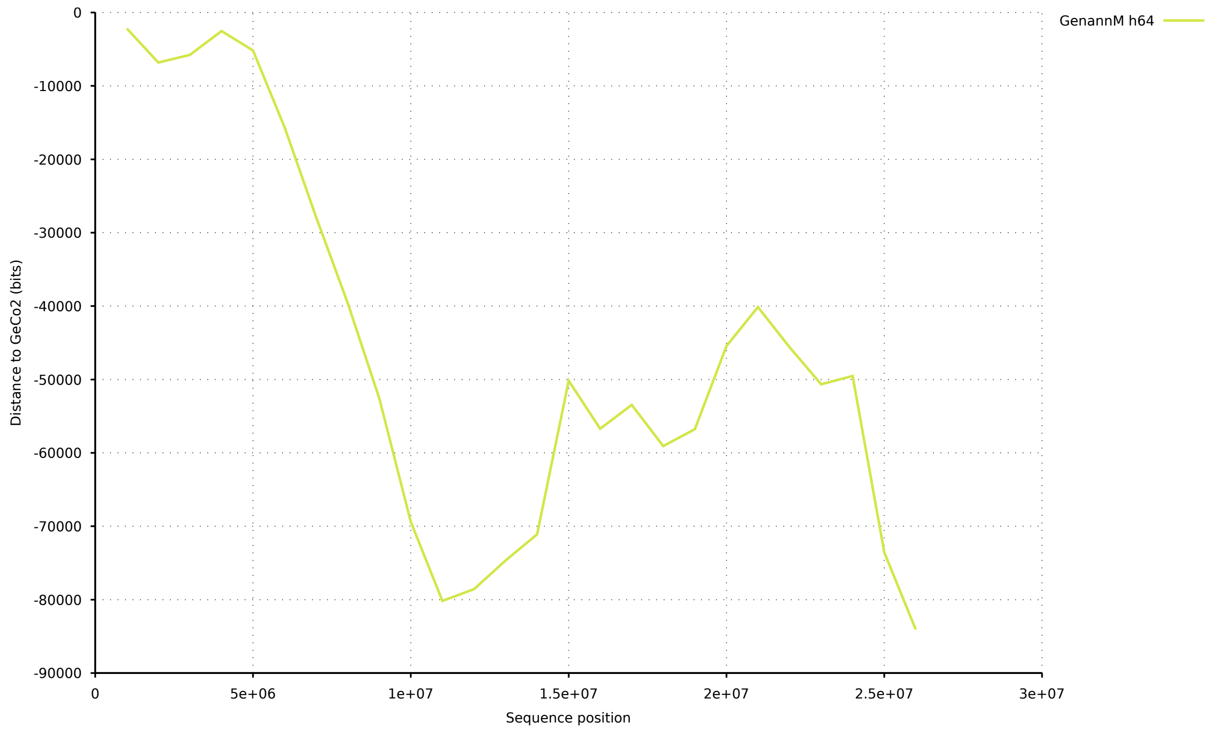


Figure 3.4: The distance of Genann and the two best neural network architectures from Fig. 3.1 to the GeCo2 for the entire EnIn DNA sequence. Positive distances indicate the network is outputting less bits than GeCo2. The Genann has worse compression than GeCo2.

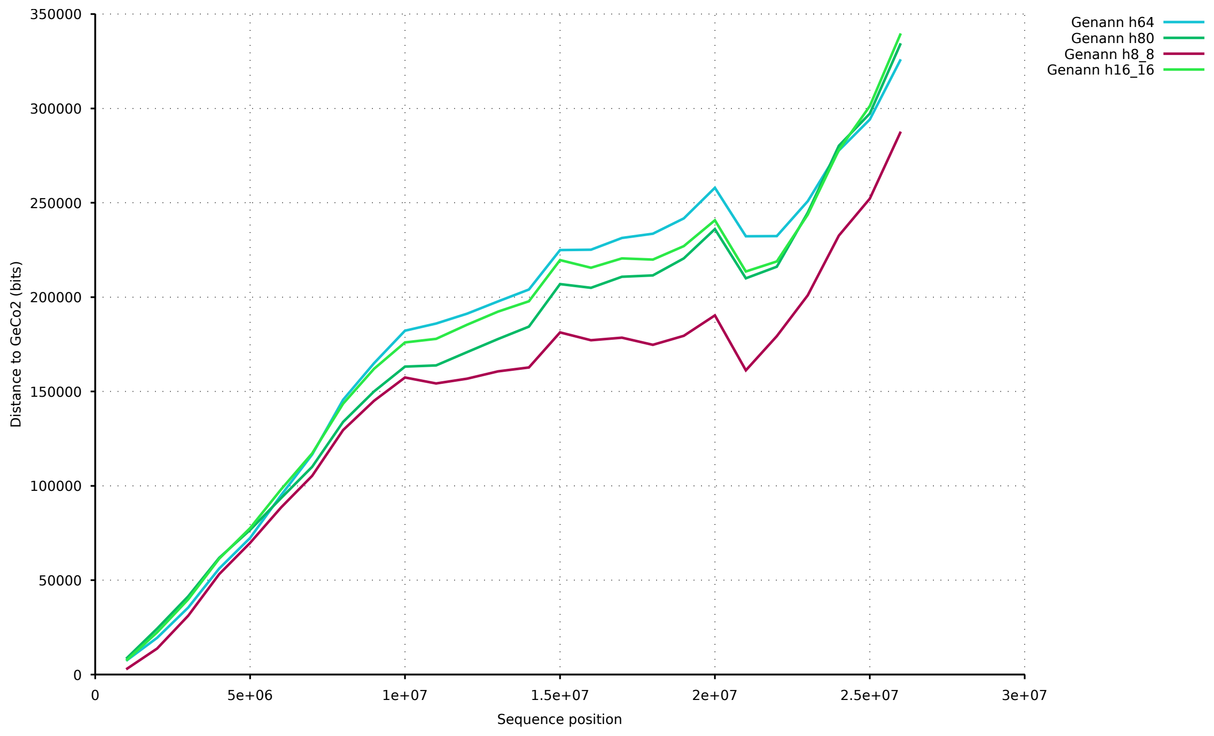


Figure 3.5: The distance of Genann to GeCo2 using the GeCo2 mixing as an input, as well as the other models. The Genann h64 and h80 uses a single hidden layer with 64 and 80 hidden nodes, respectively. The Genann h8_8 and h16_16 uses two layers of eight and sixteen hidden nodes each, respectively.

3.2 GeCo3 mixer

We now present the mixer for GeCo3, which is mostly similar to that of AC2, but due to the large number of symbols (≈ 20 instead of 4), we found that different inputs had better performance. And because AC2 typically compresses smaller sequences, the percentage of time to train the network is larger, so we applied a heuristic to improve the mixing, as well as a pre-training phase.

The GeCo3, like GeCo2, uses a combination of multiple context models and substitution tolerant context models of several order-depths. The neural network provides an efficient combination of these models. Therefore, we describe the new method with the main focus on the neural network, including the inputs, updates, outputs, and training process.

3.2.1 Neural network structure

The model mixing is constructed using a feed-forward artificial neural network trained with stochastic gradient descent [148]. This choice is motivated by implementation simplicity and competitive performance compared to more complex neural networks [140]. The activation function for this network is the sigmoid, and the loss function is the mean squared error. The network structure is fully connected with one hidden layer, as seen in Fig. 3.7. One bias neuron is used for the input and hidden layer, while the weights respect the Xavier initialization according to [149]. Although we empirically tested different activation functions (ReLu, TanH) and a higher number of hidden layers, the most efficient structure was obtained with the previous description.

We introduced two parameters for the GeCo3 compression tool in order to control the number of nodes of the hidden layer and the learning rate. These parameters are written in the compressed file header to ensure a lossless decompression.

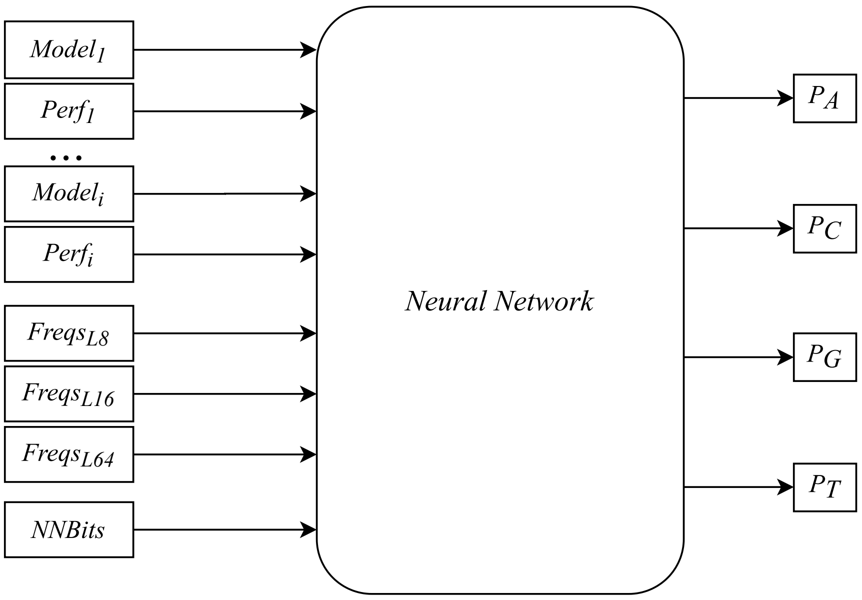


Figure 3.6: High level overview of inputs to the neural network (mixer) used in GeCo3. *Model₁* through *Modelᵢ* represent the GeCo2 model outputs (probabilities for A, C, T, G). *Perf* represents the performance metrics (*hit*, *best*, *bits*) for each model. *Freqs* are the frequencies for the last 8, 16, and 64 symbols. *NNBits* is a moving average of the approximate number of bits that the neural network is producing. The network outputs represent the non-normalized probabilities for each DNA symbol.

3.2.2 Neural network inputs

As inputs to the network, the stretched probabilities of each symbol are used. These are given by

$$p_{i,j} = stretch\left(\frac{1+f_{i,j}}{\sum\limits_{m\in\Theta} 1+f_{i,m}}\right) - stretch\left(mean_p\right), \tag{3.1}$$

where $f_{i,j}$ is the frequency of symbol j for model i with Θ as the set of all symbol and $mean_p$ is the mean probability of each symbol.

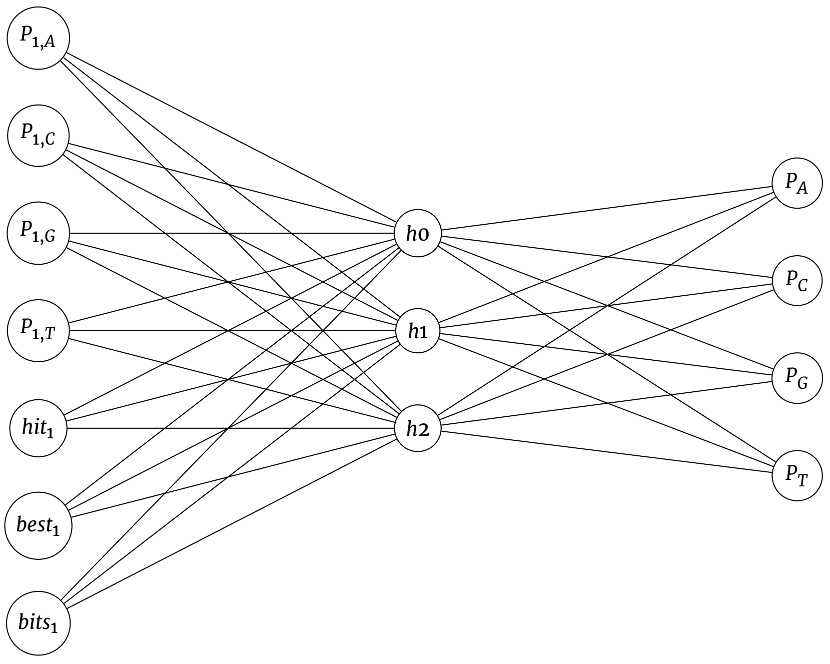


Figure 3.7: A fully connected neural network with one hidden layer. For illustration purposes, this neural network only has the inputs corresponding to one model and the three features that evaluate the model performance. The frequencies of the last 8, 16, and 64 symbols, as well as the $NNBits$ and the bias neurons, are omitted.

We stretch the probabilities according to the work of Mahoney [13]. The effects of stretching can be seen in Fig. 3.8. The inputs are normalized for forcing the average to be close to zero by subtracting the stretched mean probability, which, for the case of DNA, we assume to be 0.25. The normalization and its motivation are explained in [146]. Stretching the probabilities has the effect of scaling them in a non-linear way, which increases the weights of probabilities near zero and one.

The context models, substitution tolerant context models, and the mixed probabilities of GeCo2 are used as input models. This inclusion means that the mixing done in GeCo2 is not discarded, but are used as an additional input to the neural network.

We extract features from the context (the last n symbols) and also calculate model and network performance indicators to improve the network predictions. These are used as inputs to the neural network. Three performance indicators are derived for each mode according to the names *hit*, *best*, and *bits*. These features correspond to three input nodes per model, as seen in Fig. 3.7.

To measure how precise model i is voting, we use

$$hit_{i,n} = \begin{cases} hit_{i,n-1}, & \text{if } \forall x, y \in \Theta : p_{i,x} = p_{i,y} \\ hit_{i,n-1} + 0.1, & \text{if } \forall x \in \Theta : p_{i,sym} > p_{i,x} \\ hit_{i,n-1} - 0.1, & \text{otherwise.} \end{cases} \tag{3.2}$$

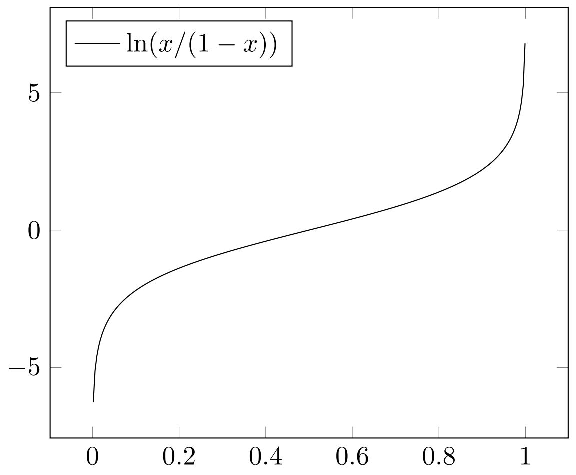


Figure 3.8: Stretching function applied to the models' probabilities.

The symbol with the highest probability is considered the vote of the model. Each time the model votes correctly, *hit* is increased. If the model abstains (probabilities of each symbol are equal), then *hit* remains the same; otherwise, it decreases.

For each model, we also measure if it has assigned the highest probability to the correct symbol, compared to all other models. This is given by

$$best_{i,n} = \begin{cases} best_{i,n-1}, & \text{if } \forall x, y \in \Theta : p_{i,x} = p_{i,y} \\ best_{i,n-1} + 0.1, & \text{if } p_{i,sym} \geq p_{k,sym} \\ best_{i,n-1} - 0.1, & \text{otherwise.} \end{cases} \tag{3.3}$$

The update rules for *best* are similar to *hit* and both have a domain of $[-1, 1]$.

As an approximation to the average number of bits the model would output, we use an exponential moving average

$$bits_{i,n} = \alpha_1 \cdot (-\log_2(p_{i,sym}) + \log_2(mean_p)) + (1 - \alpha_1) \cdot bits_{i,n-1}, \tag{3.4}$$

with $\alpha_1 = 0.15$. This input is also normalized such that the average value is close to zero.

In Eqs. (3.2), (3.3) and (3.4), $p_{i,sym}$ is the probability assigned by model i to the actual symbol in the sequence. To reach these features and their constants, we tested each with a couple of files from one dataset and adjusted until finding a value that produced satisfactory results.

The features extracted from the context are the probabilities of each symbol for the last 8, 16, and 64 symbols. These represent a total of twelve input nodes. In Fig. 3.6, these nodes are represented by $Freqs_{L8}$, $Freqs_{L16}$ and $Freqs_{L64}$. For example, to obtain the probabilities for the last eight symbols with the sequence ACAGTAAA, the number of A's is divided by the number of total symbols, so the frequency of symbol A is 5/8 and for the other symbols is 1/8. These probabilities are then scaled to fit between -1 and 1.

In Fig. 3.6, $NNBits$ represents the exponential moving average of the approximate number of bits and is given by

$$nnbits_n = \alpha_2 \cdot (-\log_2(p_{sym}) + \log_2(mean_p)) + (1-\alpha_2) \cdot nnbits_{n-1}, \tag{3.5}$$

with p_{sym} as the probability the network assigned to the correct symbol and $\alpha_2 = 0.5$.

3.2.3 Updating model performance features

As an example of how to update the features, consider two symbols and three models, and assume all features start equal to zero. Model 1 assigns the probabilities $[0.5, 0.5]$, meaning that the model abstains and, as such, no change is made to hit or $best$. Also, $bits_1$ would be equal to zero. The probabilities for model 2 and 3 are $[0.7, 0.3]$ and $[0.8, 0.2]$, respectively. Assuming the models voted correctly, then hit is now $0 + 0.1 = 0.1$ for both. Because model 3 assigned the highest probability to the correct symbol then $best_3$ is now $0 + 0.1 = 0.1$, and $best_2$ becomes -0.1. Moreover, $bits_2$ would become $bits_2 = 0.15 \cdot (-\log_2(0.7) + log_2(0.5))$ and $bits_3 = 0.15 \cdot (-\log_2(0.8) + log_2(0.5))$.

3.2.4 Neural network outputs and training

As outputs of the network, one node per symbol is used. After the result is transferred to the encoder, the network is trained with the current symbol using the learning rate specified within the program input.

When compared to GeCo2, the results of the new mixing contain two main differences. First, the sum of output nodes is different from one. This outcome is corrected by dividing the node's output by the sum of all nodes. The second difference is that the new approach outputs probabilities in the range $]0, 1[$, while in GeCo2, the mixing always yielded probabilities inside the range of the models.

3.2.5 Neural network implementation

So far we have shown some preliminary results using the Gennan network integrated into GeCo2, but given our understanding of the problem and the way the network will be used, we can do some optimizations. That is, our problem is less general than the problem Genann solves and this opens new paths for greater performance.

The biggest performance increase can be obtained by noticing that we always do a evaluation of the network before we train it. The Genann code however doesn't have this knowledge an thus always evaluate the network before training, which in our case leads to two evaluations per training. The next thing to notice is that we don't see any benefit in having more than one hidden layer, this means we can remove one loop. As a last improvement, we notice that the network is using doubles instead of floats and because the operations performed can be vectorized, converting the network to use floats should increase the performance in CPUs supporting vector operations.

Given the identified optimization opportunities we have implemented a neural network from scratch in C language. The implementation is based on the algorithm presented in [150], and is available at `https://github.com/cobilab/geco3`.

3.3 AC2 mixer

The AC2 mixer is similar to the GeCo3 mixer, the only difference is the derived features and the input transformations. Instead of stretching the model probabilities as in Fig. 3.8, we center the values around zero and multiply them by five, except for the AC mixer output which is multiplied by ten. For the derived features we used the same *Freqs* (8, 16 and 64) as in GeCo3, but we multiply them by five. Finally, we use an exponential moving average for all symbols, such that when a symbol occurs the average for symbol i is updated according to

$$avg_i \coloneqq 0.8 + 0.2 * avg_i, \tag{3.6}$$

if symbol i does not occur then the update rule is

$$avg_i \coloneqq 0.2 * avg_i. \tag{3.7}$$

Additionally, because of the valley seen in Figs. 3.1 and 3.3 and typically smaller amino acid sequences we introduced an heuristic that selects between the AC mixer and the neural network output. The mixer used is the best performing one. This is determined by an exponential moving average of the number of estimated bits produced.

Finally, we pre-train the neural network by activating the same symbol in all models with a value of one and training with that same symbol. Essentially, this is forcing the bias that if all models agree on the same symbol, with absolute certainty, than the output should be that symbol.

Most of theses changes are used to improve the compression of the smaller sequences that AC2 has to process.

Chapter 4

Results and analysis

This chapter presents the main results for the GeCo3 and AC2 compressors. We start by presenting the GeCo3 results, including the datasets and other compressors used for the benchmark and the motivation for that selection. We explore the benefits and costs of our approach and the influence of the new parameters for the compressor. Finally, we show the results for AC2 which are analogous to those of GeCo3.

4.1 GeCo3 results

In this section, we benchmark GeCo3 against state-of-the-art tools in both reference-free and referential compression approaches. In the following subsection, we describe the datasets and materials used for the benchmark, followed by the comparison with GeCo2 using different characteristics, number of models, and data redundancy. Finally, we provide the full benchmark for the nine datasets.

4.1.1 Datasets and materials

The benchmark includes nine datasets. Five datasets are selected for reference-free compression, including

- **DS1**: two compilations of FASTQ data, namely a human virome (Virome) [151] and ancient DNA from a Denisova individual (Denisova) [152];
- **DS2**: four whole genomes: human (HoSaC), chimpanzee (PaTrC), gorilla (GoGoC), and the Norway spruce (PiAbC);
- **DS3**: two compilations of archaeal (Archaea) and viral genomes (Virus);
- **DS4**: highly repetitive DNA with the human Y-chromosome (HoSaY) and a human mitogenome collection (Mito) (proposed in [153]);
- **DS5**: a comprehensive-balanced dataset (proposed in [147]), containing the following sequences:

- HoSa: chromosome 4 of the reference human genome
- GaGa: chromosome 2 of *G. gallus*;
- DaRe: chromosome 3 of *D. rerio*;
- OrSa: chromosome 1 of *O. sativa Japonica*;
- DrMe: chromosome 2 of *D. miranda*;
- EnIn: genome of *E. invadens*;
- ScPo: genome of *S. pomb*;
- PlFa: genome of *P. falciparum*;
- EsCo: genome of *E. coli*;
- HaHi: genome of *H. hispanica*;
- AeCa: genome of *A. camini*;
- HePy: genome of *H. pylori*;
- YeMi: genome of *Yellowstone lake* mimivirus;
- AgPh: genome of *Aggregatibacter* phage S1249;
- BuEb: genome of *Bundibugyo ebolavirus.*

On the other hand, to benchmark the reference-based approach, we use the complete genomes of four primates (human, gorilla, chimpanzee, and orangutan) with a pairwise chromosomal compression. Non-human chromosomes are concatenated to match the human chromosomal fusion [154]. For each chromosomal pair, the following compression was performed

- **DSR1**: chimpanzee (PT) using human (HS) as a reference;
- **DSR2**: orangutan (PA) using human (HS) as a reference;
- **DSR3**: gorilla (GG) using human (HS) as a reference;
- **DSR4**: human (HS) using gorilla (GG) as a reference.
- **EDSR5**: Korean human sequence (HSK1) and for the same human, a re-sequence (HSK2).

4.1.2 Compression sizes

In order to assess the performance of the neural network mixing, we compare GeCo2 with GeCo3. To ensure a fair comparison, the compression modes, including the models and parameters, are kept identical for both programs.

In Table 4.1, GeCo2 and GeCo3 are compared using the compression modes published in [23]. The overall compression improves by 1.93%, and the mean improvement is 1.06%.

Table 4.1: Number of bytes needed to represent each DNA sequence for GeCo2 and GeCo3 compressors. The column mode applies to both compression methods, while the learning rate and the number of hidden nodes only apply to the latter.

ID	GeCo2 bytes	GeCo3 bytes	GeCo2 secs	GeCo3 secs	Mode	L.Rate	H.Nodes
HoSa	38,845,642	**37,891,143**	223	598	12	0.03	64
GaGa	33,877,671	**33,411,628**	160	424	11	0.03	64
DaRe	11,488,819	**11,189,716**	64	189	10	0.03	64
OrSa	8,646,543	**8,434,878**	44	133	10	0.03	64
DrMe	7,481,093	**7,379,992**	33	99	10	0.03	64
EnIn	5,170,889	**5,066,670**	26	75	9	0.05	64
ScPo	2,518,963	**2,511,054**	11	24	8	0.03	40
PlFa	1,925,726	**1,906,919**	10	22	7	0.03	40
EsCo	1,098,552	**1,094,298**	2	8	6	0.03	40
HaHi	902,831	**896,037**	2	6	5	0.04	40
AeCa	380,115	**377,343**	1	2	5	0.04	16
HePy	375,481	**373,583**	1	3	4	0.04	40
YeMi	16,798	**16,793**	0	0	3	0.09	24
AgPh	**10,708**	10,715	0	0	2	0.06	16
BuEb	**4,686**	**4,686**	0	0	1	0.06	8
Total	112,744,517	**110,565,455**	577	1,583			

The larger sequences (larger than ScPo) have mean improvements of 2.04%, while the remaining have modest improvements of 0.4%. Only the two smallest sequences show negative improvement, given the absence of enough time to train the network. Additionally, the eight bytes that are used to transmit the two network parameters to the decompressor are a significant percentage of the total size, unlike in larger sequences. Overall, GeCo3 improves the compression of the whole dataset by more than 1.9%.

4.1.3 Computational resources

In Table 4.2, we show a comparison of our custom neural network and the Genann network. In terms of bytes the results are mostly the same with the differences stemming from the weight initialization and the approximations done on the logistic function as well as the use of floats instead of doubles for the floating point numbers. The Genann was modified to not run the forward stage twice per training. Even with this optimization, the results show that GeCo3 is approximately two times faster when using our network.

Regarding computational resources, the mixing modification is 2.7× slower, as shown in Table 4.1. The computation was performed on an Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz running Linux 5.4.0 with the scaling governor set to performance and 32GB of RAM. The new mixing approach is always slower, because GeCo2's mixing is still used, not as a result of the encoder, but rather as an input to the network. Even if this was

Table 4.2: Number of bytes and seconds needed to represent each DNA sequence for GeCo3 compressor using our custom tailored network and the Genann network. All parameters are the same as in Table 4.1.

ID	Genann bytes	Custom bytes	Genann secs	Custom secs
HoSa	37,893,427	**37,891,143**	1,222	598
GaGa	33,415,307	**33,411,628**	804	424
DaRe	11,192,291	**11,189,716**	388	189
OrSa	8,436,102	**8,434,878**	267	133
DrMe	7,380,539	**7,379,992**	198	99
EnIn	5,068,775	**5,066,670**	140	75
ScPo	2,511,104	**2,511,054**	40	24
PlFa	1,907,538	**1,906,919**	37	22
EsCo	1,094,379	**1,094,298**	12	8
HaHi	896,056	**896,037**	10	6
AeCa	377,407	**377,343**	3	2
HePy	373,613	**373,583**	5	3
YeMi	16,795	**16,793**	0	0
AgPh	10,726	**10,715**	0	0
BuEb	4,693	**4,686**	0	0
Total	110,578,752	**110,565,455**	3,126	1,583

not the case, it is unlikely that GeCo3 would be faster because the mixer has to do more calculations.

The difference in RAM usage of both approaches is less than 1 MB, which corresponds to the size of the neural network and the derived features for each model.

The number of hidden nodes is chosen to fit in the vector registers, in order to take full advantage of the vectorized instructions. Accordingly, we set the number of hidden nodes as a multiple of eight, where floating points of four bytes represent the nodes and 32 bytes represent the vector registers.

4.1.4 Effects of the hidden layer size on mixing

Increasing or decreasing the number of hidden nodes affects the number of weights, and it also affects compression, as can be seen in Fig. 4.1. Increasing the number of nodes increases the compression up to a point. This point varies from sequence to sequence; however, the abruptest gains in compression generally occur until 24 hidden nodes. As expected, increasing the number of hidden nodes leads to an increase in execution time and a progressive decline of compression gain. These results are also consistent in referential compression as seen in Fig. 4.2.

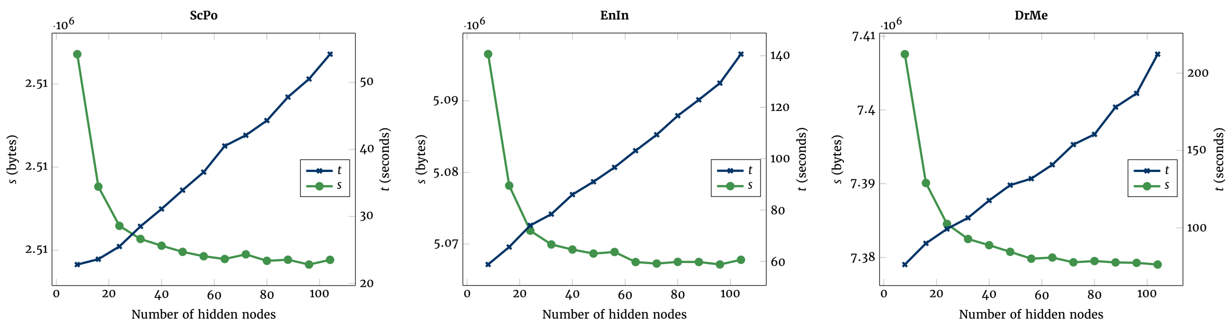


Figure 4.1: Number of bytes (s) and time (t) according to the number of hidden nodes for reference-free compression of ScPo, EnIn, and DrMe sequence genomes.

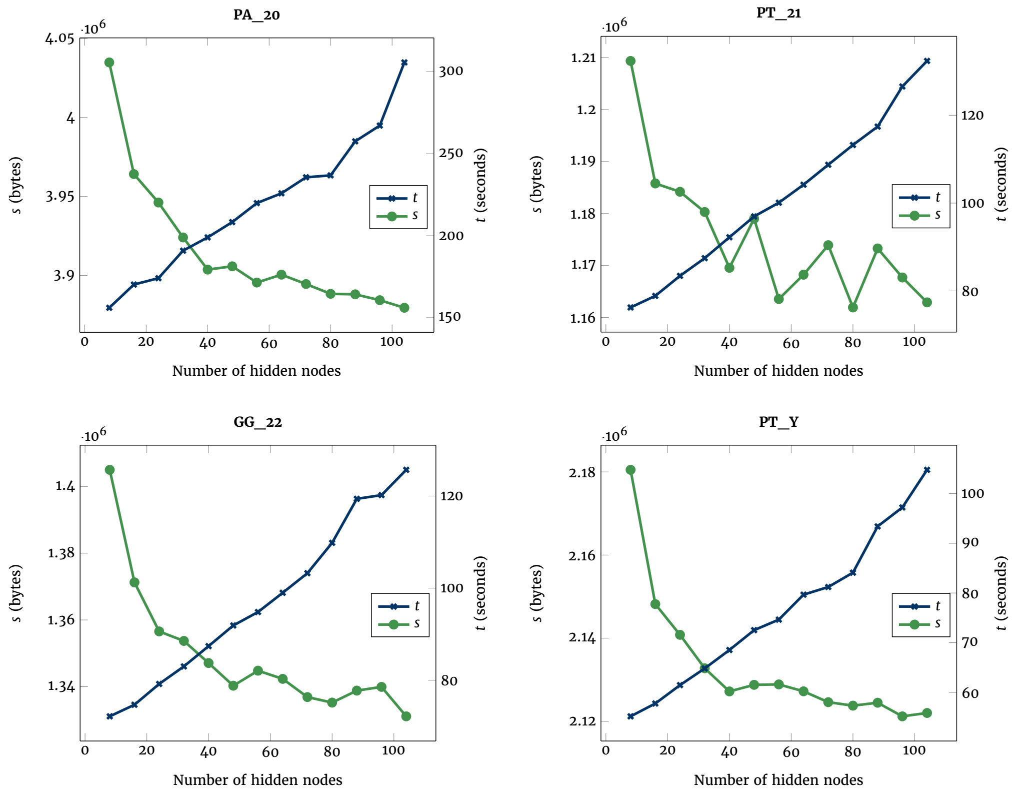


Figure 4.2: Number of bytes (s) and time (t) according to the number of hidden nodes for reference compression of four primate chromosomes. PA_20 - Chromosome 20 from *Pongo abelii*. PT_21(Y) - Chromosome 21(Y) from *Pan troglodytes*. GG_22 - Chromosome 22 from *Gorilla gorilla*. All use the corresponding human chromosomes as a reference.

4.1.5 The importance of derived features on mixing

We removed the derived features from the inputs to the network to assess its impact on the mixing performance. The results are present in Table 4.3.

When using just the models' probabilities as inputs, the compression is more efficient than GeCo2 by a small margin (0.18%), while, in the majority of the sequences, there is no improvement. By adding the result of the GeCo2 mixing as an input, the improvement increases to 1.36%. The gain escalates, having an improvement of 1.73%, when using the context models and substitution tolerant context models as inputs and the derived features.

Table 4.3: Number of bytes needed to represent each DNA sequence using the GeCo3 compressor with specific conditions. For the column named Models, only the context models and tolerant context models of GeCo2 were used as network inputs. For "Models + GeCo2", the result of GeCo2 mixing was also used as input. With "Models + Derived" the inputs for the network were the same as "Models" with the derived features added. The compression modes are the same as in Table 4.1.

ID	Models	Models + GeCo2	Models + Derived
HoSa	38,556,039	38,153,358	**37,943,933**
GaGa	33,758,606	33,548,929	**33,444,816**
DaRe	11,615,937	11,280,688	**11,251,390**
OrSa	8,694,790	8,517,947	**8,471,715**
DrMe	7,475,341	7,414,919	**7,392,290**
EnIn	5,183,237	5,095,391	**5,087,359**
ScPo	2,524,818	2,514,188	**2,513,085**
PlFa	1,928,282	1,912,745	**1,912,176**
EsCo	1,104,646	1,095,589	**1,096,255**
HaHi	903,019	898,280	**898,145**
AeCa	378,226	377,857	**377,696**
HePy	379,285	**374,364**	374,975
YeMi	16,901	**16,827**	16,882
AgPh	10,744	**10,727**	10,731
BuEb	**4,694**	4,696	4,698
Total	112,534,565	111,216,505	**110,796,146**

4.1.6 Scaling the number of models

GeCo2 and GeCo3 contain several modes (compression levels), which are parameterized combinations of models with diverse neural network characteristics. To see how the compression of the new approach scales with more models, we introduced mode 16 with a total of 21 models. This new mode was used to compress the sequences of HoSa to HePy (by size order). For the remaining sequences, the same models were used as in Table 4.1. We used this approach because increasing the number of models was incapable of improving the compression of GeCo3 and GeCo2, given the smaller dimensions of these sequences. The number of hidden nodes was also adjusted until no tangible improvements in compression were observed.

The results in Table 4.5 show that the distance between the approaches increases from 1.93% to 2.43%. The time difference reduces from 2.7× to 2.0×. This reduction is due to the increased percentage of time spent by the higher-order context models. These results show that neural network mixing can scale with the number of models. The forgetting factors for this new mode were not tuned, due to the use of a large number of models. Therefore, with this tuning, additional gains can be observed. Nevertheless, this shows another advantage of this new mixing, which is that there are only two parameters that need tuning regardless of the number of models. As the sequence size and the number of models increases, there is almost no tuning required, with the optimal values being around 0.03 for the learning rate and 64 hidden nodes.

4.1.7 Compressing highly repetitive and large sequences

In this subsection, we show how the reference-free compression scales with the new mixing using highly repetitive and extensive sequences, namely in the gigabyte scale. Four datasets are selected, and the results are shown in Table 4.5.

According to the results from Table 4.5, GeCo3 compresses the highly repetitive sequences (DS3 and DS4) with an mean of 6.6% compared to GeCo2 using more 1.9× time. For the larger sequences of DS1 and DS2, GeCo3 has an mean compression improvement of 3.2% in the primates, 8.2% in the spruce (PiAbC), 11.8% for the Virome and 5.2% for Denisova, with a 2.6× mean slower execution time. These results show that the compression of longer repetitive sequences present higher compression gains.

4.1.8 Reference-free sequence compression benchmark

In this subsection, we compare GeCo3 with other specialized reference-free compressors, namely XM (v3.0) [155], GeCo2 (previously compared), Jarvis [86], and NAF [11]. As presented in Table 4.5, GeCo3 achieves the best total size in three out of five datasets. In DS3 and DS4, GeCo3 was unable to achieve the best compression, delivered by Jarvis. These types of datasets justify this performance. Specifically, DS3 and DS4 contain a high number of identical sequences. These are collection of mitogenomes, archeal and virus where the variability is very low, which gives an advantage to models of extremely repetitive nature. Such models, also known as weighted stochastic repeat models, are present in Jarvis, unlike in GeCo3. The reason why we excluded the inclusion of these models in GeCo is that they fail in scalability because the RAM increases according to the sequence length. For the larger datasets, DS1 and DS2, Jarvis was unable to compress the sequences even with 32GB of RAM. On the other hand, GeCo3 has constant RAM, which is not affected by the sequence length but rather only by the mode used.

Comparing GeCo3 against the second best compressor for each dataset, the compression gain is 6% (vs GeCo2), 5.8% (vs GeCo2), −0.8% (vs Jarvis), −3.2% (vs Jarvis) and 1.9% (vs Jarvis), for DS1, DS2, DS3, DS4 and DS5, respectively. For the individual sequences in the datasets, GeCo3 compresses more than the other compressors, except for the AgPh, BuEb, Mito, Virus and Archaea. Tiny sequences compose the AgPh and BuEb dataset, and the neural network does not have enough time to learn, while Mito, Virus and Archaea have already been mentioned above.

Regarding computational time, GeCo3 is faster than XM per dataset, spending on average only 0.6× the time. Against GeCo2, it is slower 2.1× on average, and compared to Jarvis, it is 1.1× slower. NAF is the fastest compressor in the benchmark. Compared to NAF, GeCo3 is between 12× slower for DS5 and 3× for DS1.

Regarding computational memory, the maximum amount of RAM used for GeCo2 and GeCo3 was 12.6GB, Jarvis peaked at 32GB, XM at 8GB, and NAF used at most 0.7GB. Jarvis could not complete the compression for DS1 and DS2 due to a lack of memory. This issue is a limitation that was mentioned earlier. We also note that the XM is unable to decompress some of the sequences. In these cases, the decompressed file has the correct size, but the sequence does not fully match the original file. NAF, GeCo2, and GeCo3 were the only compressors that have been able to compress all the sequences losslessly, independently from the size. The overall results of these compressors show that GeCo3 provides a total compression improvement of 25% and 6% over NAF and GeCo2, respectively.

Compared with general-purpose compressors that achieve the best compression ratios, such as CMIX and DeepZip, GeCo3 is approximately 100 times faster. GeCo3 also has better total compression ratio compared to CMIX (7.7%). We could not obtain enough results with DeepZip to make a meaningful comparison. The results can be seen in Table 4.4.

4.1.9 Reference-based sequence compression benchmark

In this subsection, we benchmark GeCo3 with state-of-the-art referential compressors. The comparison is done between the genomes of different species and not for re-sequenced genomes. Re-sequencing is applied to the same species and, in a general case, limits the domain of applications; for example, phylogenomic, phylogenetic, or evolutionary analysis.

To run the experiments, we used four complete genomes of closely related species: *Homo sapiens* (HS), *Pan troglodytes* (PT), *Gorilla gorilla* (GG) and *Pongo abelii* (PA). The compression for PT, GG, and PA was done using HS as the reference. HS was compressed using GG as a reference. Each chromosome was paired with the corresponding one of the other species. Due to the unavailability of chromosome Y for GG and PA, comparisons that involved these chromosomes were not made. The compressors used in this benchmark are GeCo3, GeCo2, iDoComp [94], GDC2 [93], and HRCM [97]. The FASTA files were filtered such that the resulting file only contained the symbols $\{A, C, G, T\}$, and a tiny header line. HRCM needs the line size to be limited; therefore, line breaks were added for the files under its compression. However, this approach prevents a direct comparison of total compressed size and time, which we solved using the compression ratio percentage ($output\_size \div input\_size \times 100$) and the speed in kilobytes per seconds ($input\_size \div 1000 \div seconds\_spent$). For GeCo2 and GeCo3, two approaches of referential compression are considered. One approach is based on conditional compression, where a hybrid of both reference and target models are used. The other approach, called relative approach, uses

Table 4.4: Number of bytes and time needed to represent a DNA sequence for CMIX, DeepZip and ZPAQ. CMIX and DeepZip were run with the default configuration and ZPAQ was run with level 5. Some tests were not run (NR) due to time constraints and DeepZip forced the computer to reboot (SF) with some sequences.

		CMIX		DeepZip		ZPAQ	
DS	ID	size	time	size	time	size	time
2	PiAbC	NR	NR	NR	NR	2.75 GB	2h01m
	HoSaC	NR	NR	NR	NR	629.76 MB	29m
	PaTrC	NR	NR	NR	NR	614.37 MB	29m
	GoGoC	NR	NR	NR	NR	597.90 MB	28m
	Total	NR	NR	NR	NR	NR	NR
3	Archaea	NR	NR	NR	NR	138.05 MB	8m09s
	Virus	NR	NR	NR	NR	106.40 MB	8m12s
	Total	NR	NR	NR	NR	244.45 MB	16m21s
4	Mito	NR	NR	NR	NR	44.85 MB	2m59s
	HoSaY	4.80 MB	5h17m	SF	SF	5.28 MB	1m07s
	Total	NR	NR	NR	NR	50.13 MB	4m07s
5	HoSa	NR	NR	NR	NR	41.49 MB	2m56s
	GaGa	NR	NR	NR	NR	34.69 MB	2m54s
	DaRe	12.19 MB	12h20m	NR	NR	13.18 MB	2m34s
	OrSa	9.04 MB	8h55m	SF	SF	9.64 MB	1m48s
	DrMe	7.46 MB	6h41m	SF	SF	7.61 MB	1m20s
	EnIn	5.58 MB	5h21m	SF	SF	6.13 MB	1m05s
	ScPo	2.54 MB	2h08m	SF	SF	2.58 MB	25s
	PlFa	1.93 MB	1h48m	SF	SF	1.99 MB	21s
	EsCo	1.09 MB	56m37s	SF	SF	1.11 MB	11s
	HaHi	882.71 KB	46m57s	883.07 KB	1h19m	902.85 KB	9s
	AeCa	370.61 KB	19m01s	371.88 KB	32m41s	380.13 KB	3s
	HePy	376.61 KB	20m03s	377.53 KB	34m49s	384.42 KB	3s
	YeMi	16.68 KB	58s	19.53 KB	1m33s	17.83 KB	0s
	AgPh	10.70 KB	36s	12.24 KB	59s	11.77 KB	0s
	BuEb	4.68 KB	17s	6.23 KB	31s	5.77 KB	0s
	Total	NR	NR	NR	NR	120.13 MB	13m53s

exclusively models loaded from the reference sequence. Both types of compression assume causality, which means that with the respective reference sequence, the decompressor is able to decompress without loss. The reason why we benchmark these two approaches is that there are many sequence analysis applications for both approaches.

The results are presented in Table 4.7, showing the total compression ratio and speed

Table 4.5: Size and time needed to represent a DNA sequence for NAF, XM, Jarvis, GeCo2, and GeCo3. For DS5, Jarvis uses the same configuration as in [86], for DS4 and DS3 it uses level 7. XM uses the default configuration. NAF uses the highest compression level (22). GeCo2 and GeCo3 use mode 16 for DS5, except for BuEb, AgPh and YeMi which use the configurations of Table 4.1. For DS4 and DS3 the models are *"-tm 3:1:1:1:0.8/0:0:0 -tm 6:1:1:1:0.85/0:0:0 -tm 9:1:1:1:0.85/0:0:0 -tm 12:10:0:1:0.85/0:0:0 -tm 15:200:1:10:0.85/2:1:0.85 -tm 17:200:1:10:0.85/2:1:0.85 -tm 20:500:1:40:0.85/5:20:0.85"*, DS2 uses *"-tm 3:1:1:1:0.70/0:0:0 -tm 8:1:1:1:0.85/0:0:0 -tm 13:10:0:1:0.85/0:0:0 -tm 19:500:1:40:0.85/5:20:0.85"*, and Virome uses *"-tm 7:1:1:1:0.8/0:0:0 -tm 13:10:0:1:0.95/0:0:0 -tm 19:500:1:40:0.95/5:20:0.95"*. Denisova uses the same models as Virome but with inversions turned off. GeCo3 uses a learning rate of 0.03 and 64 hidden nodes for all sequences. The character '*' indicates the sequence was not compressed due to an error and '/' due to out of memory. Results where the decompression produces different results than the input file are appended by the character '?'.

		NAF		XM		Jarvis		GeCo2		GeCo3	
DS	ID	size	time	size	time	size	time	size	time	size	time
1	Denisova	25.36 GB	25h22m	*	*	/	/	20.61 GB	23h18m	**19.55 GB**	71h19m
	Virome	4.72 GB	6h01m	*	*	/	/	3.17 GB	8h45m	**2.79 GB**	24h32m
	Total	30.08 GB	31h23m	*	*	/	/	23.78 GB	32h04m	**22.34 GB**	95h51m
2	PiAbC	2.29 GB	2h45m	*	*	/	/	1.86 GB	4h02m	**1.71 GB**	9h21m
	HoSaC	634.07 MB	38m	*	*	/	/	579.66 MB	53m12s	**560.88 MB**	2h14m
	PaTrC	619.48 MB	37m	*	*	/	/	569.40 MB	51m40s	**551.54 MB**	2h08m
	GoGoC	603.39 MB	36m	*	*	/	/	556.54 MB	49m57s	**539.30 MB**	2h04m
	Total	4.15 GB	4h36m	*	*	/	/	3.57 GB	6h37m	**3.36 GB**	15h49m
3	Archaea	128.09 MB	7m	103.01 MB?	1h41m	**96.66 MB**	57m	103.70 MB	30m	97.87 MB	55m
	Virus	85.51 MB	6m	63.93 MB?	1h35m	**61.19 MB**	1h35m	65.63 MB	29m	61.19 MB	55m
	Total	213.60 MB	14m	166.93 MB?	3h16m	**157.84 MB**	2h32m	169.34 MB	1h00m	159.07 MB	1h51m
4	Mito	35.93 MB	2m32s	28.12 MB?	47m11s	**27.11 MB**	16m1s	30.40 MB	11m26s	28.17 MB	21m31s
	HoSaY	5.17 MB	11s	3.88 MB?	3m25s	3.93 MB	1m45s	4.08 MB	1m15s	**3.85 MB**	2m21s
	Total	41.10 MB	2m43s	32.01 MB?	50m36s	**31.04 MB**	17m46s	34.48 MB	12m41s	32.03 MB	23m52s
5	HoSa	41.73 MB	2m06s	38.66 MB?	29m26s	38.66 MB	4m33s	38.79 MB	11m17s	**37.56 MB**	22m39s
	GaGa	35.57 MB	1m38s	33.83 MB?	22m20s	33.70 MB	2m38s	33.75 MB	8m43s	**33.26 MB**	17m38s
	DaRe	12.83 MB	32s	11.17 MB?	8m59s	11.17 MB	1m32s	11.44 MB	3m40s	**10.97 MB**	7m32s
	OrSa	9.53 MB	21s	8.48 MB?	6m39s	8.45 MB	1m14s	8.60 MB	2m37s	**8.34 MB**	5m17s
	DrMe	7.85 MB	15s	7.53 MB?	5m01s	7.49 MB	22s	7.47 MB	1m57s	**7.36 MB**	3m50s
	EnIn	5.87 MB	12s	5.12 MB?	3m19s	5.09 MB	36s	5.14 MB	1m37s	**5.02 MB**	3m12s
	ScPo	2.59 MB	4s	2.53 MB	55s	2.52 MB	11s	2.52 MB	44s	**2.51 MB**	1m21s
	PlFa	2.02 MB	4s	1.92 MB	59s	1.92 MB	10s	1.93 MB	37s	**1.90 MB**	1m09s
	EsCo	1.15 MB	2s	1.11 MB	13s	1.10 MB	4s	1.10 MB	24s	**1.09 MB**	39s
	HaHi	948.69 KB	2s	914.87 KB	16s	899.47 KB	2s	899.17 KB	21s	**889.51 KB**	34s
	AeCa	396.82 KB	1s	387.00 KB	3s	380.51 KB	1s	381.29 KB	13s	**376.97 KB**	18s
	HePy	404.55 KB	1s	384.30 KB	4s	374.37 KB	1s	375.66 KB	13s	**371.62 KB**	19s
	YeMi	17.35 KB	1s	16.84 KB	0s	16.87 KB	0s	16.80 KB	0s	**16.79 KB**	0s
	AgPh	11.02 KB	1s	10.71 KB	0s	10.75 KB	0s	**10.71 KB**	0s	10.72 KB	0s
	BuEb	4.81 KB	1s	**4.64 KB**	0s	4.70 KB	0s	4.69 KB	0s	4.69 KB	0s
	Total	120.94 MB	5m22s	112.07 MB	1h18m14s	111.79 MB	11m24s	112.42 MB	32m23s	**109.68 MB**	1h04m28s

for the four comparisons. The total compression ratio is the

$$total\_output\_size \div total\_input\_size \times 100, \tag{4.1}$$

and the total speed is

$$total\_input\_size \div 1000 \div total\_seconds\_spent. \tag{4.2}$$

The results show GeCo3 achieving the best compression ratio, both in relative and conditional compression. The latter shows improved compression capabilities, with mean improvements of 11%, 35%, 38% and 50% over GeCo2, iDoComp, GDC2 and HRCM, respectively. This comes at a cost of being the slowest. The mean increase in time over GeCo2, iDoComp, GDC2 and HRCM is 1.7×, 9.8×, 2.6× and 7.3×, respectively. Compared with GeCo2, the total improvement for PT, PA, GG, and HS is 12.4%, 11.7%, 10.8% and 10.1%. The total improvements are similar to the mean improvement per chromosome. The computational RAM of GeCo3 is similar to GeCo2. For the majority of chromosome pairs, GeCo3 offers better compression.

Table 4.6: Total referential compression ratio and speed in kB/s for a re-sequenced Korean human genome. GeCo3 uses 64 hidden nodes and has 0.03 learning rate. The configuration for GeCo2-r and GeCo3-r (relative approach) is "-rm 20:500:1:35:0.95/3:100:0.95 -rm 13:200:1:1:0.95/0:0:0 -rm 10:10:0:0:0.95/0:0:0". For GeCo2-h and GeCo3-h (conditional approach) the following models where added "-tm 4:1:0:1:0.9/0:0:0 -tm 17:100:1:10:0.95/2:20:0.95". iDoComp, GDC2 and HRCM use the default configuration.

HRCM		GDC2		iDoComp		GeCo2-r		GeCo3-r		GeCo2-h		GeCo3-h	
ratio	speed	ratio	speed	ratio	speed	ratio	speed	ratio	speed	ratio	speed	ratio	speed
0.27	9,552	0.29	2,620	0.27	3,228	1.55	547	1.30	306	1.55	384	1.24	229

In Table 4.6, we show the results for compression of a re-sequenced genome. In this dataset HRCM achieves the best results, with GeCo3 trailing both in speed (42×) and ratio (−363%). While these results show that GeCo2 and GeCo3 are not suitable for compressing this type of dataset, the substantial improvement over GeCo2 (20%), hint at the possibility that the new mixer might be useful when integrated into a different type of compressor.

4.1.10 Estimating the cost for long term storage

To estimate the cost of long term storage, we developed a model with the following simplifying assumptions:

- two or more copies are stored;
- compression is done once and the result is copied to the different backup media;
- one CPU core is at 100% utilization during compression;
- the cooling and transfer costs are ignored;

Table 4.7: Total referential compression ratio and speed in kB/s. GeCo3 uses 64 hidden nodes and has 0.03 learning rate. The configuration for GeCo2-r and GeCo3-r (relative approach) is *"-rm 20:500:1:35:0.95/3:100:0.95 -rm 13:200:1:1:0.95/0:0:0 -rm 10:10:0:0:0.95/0:0:0"*. For GeCo2-h and GeCo3-h (conditional approach) the following models where added *"-tm 4:1:0:1:0.9/0:0:0 -tm 17:100:1:10:0.95/2:20:0.95"*. iDoComp, GDC2 and HRCM use the default configuration.

		HRCM		GDC2		iDoComp		GeCo2-r		GeCo3-r		GeCo2-h		GeCo3-h	
DSR	ID	ratio	speed	ratio	speed	ratio	speed	ratio	speed	ratio	speed	ratio	speed	ratio	speed
1	HSxPT	6.29	2,006	5.01	841	4.78	2,430	4.16	527	3.65	296	4.02	374	**3.52**	224
2	HSxPA	15.27	1,260	12.24	382	11.31	1,891	7.51	513	6,57	294	7.26	367	**6.41**	222
3	HSxGG	8.80	1,691	7.06	588	6.70	2,201	5.58	516	4.96	293	5.43	369	**4.84**	222
4	GGxHS	9.48	1,773	8.11	712	7.80	2,332	6.43	558	5.81	301	5.77	389	**5.19**	230
	Total	9.96	1,635	8.11	580	7.66	2,195	5.92	529	5.26	296	5.62	375	**4.99**	225

- the computing platform is idle when not compressing;
- no human operator is waiting for the operations to terminate.

Given the assumptions we now show the cost model:

$$\begin{aligned}
Total_{cost} &= Processing_{cost} + Storage_{cost}\\
Processing_{cost} &= Processing_{time} \times Power \times Energy_{price}\\
Storage_{cost} &= N_{copies} \times Size \times Size_{price},
\end{aligned}$$

where $Processing_{time}$ is the total time to compress and decompress the sequence.

From [156], we use the single thread load subtracted by the idle value to calculate the power (watts) a system uses during processing. The mean result for all systems is 34 watts. The mean cost of electricity in the world is €0.12 per kWh, according to [157]. The mean storage costs per GB for HDDs is €0.04 [158] and for SSDs is €0.13 [159].

Assuming €0.13 per GB and three copies, the costs for DS1 are €11.86, €9.54 and €9.5 for NAF, GeCo2, and GeCo3, respectively. Using €0.04 per GB and three copies, GeCo2 is more cost effective at €3.12, followed by GeCo3 (€3.46) and NAF (€3.74). In Fig. 4.3, we show the costs of storing each sequence in DS1 and DS2 with GeCo3 relative to NAF and GeCo2. As hinted by Fig. 4.1, we also show that the cost of compressing the Denisova sequence is improved when using 32 instead of 64 hidden nodes. The reduction of hidden nodes leads to a negligible drop in compression ratio (5.2% to 4.9% vs GeCo2), but a substantial time decrease (3.1× to 2.4× vs GeCo2).

These results use the mean costs, though given the variability of electricity prices, CPU power efficiency and storage costs, the analysis would need to be done for each specific case.

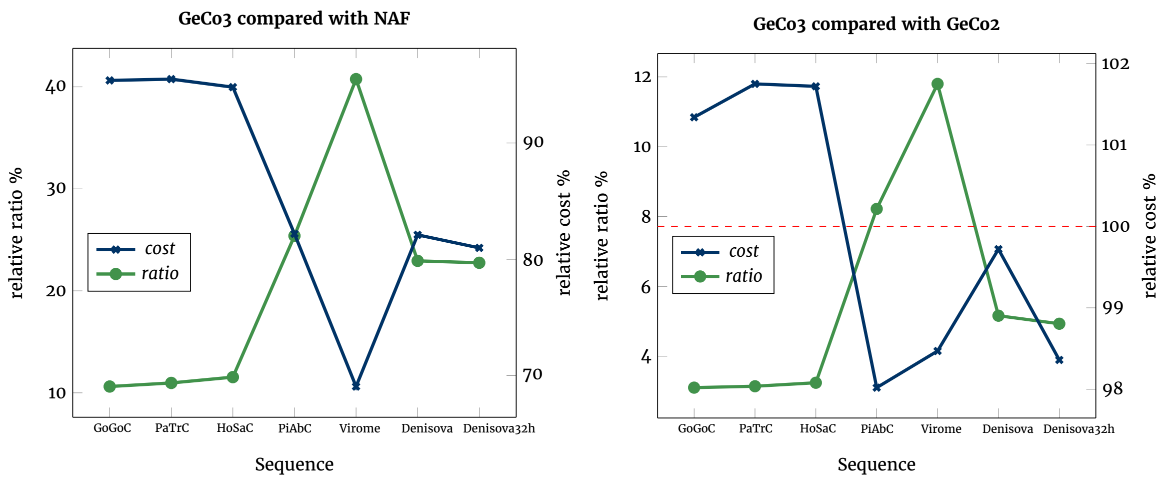


Figure 4.3: Relative ratio and cost of GeCo3 compared with NAF and GeCo2 for sequences in DS1 and DS2. Higher relative ratios represent greater compression improvements by GeCo3. The cost is calculated assuming €0.13 per GB and the storage of three copies. The red dashed line shows the cost threshold. Cost points above the line indicate that GeCo3 is more expensive. Denisova32h represents the results of running the Denisova sequence with 32 instead of 64 hidden nodes.

4.1.11 Why is the neural network better for mixing?

One of the possible reasons this approach has higher compression than GeCo2 is due to the mixing output not being constrained by the inputs. By comparing the histograms in Fig. 4.5 for the sequence EnIn and OrSa (two of the sequences with higher gains), we can verify that GeCo3 appears to correct the models' probabilities greater than 0.8 to probabilities closer to 0.99. Therefore, in some way, it is betting more if at least four in five chances are accomplished. Referential histograms are presented in Fig. 4.4; these are similar to the ones for reference-free compression.

Another reason the network performs better, especially in reference-based compression is that the network can learn to predict the sequence even when all the models are ignorant or have wrong opinions. This however means that this approach can't be used for pure referential compression, because the mixer will always use the information of the current sequence to supplement the inputs.

Another improvement is due to the higher percentage of symbols inferred correctly as seen in Table 4.8. For dataset five (DS5), GeCo3 has an mean improvement of 1.5% in the number of symbols inferred correctly, where only the smallest sequence has a lower hit rate than GeCo2.

For referential compression, we show a complexity profile in Fig. 4.6. This profile reveals that GeCo3 consistently outputs a lower number of bits per symbol. The gains appear to be larger in places of higher sequence complexity, namely in the higher Bps regions. These regions are typically where rapid switching between smaller models should

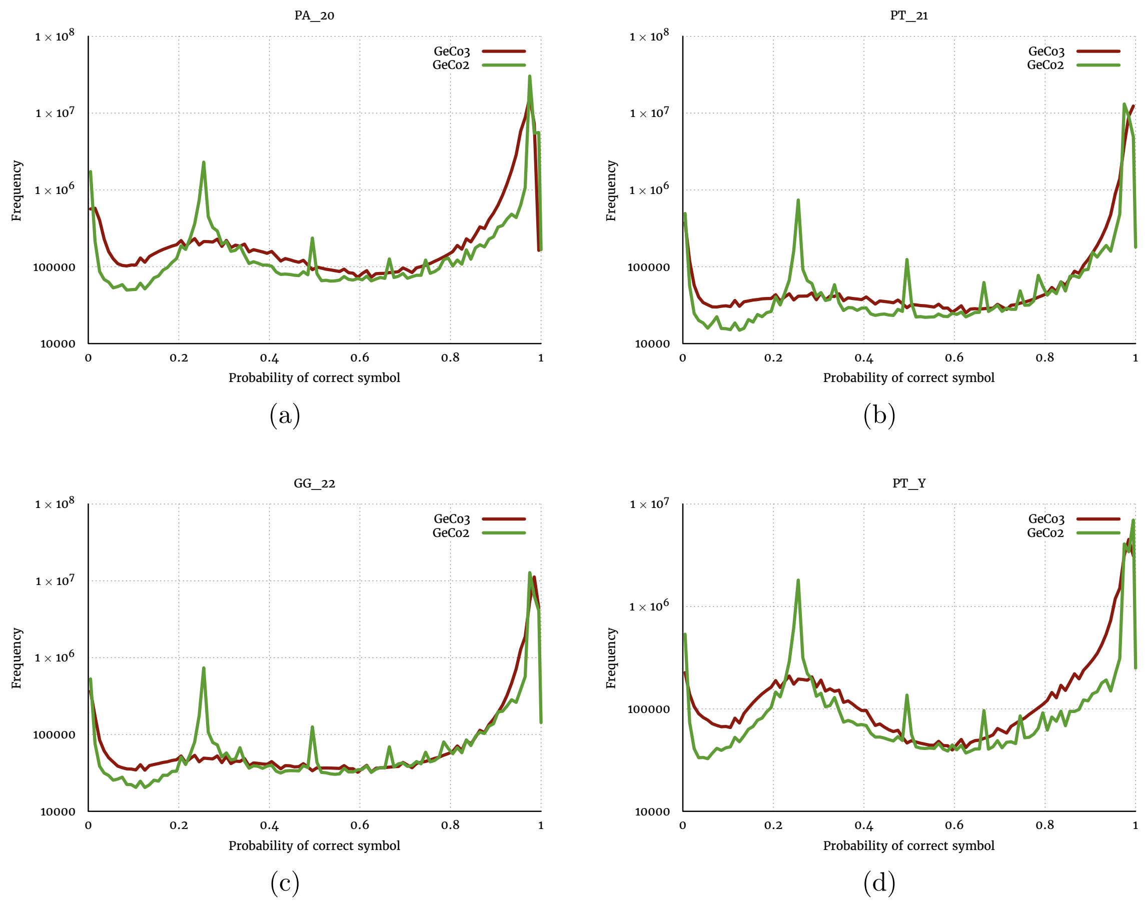
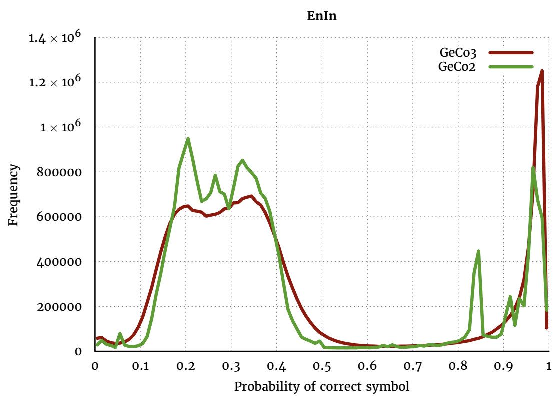


Figure 4.4: Histograms for GeCo2 and GeCo3 using referential compression, with the vertical axis in a log 10 scale. Four chromosome pairs are presented. The PA_20 is the chromosome 20 of the orangutan (PA) using human (HS) as a reference. The PT_21 and PT_Y are the chromosome 21 and Y of the chimpanzee (PT) using HS as reference. The GG_22 is the chromosome 22 gorilla (GG) using HS as reference.

---

occur, suggesting that the neural network mixer can adapt faster than the approach used in GeCo2.

Finally, the training is maintained during the entire sequence. Because, we found that doing early stopping leads to worse outcomes. This characteristic might be due to the advantages of over-fitting for non-stationary time series reported in [114] and also due to the constant need to adjust for concept drift.



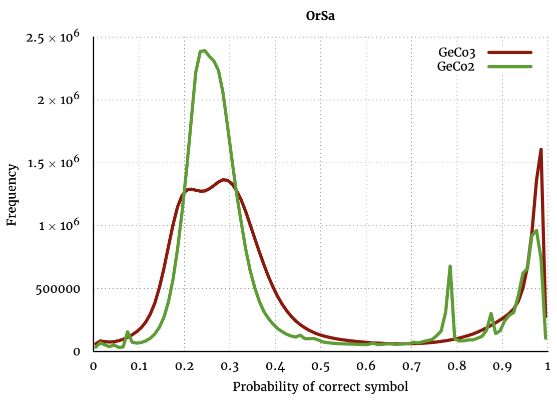
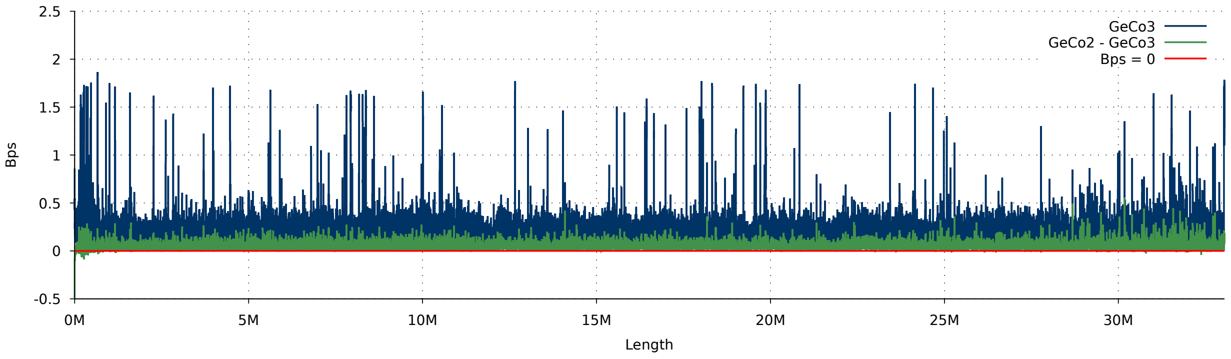


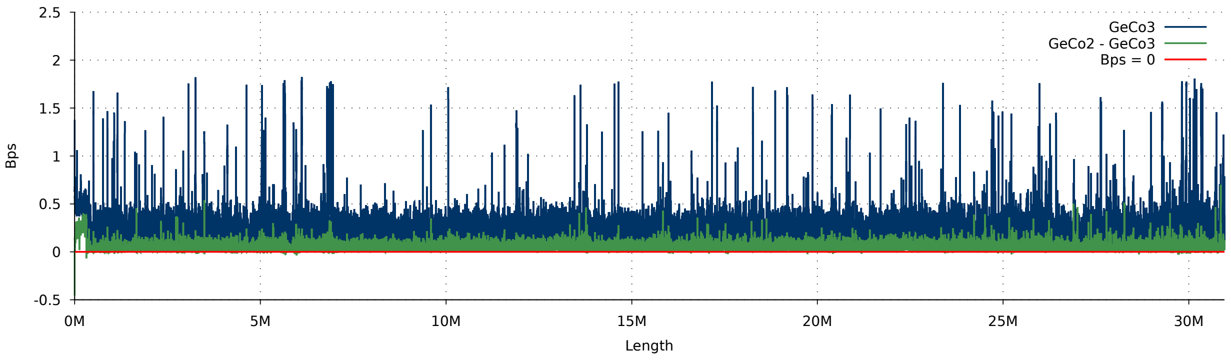
Figure 4.5: Comparison of histograms using the EnIn (*Entamoeba invadens*) and OrSa (*Oryza sativa*) genome sequences and the GeCo2 and GeCo3 as data compressors.

Table 4.8: Percentage of symbols predicted correctly by GeCo2 and GeCo3 for all sequences in dataset four (DS4). The improvement percentage of GeCo3 over GeCo2 is the diff.

ID	GeCo2	GeCo3	diff
HoSa	47.1	48.5	2.9
GaGa	38.9	40.0	2.8
DaRe	54.9	55.9	1.8
OrSa	48.2	49.7	3.0
DrMe	38.5	39.7	3.0
EnIn	50.3	51.2	1.8
ScPo	35.8	36.2	1.1
PlFa	44.4	45.3	2.0
EsCo	35.9	36.5	1.6
HaHi	40.0	40.4	1.0
AeCa	36.7	37.3	1.6
HePy	42.2	42.7	1.2
YeMi	41.7	42.1	1.0
AgPh	35.1	35.2	0.3
BuEb	33.2	32.4	-2.5



(a) PT_21 - Chromosome 21 from *Pan troglodytes* compressed with the corresponding *Homo sapiens* chromosome.



(b) GG_22 - Chromosome 22 from *Gorilla gorilla* compressed with the corresponding *Homo sapiens* chromosome.

Figure 4.6: Smoothed number of bits per symbol (Bps) of GeCo2 subtracted by GeCo3 Bps. The Bps were obtained by referential compression of PT_21 and GG_22, with the same parameters as in Table 4.7. Places where the line rises above zero indicate that GeCo3 has better compression than GeCo2.

4.2 AC2 results

In this section we evaluate the performance and accuracy of AC2, the tool derived from AC with the new mixer based on neural networks. Many of the main results of GeCo3, such as the effect of hidden nodes, scaling the number of models, computational resources, importance of derived features, among others, translate directly to AC2, therefore, we avoid to repeat them.

4.2.1 Datasets and materials

For benchmarking AC2 as a reference-free compressor, we use two datasets, namely

- **DS1**: three protein databases used in [153]:
    - uniprot : The UniProt collection of sequences [160];
    - pdbaa: The Protein Data Bank [161];
    - GRCh38: The human reference genome [162];
- **DS2**: a comprehensive dataset (proposed in [24]), containing the following sequences:
    - BT: Bos taurus;
    - HS: Homo sapiens;
    - SC: Saccharomyces cerevisiae;
    - HT: Haloterrigena turkmenica;
    - EC: Escherichia coli;
    - LC: Lactobacillus casei;
    - SA: Staphylococcus aureus;
    - HI: Haemophilus influenzae;
    - MJ: Methanococcus jannaschii;
    - DA: Desulfurococcus amylolyticus;
    - AP: Acanthamoeba polyphaga;
    - HA: Hadesarchaea archaeon;
    - FM: Fomitiporia mediterranea;
    - FV: Fowlpox virus;
    - XV: Xanthomonas virus Xp10;
    - EP: Enterococcus phage;

4.2.2 Amino acid compression benchmark

We were unable to find working tools for amino acid compression besides AC and NAF, therefore, we added general purpose compressors to make a more complete comparison. The general purpose compressors added are the Big Block BWT (Burrows-Wheeler transform) [163], the CMIX [43] and the LZMA [164].

Compared to AC, the results in Table 4.9 show compression improvements that range from 7.5% (5 Mb) for DS1, to 1.6% (175 kB) for DS2. These results are followed by $3-4\times$ slower execution. These results are in line with the GeCo3's, where the highest gains are in the bigger sequences. The AC2 has a higher performance gap than GeCo3, because the number of input and output nodes is $5\times$ larger. This means, that for the same number of models, more time is spent on mixing compared to GeCo3.

The AC2 achieves the overall best compression ratio for DS1 and DS2, but the lack of cache hash, which is present in GeCo3, means that the uniprot sequence needs more than 100GB of RAM to compress. The uniprot was compressed in a different machine, without support for the fused multiply–add (FMA) instruction, which is crucial for good performance of the neural network. As a consequence, the results for this sequences show that AC2 is $6\times$ slower while using the same number of hidden nodes as the other sequences of DS1.

4.2.3 Adaptable learning rate

GeCo3 and AC2 both take two new parameters. The hidden nodes is relative straight forward, since it typically increases the compression at the cost of greater computational resources until a certain threshold. The learning rate is also easy to set for large sequences, with the optimal values being around 0.03 and 0.16 for GeCo3 and AC2, respectively. Amino acid sequences are typically much shorter than DNA sequences, which means that setting the optimal learning rate becomes a trial and error process as seen by the spread of learning rates in Table 4.10. To combat this issue we decided to try the Adam optimizer which automatically adjusts the learning rate depending on the gradient and squared gradient.

The results in Table 4.10, show that by using Adam, we can use the same learning rate for all sequences and get a overall better result for the evaluated sequences, however we can still get better results by tuning the learning rate. For example using the learning rate equal to 0.0003, the compressed sequence EP becomes three bytes smaller. The same for FV, which using 0.0004, results in a compressed size 33 bytes smaller. The compression gains are very modest, at 0.2% for DS2 and almost no difference for DS1 (excluding uniprot).

A problematic feature of the Adam algorithm, as we implemented it, is that AC2 becomes between $4-15\times$ slower. This is due to the need to keep two exponential moving averages for each individual weight in the network.

AC2, with the Adam optimizer, achieves the best compression ratio for DS1, with a 1.8% (200 kB) improvement over AC, but at the cost of being $23\times$ slower. For the network sizes used, the memory requirements are very similar to those of AC, with the difference

Table 4.9: The bits per symbol (bps) and time needed to represent an amino acid sequence for BBB, LZMA, CMIX, NAF, AC and AC2. NAF uses the highest compression level (22) for all sequences. BBB uses the parameters 'cfm100q' for all sequences. LZMA uses the highest level (-9 -e) for all sequences. For DS2, AC and AC2 use the same levels as in [24]. For DS1, the models used by AC and AC2 are "-tm 1:1:0.76/1:1:0.88 -tm 2:10:0.83/1:1:0.86 -tm 3:20:0.83/2:1:0.87 -tm 4:50:0.88/2:10:0.89 -tm 5:200:0.94/3:20:0.89 -tm 6:300:0.91/5:20:0.88 -tm 7:500:0.91/6:60:0.87 -tm 8:500:0.92/7:15:0.89 -tm 9:1000:0.92/8:15:0.9 -tm 10:1500:0.92/9:80:0.9 -tm 11:1500:0.93/10:200:0.92 -tm 12:1500:0.94/11:200:0.93 -tm 13:1500:0.96/12:30:0.92 -tm 14:1750:0.95/13:150:0.93 -tm 15:2000:0.94/14:250:0.92 -tm 17:2200:0.95/16:350:0.93 -tm 20:2500:0.96/19:500:0.95". The asterisk (*) next to the time means that the compression was run on a different machine, with more RAM but slower CPUs.

		BBB		LZMA		CMIX		NAF		AC		AC2	
DS	ID	bps	time	bps	time	bps	time	bps	time	bps	time	bps	time
1	uniprot	2.872	2m34s	1.939	3m20s	**1.887**	47h08m07s	2.013	3m26s	2.071	2h03m37s*	1.894	12h15m54s*
	GRCh38	1.906	29s	1.196	22s	1.168	9h24m44s	1.203	35s	1.216	6m38s	**1.167**	27m36s
	pdbaa	2.541	25s	1.851	28s	1.847	7h06m24s	1.824	34s	1.790	6m55s	**1.735**	21m17s
	Total	2.689	3m28s	1.817	4m10s	1.774	63h39m16s	1.869	4m36s	1.910	2h17m12s*	**1.766**	13h04m48s*
2	BT	3.711	9s	3.208	10s	3.081	2h54m59s	3.114	12s	3.049	1m35s	**2.979**	3m53s
	HS	4.076	2s	4.022	2s	3.859	44m36s	3.905	2s	3.786	21s	**3.725**	48s
	SC	4.093	2s	4.030	1s	3.914	38m59s	3.956	2s	3.876	16s	**3.841**	38s
	HT	4.006	2s	3.971	1s	3.867	19m42s	3.929	1s	3.825	10s	**3.774**	23s
	EC	4.150	1s	4.209	1s	4.051	17m51s	4.108	1s	4.038	6s	**4.010**	17s
	LC	4.129	1s	4.188	0s	4.051	10m59s	4.119	1s	4.055	4s	**4.027**	10s
	SA	4.142	1s	4.213	0s	4.036	10m52s	4.109	1s	4.056	4s	**4.017**	8s
	HI	4.155	1s	4.239	0s	**4.087**	6m59s	4.155	1s	4.102	1s	4.090	4s
	MJ	4.059	0s	4.141	0s	3.974	6m15s	4.069	1s	3.997	1s	**3.971**	3s
	DA	4.083	0s	4.182	0s	**4.014**	5m31s	4.101	1s	4.028	1s	4.016	3s
	AP	4.084	0s	4.106	0s	**3.951**	4m46s	4.073	1s	3.985	1s	3.953	4s
	HA	4.122	0s	4.214	0s	**4.081**	3m03s	4.145	0s	4.082	0s	4.081	1s
	FM	3.968	0s	3.508	0s	3.538	2m19s	3.537	1s	3.426	1s	**3.388**	2s
	FV	4.130	0s	4.176	0s	4.063	1m12s	4.118	1s	4.063	0s	**4.055**	0s
	XV	4.188	0s	4.258	0s	4.140	14s	4.176	0s	**4.137**	0s	4.139	0s
	EP	4.262	0s	4.434	0s	**4.228**	7s	4.348	1s	4.323	0s	4.338	0s
	Total	3.900	19s	3.642	15s	3.507	48m30s	3.553	29s	3.476	2m48s	**3.421**	6m41s

being less than 20 megabytes. Compared with the other compressors the improvements are 12.5%, 6.3%, 2.7% and 3.9% for BBB, LZMA, CMIX and NAF respectively. The execution is 261× slower than LZMA and is 5× faster than CMIX. For reference, the AC2 with SGD is 31× slower than LZMA and is 45× faster than CMIX. The DS2 sequences benchmarked show no compression improvements.

4.2.4 Softmax activation and cross-entropy loss function

Instead of changing the optimization algorithm, we can experiment with different activation and loss functions. In the last layer, we can replace the logistic with the softmax function. This activation is more in line with what we need, as it ensures the results sum to one, which is what we want for the probabilities of the symbols. For the loss function,

Table 4.10: Number of bytes necessary to represent a subset of sequences from Table 4.9, with manually optimized learning rates (SGDv), a fixed learning (SGDf) and an adaptive learning rate (Adam). The L.Rate corresponds to the learning rates of SGDv. The SGDf uses a learning rate of 0.16 for all sequences and Adam uses 0.0002.

ID	SGDv	SGDf	Adam	L.Rate
GRCh38	6,070,584	6,070,584	**6,067,861**	0.16
pdbaa	**6,775,168**	**6,775,168**	6,777,180	0.16
Total	12,845,752	12,845,752	**12,845,041**	
BT	4,783,390	4,783,390	**4,763,349**	0.16
HS	1,534,479	1,534,479	**1,533,247**	0.16
SC	1,392,700	1,392,700	**1,391,935**	0.16
HT	695,259	695,259	**693,913**	0.16
EC	656,030	656,147	**655,278**	0.30
LC	407,400	407,458	**407,272**	0.25
SA	400,064	400,122	**399,904**	0.30
HI	260,482	260,494	**260,459**	0.30
MJ	222,768	222,768	**222,758**	0.16
DA	201,055	201,056	**200,944**	0.17
AP	168,810	168,810	**168,695**	0.16
HA	111,535	111,537	**111,484**	0.20
FM	**68,495**	68,508	68,787	0.19
FV	**40,927**	40,935	40,959	0.30
XV	**6,919**	6,920	6,923	0.40
EP	**2,269**	**2,269**	**2,269**	0.40
Total	10,952,582	10,952,852	**10,928,176**	

the cross-entropy is another popular alternative to the mean squared error (MSE). The results for AC2 with these functions are presented in Table 4.11. We observe that the time is similar, but the compression improves from 1.6% to 2.0% for DS2 and from 7.5% to 8.7% for DS1. This is a better result than using Adam and at no additional cost. In fact with these changes AC2 achieves the best compression ratio in all sequences, but the smaller two (EP and XV).

The results also show reduced execution times and the program uses less memory. For the uniprot sequence the RAM usage decreases from 111GB to 25GB. These improvements stem from micro-optimizations to the neural network code, such as using a faster version of the logistic activation function, but more importantly, we are able to run the uniprot sequence in the same test machine due to the modifications described in the next subsection.

Table 4.11: The bits per symbol (bps) and time needed to represent an amino acid sequence for CMIX and AC2 using the cache hash and softmax with cross-entropy. For DS2, AC2 uses the same levels as in Table 4.9, and for DS1 it uses level 8. All parameters are the same configurations previously used, but with the added cache hash parameter.

		CMIX		AC2	
DS	ID	bps	time	bps	time
1	uniprot	1.887	47h08m07s	**1.868**	<u>2h26m28s</u>
	GRCh38	1.168	9h24m44s	**1.154**	<u>26m22s</u>
	pdbaa	1.847	7h06m24s	**1.718**	<u>20m09s</u>
	Total	1.774	63h39m16s	**1.743**	<u>3h13m00s</u>
2	BT	3.081	2h54m59s	**2.961**	<u>4m21s</u>
	HS	3.859	44m36s	**3.717**	<u>49s</u>
	SC	3.914	38m59s	**3.836**	<u>47s</u>
	HT	3.867	19m42s	**3.764**	<u>26s</u>
	EC	4.051	17m51s	**4.000**	<u>18s</u>
	LC	4.051	10m59s	**4.019**	<u>9s</u>
	SA	4.036	10m52s	**4.009**	<u>9s</u>
	HI	4.087	6m59s	**4.083**	<u>4s</u>
	MJ	3.974	6m15s	**3.965**	<u>3s</u>
	DA	4.014	5m31s	**4.010**	<u>4s</u>
	AP	3.951	4m46s	**3.938**	<u>5s</u>
	HA	4.081	3m03s	**4.072**	<u>2s</u>
	FM	3.538	2m19s	**3.373**	<u>3s</u>
	FV	4.063	1m12s	**4.052**	<u>1s</u>
	XV	4.140	14s	**4.138**	<u>0s</u>
	EP	**4.228**	7s	4.321	<u>0s</u>
	Total	3.507	5h48m31s	**3.408**	<u>7m27s</u>

4.2.5 Memory reduction

An important goal for us, is to allow AC2 to be as usable as possible. With that in mind, we introduced two modifications to tool. First, we reduced the size of the counters for the models that used a hash table. AC uses eight bits per symbol, we reduced this to two bits per symbol. Furthermore, we introduced a similar caching mechanism to GeCo3. This allows the precise control of how much memory the program uses, it also speeds up the program and it enables compression of larger sequences.

Chapter 5

Conclusion and future work

This chapter serves to summarize the main contributions and results of our work, as well as provide some possible directions that we think are worthwhile to explore.

In essence, this dissertation considers the GeCo2 and AC as a base, collecting its specific DNA and amino acid models, and augments the mixture of models by using a neural network. The primary outcome is two new efficient tools, GeCo3 and AC2. The results show a compression improvement at the cost of longer execution times and equivalent RAM.

For the evaluated datasets, this approach delivers the best results for the most significant and the highest repetitive sequences. One of the reasons for this is that for small sequences, the network spends a significant percentage of time adjusting. Moreover, we show the importance of selecting and deriving the appropriate network inputs as well as the influence of the number of hidden nodes. These can be used to increase compression at the cost of higher execution times.

Compared to other state-of-the-art compressors, this approach is typically slower, but achieves better compression ratios both in reference-free and referential compression. Nevertheless, the compression times can be reduced by decreasing the number of hidden nodes while still improving the ratio.

The GeCo3 reference-free results, show an improvement of 25%, and 6% over NAF and GeCo2, respectively. In reference-based compression, GeCo3 is able to provide compression gains of 11%, 35%, 38%, and 50% over GeCo2, iDoComp, GDC2, and HRCM, respectively.

The time trade-off and the symmetry of compression-decompression establish GeCo3 as a non-appropriate tool for on-the-fly decompression. Tools such as NAF [11] are efficient for this purpose, namely because the computational decompression speed is very high, which for industrial usage is mandatory. The purposes of tools such as GeCo3 are in another domain, namely long-term storage and data analysis. In particular, the results suggest that long-term storage of extensive databases, for example, as proposed in [165], would be a good fit for GeCo3.

The steady rise of analysis tools based on DNA sequence compression is showing its potential, with increasing applications and surprising results. Some of the applications are the estimation of the Kolmogorov complexity of genomes [166], rearrangements detection [167], sequence clustering [168], measurement of distances and phylogenetic trees computation [169], and metagenomics [170].

The AC2 results show improvement between 2.0% and 8.7% over AC, depending if the compression is done for smaller (the former) or larger sequences (the latter).

The main advantage of using efficient (lossless) compression-based data analysis is non-overestimation. Many analysis algorithms include multiple thresholds that use a consensus value for what is considered balanced and consistent, leaving space for overestimation. The problem is that using a consensus or average parameter for a specific analysis may overtake the limit of the estimation balance. Since data compression needs the appropriate decompressor to ensure the full recovery of the data, the compressor acts under a boundary that ensures that the limit is never crossed (Kolmogorov complexity). This property is critical in data analysis because the data in use may be vital and sensitive, mainly when multiple models are used. Without a channel information limit and an efficient mixing model, the information that is embedded in the probabilities estimation of each model transits to the model choice.

The mixing method used to achieve these results assumes only that probabilities for the symbols are available. Because of this, it permits to be easily exported to other compressors or compressed-based data analysis tools that use multiple models. GeCo3 and AC2 shows what compression improvements and execution times can be expected when using neural networks for the mixture of experts in DNA and amino acid sequence compression.

This dissertation highlights the importance of expert mixing. Mixing has applications in all areas where there is the uncertainty of outcomes, and many expert opinions are available. This ranges from data compression, to climate modeling and, in the future, possibly the creation of legislation. While more traditional methods, such as weighted majority voting, are more efficient and can achieve good accuracy, neural networks show promising results. With the development of specialized hardware instructions and data-types to be included in general-purpose CPUs [27, 28], neural networks should become an even more attractive option for expert mixing.

Additional improvements on the compression of large FASTQ data, for example, from the Virome and Denisova datasets can be achieved with complementary techniques based on reordering or metagenomic composition identification. Specifically, the reads of these datasets can be split according to their composition using fast assembly and alignment-free methods, namely extensions of Read-SpaM [171], in order to take advantage of the similarity read proximity to improve substantially the compression.

Because of the tremendous impact that the derived features and the input transformations have in the mixing performance, one of the future directions would be to use some form of automated feature extraction. The deep models we used didn't provide this capability, but maybe other CNN architectures are more suited for this type of problem. Another direction to explore would be the potential usage of hardware acceleration (e.g., GPUs) to speedup the compression, maybe by using an API that can target both CPUs and GPUs in a transparent fashion. It would also be interesting to test other machine learning tools to do the mixing, as these might have better performance or greater compression gains. Finally, it would be interesting to integrate other types of specific DNA models in the same mixing framework, to see what types of gains could be achieved.

Whichever the technology and application, the core method that we provide here, namely for combining the specific DNA models with neural networks, enables a substantial improvement in the precision of DNA sequence compression-based data analysis tools and provides a significant reduction of storage associated with DNA sequences. In the case of amino acid compression, neural networks also show similar improvements. Both AC2 and GeCo3 almost always surpass the current state of the art compression ratios for biosequence data compression.

Bibliography

[1] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, “The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants,” *Nucleic acids research*, vol. 38, no. 6, pp. 1767–1771, 2010.

[2] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, “Big data: astronomical or genomical?” *PLoS biology*, vol. 13, no. 7, p. e1002195, 2015.

[3] M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney, “Efficient storage of high throughput DNA sequencing data using reference-based compression,” *Genome research*, vol. 21, no. 5, pp. 734–740, 2011.

[4] S. Christley, Y. Lu, C. Li, and X. Xie, “Human genomes as email attachments,” *Bioinformatics*, vol. 25, no. 2, pp. 274–275, 2009.

[5] R. Giancarlo, D. Scaturro, and F. Utro, “Textual data compression in computational biology: a synopsis,” *Bioinformatics*, vol. 25, no. 13, pp. 1575–1586, 2009.

[6] A. Milosavljević and J. Jurka, “Discovering simple DNA sequences by the algorithmic significance method,” *Bioinformatics*, vol. 9, no. 4, pp. 407–411, 1993.

[7] B. T. Korber, R. M. Farber, D. H. Wolpert, and A. S. Lapedes, “Covariation of mutations in the V3 loop of human immunodeficiency virus type 1 envelope protein: an information theoretic analysis,” *Proceedings of the National Academy of Sciences*, vol. 90, no. 15, pp. 7176–7180, 1993.

[8] Ö. U. Nalbantoglu, D. J. Russell, and K. Sayood, “Data compression concepts and algorithms and their applications to bioinformatics,” *Entropy*, vol. 12, no. 1, pp. 34–52, 2010.

[9] M. Hernaez, D. Pavlichin, T. Weissman, and I. Ochoa, “Genomic Data Compression,” *Annual Review of Biomedical Data Science*, vol. 2, pp. 19–37, 2019.

[10] M. Goyal, K. Tatwawadi, S. Chandak, and I. Ochoa, “DeepZip: Lossless Data Compression using Recurrent Neural Networks,” *arXiv preprint arXiv:1811.08162*, 2018.

[11] K. Kryukov, M. T. Ueda, S. Nakagawa, and T. Imanishi, "Nucleotide Archival Format (NAF) enables efficient lossless reference-free compression of DNA sequences," *Bioinformatics*, vol. 35, no. 19, pp. 3826–3828, 2019.

[12] A. Hategan and I. Tabus, "Jointly Encoding Protein Sequences and their Secondary Structure Information," in *2007 IEEE International Workshop on Genomic Signal Processing and Statistics.* IEEE, 2007, pp. 1–4.

[13] M. Mahoney. Data Compression Explained. [Online]. Available: http://mattmahoney.net/dc/dce.html

[14] K. Sayood, *Introduction to data compression.* Morgan Kaufmann, 2017.

[15] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[16] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[17] A. Said, "Introduction to arithmetic coding-theory and practice," *Hewlett Packard Laboratories Report*, pp. 1057–7149, 2004.

[18] P. Vitányi, "How incomputable is Kolmogorov complexity?" *Entropy*, vol. 22, no. 4, p. 408, 2020.

[19] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *J. of Math*, vol. 58, no. 345-363, p. 5, 1936.

[20] F. F. Abbott, *A history and description of Roman political institutions.* Ginn & Company, 1911.

[21] N. Littlestone, M. K. Warmuth *et al.*, *The weighted majority algorithm.* University of California, Santa Cruz, Computer Research Laboratory, 1989.

[22] R. Polikar, "Ensemble based systems in decision making," *IEEE Circuits and systems magazine*, vol. 6, no. 3, pp. 21–45, 2006.

[23] D. Pratas, M. Hosseini, and A. J. Pinho, "GeCo2: An Optimized Tool for Lossless Compression and Analysis of DNA Sequences," in *International Conference on Practical Applications of Computational Biology & Bioinformatics.* Springer, 2019, pp. 137–145.

[24] M. Hosseini, D. Pratas, and A. J. Pinho, "AC: A compression tool for amino acid sequences," *Interdisciplinary Sciences: Computational Life Sciences*, vol. 11, no. 1, pp. 68–76, 2019.

[25] D. Pratas, M. Hosseini, and A. J. Pinho, "Substitutional tolerant Markov models for relative compression of DNA sequences," in *International Conference on Practical Applications of Computational Biology & Bioinformatics.* Springer, 2017, pp. 265–272.

[26] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[27] BFLOAT16 – Hardware Numerics Definition. [Online]. Available: https://software.intel.com/sites/default/files/managed/40/8b/bf16-hardware-numerics-definition-white-paper.pdf

[28] IBM Reveals Next-Generation IBM POWER10 Processor. [Online]. Available: https://newsroom.ibm.com/2020-08-17-IBM-Reveals-Next-Generation-IBM-POWER10-Processor

[29] M. Silva, D. Pratas, and A. J. Pinho, "Efficient DNA sequence compression with neural networks," *GigaScience*, vol. 9, no. 11, Nov. 2020. [Online]. Available: https://doi.org/10.1093/gigascience/giaa119

[30] ——, "Supporting data for "Efficient DNA sequence compression with neural networks"," 2020. [Online]. Available: http://gigadb.org/dataset/100808

[31] S. Hornblower, A. Spawforth, and E. Eidinow, *The Oxford classical dictionary.* Oxford University Press, 2012.

[32] L. Russo *et al.*, *The forgotten revolution: How science was born in 300 BC and why it had to be reborn.* Springer Science & Business Media, 2013.

[33] R. Hoffman, *Data compression in digital systems.* Springer Science & Business Media, 2012.

[34] L. Coe, *The telegraph: A history of Morse's invention and its predecessors in the United States.* McFarland, 2003.

[35] C. C. Cutler, "Differential quantization of communication signals," Jul. 29 1952, uS Patent 2,605,361.

[36] A. Robinson and C. Cherry, "Results of a prototype television bandwidth compression scheme," *Proceedings of the IEEE*, vol. 55, no. 3, pp. 356–364, 1967.

[37] D. Salomon, *Data compression: the complete reference.* Springer Science & Business Media, 2004.

[38] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[39] ——, "Compression of individual sequences via variable-rate coding," *IEEE transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[40] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM Journal of research and development*, vol. 23, no. 2, pp. 149–162, 1979.

[41] P. G. Howard and J. S. Vitter, "Arithmetic coding for data compression," *Proceedings of the IEEE*, vol. 82, no. 6, pp. 857–865, 1994.

[42] J. Schmidhuber and S. Heil, "Sequential neural text compression," *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 142–146, 1996.

[43] B. Knoll. CMIX. [Online]. Available: http://www.byronknoll.com/cmix.html

[44] M. Mahoney. Large text compression benchmark. [Online]. Available: http://mattmahoney.net/dc/text.html

[45] P. G. Smith, *Professional Website Performance: Optimizing the Front-End and Back-End*. John Wiley & Sons, 2012.

[46] S. Dominé, "Using texture compression in OpenGL," *Nvidia Corporation*, 2000.

[47] M. Rabbani, "JPEG2000: Image compression fundamentals, standards and practice," *Journal of Electronic Imaging*, vol. 11, no. 2, p. 286, 2002.

[48] D. Y. Pan, "Digital audio compression," *Digital Technical Journal*, vol. 5, no. 2, pp. 28–40, 1993.

[49] D. Le Gall, "MPEG: A video compression standard for multimedia applications," *Communications of the ACM*, vol. 34, no. 4, pp. 46–59, 1991.

[50] H. Jiang, "The Intel® Quick Sync Video technology in the 2nd-generation Intel Core processor family," in *2011 IEEE Hot Chips 23 Symposium (HCS)*. IEEE, 2011, pp. 1–23.

[51] A. Patait and E. Young, "High performance video encoding with NVIDIA GPUs," in *2016 GPU Technology Conference (https://goo. gl/Bdjdgm)*, 2016.

[52] A. Mazer, *The ellipse: a historical and mathematical journey*. John Wiley & Sons, 2011.

[53] K. E. Iverson, "Notation as a tool of thought," in *ACM Turing award lectures*, 2007, p. 1979.

[54] B. Victor, "Media for thinking the unthinkable," *Vimeo, May*, 2013.

[55] C. Muratori. Semantic Compression. [Online]. Available: https://caseymuratori.com/blog_0015

[56] L. H. Rieseberg, "Chromosomal rearrangements and speciation," *Trends in ecology & evolution*, vol. 16, no. 7, pp. 351–358, 2001.

[57] G. S. Roeder and G. R. Fink, "DNA rearrangements associated with a transposable element in yeast," *Cell*, vol. 21, no. 1, pp. 239–249, 1980.

[58] K. Harris, "Evidence for recent, population-specific evolution of the human mutation rate," *Proceedings of the National Academy of Sciences*, vol. 112, no. 11, pp. 3439–3444, 2015.

[59] C. Jeong and A. Di Rienzo, "Adaptations to local environments in modern human populations," *Current opinion in genetics & development*, vol. 29, pp. 1–8, 2014.

[60] S. Beres, P. Kachroo, W. Nasser, R. Olsen, L. Zhu, A. Flores, I. de la Riva, J. Paez-Mayorga, F. Jimenez, C. Cantu *et al.*, "Transcriptome remodeling contributes to epidemic disease caused by the human pathogen," *Streptococcus pyogenes*, pp. 00 403–16, 2016.

[61] M. Fumagalli and M. Sironi, "Human genome variability, natural selection and infectious diseases," *Current opinion in immunology*, vol. 30, pp. 9–16, 2014.

[62] H. Long, W. Sung, S. Kucukyildirim, E. Williams, S. F. Miller, W. Guo, C. Patterson, C. Gregory, C. Strauss, C. Stone *et al.*, "Evolutionary determinants of genome-wide nucleotide composition," *Nature ecology & evolution*, p. 1, 2018.

[63] A. Golan, *Foundations of Info-Metrics: Modeling and Inference with Imperfect Information.* Oxford University Press, 2017.

[64] C. G. Nevill-Manning and I. H. Witten, "Protein is incompressible," in *Proceedings DCC'99 Data Compression Conference (Cat. No. PR00096).* IEEE, 1999, pp. 257–266.

[65] S. Grumbach and F. Tahi, "Compression of DNA sequences," 1993, pp. 340–350.

[66] ——, "A new challenge for compression algorithms: genetic sequences," *Information Processing & Management*, vol. 30, no. 6, pp. 875–886, 1994.

[67] E. Rivals, J.-P. Delahaye, M. Dauchet, and O. Delgrange, "A guaranteed compression scheme for repetitive DNA sequences," in *Proceedings of Data Compression Conference-DCC'96.* IEEE, 1996, p. 453.

[68] P. N. Yianilos, "Significantly lower entropy estimates for natural dna sequences," 1996.

[69] L. Allison, T. Edgoose, and T. I. Dix, "Compression of strings with approximate repeats." in *ISMB*, 1998, pp. 8–16.

[70] X. Chen, S. Kwong, and M. Li, "A compression algorithm for DNA sequences and its applications in genome comparison," *Genome informatics*, vol. 10, pp. 51–61, 1999.

[71] X. Chen, M. Li, B. Ma, and J. Tromp, "DNACompress: fast and effective DNA sequence compression," *Bioinformatics*, vol. 18, no. 12, pp. 1696–1698, 2002.

[72] I. Tabus, G. Korodi, and J. Rissanen, "DNA sequence compression using the normalized maximum likelihood model for discrete regression," in *Data Compression Conference, 2003. Proceedings. DCC 2003.* IEEE, 2003, pp. 253–262.

[73] G. Korodi and I. Tabus, "An efficient normalized maximum likelihood algorithm for DNA sequence compression," *ACM Transactions on Information Systems (TOIS)*, vol. 23, no. 1, pp. 3–34, 2005.

[74] M. D. Cao, T. I. Dix, L. Allison, and C. Mears, "A simple statistical algorithm for biological sequence compression," in *2007 Data Compression Conference (DCC'07).* IEEE, 2007, pp. 43–52.

[75] K. N. Mishra, A. Aaggarwal, E. Abdelhadi, and D. Srivastava, "An efficient horizontal and vertical method for online DNA sequence compression," *International Journal of Computer Applications*, vol. 3, no. 1, pp. 39–46, 2010.

[76] A. Gupta and S. Agarwal, "A novel approach for compressing DNA sequences using semi-statistical compressor," *International Journal of Computers and Applications*, vol. 33, no. 3, pp. 245–251, 2011.

[77] A. J. Pinho, D. Pratas, and P. J. Ferreira, "Bacteria DNA sequence compression using a mixture of finite-context models," in *2011 IEEE Statistical Signal Processing Workshop (SSP).* IEEE, 2011, pp. 125–128.

[78] Z. Zhu, J. Zhou, Z. Ji, and Y.-H. Shi, "DNA sequence compression using adaptive particle swarm optimization-based memetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5, pp. 643–658, 2011.

[79] D. Satyanvesh, K. Balleda, A. Padyana, and P. Baruah, "Gencodex-A novel algorithm for compressing DNA sequences on multi-cores and GPUs," in *Proc. IEEE, 19th International Conf. on High Performance Computing (HiPC), Pune, India*, no. 37, 2012.

[80] P. Li, S. Wang, J. Kim, H. Xiong, L. Ohno-Machado, and X. Jiang, "DNA-COMPACT: DNA COMpression Based on a Pattern-Aware Contextual Modeling Technique," *PloS one*, vol. 8, no. 11, p. e80377, 2013.

[81] D. Pratas and A. J. Pinho, "Exploring deep Markov models in genomic data compression using sequence pre-analysis," in *2014 22nd European Signal Processing Conference (EUSIPCO).* IEEE, 2014, pp. 2395–2399.

[82] X. Xie, S. Zhou, and J. Guan, "CoGI: Towards compressing genomes as an image," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 12, no. 6, pp. 1275–1285, 2015.

[83] D. Pratas, A. J. Pinho, and P. J. Ferreira, "Efficient compression of genomic sequences," in *2016 Data Compression Conference (DCC)*. IEEE, 2016, pp. 231–240.

[84] M. Chen, J. Shao, and X. Jia, "Genome sequence compression based on optimized context weighting," *Genet. Mol. Res*, vol. 16, no. 2, 2017.

[85] D. Mansouri and X. Yuan, "One-Bit DNA Compression Algorithm," in *International Conference on Neural Information Processing*. Springer, 2018, pp. 378–386.

[86] D. Pratas, M. Hosseini, J. M. Silva, and A. J. Pinho, "A Reference-Free Lossless Compression Algorithm for DNA Sequences Using a Competitive Prediction of Two Classes of Weighted Models," *Entropy*, vol. 21, no. 11, p. 1074, 2019.

[87] S. Kuruppu, S. J. Puglisi, and J. Zobel, "Relative Lempel-Ziv compression of genomes for large-scale storage and retrieval," in *International Symposium on String Processing and Information Retrieval*. Springer, 2010, pp. 201–206.

[88] C. Wang and D. Zhang, "A novel compression tool for efficient storage of genome resequencing data," *Nucleic acids research*, vol. 39, no. 7, pp. e45–e45, 2011.

[89] S. Deorowicz and S. Grabowski, "Robust relative compression of genomes with random access," *Bioinformatics*, vol. 27, no. 21, pp. 2979–2986, 2011.

[90] S. Kuruppu, B. Beresford-Smith, T. Conway, and J. Zobel, "Iterative dictionary construction for compression of large DNA data sets," *IEEE/ACM transactions on Computational Biology and Bioinformatics*, vol. 9, no. 1, pp. 137–149, 2011.

[91] A. J. Pinho, D. Pratas, and S. P. Garcia, "GReEn: a tool for efficient compression of genome resequencing data," *Nucleic acids research*, vol. 40, no. 4, pp. e27–e27, 2012.

[92] S. Wandelt and U. Leser, "FRESCO: Referential compression of highly similar sequences," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 10, no. 5, pp. 1275–1288, 2013.

[93] S. Deorowicz, A. Danek, and M. Niemiec, "GDC 2: Compression of large collections of genomes," *Scientific reports*, vol. 5, p. 11565, 2015.

[94] I. Ochoa, M. Hernaez, and T. Weissman, "iDoComp: a compression scheme for assembled genomes," *Bioinformatics*, vol. 31, no. 5, pp. 626–633, 2015.

[95] S. Saha and S. Rajasekaran, "ERGC: an efficient referential genome compression algorithm," *Bioinformatics*, vol. 31, no. 21, pp. 3468–3475, 2015.

[96] Y. Liu, H. Peng, L. Wong, and J. Li, "High-speed and high-ratio referential genome compression," *Bioinformatics*, vol. 33, no. 21, pp. 3364–3372, 2017.

[97] H. Yao, Y. Ji, K. Li, S. Liu, J. He, and R. Wang, "HRCM: An Efficient Hybrid Referential Compression Method for Genomic Big Data," *BioMed Research International*, vol. 2019, 2019.

[98] A. Hategan and I. Tabus, "Protein is compressible," in *Proceedings of the 6th Nordic Signal Processing Symposium, 2004. NORSIG 2004.* IEEE, 2004, pp. 192–195.

[99] D. Adjeroh and F. Nan, "On compressibility of protein sequences," in *Data Compression Conference (DCC'06).* IEEE, 2006, pp. 10–pp.

[100] N. M. Daniels, A. Gallant, J. Peng, L. J. Cowen, M. Baym, and B. Berger, "Compressive genomics for protein databases," *Bioinformatics*, vol. 29, no. 13, pp. i283–i290, 2013.

[101] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1249, 2018.

[102] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Information Fusion*, vol. 37, pp. 132–156, 2017.

[103] N. Bouaynaya and D. Schonfeld, "Non-stationary analysis of DNA sequences," in *2007 IEEE/SP 14th Workshop on Statistical Signal Processing.* IEEE, 2007, pp. 200–204.

[104] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *Journal of Machine Learning Research*, vol. 8, no. Dec, pp. 2755–2790, 2007.

[105] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky, "Bayesian model averaging: a tutorial," *Statistical science*, pp. 382–401, 1999.

[106] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine learning*, vol. 37, no. 3, pp. 297–336, 1999.

[107] S. Mao, L. Jiao, L. Xiong, S. Gou, B. Chen, and S.-K. Yeung, "Weighted classifier ensemble based on quadratic form," *Pattern Recognition*, vol. 48, no. 5, pp. 1688–1706, 2015.

[108] R. Wang, Y. Bai, Y.-S. Chu, Z. Wang, Y. Wang, M. Sun, J. Li, T. Zang, and Y. Wang, "DeepDNA: a hybrid convolutional and recurrent neural network for compressing human mitochondrial genomes," in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM).* IEEE, 2018, pp. 270–274.

[109] M. V. Mahoney, "Fast Text Compression with Neural Networks." in *FLAIRS Conference*, 2000, pp. 230–234.

[110] C. Baldassi, C. Borgs, J. T. Chayes, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina, "Unreasonable effectiveness of learning neural networks: From accessible states and robust ensembles to basic algorithmic schemes," *Proceedings of the National Academy of Sciences*, vol. 113, no. 48, pp. E7655–E7662, 2016.

[111] L. Flagel, Y. Brandvain, and D. R. Schrider, "The unreasonable effectiveness of convolutional neural networks in population genetic inference," *Molecular biology and evolution*, vol. 36, no. 2, pp. 220–238, 2019.

[112] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks," *Andrej Karpathy blog*, vol. 21, p. 23, 2015.

[113] P. L. Rosin and F. Fierens, "Improving neural network generalisation," in *1995 International Geoscience and Remote Sensing Symposium, IGARSS'95. Quantitative Remote Sensing for Science and Applications*, vol. 2. IEEE, 1995, pp. 1255–1257.

[114] T. Y. Kim, K. J. Oh, C. Kim, and J. D. Do, "Artificial neural networks for non-stationary time series," *Neurocomputing*, vol. 61, pp. 439–447, 2004.

[115] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[116] M. Minsky and S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, 1969.

[117] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[118] P. J. Werbos, *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting.* John Wiley & Sons, 1994, vol. 1.

[119] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[120] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[121] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[122] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[123] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[124] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[125] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[126] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[127] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[128] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, 2013, pp. 1310–1318.

[129] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[130] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[131] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural networks*, vol. 1, no. 2, pp. 119–130, 1988.

[132] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets.* Springer, 1982, pp. 267–285.

[133] K. J. Lang, A. H. Waibel, and G. E. Hinton, "A time-delay neural network architecture for isolated word recognition," *Neural networks*, vol. 3, no. 1, pp. 23–43, 1990.

[134] Y. LeCun *et al.*, "Generalization and network design strategies," *Connectionism in perspective*, vol. 19, pp. 143–155, 1989.

[135] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[136] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, “The marginal value of adaptive gradient methods in machine learning,” in *Advances in neural information processing systems*, 2017, pp. 4148–4158.

[137] N. S. Keskar and R. Socher, “Improving generalization performance by switching from adam to sgd,” *arXiv preprint arXiv:1712.07628*, 2017.

[138] G. A. Churchill, “Stochastic models for heterogeneous DNA sequences,” *Bulletin of mathematical biology*, vol. 51, no. 1, pp. 79–94, 1989.

[139] P. Lara-Benítez, M. Carranza-García, F. Martínez-Álvarez, and J. C. Riquelme, “On the performance of deep learning models for time series classification in streaming,” *arXiv preprint arXiv:2003.02544*, 2020.

[140] M. Hiransha, E. A. Gopalakrishnan, V. K. Menon, and K. Soman, “NSE stock market prediction using deep-learning models,” *Procedia computer science*, vol. 132, pp. 1351–1362, 2018.

[141] J. Struye and S. Latré, “Hierarchical temporal memory and recurrent neural networks for time series prediction: An empirical validation and reduction to multilayer perceptrons,” *Neurocomputing*, vol. 396, pp. 291–301, 2020.

[142] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep learning for time series classification: a review,” *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[143] Z. Wang, W. Yan, and T. Oates, “Time series classification from scratch with deep neural networks: A strong baseline,” in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 1578–1585.

[144] T. Lin, T. Guo, and K. Aberer, “Hybrid neural networks for learning the trend in time series,” in *Proceedings of the twenty-sixth international joint conference on artificial intelligence*, no. CONF, 2017, pp. 2273–2279.

[145] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[146] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[147] D. Pratas and A. J. Pinho, “A DNA sequence corpus for compression benchmark,” in *International Conference on Practical Applications of Computational Biology & Bioinformatics*. Springer, 2018, pp. 208–215.

[148] H. Robbins and S. Monro, “A stochastic approximation method,” *The annals of mathematical statistics*, pp. 400–407, 1951.

[149] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[150] S. Russel, P. Norvig *et al.*, *Artificial intelligence: a modern approach.* Pearson Education Limited, 2013.

[151] D. Pratas, M. Toppinen, L. Pyöriä, K. Hedman, A. Sajantila, and M. F. Perdomo, “A hybrid pipeline for reconstruction and analysis of viral genomes at multi-organ level,” *GigaScience*, vol. 9, p. giaa086, 2020.

[152] M. Meyer, M. Kircher, M.-T. Gansauge, H. Li, F. Racimo, S. Mallick, J. G. Schraiber, F. Jay, K. Prüfer, C. De Filippo *et al.*, “A high-coverage genome sequence from an archaic Denisovan individual,” *Science*, vol. 338, no. 6104, pp. 222–226, 2012.

[153] K. Kryukov, M. T. Ueda, S. Nakagawa, and T. Imanishi, “Sequence Compression Benchmark (SCB) database—A comprehensive evaluation of reference-free compressors for FASTA-formatted sequences,” *GigaScience*, vol. 9, no. 7, p. giaa072, 2020.

[154] J. Ijdo, A. Baldini, D. Ward, S. Reeders, and R. Wells, “Origin of human chromosome 2: an ancestral telomere-telomere fusion.” *Proceedings of the National Academy of Sciences*, vol. 88, no. 20, pp. 9051–9055, 1991.

[155] M. D. Cao, T. I. Dix, L. Allison, and C. Mears, “A simple statistical algorithm for biological sequence compression,” Mar. 2007, pp. 43–52.

[156] H. Hagedoorn. AMD Ryzen 5 3600 review - Power Consumption and temperatures. [Online]. Available: https://www.guru3d.com/articles-pages/amd-ryzen-5-3600-review,7.html

[157] Electricity prices. [Online]. Available: https://www.globalpetrolprices.com/electricity_prices/

[158] Amazon.de HDD prices. [Online]. Available: https://diskprices.com/?locale=us&condition=new&units=gb&disk_types=internal_hdd25,internal_sshd,internal_sas

[159] Amazon.de SSD prices. [Online]. Available: https://diskprices.com/?locale=us&condition=new&units=gb&disk_types=internal_ssd,m2_ssd,m2_nvme

[160] U. Consortium, “Uniprot: a worldwide hub of protein knowledge,” *Nucleic acids research*, vol. 47, no. D1, pp. D506–D515, 2019.

[161] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, “The protein data bank,” *Nucleic acids research*, vol. 28, no. 1, pp. 235–242, 2000.

[162] A. D. Yates, P. Achuthan, W. Akanni, J. Allen, J. Allen, J. Alvarez-Jarreta, M. R. Amode, I. M. Armean, A. G. Azov, R. Bennett *et al.*, "Ensembl 2020," *Nucleic acids research*, vol. 48, no. D1, pp. D682–D688, 2020.

[163] M. Mahoney. Big Block BWT. [Online]. Available: http://mattmahoney.net/dc/#bbb

[164] I. Pavlov, "Lzma sdk (software development kit)," 2007.

[165] H. A. Lewin, G. E. Robinson, W. J. Kress, W. J. Baker, J. Coddington, K. A. Crandall, R. Durbin, S. V. Edwards, F. Forest, M. T. P. Gilbert *et al.*, "Earth BioGenome Project: Sequencing life for the future of life," *Proceedings of the National Academy of Sciences*, vol. 115, no. 17, pp. 4325–4333, 2018.

[166] D. Pratas and A. J. Pinho, "On the approximation of the Kolmogorov complexity for DNA sequences," in *Iberian Conference on Pattern Recognition and Image Analysis*. Springer, 2017, pp. 259–266.

[167] M. Hosseini, D. Pratas, B. Morgenstern, and A. J. Pinho, "Smash++: an alignment-free and memory-efficient tool to find genomic rearrangements," *GigaScience*, vol. 9, no. 5, 05 2020.

[168] R. Cilibrasi and P. M. Vitányi, "Clustering by compression," *IEEE Transactions on Information theory*, vol. 51, no. 4, pp. 1523–1545, 2005.

[169] M. Li, X. Li, B. Ma, and P. Vitányi, "Normalized information distance and whole mitochondrial genome phylogeny analysis," *arXiv preprint cs/0111054*, 2008.

[170] D. Pratas and A. J. Pinho, "Metagenomic composition analysis of sedimentary ancient DNA from the Isle of Wight," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 1177–1181.

[171] A.-K. Lau, S. Dörrer, C.-A. Leimeister, C. Bleidorn, and B. Morgenstern, "Read-SpaM: assembly-free and alignment-free comparison of bacterial genomes with low sequencing coverage," *BMC bioinformatics*, vol. 20, no. 20, p. 638, 2019.