

HYPERDIMENSIONAL COMPUTING

FOR PROTEIN LANGUAGE

MODELING

...

Michael Fatjanov

Student ID: ...

Promotors: Prof. dr. Bernard De Baets, dr. Michiel Stock

Supervisor: Dimitri Boeckaerts

A dissertation submitted to Ghent University in partial fulfilment of the requirements for the degree of master in Bioinformatics (Systems Biology track).

Academic year: 2022-2023

De auteur en promotor geven de toelating deze scriptie voor consultatie beschikbaar te stellen en delen ervan te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting uitdrukkelijk de bron te vermelden bij het aanhalen van resultaten uit deze scriptie.

The author and promoter give the permission to use this thesis for consultation and to copy parts of it for personal use. Every other use is subject to the copyright laws, more specifically the source must be extensively specified when using results from this thesis.

Gent, FILL IN THE DATE

The promotor,

Prof. dr. Bernard De Baets

The author,

Michael Fatjanov

ACKNOWLEDGEMENTS

Thank you, all of you!

CONTENTS

Acknowledgements	i
Contents	iii
Abstract	v
1 Introduction	1
1.1 A historical perspective on bioinformatics	1
1.2 Protein biology	4
1.3 Computational approaches to understand protein biology	5
1.3.1 State-of-the-art approaches	5
1.3.2 Protein language modeling	7
1.4 Aims	9
2 Hyperdimensional computing	11
2.1 Operations on hyperdimensional vectors	12
2.2 Examples	17
2.3 Examples of hyperdimensional computing with real datasets	18
3 Amino acid and protein encoding	25
3.1 Introduction	25
3.2 Encoding single amino acids into hyperdimensional vectors	25
3.3 Encoding proteins into hyperdimensional vectors	27
3.4 Methods	27
3.5 Results	32
3.6 Discussion	37
4 PhaLP case study	39
4.1 PhaLP database	39
4.2 Type classification	40
4.3 Methods	41
4.4 Results	44
4.5 Discussion	51
5 Methods for amino acid-level predictions	53
5.1 Methods	54
5.2 Results	56
5.3 Discussion	59
Bibliography	60
Appendix A Additional information on chapter 3	71
Appendix B Additional information on chapter 5	75

ABSTRACT

insert english summary here...

1. INTRODUCTION

1.1 A historical perspective on bioinformatics

Many decades ago, around the 1950s, we did not know much about the molecules that carry our genetic information and how this would be translated into higher levels of biology. All that was known about deoxyribonucleic acid (DNA) was that it carries nucleotides in equimolar proportions so that there is as much guanine as cytosine and as much adenine as thymine. A major breakthrough came with Watson and Crick's discovery of the structure of DNA in 1953 [1]. Despite that, it took some more decades before the genetic code was deciphered and how this information is further transferred. Researchers came to know that DNA is essentially built up of a linear sequence of the four aforementioned nucleic acids. This sequence encodes information that undergoes transcription into ribonucleic acid (RNA) that in turn gets translated into proteins. The encoding of proteins is done in groups of three nucleic acids at a time, known as codons. All of this was later stated as the *central dogma of molecular biology*, also by Crick in 1958 [2]. This was hugely important for later research since it gives us more insight into how the genetic code is translated and transferred. In the same decade, major leaps were made in the research of protein structure and sequences. The first three-dimensional protein structures were determined via X-ray crystallography [3], which is still mostly the preferred method to this day. On top of that, the arrangement of the primary structure of a protein has been resolved after the first sequencing of a polypeptide. Sanger determined by sequencing insulin in 1953 [4] that a protein is built up of a sequence of amino acids, all connected by a peptide bond into a polypeptide. This established the idea that proteins are biological macromolecules that carry lots of information [5], which started a boom of research on more efficient methods for obtaining protein sequences. The most popular method of that time was the Edman degradation method [6]. A major issue with this method was that only a theoretical maximum of 50 to 60 sequential amino acids could be sequenced. Larger proteins had to be cleaved into fragments that were

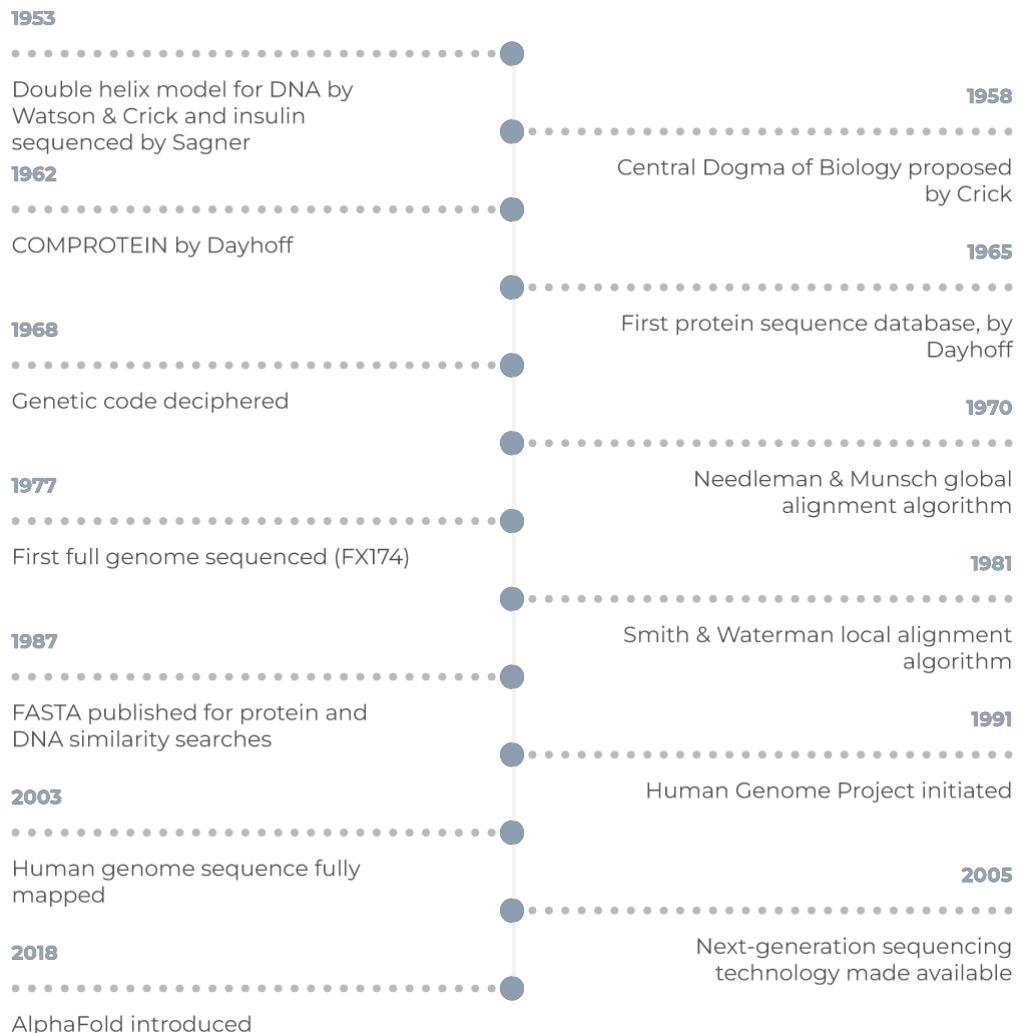


Figure 1.1: A brief timeline of important developments in bioinformatics.

small enough to be sequenced. Tracing back the input sequence from this data was a cumbersome process and thus published Dayhoff the first computational program applied to biological data, COMPROTEIN [7] in 1962. This program was essentially a *de novo* sequence assembler for Edman degradation data. Furthermore, the sequencing amino acids was also increasingly made automated later in the 1960s. These innovations assisted the creation of the first published protein sequence database [8] in 1965.

Following figure 1.1 further into the 1960s, researchers discovered the evolutionary value of having protein sequence data of different species. The problem to be solved back then was the quantification of the similarity between sequences. Pairwise alignment algorithms such as the algorithm of Needleman & Wunsch [9] for global alignments and that of Smith & Waterman [10] for local alignments from the 1970s solved this issue and align-

1. Introduction

ment is still considered to be a key bioinformatics task to this day. Together with this, mathematical frameworks for amino acid substitutions in the context of evolution such as PAMs and BLOSUMs also contributed to bioinformatics. These pairwise alignment algorithms are sufficient for comparing two sequences but unfeasible for searching databases for homologous sequences, hence faster algorithms like FASTA [11], BLAST [12] were developed in the 1980s and 1990s. These methods were and are still important for discovering functional, structural and evolutionary information in biological sequences since sequences that are in some way similar, have a high chance to have the same biological function. This also means that such sequences might be derived from a common ancestor and it became commonplace that sequence patterns may lead to structural and functional relevance. A natural extension of pairwise alignments is multiple sequence alignment (MSA) [13], which is to align multiple related sequences. The most popular and still widely used tools today include Clustal [14] and MUSCLE [15]. This reveals much more information than pairwise alignment can. It allows for the identification of conserved sequence patterns and critical amino acid residues with much more statistical significance which is of great value for constructing phylogenetic profiles of gene families [16]. All of this showed the importance of computational biology and established bioinformatics as a beneficial field of science.

Development of DNA-based applications took some more time since the genetic code and how it translates to amino acids was not deciphered yet until 1968 [17]. Early DNA and RNA sequencing methods were first demonstrated in the 1970s [18, 19] and like with protein sequencing methods, these methods only became faster, more efficient and more scalable. The 1990s saw the appearance of whole genome sequencing and internet-accessible databases that we still use to this day, such as Genbank, Genomes and PubMed. This resulted in advances in computational biology followed the ever-increasing amount of data and need for processing power. With the advent of second-generation sequencing in the 2000s came even more Big Data issues, further challenging bioinformaticians by allowing us to sequence millions of DNA and RNA molecules in a single run. The sheer size and complexity of biological data can make it difficult to store and manage, as for instance implementing error identification, security, quality standards and easy data retrievability. On top of that, the analysis of such large-scale data is not straightforward and traditional software tools won't be sufficient

anymore. Lastly, the need for high-performance computing resources to handle all of these computational demands will only rise.

Today we see that research in the field of biology is becoming more and more computationally driven and that this trend will not slow down any time soon. As for this thesis, we will focus on this matter, more specifically on the level of protein research which is discussed in the next section.

1.2 Protein biology

Proteins are an essential part of molecular biology and are responsible for almost all cellular functions. They are part of the important biological macromolecules that make up life, hence a lot of effort has gone towards trying to understand the functions of protein and disruptions in its mechanisms that lead to many kinds of diseases. Proteins are composed of a linear chain of amino acids (AA) with a length ranging from 50 to tens of thousands of AAs, all connected by peptide bonds into a polypeptide. This is also referred to as the *primary structure* of a protein as mentioned earlier [5]. A sequence of amino acids is mostly determined by the genetic code without considering post-translational and post-transcriptional modifications etc. In the genetic code of all living organisms, there are 20 different kinds of amino acids coded in that make up the 'language' of proteins. Each amino acid has the same backbone but differs by the chemical properties of its side chain, also known as the R-group. This sequence of amino acids does not occur as a mere linear chain of peptides and consists of much more intricacies, however. The polypeptide can form locally folded structures due to chemical interactions within the backbone (the polypeptide chain without the R-group), referred to as the *secondary structure* of a protein. α -helices and β -sheets are the most well-known and common examples of these structures. The overall three-dimensional structure that forms out of these structures is referred to as the *tertiary structure*. These are formed due to interactions between the R-groups and are much harder to classify due to the quasi-infinite amount of different combinations that can occur in an amino acid sequence. And lastly, a protein can also be made up of multiple polypeptide chains, referred to as its subunits. These subunits together form the *quaternary structure* of a protein.

1. Introduction

The sought-after biochemical and cellular functions of a protein emerge from a combination of all of these structures. While a protein's 3D structure and function are dynamic and dependent on its surroundings such as the cellular state and other proteins and molecules, it is still defined by its underlying sequence. This means that a lot of the 3D-structural and functional information of a protein should be retrievable from its amino acid sequence [20]. Understanding how a protein's sequence translates to its structure and function, otherwise known as the 'protein folding problem', is the central problem of protein biology and is crucial for understanding disease mechanisms and designing proteins and drugs for therapeutic and bioengineering applications. Therefore, a lot of effort has gone into computational methods for structure and function predictions from protein sequences but this sequence-structure-function relationship continues to challenge bioinformaticians (insert visual aid). Further in this chapter, we will discuss traditional and state-of-the-art bioinformatics methods to obtain more knowledge in this field.

1.3 Computational approaches to understand protein biology

1.3.1 State-of-the-art approaches

As mentioned earlier, the Edman degradation method was mostly used before to determine the primary structure of a protein. The current state-of-the-art methods for the identification of protein sequences are *de novo sequencing* algorithms applied to tandem mass spectrometry data [21] and allow simultaneous sequencing of thousands of proteins per given sample. Plenty of valuable biological and evolutionary information is already retrievable from just the primary structure *via* traditional tools such as BLAST and MSA as discussed earlier and could provide more information for the prediction of secondary and tertiary structures. To cope with the number of recorded protein sequences rising exponentially, far more compute-efficient methods based on multiple sequence alignments had to be developed like PSI-BLAST [22], HHblits [23] and MMseqs [24]. However, these methods might not be able to keep up with the ever-increasing number of protein sequences stored in databases.

At the other end of the spectrum, the most common way to determine the 3D structure of a protein has remained to be X-ray crystallography for more than half a century [3], with cyro-electron microscopy now catching up rapidly [25]. However, these kinds of laboratory approaches for structure determination of proteins are complex, expensive and in some cases not possible for the protein in question whilst sequence determination is relatively much easier to perform. Because of that, the structures and functions for a large fraction of the approximately 20000 known human proteins remain unknown. The number of verified three-dimensional structures in protein databases consequently has not kept up with the explosive growth in sequence information, further increasing the demand for computational structure/function prediction models.

A lot of methods have been developed to tackle this problem. Until recently, these methods were mainly based on statistical sequence models or physics-based structural simulations. *Ab initio* physics-based approaches such as ROSETTA [26] solve this problem by searching the protein's conformational space using atom energy functions and minimizing the total free energy of the system. ROSETTA has shown to be effective at predicting unknown structures and has been widely used for varying applications, but also assumes simplified energy models, is extremely computationally intensive and has limited accuracies [27]. Statistical sequence modeling of a set of related proteins, on the other hand, has proven to be very useful for discovering evolutionary constraints, homology searches and predicting residue-residue contacts. Improvement of these models has mainly been data-driven; exploiting databases to build large deep learning systems which culminated in the recent success of DeepMind's AlphaFold2 [28] at the Critical Assessment of Protein Structure Prediction (CASP) 14. This success of AlphaFold continued into more recently CASP 15 [29]. Even though, DeepMind did not participate, the most successful participants integrated AlphaFold into their methods. However, all of these models are supervised methods that require labels. Labeled protein structure data essentially means retrieving the 3D coordinates of every atom in a protein which is very labor-intensive and time-consuming. Also, such kind of models would likely perform poorly when working with completely unrelated proteins since the model won't be trained on this kind of data.

One underlying theme of these computational methods for protein structure prediction is being able to translate the protein 'language' into numerical

1. Introduction

representations which computers can learn from. This is now known as the field of protein language modeling.

1.3.2 Protein language modeling

It is intuitive to represent a protein as a sequence of letters with each letter corresponding to an amino acid. As with natural languages, we can find common elements between naturally evolved proteins. Noticeable patterns reoccurring in multiple (related) protein sequences are highly likely to be biologically relevant. These motifs and domains are essential to many biological processes and can easily be represented as words, phrases or sentences of amino acids in a language model perspective. This is why researchers are taking inspiration from the recent successes of natural language processing (NLP) and applying this to a biological context. NLP is a branch of artificial intelligence (AI) concerning itself with creating the ability for computers to learn and understand human languages by using statistical, machine learning and in recent years deep learning models [30]. Common tasks in NLP include part-of-speech tagging (grouping words based on their function in a sentence), named entity recognition (recognizing specific entities in a sentence such as locations, persons, dates etc.) and natural language generation (letter/word prediction).

As with protein modeling, applying labels to millions of natural language containing web pages, articles, journals etc. is a labor-intensive procedure and thus state-of-the-art NLP models use a form of *self-supervised learning*, a form of unsupervised learning in which the context of the text is learned to fill in missing words, predict the next word in a sentence etc. during the training as shown in figure 1.2. This masked learning paradigm involves tokenizing input text into smaller units and converting them into numerical representations, followed by random masking of a certain percentage of tokens in the input sequence. The masked input sequence is then fed through multiple layers of a neural network architecture, which capture different levels of semantic and syntactic information, resulting in contextualized embeddings. Finally, the model can be fine-tuned on a specific task using supervised learning. Through self-supervised masked learning, language models can learn a wide range of linguistic patterns and structures, enabling them to generate meaningful text, answer questions etc. The principles of self-supervised learning and masked learning from NLP can be adapted to the "language" of proteins to learn how a protein will look like.

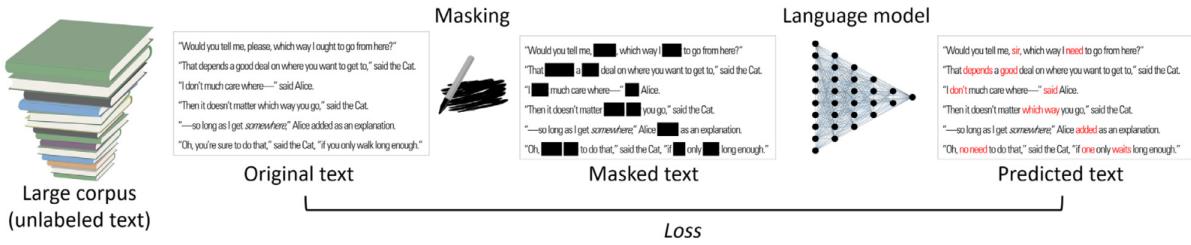


Figure 1.2: Demonstration of a self-supervised masked language learning model. A fraction of the unlabeled text is masked randomly and the language model will attempt to predict these masked tokens. The loss function is a method to assess the performance of the model on its predictions

Well-known NLP methods which adapt some kind of self-supervision include bi-directional long-short term recurrent neural networks (biLSTMs) such as ELMo [31] and more recently transformers such as Google’s BERT [32] and OpenAI’s GPT-3 [33] and GPT-4 [34]. Despite the simplicity of the idea behind self-supervision, such models pre-trained at scale have shown interesting capabilities with very little training on a specific task, now mostly referred to as *few-shot learning*.

In recent years, transformers have become the backbone of many state-of-the-art NLP and protein language models. Transformers are a class of neural network architectures that have revolutionized the field of natural language processing (NLP) since their introduction by Vaswani *et al.* in 2017 [35]. The key innovation of transformers is the self-attention mechanism, which allows the model to weigh the importance of each input token in relation to the others, enabling the efficient handling of long-range dependencies in text. Similar to NLP tasks, protein language models can be pre-trained on large-scale unlabeled protein sequence databases, to learn general features of protein sequences in an unsupervised manner. These pre-trained models can then be fine-tuned on specific protein-related tasks with limited labeled data.

State-of-the-art protein models leverage other advanced machine learning techniques to capture complex patterns in amino acid sequences and predict protein properties. These models employ a combination of sequence alignments, evolutionary information, and NLP-inspired transformer-based architectures to achieve remarkable performance in predicting protein structures and function with the most notable latest projects being TAPE [36], ProtTrans [37] and Meta AI’s ESM-2 [38], one of the most recent and largest protein language models to ever have been developed at the time of writing. ESM-2 has been shown to accurately capture evolutionary information

1. Introduction

and to perform well on structure prediction tasks. It consists of transformer-based protein language models with up to 15 billion parameters trained on 65 million unique sequences.

As with natural language models, a consistent trend has been observed in protein language models wherein increasing the number of parameters and the amount of training data leads to ever-increasing performance. Remarkably, performance does not appear to saturate even with the largest models [30]. However, the computational power required to train such models and perform predictions with them grows exponentially with the number of parameters, reaching a point where modern computers struggle to keep up in a sustainable manner. For instance, training the 175 billion parameter GPT-3 model with 300 billion tokens is estimated to take at least 34 days using 1024 NVIDIA A100 GPUs [39]. While these advancements in computational modeling are undeniably impressive, they come with significant environmental costs in terms of energy consumption and material resources required to support such systems. As researchers continue to research natural language processing, it is crucial to balance the pursuit of ever-larger models with the imperative for sustainable and responsible development.

1.4 Aims

2. HYPERDIMENSIONAL COMPUTING

Hyperdimensional computing (HDC) is a relatively new paradigm of computing in which data is represented and manipulated by high-dimensional (or hyperdimensional) vectors in the range of tens of thousands bit. This framework, developed by Kanerva [40], is inspired by the workings of the human brain and its ability to adapt, learn fast and easily understand semantic relations. The human brain consists of about 100 billion neurons (nerve cells) and 1000 trillion synapses that connect these neurons. Each neuron is connected to up to 10,000 other neurons, creating massive circuits. This is likely fundamental to the workings of the human brain and what separates our brains from modern von Neumann computer architectures which operate on 8 to 64-bit vectors. This becomes clear when we compare the relative simplicity for a human to learn a language compared to computers. Computers use a large and complicated set of arithmetic operations in the form of deep learning networks which require terabytes of data and thousands of Watts of computing power to come close to mastering a language whilst a human can recognize other languages relatively easily when they don't even speak it. Likewise languages, we can very easily memorize and compare other intrinsically complex and contextual concepts such as images. A computer would have a hard time finding similarities between a set of images and faces because this requires very complex machine learning models. The human brain can do this all with a very large efficiency by consuming only roughly 20 W of energy.

Achieving these kinds of flexible brain-like models based on high dimensionality is not entirely new and is being explored since the 1990s. Some of these earlier models include Holographic Reduced Representations [41], Spatter Code [42] and others. A hyperdimensional vector (HDV) can represent anything from a scalar number to any kind of concept. This vector is initially made up of totally random elements, but with a simple set of operations which will be explained later, we can use other vectors to combine some concepts into new similar or dissimilar concepts. For example, to show

the essence of HDC and how it tries to simulate the brain, we can compare the concept of a *table* to the concept of a *broccoli*. We would not immediately conclude that they are in any way similar but as humans, we can trace back *table* to *plate* which has some similarities with *food* from which we can easily extract the concept of *broccoli* as in equation 2.1. These kinds of operations are not very obvious for a classical computer but creating these semantic pathways and recognizing links between distant objects are rather easy for humans.

$$\begin{aligned} \text{table} &\neq \text{broccoli} \\ \text{table} &\approx \text{plate} \approx \text{food} \approx \text{broccoli} \end{aligned} \tag{2.1}$$

The elements in an HDV can be made up of binary bits (values from the set 0, 1) like in classical computing but also of bipolar (values from the set -1, 1) or real numbers. The choice of the nature of the elements has also implications on the nature of the different operations and possibly the results.

An initial HDV is made up fully randomly. This *holistic* or *holographic* representation of a concept smeared out over a vector consisting of thousands of bits gives rise to interesting properties such as its robustness. These kinds of systems are very tolerant to noise and failure of bits since we introduce a lot of redundancy in the vector just by stochastics. This is very unlike classical computing where every bit counts and one failure in a bit can lead to immediate data corruption. Besides its robustness, it also has the potential to show much faster and more efficient calculations than traditional computer systems since it allows for more efficient data storage by encoding multiple objects into a vector.

2.1 Operations on hyperdimensional vectors

The interesting properties of HDC are based on only four basic operations we can perform on HDVs. We will discuss these for bipolar and binary vectors. From here, all implementations are written in the programming language Julia [43] unless noted otherwise. Known for its efficiency, Julia's blend of high-level, interpreter-based features makes it possible to write powerful programs. This is particularly beneficial when working with high-dimensional spaces. Julia's in-built support for bitvectors and bitmatrices, along with parallel computing, will be useful for this thesis. Furthermore, the lively Julia

2. Hyperdimensional computing

community supports a rich ecosystem and a broad array of packages to utilize.

Before the operations are demonstrated, we show how a random hyperdimensional vector could be generated in Julia. In the following code block, functions to generate binary and vectors can easily be made with only one line each.

```
bithdv(N::Int=10000) = bitrand(N) # binary vector  
hdv(N::Int=10000) = rand!(zeros(N), (-1,1)) # bipolar vector
```

Addition

Also referred to as *bundling* or *aggregation*, the element-wise addition as in equation 2.2 of n input vectors $\{X_1 + X_2 + \dots + X_n\}$ creates a vector X that is similar to the input vectors.

$$X = X_1 + X_2 + \dots + X_n \quad (2.2)$$

For bipolar vectors, this entails a straightforward element-wise addition. The resulting vector is restricted to a bipolar nature too depending on the sign of each element, thus containing only -1 , 1 but allowing 0 for elements that are in disagreement as shown in the following 6-dimensional example.

$$\begin{array}{rccccccc} X_1 & = & +1 & -1 & +1 & +1 & -1 & -1 \\ X_2 & = & +1 & +1 & +1 & -1 & -1 & -1 \\ X_3 & = & -1 & -1 & +1 & +1 & -1 & +1 \\ X_4 & = & -1 & -1 & -1 & +1 & -1 & +1 \\ \hline X_1 + X_2 + X_3 + X_4 & = & 0 & -1 & +1 & +1 & -1 & 0 \end{array}$$

For binary vectors, the vectors are element-wise bundled based on the majority element. This is no problem if an odd number of input vectors are considered but ambiguity rises when bundling an even set of vectors. This can be solved by setting the element in question randomly. [44] Another possibility is to add another random vector however this may seem to add more unnecessary noise, especially when bundling a low number of vectors. We can also reverse this by an *inverse addition*. For bipolar vectors, this

means just multiplying the vector of interest by -1. A binary vector can be flipped bit-wise. These kinds of operations are not directly built into Julia, but are easily done as followed:

```
# Adds element-wise bitvectors via based on the majority element.  
# Random if tied.  
# Reduce function takes another function (here .+)  
# and applies it to all given vectors.  
# 'Dotted' operations such as .+ are vectorized  
# making it element-wise.  
function bitadd(vectors::BitVector ...)  
    v = reduce(.+, vectors)  
    n = length(vectors) / 2  
    x = [i > n ? 1 : i < n ? 0 : rand(0:1) for i in v]  
    return convert(BitVector, x)  
end  
  
# Adds bipolar vectors element-wise and rounds off to -1,0 or 1  
# depending on the resulting sign.  
add(vectors...) = reduce(.+, vectors) .|> sign
```

Similar to an ordinary arithmetic summation, the bundling addition of hyper-dimensional vectors is commutative so the result is not dependent on the order of addition.

$$X_1 + X_2 = X = X_2 + X_1 \quad (2.3)$$

Multiplication

Also referred to as *binding*, two vectors can be multiplied element-wise resulting in a vector maximally dissimilar to the input vectors. Vectors X and Y are bound together forming Z being orthogonal to X and Y as shown in equation 2.4.

$$Z = X * Y \quad (2.4)$$

This *binding* operation translates to a simple arithmetic element-wise multiplication for bipolar vectors. For binary vectors, this is represented by a *XOR*

2. Hyperdimensional computing

bit-operation shown as follows.

$$\begin{array}{r} X = \\ \hline Y = \\ \hline X * Y = \end{array} \begin{array}{ccccccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{array}$$

In Julia, there is a built-in function to carry out bit-operations such as *XOR*. For bipolar vectors, a simple element-wise multiplication suffices.

```
# Bind binary vectors
bitbind(vectors::BitVector...) = reduce((x, y) -> xor.(x, y), vectors)

# Bind bipolar vectors
bind(vectors...) = reduce(.*, vectors)
```

The binding operation can also be undone by multiplying with the same vector again. It is its own inverse so that:

$$A * A = O \text{ where } O \text{ is a vector containing only 0s} \quad (2.5)$$

Likewise an ordinary multiplication, this operation is commutative and distributive over additions, meaning that transforming a bundle of concepts with binding is equivalent to binding every element before bundling.

$$A = Z * (X + Y) = XZ + YZ \quad (2.6)$$

Permutation

The permutation operation of an HDV, also known as *shifting*, is a simple reordering of the HDV. This can be random but a circular shift is widely employed [45] and makes the operation easily reversible. This results in a vector technically dissimilar from the input vector but still encoding its information. This will become important later when it will be used to encode sequential information such as tokens in a text. This operation will be denoted by Π .

$$\begin{array}{r} X = \\ \hline \Pi(X) = \end{array} \begin{array}{ccccccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

In Julia, this line can do it for both binary and bipolar vectors:

```
# Permute by applying a circular shift  
perm(vector::Vector, k::Int=1) = circshift(vector, (0, k))
```

Similarity measurement

For many kinds of problems, it will be necessary to quantify the similarity between two HDVs. The method depends on the nature of the vectors. For binary vectors, the *Hamming distance* defined as in equation 2.7 is widely used.

$$Ham(A, B) = \frac{1}{d} \sum_{i=1}^d 1_{A(i) \neq B(i)} \quad (2.7)$$

The *cosine distance* as defined in equation 2.8 is most commonly used for bipolar vectors.

$$\cos(A, B) = \frac{A \cdot B}{\|A\| * \|B\|} \quad (2.8)$$

Both are not built-in Julia, but can be programmed as followed:

```
# Hamming distance  
hamming(x::BitVector, y::BitVector) = sum(x .!= y)/length(x)  
  
# Cosine similarity  
cosine(x::Vector, y::Vector) = dot(x, y) / (norm(x) * norm(y))
```

The results of both of these measurements are summarized in table 2.1.

Table 2.1: Overview of similarity measurements in HDC depending on the nature of the HDVs

Measurement	Dissimilar	Orthogonal	Similar
Hamming distance	1	0.5	0
Cosine similarity	-1	0	1

It is important to note that two random HDVs will be quasi-orthogonal to each other just by stochastics. Also notice that the first quantifies a distance and the latter a similarity.

2.2 Examples

There are many interesting possibilities given the relative simplicity of all these operations. In this section, we will demonstrate the power of hyperdimensional computing with some simple examples.

Simple example with simulated data

To get a feel for the operations, assume A, B, C, X, Y and Z to be random 10,000-dimensional bipolar hypervectors and that $D = X * A + Y * B + Z * C$, let us then try to retrieve A from D by using the defined operations. We generate for A, B, C, X, Y and Z each a random 10,000-D vector. To retrieve an approximation of A , D can be multiplied by X and the rest of the included vectors are then regarded as noise as done in equations 2.9. Because of the robustness of hyperdimensional vectors, a lot of information of A should still be contained within D .

$$\begin{aligned}
 A' &= X * D \\
 &= X * (X * A + Y * B + Z * C) \\
 &= \underbrace{X * X * A}_{A} + \underbrace{X * Y * B + X * Z * C}_{\text{noise}} \\
 &\approx A
 \end{aligned} \tag{2.9}$$

The procedure of equation 2.9 is repeated 10,000 times because of the stochastic nature of these vectors. The results are illustrated in figure 2.1. We see that we can retrieve a lot of information with most of the Hamming distances centering around 0.25. Notice that two completely random HDVs would have a distance close to 1 just by stochastics. A Hamming distance of 0 would mean that we retrieved all bits of A correctly, which is impossible in this model due to the consideration of noise. Although we work with a very constrained model, vector A is roughly 75 % retrievable and the calculations are very efficient as all of these can be completed in less than 2 seconds on a laptop. This same experiment was done with random bipolar 10,000-dimensional vectors and it performs slightly slower as expected but retained the accuracy.

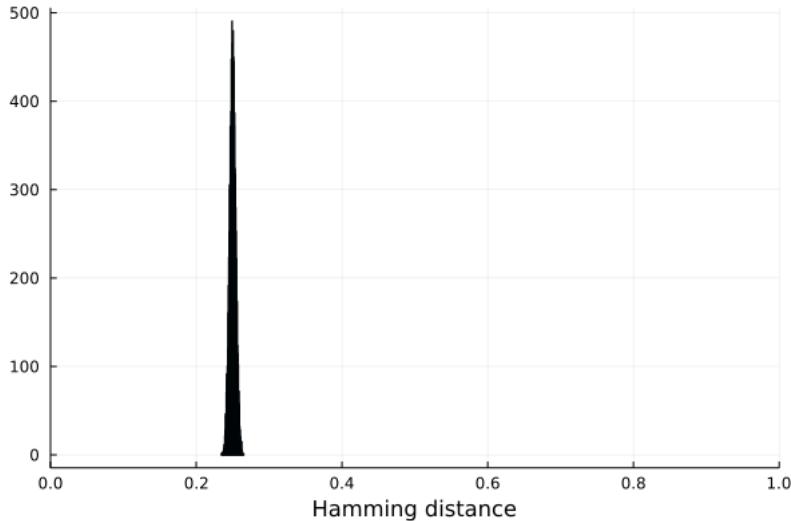


Figure 2.1: 10,000 cases of random 10,000-dimensional binary vectors are made and each time implemented following equation 2.9. The resulting Hamming distances between A and A' are then plotted in a histogram.

2.3 Examples of hyperdimensional computing with real datasets

Now, the power of these simple operations will be demonstrated by applying them to a couple of relatively small real datasets.

Zoo animal classification

As the first example, we will consider a simple dataset[46] containing 101 animals with 17 descriptors such as their number of legs, their skin covering and other physical properties. In the end, we want to create a simple model that can classify these animals and other animals that are not present in the dataset based on their descriptors. To tackle this problem, we first assign to each descriptor a random hyperdimensional vector. For each animal, all of its features can be bundled to obtain a final vector representing the animal. For example, it is known that a chicken lays eggs, is covered with feathers and has two legs so then these features can be bundled as in the following equation. C is a vector representing a chicken, E the ability to lay eggs, F the possession of feathers and T the possession two legs:

$$C = E + F + T \quad (2.10)$$

2. Hyperdimensional computing

This is simple for all the binary features but the feature for the number of legs is variable. Although it is possible to assign completely random vectors to each number of legs, it would make a slightly more biologically realistic model if an animal with 2 legs would be more similar to one with 4 legs than to one with 8 legs. To address this, a range of numbers would have to be representable by hyperdimensional vectors, the range from 0 to 8 in this case. First, a random hyperdimensional vector representing the lower bound of the interval is generated. Next, a vector representing the next step in the interval is constructed by replacing a fraction of the vector with random bits. This last step is then repeated to obtain a vector of each number in a range.

In biology, it is possible to find higher order of concepts which are combinations of directly observable characteristics. For example, an animal could lay eggs or be dependent on mother's milk, but (almost) never both. So, the growth and development of an animal depends on these characteristics. This property can be easily implemented into this HDC model by binding a HDV of a higher order concept to the descriptor HDV. So as said previously, the 'milk' (M), and 'egg' (E) features give us information about the growth of the animal, so we will bind these features to another vector representing the growth feature (G) to obtain a more expanded model. This is also done for the skin protection features (S) and all the features considering the limbs (L). This also gives us the possibility to retrieve some features of the animals as in the procedure shown in the previous example. Thus, equation 2.10 can be expanded into:

$$C = G * E + S * F + L * T \quad (2.11)$$

After conducting the various procedures, the data set now consists of 101 animals, each represented by a 10,000-dimensional feature vector. To efficiently analyze and visualize this high-dimensional data, principal component analysis (PCA) has been employed to reduce the dimensionality from 10,000 to just 2. By projecting these vectors onto a 2D space, the data can be conveniently displayed in a two-dimensional plot, as illustrated in Figure 2.2. Upon examination of the plot, three distinct and meaningful clusters of animal classes emerge, which align well with our understanding of evolutionary relationships. These clusters include: (1) mammals, (2) a group consisting of birds, reptiles, and amphibians, and (3) a cluster encompassing invertebrates and insects. While the current analysis already provides valuable insights into the relationships between different animal classes, further improvement could be achieved by incorporating additional features

that may lead to a clearer separation of these groups. By doing so, the model would be able to more accurately distinguish between the different animal classes and provide a more comprehensive understanding of their evolutionary connections.

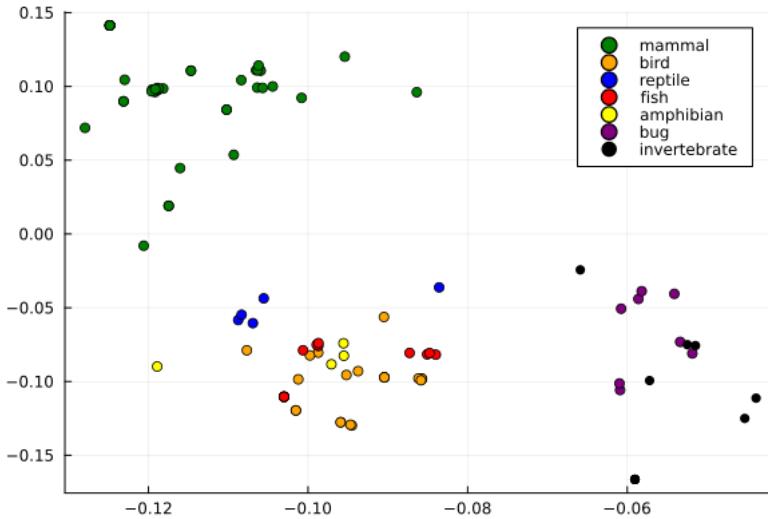


Figure 2.2: Scatter-plot of the first two principal components (PCs) of a $101 \times 10,000$ matrix containing hyperdimensional vectors for every animal in the zoo dataset after a PCA procedure. These PCs account for roughly 48 % of the variance.

After an HDV has been made for every animal, all animals of the same class can be bundled to obtain an HDV representing the said class. So for example, if we have a hyperdimensional vector for a pigeon (P), chicken (C) and a kiwi (K), an HDV representing birds (B) can be made by doing:

$$B = P + C + K \quad (2.12)$$

To classify an animal, its HDV can be compared to the HDVs of every class and the most similar vector is then assumed to be its class. The dataset was stratified and split into a training set (comprising 80 % of the sequences) and a test set. With 100 runs, it could predict the class on average 94 % of the test animals. For further improvement, it would be possible to generate a set of animals not present in the dataset and test those in order to further understand how this model can be improved. On top of this, it would also be possible to generate a confusion matrix to understand where we could use more distinguishing descriptors. From the PCA, we could already predict that reptiles and amphibians would be easily confused, as for invertebrates and bugs.

Protein classification

To illustrate an example more akin to this research topic, a model based on the principles of hyperdimensional computing will be built to classify a protein sequence dataset[47]. It contains 949 manually curated peptide sequences with their membranolytic anti-breast cancer activity level (very active, moderately active, experimentally inactive and virtually inactive). The virtually inactive peptides are predicted to be inactive. The model will be built with mostly the same procedure as for the animal classifier, but instead of animals, sequences have to be encoded into HDVs now. First, a random HDV is generated for every amino acid. Physicochemical properties, evolutionary constraints etc. could be introduced to make this model more realistic but that is not necessary for this demonstration. Next, a peptide sequence is to be considered as a bag of trimers as seen in figure 2.4. A vector representing a trimer is generated by binding the three amino acids whilst retaining sequential information by shifting as in equation 2.13. All retrievable trimers from a given sequence are then bundled together, forming a vector representative of the sequence.

$$ABC = A * \Pi(B) * \Pi(\Pi(C)) \quad (2.13)$$

From here on, the same procedures as in the last example can be applied here too, so all HDVs of a class are bundled for further analysis. The PCA procedure did not generate interesting results because the two first principal components explained only 5 % of the variance, aside from the clear clustering of the predicted inactive peptides. This means that it is not feasible to reduce the 10,000 dimensions of the vectors to 2, likely because the information is too smeared out over the vectors. This occurrence is highly dependent on the training data. Nevertheless, this follows the philosophy of hyperdimensional computing in keeping holistic representations of concepts.

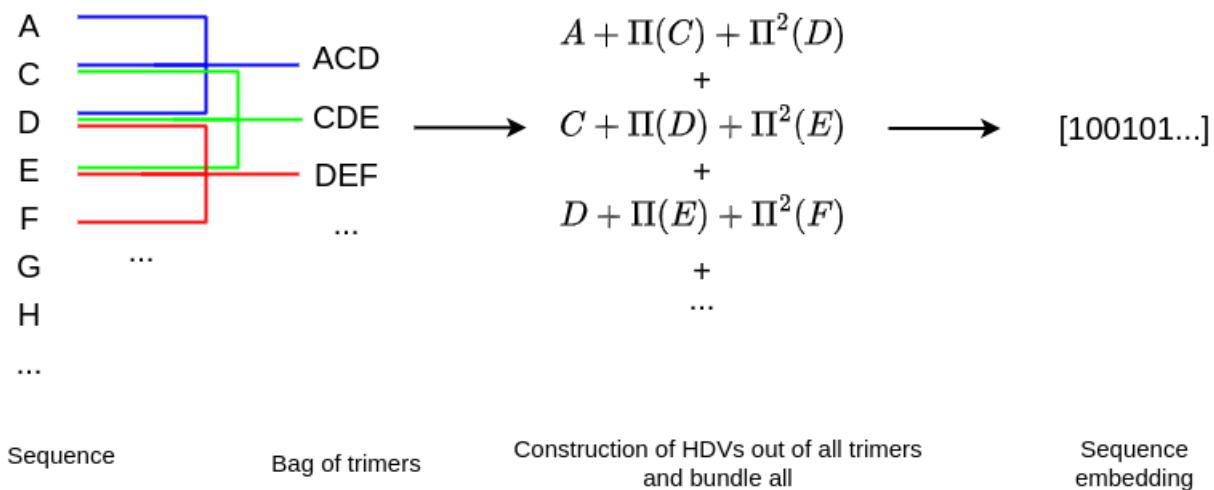


Figure 2.3: Overview of operations done to obtain an HDV of a protein sequence for the example concerning the real peptide dataset. First, a random HDV is generated for every amino acid. Next, a peptide sequence is to be considered as a bag of trimers. A vector representing a trimer is generated by binding the three amino acids whilst retaining sequential information by shifting as in equation 2.13. All retrievable trimers from a given sequence are then bundled together, forming a hyperdimensional vector representative of the sequence.

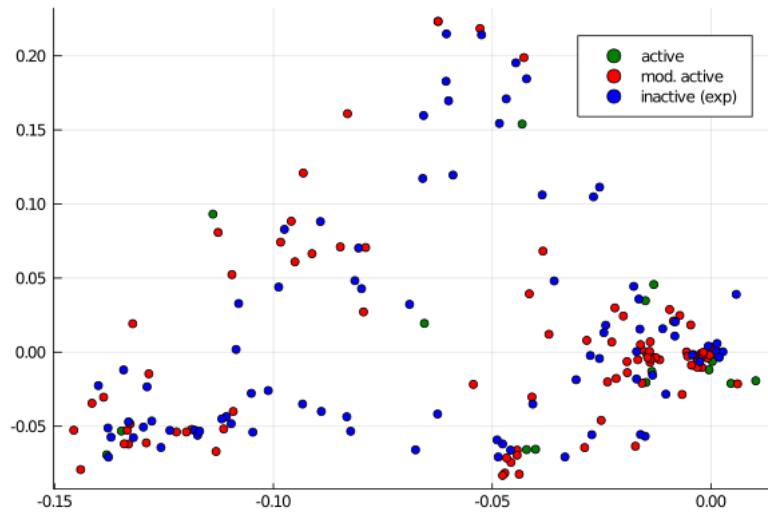


Figure 2.4: Scatter-plot of the first two principal components, decomposing the 10,000-dimensional vectors for every peptide into 2. These PCs account for roughly 5 % of the total variance

Next, a classifier was made correspondingly. The dataset was stratified and split into a training set (comprising 80 % of the sequences) and a test set. With 100 runs, it could predict the class on average 85 % of the test sequences. It has to be taken into account that the predicted inactive peptides account for 80 % of the sequences of the dataset, thus this model performs slightly better than if we would predict at random. There are many

2. Hyperdimensional computing

possible improvements to be made however, such as using more suitable performance metrics, introducing similarities between amino acids instead of setting them randomly and using more suitable frameworks for our protein language models, which will all be research topics further on in this project.

3. HYPERDIMENSIONAL COMPUTING METHODS TO ENCODE AMINO ACIDS AND PROTEIN SEQUENCES

3.1 Introduction

To research the possibilities of the hyperdimensional computing framework applied to protein language modeling, we introduce a rudimentary pipeline. First, amino acids would have to be encoded into hyperdimensional vectors. These vectors for every separate amino acid could be then further encoded into vectors representing a full protein. And lastly, these embeddings could be then utilized to perform predictions such as classifications as shown in figure 3.1. The first two steps of this pipeline will be discussed further in this chapter.

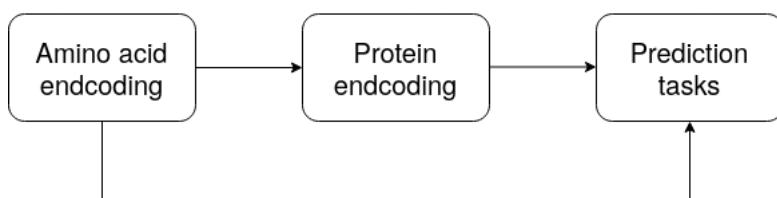


Figure 3.1: A simple demonstration of our workflow. First, we encode single amino acids into hyperdimensional vectors. With these, protein sequences could be encoded into hyperdimensional vectors. Lastly, both kinds of embeddings are then used for prediction tasks.

3.2 Encoding single amino acids into hyperdimensional vectors

Currently, in most of the research in hyperdimensional computing, there is an emphasis on creating and assigning hyperdimensional vectors to certain

concepts at random. This is useful for optimizing speed and efficiency and is not a problem for many prevalent research areas such as natural language processing, where, for example, it is typically assumed that a letter's similarity to other letters in the alphabet remains constant. However, hyperdimensional computing sees minimal application in the context of biological data. Some notable studies include HDNA which assigns random hyperdimensional vectors to DNA bases to encode DNA sequences and BioHD [48] which encodes protein sequences using random mRNA hypervectors to use in genome sequence searches. In the context of DNA sequencing, the assumption of assigning random hyperdimensional vectors may not present as significant an issue as with proteins. This is largely due to the distinct nature of DNA bases as they do not display the same degree of physicochemical diversity that amino acids in proteins do. DNA bases serve primarily as the information carrier in the biological system, with their sequence dictating the sequence of amino acids in proteins. In this role, the relevance of their physicochemical properties to their informational role is rather limited in comparison to amino acids, which main function is to dictate the structure and function of the resulting protein. Whilst there are phenomena in DNA structures such as base stacking and hydrogen bonding that play a role, the simple base-to-vector mapping often suffices.

When considering protein language modeling, however, this assumption might be suboptimal since some amino acids are chemically more similar to each other than to others which is crucial to the protein's structure and function. We can already estimate physicochemical distances between amino acids based on their physicochemical properties [49] such as volume, polarity, chemical groups etc. *via* many kinds of distance measures. The different similarities between amino acids are tied into the structure and thus function of amino acid sequences and shape our view of protein language. It explains why some amino acid substitutions can result in almost no phenotypical changes or on the other hand detrimental changes. Proteins have evolved to maintain their structure and function, and drastic changes in physicochemical properties can disrupt these characteristics. Therefore, amino acid substitutions that preserve the physicochemical properties of the original amino acid are more likely to be selected, resulting in a negative correlation with physicochemical distance. To account for this, we experimented with several methods to encode physicochemical distance into amino acid hypervectors.

3. Amino acid and protein encoding

Besides encoding amino acids as building blocks, it is also useful to encode amino acids in the context they are found in. State-of-the-art protein language models have the ability to gather information on long-range dependencies around a single amino acid and encode this information into neural networks and in dense numerical vectors. These models are very powerful, but as discussed earlier very resource-intensive too. To investigate the possibilities of developing contextual embeddings on the level of amino acids, we propose a novel encoding technique within the hyperdimensional computing framework.

3.3 Encoding proteins into hyperdimensional vectors

In section 2.3, we introduced the bag-of-words (BoW) method of embedding sequences in hyperdimensional space which has been extensively applied to cases related to natural language processing in a hyperdimensional computing context. We also introduce a novel sequence embedding method in hyperdimensional computing.

3.4 Methods

From here on, every hyperdimensional vector is made to be 10,000-dimensional and binary unless noted otherwise.

Encoding single amino acids into hyperdimensional vectors

Extended ESM-2 embeddings

The last layer of the 3 billion-parameter ESM-2 model [38] of every amino acid was extracted, resulting in 1024-dimensional real-valued embeddings for every amino acid. To project these into hyperdimensionality, a simple matrix multiplication has been employed:

$$A_{1 \times 1024} \times B_{1024 \times 10,000} = C_{1 \times 10,000}$$

Where A is a 1024-dimensional ESM-2 embedding and B a matrix of 1024 random 10,000-D vectors. The resulting vectors are then min-max scaled and rounded depending on the desired nature of the vectors. To visually assess these, the vectors for each amino acid are reduced in dimensionality via PCA into 2 dimensions. This has been compared to a PCA of the unaltered 1024-dimensional ESM-2 embeddings.

Real-valued projected ESM-2 embeddings

It is possible to step away from binary hyperdimensional vectors, which are typically used in hyperdimensional computing research, and allow the vectors to be real-valued. This gives us the possibility to store more data in a vector and utilize more complex mathematical operations with the cost of losing efficiency of bit-operations. We generated random real-valued vectors with random values in [-1,1]. Real-valued ESM-2 embeddings were generated by the same procedure as above, but without the rounding.

Enforcing similarities using genetic algorithms

Instead of utilizing embeddings coming from other large protein language models, we also experimented with encoding predetermined target pairwise distances onto initially random hyperdimensional vectors. First, a suitable matrix with predetermined pairwise distances has to be considered. This also implies that the matrix has to be symmetric. If we then consider the 20 essential amino acids, the problem at hand would involve a set of 20 binary vectors of length 10,000 to conform to a target distance matrix based on Hamming distance. This can be classified as a combinatorial optimization problem as it involves searching for an optimal or near-optimal configuration of binary vectors that satisfy a specific criterion. To minimize the difference between the target and actual Hamming distances for all pairs of binary vectors, we could adjust the vectors by randomly bit-flipping them until they meet the desired criteria. However, the search space in this problem is vast ($2^{10,000 \times 20}$), making exhaustive search methods computationally infeasible. Thus, more efficient algorithms are needed to solve this problem such as genetic algorithms [50] (GAs). GAs, a subfield of evolutionary algorithms, draw inspiration from the process of natural selection and emulate the evolutionary mechanisms of crossover, mutation, and selection to explore a vast search space and converge toward an optimal or near-optimal solution.

3. Amino acid and protein encoding

A genetic algorithm was implemented with the *Evolutionary.jl v0.11.1* [51] package in *Julia* to encode target similarities. The primary steps of a genetic algorithm include:

- Initialization: random candidate solutions are initiated with a given population size.
- Crossover: combine genetic material offspring of two parents (in this case vectors). There are many recombination techniques such as single-point, multi-point and uniform crossover.
- Mutation: randomly alterate genes (in this case bits) to explore other possibilities of configurations and prevent premature convergence due to local optima.
- Evaluation: Each individual in the population (in this case a set of vectors) is assessed using a fitness function, which measures how well the solution solves the given problem.
- Selection: individuals from the population are selected based on their fitness to create a mating pool. Fitter individuals have a higher probability of being selected, mimicking the concept of survival of the fittest in natural evolution.

These steps are reiterated over a number of generations to obtain a set of vectors that correspond to the best fitness. A *BLOSUM62* substitution matrix [52] and Grantham's distance matrix [53] were considered as target pairwise similarity matrices for the GA. These were then normalized to obtain the target pairwise Hamming distances. The fitness is determined by the sum of the squared differences between the computed distance matrix of an individual and the target distance matrix. The lower the fitness value, the more optimal the individual. The population size was set to 25000 and the number of generations to 250. The mutation rate was set to 0.15 and the crossover rate to 0.2, these are high and low respectively compared to more commonly used parameters because we want to emphasize the bit-flipping and avoid recombination of vectors for faster convergence.

Contextualized neighborhood-encoding of amino acids

To encode the neighborhood of an amino acid in a sequence, all possible pairwise interactions with the central amino acid in question in a given win-

dow are made *via* binding and then all encoded into one vector *via* bundling as shown figure 3.2. Our neighborhood-encoder was tested on the human reference proteome in UniProt, entry *UP000005640*, containing 20591 proteins. We started with real-valued random vectors and real-valued projected ESM-2 vectors for every amino acid. Windows of $n = 4$ and $k = 50$ were considered resulting in 4 different experiments. Every single residue in the human reference proteome was encoded with information within the k-range window and an average vector was made for every amino acid. For all 20591 peptides in the reference proteome, this procedure took only 3 hours for $n = 4$, but upwards to 15 hours for $n = 50$ on a high-performance computing cluster (HPC). After all amino acids were encoded, an element-wise average was made for every amino acid. The resulting hyperdimensional vectors were kept to a real-numbered nature to not lose information for illustrative purposes. Principal component analysis was then done for these 4 experiments as seen in figure 3.6.

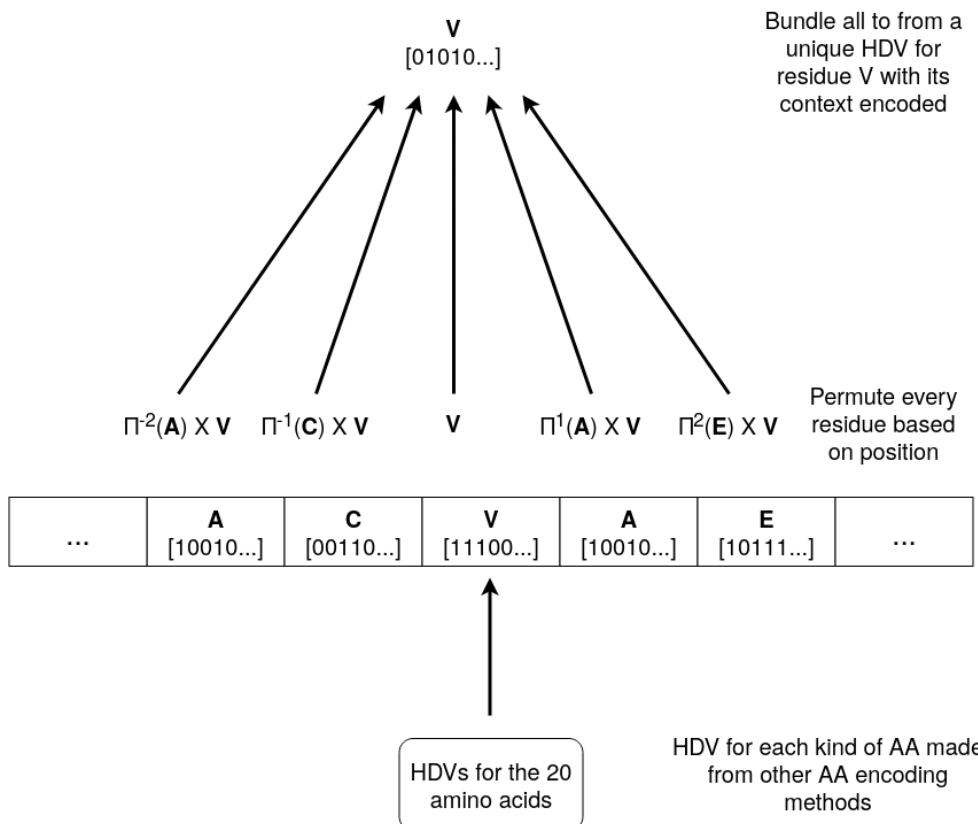


Figure 3.2: A simple demonstration of our amino acid encoder. First, HDVs are generated for every kind of amino acid, this can be done using just random HDVs, projected ESM-2 embeddings, GA-optimized vectors etc. It considers an amino acid and all amino acids in a predetermined neighborhood (here $n = 2$). It produces all possible interactions of the central amino acid in the window by binding and then bundles all the pairwise interactions into one hyperdimensional vector that represents the central amino acid.

Encoding proteins into hyperdimensional vectors

We implemented two different algorithms to encode amino acid vectors into vectors representing proteins. We already introduced the BoW-method in section 2.3, fully explained in figure 2.3. We also introduce a novel sequence embedding method in hyperdimensional computing. It is similar to the bag-of-words method in the sense that it bundles vectors of k-mers, but here, the k-mer's positional information will be encoded into the k-mer by permuting it before bundling as seen in figure 3.3, similarly to how a convolutional layer in a neural network operates so we will name it the convolutional embedding method from here on. These methods will be extensively utilized and tested in chapters 4 and 5 where several case studies consisting of protein sequences datasets are tackled.

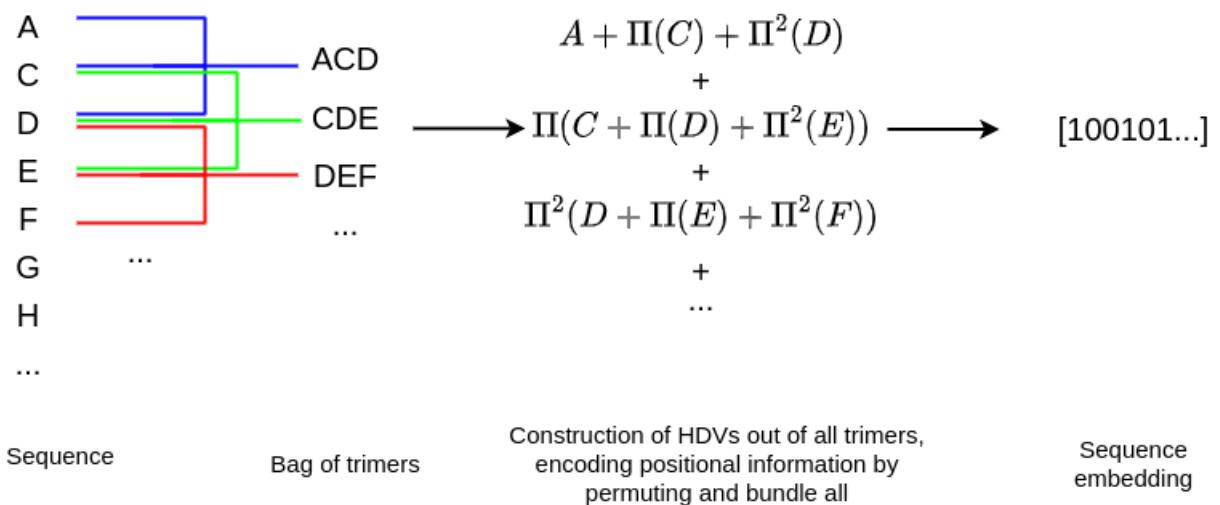


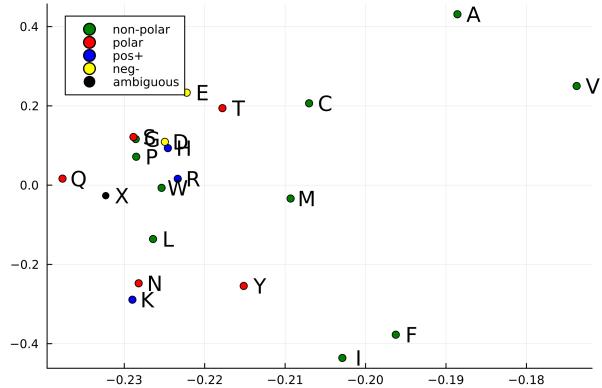
Figure 3.3: Overview of operations done to obtain an HDV from a sequence using the convolutional embedding method. First, an HDV is generated for every amino acid. Next, a peptide sequence is to be considered as a bag of trimers. A vector representing a trimer is generated by binding the three amino acids whilst retaining sequential information by shifting as in equation 2.13, just as in the BoW-method (figure 2.4). On top of that, every trimer will also be positionally encoded via a permutation operation. All retrievable trimers from a given sequence are then bundled together, forming a hyperdimensional vector representative of the sequence.

3.5 Results

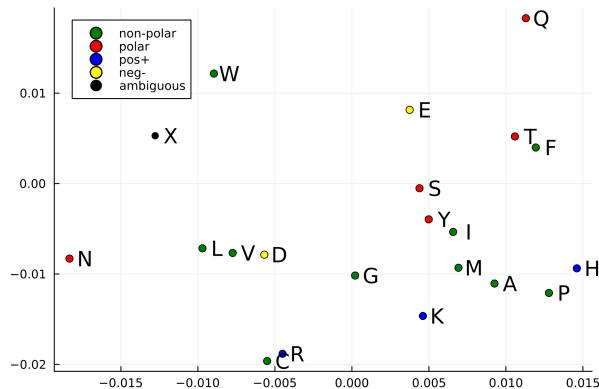
Extended ESM-2 embeddings

Visually, no interesting patterns can be deducted from the PCA decompositions of the ESM-2 embeddings. Yet, if we also perform a principal component analysis on random vectors, we can see there is significantly more variance encoded into the first two principal components of the ESM embeddings (22 %) compared to random vectors (10.5 %, can deviate slightly depending on the run), meaning that there should be a significant amount of similarity encoded into the hyperdimensional vectors. PCA decomposition of our projected ESM-2 embeddings can't fully retrieve the variance in the hyperdimensional vectors as well as in the unaltered ESM-2 embeddings, as to be expected. The performance and usefulness may be shown more clearly when tested in other computational biology problem settings.

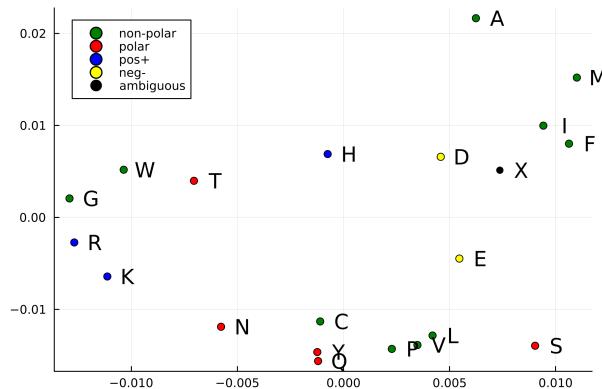
3. Amino acid and protein encoding



(a) Unaltered 1024-dimensional ESM-2 embeddings



(b) ESM-2 embeddings projected into binary hyperdimensionality



(c) Real-value projected ESM-2 embeddings.

Figure 3.4: Scatter-plots of the first two principal components of ESM-2 embeddings for amino acids. (a) Unaltered 1024-dimensional ESM-2 embeddings, accounting for roughly 79 % of the total variance. (b) ESM-2 embeddings projected into hyperdimensionality, accounting for roughly 22 % of the total variance. (c) real-valued projected ESM-2 embeddings, accounting for roughly 28.5 %. In all subfigures, amino acids are annotated and colored based on their chemical property of polarity.

Real-valued projected ESM-2 embeddings

The PCA of the real-valued projected ESM-2 vectors, as seen in figure 3.4c, appears to capture a greater proportion of the variance compared to their

binary counterparts. This increased detected variance is likely attributed to the reduction in data loss that occurs when avoiding the rounding off of values. The effectiveness of these real-valued projected ESM-2 vectors in various tasks will be further investigated and evaluated in subsequent analyses.

Enforcing similarities using genetic algorithms

The outcomes of employing a genetic algorithm to infer our target distances can be found in the tables located in section A of the appendices. These are visually presented as heatmaps in figure 3.5. Taking into account that two random hyperdimensional vectors would be quasi-orthogonal just by stochastics, their Hamming distances would be close to 0.5. Hence, one could see that despite our efforts, our obtained distances remain significantly different from the desired values and still indicate quasi-orthogonality, which led us to discontinue further exploration of these vectors. This discrepancy is likely attributable to the vast search space associated with the problem, resulting in substantial computational time required to reach the target matrix. The optimization process ran for approximately 58 hours on a high-performance computing (HPC) system. Unfortunately, utilizing a genetic algorithm does not appear to be a viable method for optimizing our vectors within an acceptable timeframe.

3. Amino acid and protein encoding

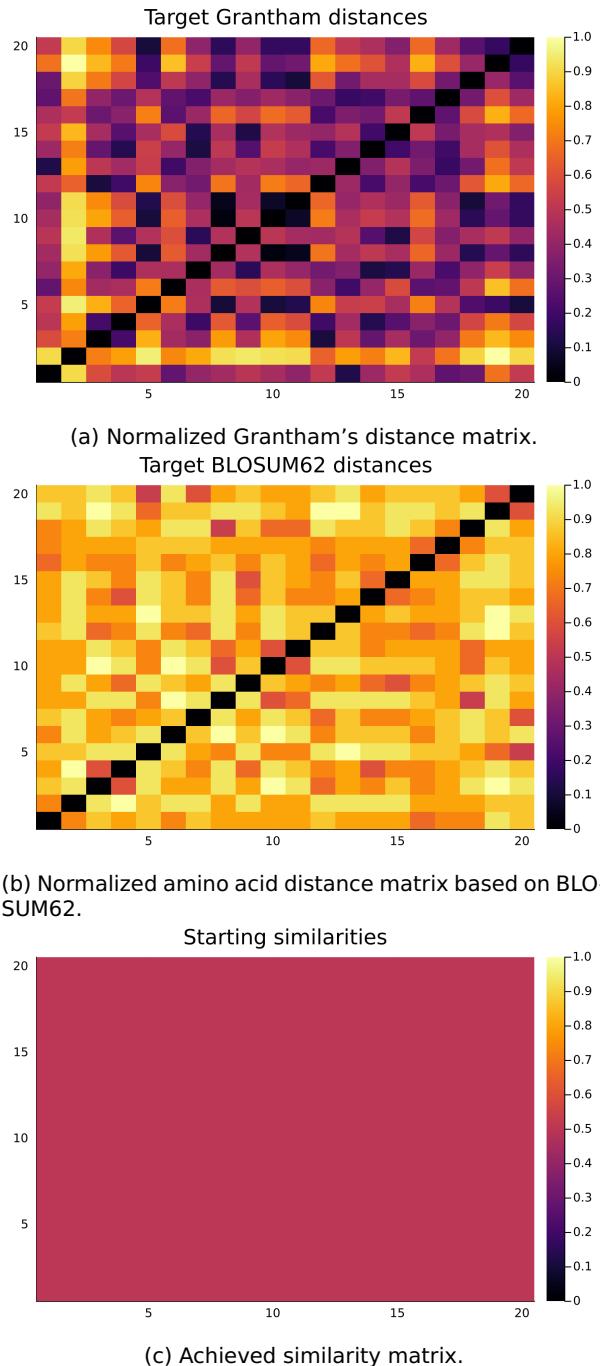


Figure 3.5: Heatmap of the target pairwise amino acid distances, based on (a) Grantham's distance matrix (b) BLOSUM62 matrix and heatmap of (c) the achieved distances.

Contextualized neighborhood-encoding of amino acids

Looking at the average neighborhood-encoded amino acids starting from ESM-2 embeddings and the human proteome (figures 3.6a and 3.6b), the charged amino acids seem to be grouped vertically by PC 2, roughly dividing the polar and non-polar amino acids, albeit slightly more pronounced for

$n = 50$. Also interesting, to see very similar groupings in the PCA plots might indicate that the first four amino acids before and after a residue are the most crucial.

The PCA scatter plots generated from random vectors (figures 3.6c and 3.6d) show slightly different results as compared to the ones starting from ESM embeddings. These PCAs also capture less variance, which may indicate that encoding biological information/similarities into the vectors beforehand might be useful. For $k = 4$, some typicalities are noticed such as the close grouping of the negatively charged amino acids (D and E) and the consistent triangular grouping of H, L and R. Nonetheless, for $k = 50$ with random vectors, we see results deviating from all the PCA scatter plots of other neighborhood-encoded amino acids, which could mean that a PCA is not able to capture the sparse information for this case or that these vectors might be oversaturated.

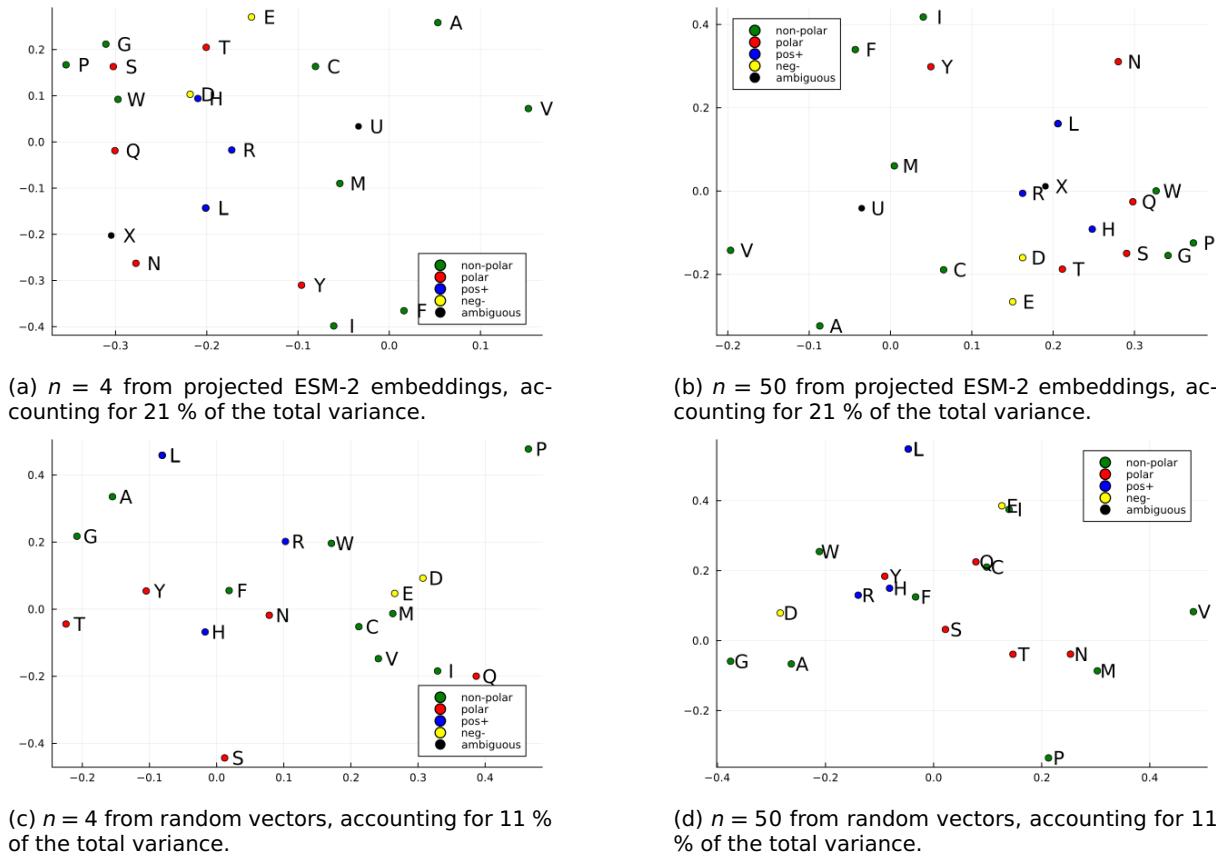


Figure 3.6: Scatter-plots of the first two principal components of the average amino acid HDVs with neighborhood-information of $n = 4$ and $n = 50$ encoded, learned from the human reference proteome. The top row represents encoded amino acids starting from projected ESM-2 embeddings, and the bottom row represents encoded amino acids starting from random hyperdimensional vectors. The amino acids are annotated and colored based on their chemical property of polarity.

3.6 Discussion

In Rives *et al.* (2011), they conducted a similar experiment with their transformer model [?]. Our results are comparable to theirs, but not as cleanly grouped which is likely due to several factors. For instance, due to the intrinsic stochastic nature of hyperdimensional computing, it is prone to capture some amount of noise. On top of this, it is difficult for dimensionality-reduction methods such as PCA to accurately capture the intricacies of 10,000-dimensional vectors into 2 dimensions. UMAP projections were also made for the resulting vectors of the neighborhood-encoder method, shown in appendix A, but these did not reveal any new information. For specifically the neighborhood-encoder method, the amount of data we used is not comparable to theirs: our method learned from less than 21000 sequences whilst their model was trained on 250 million sequences. They also included sequences originating from all recorded organisms in the UniProt database at that time whilst we confined ourselves to human sequences. Secondly, hyperdimensional vectors have a limited capacity, meaning long-range dependencies of a residue will saturate the hyperdimensional vector depending on the range.

We can predict the angle between a bundled vector and a randomly selected vector from said bundle vector by $\Theta = \arccos\left(\frac{2^k}{k}\right)/2^{2k}$ with $2k + 1$ equal to the number of sequences in the class [54]. This approximation is valid for bipolar/binary vectors in hyperdimensions (≥ 10000). This equation also suggests that an increase in dimensions will not influence the angle. Evaluating this equation by considering random 1001 vectors in a bundled vector, so $k = 500$, results in an angle of 88.6° . This indicates that a vector has a limited capacity: the more vectors we bundle together, the closer the angle will be to 90° and thus the more dissimilar the bundled vector becomes to its components. This results in the bundled vectors not being representative anymore of a given dataset. This equation assumes that the bundled vector is a bundle of purely random vectors which is not the case for most of our embeddings; however, it provides us a rough idea about the bundling capacity of a hyperdimensional vector warranting further research in this domain.

In the forthcoming two chapters, we will delve into practical case studies that draw upon the methodologies expounded in this chapter, engaging with protein sequence datasets and executing predictive tasks therein. Chapter

4 is focussed on exploring classification tasks employing protein sequence embeddings, wherein we will research the power of hyperdimensional computing, whether or not together with established machine learning methods. In chapter 5, we shift our focus toward the classification tasks on amino acid vectors by using our neighborhood-encoder method to create contextualized embeddings of amino acids and neural networks to classify these.

4. PHALP CASE STUDY: HYPERDIMENSIONAL PROTEIN SEQUENCE EMBEDDING AND BINARY PROTEIN-LEVEL CLASSIFICATION

4.1 PhaLP database

To implement and evaluate hyperdimensional computing in typical problem settings in computational biology, the potential of hyperdimensional computing will be evaluated on the PhaLP dataset [55] in this chapter. PhaLP is a comprehensive database currently comprising more than 17000 entries of phage lytic proteins, including much of their information such as their type, domains and tertiary structures. Phage lytic proteins are used by bacteriophages to infect bacterial cells. To cross the bacterial cell walls, phages use two different types of phage lytic proteins: virion-associated lysins (VALs) and endolysins. Phage lytic proteins also comprise one or more functional domains categorized into two classes: enzymatically active domains (EADs) and cell wall binding domains (CBDs) [56].

The escalating global antibiotic resistance crisis has necessitated the development of alternative strategies to combat bacterial infections [57]. One such promising alternative is enzybiotics, a class of enzyme-based antibiotics derived from phage lytic proteins. Phage lytic proteins are produced by bacteriophages during their lytic replication cycle and are responsible for breaking down the bacterial cell wall. As these proteins exhibit a high level of diversity, it is critical to make well-informed selections during the early stages of research and development. In response to this need, Criel *et al.* introduced PhaLP [55], a comprehensive, automatically updated, and easily accessible database containing more than 17,000 phage lytic proteins.

PhaLP aims to serve as a portal for researchers, allowing them to access all relevant information about the current diversity of phage lytic proteins through user-friendly search engines. This database is specifically designed to facilitate the development and application of enzybiotics by providing a wealth of data on protein architecture, evolution, and bacterial hosts corresponding to the phages. PhaLP not only serves as a valuable starting point for the broad community of enzybiotic researchers but also offers continually improving evolutionary insights that can act as a natural inspiration for protein engineers. By enabling researchers to make well-considered selections of phage lytic proteins during the early stages of their projects, PhaLP plays a significant role in the development of highly effective, narrow-spectrum antibiotics. These enzybiotics have the potential to revolutionize the field of antibacterial agents, offering a much-needed response to the alarming threat of antibiotic resistance that plagues healthcare systems worldwide.

To fully utilize the rich content of PhaLP, the researchers conducted a series of analyses at three levels to gain insights into the host-specific evolution of phage lytic proteins. First, they provided an overview of the modular diversity of these proteins. This was followed by the adoption of data mining and interpretable machine learning approaches to reveal host-specific design rules for domain architectures in endolysins. Lastly, the evolution of phage lytic proteins at the protein sequence level was explored, uncovering host-specific clusters.

In this chapter, we will explore the authors' experiment on protein classification based on sequence data and evaluate the potential of hyperdimensional computing in the context of protein functional annotation. The primary objective is to better understand the role of hyperdimensional computing in protein classification and to elucidate its potential advantages over conventional machine learning techniques.

4.2 Type classification

The developers of PhaLP aimed to classify protein sequences based on their type, with a focus on two types of phage lytic proteins: virion-associated lysins (VALs) and endolysins. Both of these proteins play crucial roles in the lytic replication cycle of bacteriophages, as they help the viruses breach the bacterial cell wall. VALs are an integral component of the viral particle,

4. PhaLP case study

and their primary function is to create a small pore in the peptidoglycan layers of the bacterial cell wall at the infection site. This process allows the bacteriophage to gain access to the interior of the bacterial cell, initiating the lytic replication cycle. On the other hand, endolysins are produced within the infected bacterial cell and act toward the end of the replication cycle. These enzymes degrade the peptidoglycan layer of the bacterial cell wall, leading to cell lysis and the release of new viral particles [55].

Only a fraction of the database is manually annotated to include the protein's type because the amount of phage lytic proteins whose type is described in the literature is relatively small. The developers of PhaLP resorted to a machine learning approach for the classification of unannotated sequences. The authors embedded each protein sequence *via* SeqVec [58] and trained a random forest classifier [59] with 100 estimators and balanced weights to classify the proteins whose types were unknown. For this case study, we attempted to simulate their experiments of classifying the proteins into the two types based on their sequence using several techniques based on hyperdimensional computing.

4.3 Methods

As of March 2023, the latest version of the PhaLP database, *v2021_04*, has been used to test our models. This dataset consists of 17356 unique amino acid sequences of phage-lytic proteins.

Embedding of sequences into hyperdimensional vectors

First, we used the two sequence encoding techniques as discussed in section 3.4 to embed the protein sequences in hyperdimensional space. These methods have been applied to all sequences in the dataset that were manually annotated on their type using both random hyperdimensional vectors and projected ESM-2 embeddings. These were then visually assessed *via* PCA.

Classification of hyperdimensional protein sequence embeddings

To discriminate, we experimented with 3 different methods: the rudimentary hyperdimensional computing classification as seen in section 2.3, an XG-Boost classifier and by A. Hernandez-Cano *et al.* [60]. As in the PhaLP study, we use all non-ML annotated proteins in the dataset. Out of the 11549 unambiguous UniParc accessions in the newest version of the database, 4829 are manually annotated on their type. Out of these manually annotated proteins, 2803 are endolysins and 2026 are VALs.

Naive additive method

As a baseline level, we use the rudimentary HDC classification technique as seen in section 2.3: the HDVs of sequences of the same class are bundled to construct single HDVs representative of every class. Then, a sequence's class is inferred by comparing the sequence's HDV to both class HDV *via* a similarity measure based on the assumption that the class vector is maximally similar to its components. This model was evaluated *via* a stratified 10-fold cross validation.

XGBoost classifier

The baseline hyperdimensional classification method has been compared to a more established method, the XGBoost classifier [61]. The classification with an XGBoost classifier is done *via* the default XGBoost classifier from *XGBoost.jl* v2.2.5 using the hyperdimensional embeddings as an input and is evaluated *via* a stratified 10-fold cross validation with *MLJ.jl* v0.19.5.

OnlineHD methods

Another method we experimented with, is OnlineHD by A. Hernandez-Cano *et al.* [60]. It is an algorithm that expands on the classical hyperdimensional training methods by trying to eliminate model saturation. Instead of naively bundling vectors on top of each other, this algorithm assigns weights to every addition depending on how much new information it adds to the model to prevent class vector saturation. To train the model, assume a new

4. PhaLP case study

data point \vec{V} with label l and class vectors \vec{C}_i with each having a label i . The cosine similarity of \vec{V} with every class vector is then calculated as \cos_i . If \vec{V} with an actual label l would have been predicted as l' , the class vectors will be updated as followed (with learning rate η as a tunable hyperparameter):

$$\vec{C}_l \leftarrow \vec{C}_l + \eta(1 - \cos_l) * \vec{V} \quad (4.1)$$

$$\vec{C}_{l'} \leftarrow \vec{C}_{l'} - \eta(1 - \cos_{l'}) * \vec{V} \quad (4.2)$$

This means that if a new data point is highly dissimilar to its class vector and thus contains a high amount of new information, the weight of the update ($\eta(1 - \cos_l)$) will increase. The information is then also subtracted from the incorrectly predicted class vector. If a label would be correctly predicted for a new data point, the model will not be updated to avoid saturation. To initialize the model, the first vector of a class to be assessed is assumed to be the class vector. For example, if a query vector in our dataset is predicted to be a VAL, but its actual label is endolysin, the class vectors are then updated as follows:

$$\vec{C}_{VAL} \leftarrow \vec{C}_{VAL} + \eta(1 - \cos_{VAL}) * \vec{V} \quad (4.3)$$

$$\vec{C}_{endo} \leftarrow \vec{C}_{endo} - \eta(1 - \cos_{endo}) * \vec{V} \quad (4.4)$$

Due to the nature of this model, we cannot constrict our hyperdimensional embeddings to a bipolar or binary nature anymore and the embeddings are thus allowed to be real-numbered. Mathematical operations such as multiplications and additions are then assumed to be element-wise. On top of the single-pass method as discussed above, A. Hernandez-Cano *et al.* also implemented an iterative retraining algorithm to increase the accuracy of OnlineHD. This starts from the class vectors made *via* the single-pass OnlineHD model, but assesses the class vectors by performing inference with every training vector. If a training vector's label is wrongly predicted, equations 4.1 and 4.2 are then used to update the model. This entire sequence is iterated over a given amount of cycles.

Since these algorithms are only available as *PyTorch* implementations, implementations in *Julia* have been made here. A stratified 10-fold cross val-

idation of these models with our subject sequences has been performed. The learning rate is set at 0.035 for both the single-pass and the iterative method and the amount of retraining iterations for the iterative method is set to 120 as these are the values set by default in their *PyTorch* package. To test these algorithms, real-numbered embeddings had to be made from our subject sequences. So instead of embedding the sequences starting with random bitvectors, random vectors with values in interval $[-1, 1]$ for both the random amino acid vectors and projected ESM-2 embeddings were made to obtain real-valued hyperdimensional vectors for every amino acid. The same bag-of-words and convolutional approaches have been applied here to obtain real-valued sequence embeddings. These were assessed *via* a scatter-plot of the two first principal components of their PCA projection.

4.4 Results

Embedding of sequences into hyperdimensional vectors

There is no visual difference between the PCA plots for the sequences embedded *via* the bag-of-words method and the convolutional method, but there is a clear difference between the different starting embeddings as seen in figure 4.1. The binary sequence embeddings from both the bag-of-words method and the convolutional method made with ESM-2 AA embeddings seem to capture more of the variance between the sequences.

As with the binary embeddings, PCA of the real-valued sequence embeddings shows no distinction between those made *via* the bag-of-words method and the convolutional method whilst showing a clear difference between the different starting embeddings as seen in figure 4.3. Similarly, the explained variance is much higher for embeddings made with projected ESM-2 embeddings. However, the PCA method retrieves much more information from real-valued embeddings. From all the PCA scatter plots mentioned, we can see that VAL proteins are generally much more diverse in sequence than endolysins.

4. PhaLP case study

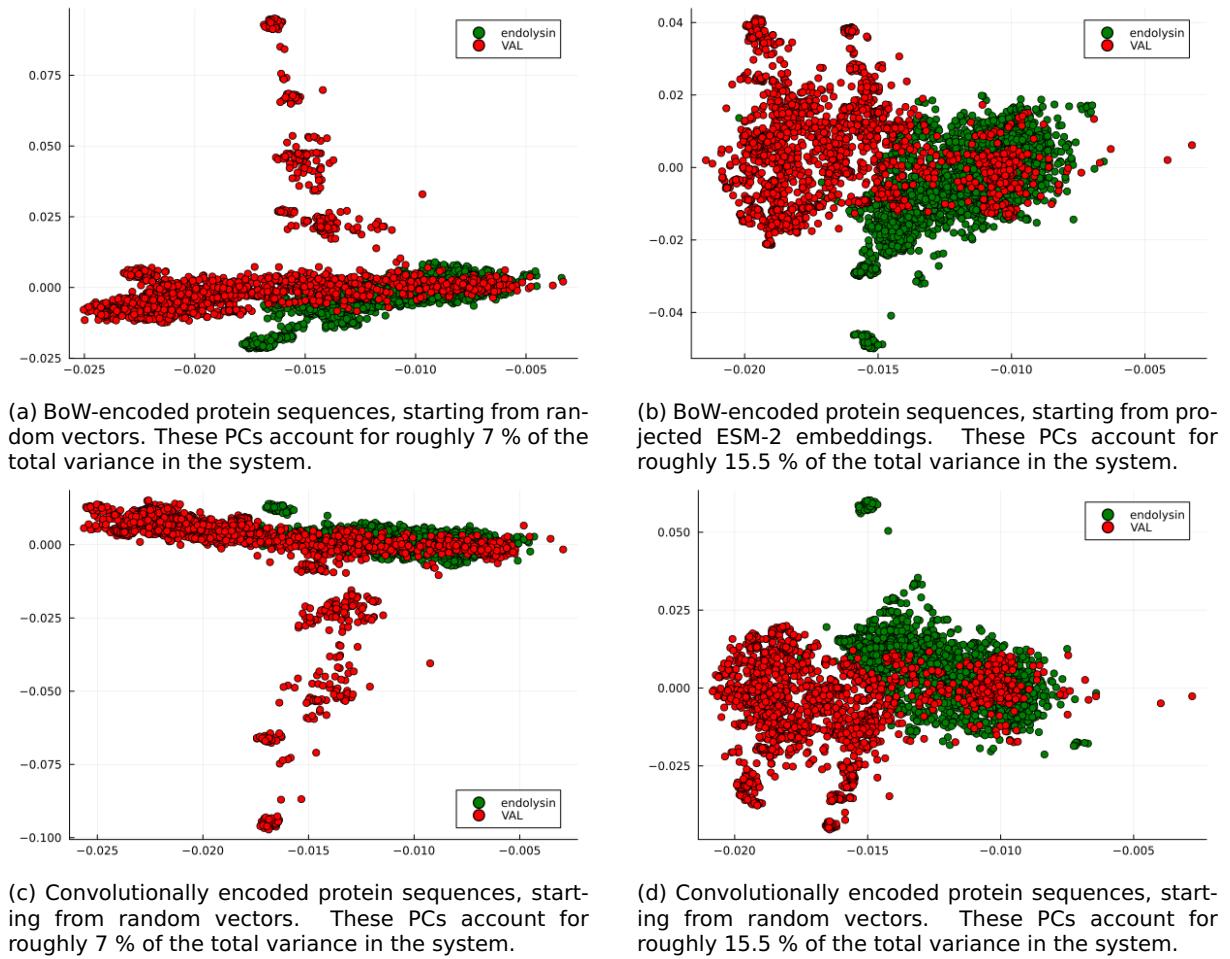
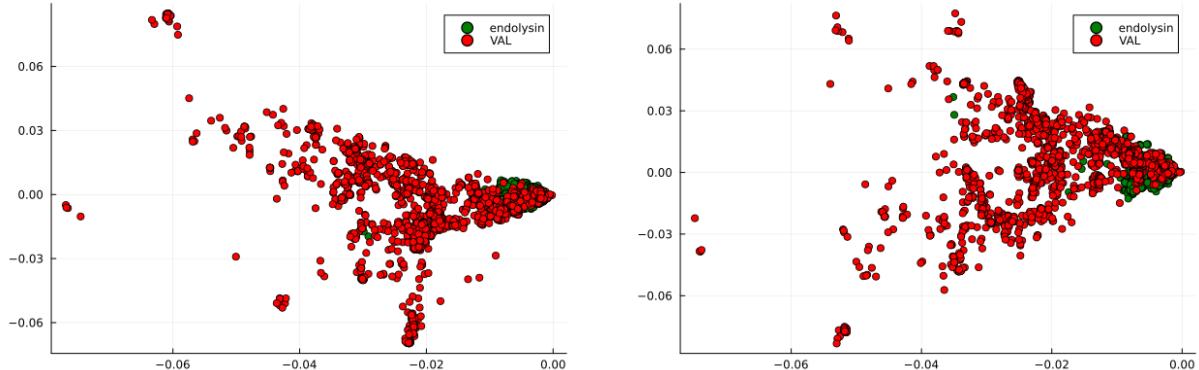


Figure 4.1: Scatter-plot of the first two principal components of the binary encoded phage lytic proteins *via* the BoW-method starting from (a) random HDVs per amino acid (b) projected ESM-2 amino acid embeddings and *via* the convolutional method from (c) random HDVs per amino acid (d) projected ESM-2 amino acid embeddings. Only manually annotated phage lytic proteins were considered and are color-coded based on their type.

Classification of hyperdimensional protein sequence embeddings

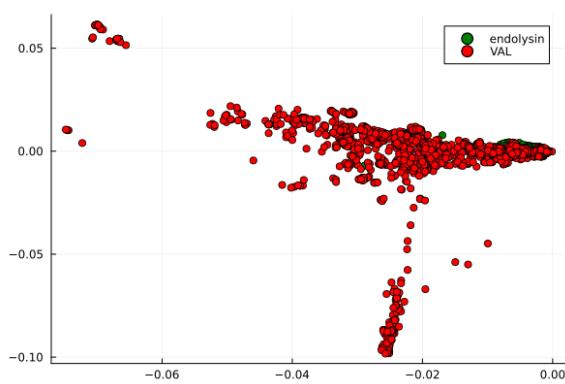
The resulting F1-scores from our 4 classification methods are listed in table 4.1. Evaluating our naive additive model using stratified 10-fold cross-validation results in F1-scores of around 0.14 for every kind of amino acid embedding.

The scores of the XGBoost classifier with our sequence embeddings are much more comparable to the results of the experiment in the PhaLP paper, with the convolutional embeddings generally performing better than the bag-of-words embeddings and also the ESM 2-based performing better than random base vectors when considering the type of amino acid vectors used. This is an indication that hyperdimensional computing can pro-

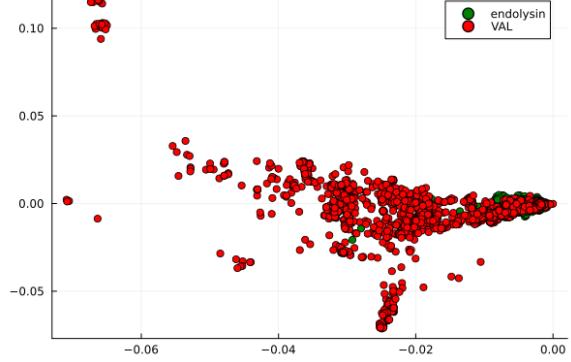


(a) BoW-encoded protein sequences, starting from random vectors. These PCs account for roughly 30.4 % of the total variance in the system.

(b) BoW-encoded protein sequences, starting from projected ESM-2 embeddings. These PCs account for roughly 57.3 % of the total variance in the system.



(c) Convolutionally encoded protein sequences, starting from random vectors. These PCs account for roughly 30.4 % of the total variance in the system.



(d) Convolutionally encoded protein sequences, starting from projected ESM-2 embeddings. These PCs account for roughly 57.3 % of the total variance in the system.

Figure 4.2: Scatter-plot of the first two principal components of the real-valued encoded phage lytic proteins *via* the BoW-method starting from (a) random HDVs per amino acid (b) projected ESM-2 amino embeddings and *via* the convolutional method from (c) random HDVs per amino acid (d) projected ESM-2 amino embeddings. Only manually annotated phage lytic proteins were considered and are color-coded based on their type.

vide a very fast and reliable method of embedding protein sequences, even without prior biological information. The drawback of this machine learning model, which is to be expected from every gradient-based model, is that training and predictions take much longer to compute compared to hyperdimensional training models. The cross validation procedure took up to 5 minutes on a consumer-grade laptop, whilst with the naive additive approach, the procedure took less than 10 seconds to finish.

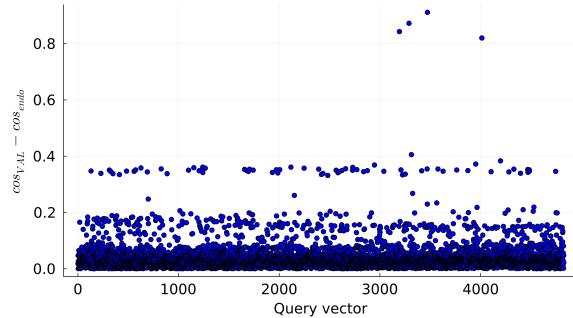
Comparing the results of the OnlineHD models to the naive additive HDC model, we can see a substantial increase in the performance of this model. The single-pass model seems to have more widely varying results depending on the type of embeddings used. As opposed to the XGBoost classifier, it performs better using bag-of-words embeddings. The model also performs better when trained with embeddings based on projected ESM-2 vectors.

4. PhaLP case study

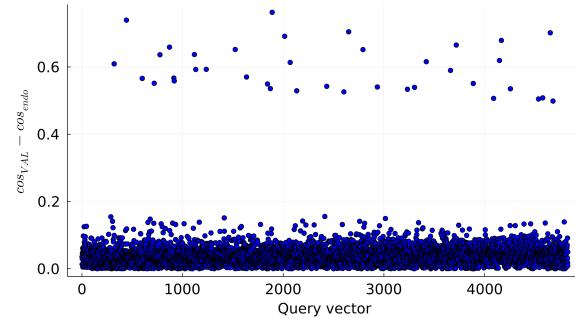
Table 4.1: Results of type classifications using the principal classification technique of hyperdimensional computing, an XGBoost classifier and OnlineHD implementations with several kinds of embeddings

F1-scores	BoW/random	BoW/ESM	Convolutional/random	Convolutional/ESM
Naive addition	0.1458	0.1468	0.1461	0.1461
XGBoost classifier	0.9667	0.9754	0.9661	0.986
Single-pass OnlineHD	0.8901	0.9214	0.7793	0.8400
Iterative OnlineHD	0.9487	0.9757	0.9486	0.9670

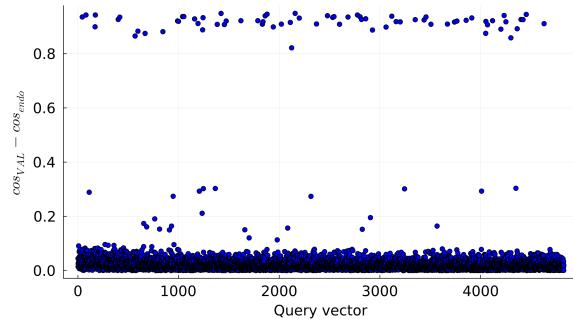
Iterative retraining of the model seems to increase its performance significantly, even coming close to the performance of an XGBoost classifier in this case. Further improvement might be found when optimizing the models' parameters. The cross validation procedure takes less than 10 seconds to run for the single-pass model, whilst doing an iterative retraining of the model adds 2 to 3 minutes. This model appears to be a decently performing extension of the rudimentary hyperdimensional classification model for protein classification, whilst still being much more efficient than the commonly used machine learning models. The drawback of the model is that we cannot use hyperefficient bit-operations anymore, which limits its efficiency compared to the binary nature of the additive model.



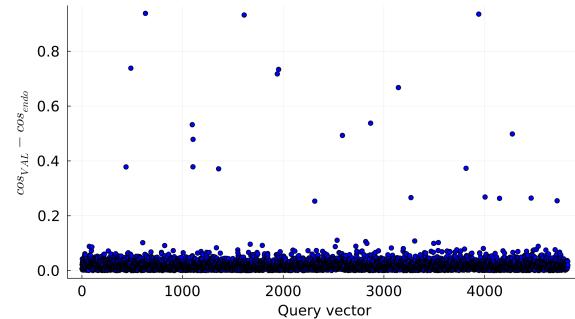
(a) Embeddings using random amino acid vectors, made via the bag-of-words embedding method



(b) Embeddings using projected ESM-2 amino acid vectors, made via the bag-of-words embedding method



(c) Embeddings using random amino acid vectors, made via the convolutional embedding method



(d) Embeddings using projected ESM-2 amino acid vectors, made via the convolutional embedding method

Figure 4.3: Absolute difference between the cosine similarity of the query vector with the VAL class vector and the cosine similarity of the query vector with the endolysin class vector while following the OnlineHD single-pass training procedure. Trained using hyperdimensional embeddings from all protein sequences in the PhaLP dataset whose types were manually annotated.

To further research the power of OnlineHD, we aim to monitor its training procedure when applied to the PhaLP dataset. We calculate the absolute difference between the cosine similarities, \cos_{VAL} and \cos_{endo} , as described in equations 4.3 and 4.4. By observing this metric, we can gain insights into the behavior of the hyperdimensional vectors in the OnlineHD model during training. A possibility to expect is that the difference in cosine similarities increases throughout the training procedure as the model learns to differentiate the two classes. However, these differences don't increase noticeably increase in this case.

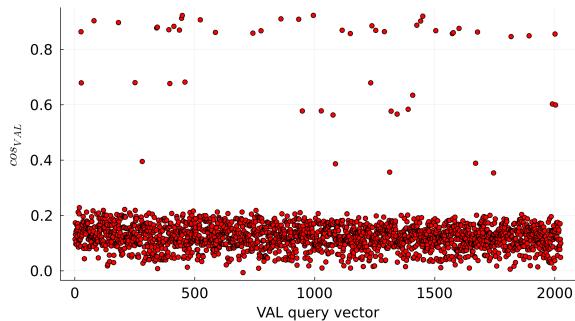
Another approach to monitor the training procedure is to track the two class vectors $\vec{C_{VAL}}$ and $\vec{C_{endo}}$ individually. For each query vector with a known label l , we calculate the cosine similarity \cos_l . This method allows us to observe the model's adaptation and refinement of its internal representation of the two classes before it predicts a query vector during the training process. The distance between a query vector and a class vector is proportional to the update weight, so we expect the cosine similarities to increase on average as the model learns. The results are shown in figure 4.4.

In this case, the VAL updates with relatively consistent weights, meaning that we have a diverse dataset. In general, we see a higher variation in cosine similarities for VAL vectors, which suggests that virion-associated lysins may have more diverse sequences compared to endolysins. We also notice higher overall similarities between the query vectors and the VAL vector when using the BoW-method with projected ESM-2 vectors compared to other embedding methods, which aligns with this model's superior performance.

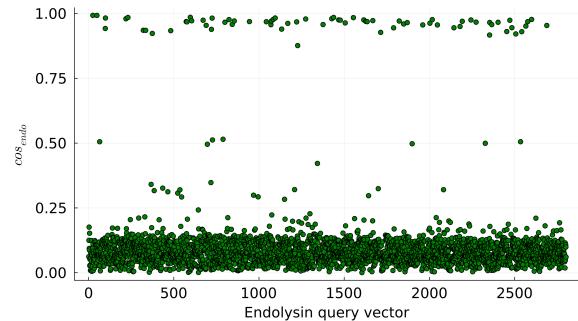
The endolysin vector appears to be updated at a constant rate too, with less variation in similarities, indicating that endolysins may have less diverse sequences. As observed with the VAL vectors, the similarities between the query vectors and the endolysin vector are higher when the embeddings are created using the BoW-method with projected ESM-2 vectors.

Instead of looking at the inner workings of the OnlineHD method, the training progress of an OnlineHD model in terms of performance has been monitored. The dataset is divided into training and testing sets, while ensuring stratification is preserved, with the test set comprising 10 % of the data. The training data is further partitioned into 500 batches, and the model is trained incrementally using these batches. After each batch is processed,

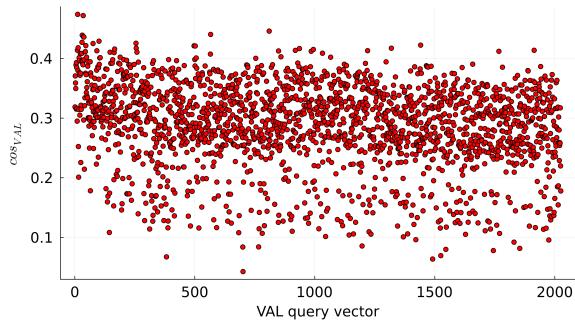
4. PhaLP case study



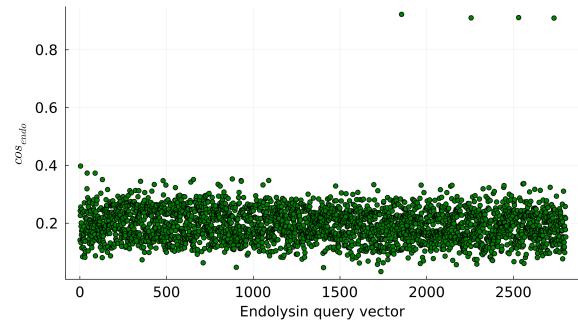
(a) VALs embedded using random amino acid vectors via the bag-of-words method



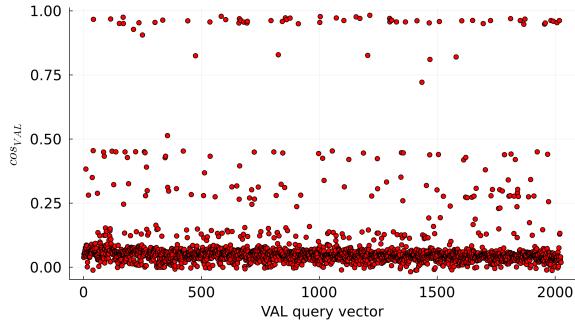
(b) Endolysins embedded using random amino acid vectors via the bag-of-words embedding method



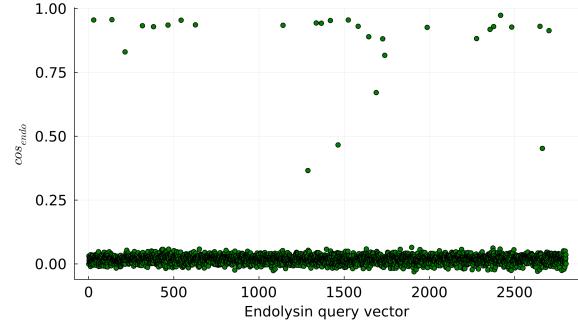
(c) VALs embedded using projected ESM-2 amino acid vectors via the bag-of-words embedding method



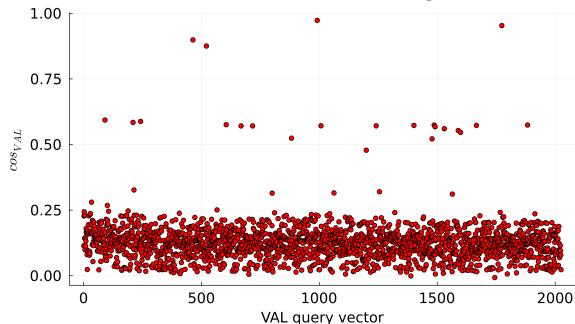
(d) Endolysins embedded using projected ESM-2 amino acid vectors via the bag-of-words embedding method



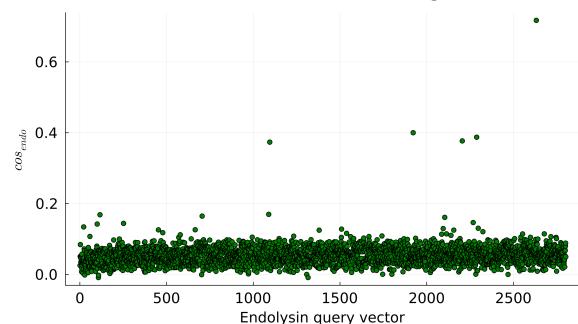
(e) VALs embedded using random amino acid vectors via the convolutional embedding method



(f) Endolysins embedded using random amino acid vectors via the convolutional embedding method



(g) VALs embedded using projected ESM-2 amino acid vectors via the convolutional embedding method



(h) Endolysins embedded using projected ESM-2 amino acid vectors via the convolutional embedding method

Figure 4.4: Cosine similarity between a query vector and the class vector during OnlineHD single-pass training procedure. Trained using hyperdimensional embeddings from all protein sequences in the PhaLP dataset whose types were manually annotated, red representing VALs and green representing endolysins.

the model's performance on the test data is assessed to provide insights into its learning trajectory with the results shown in figure 4.5.

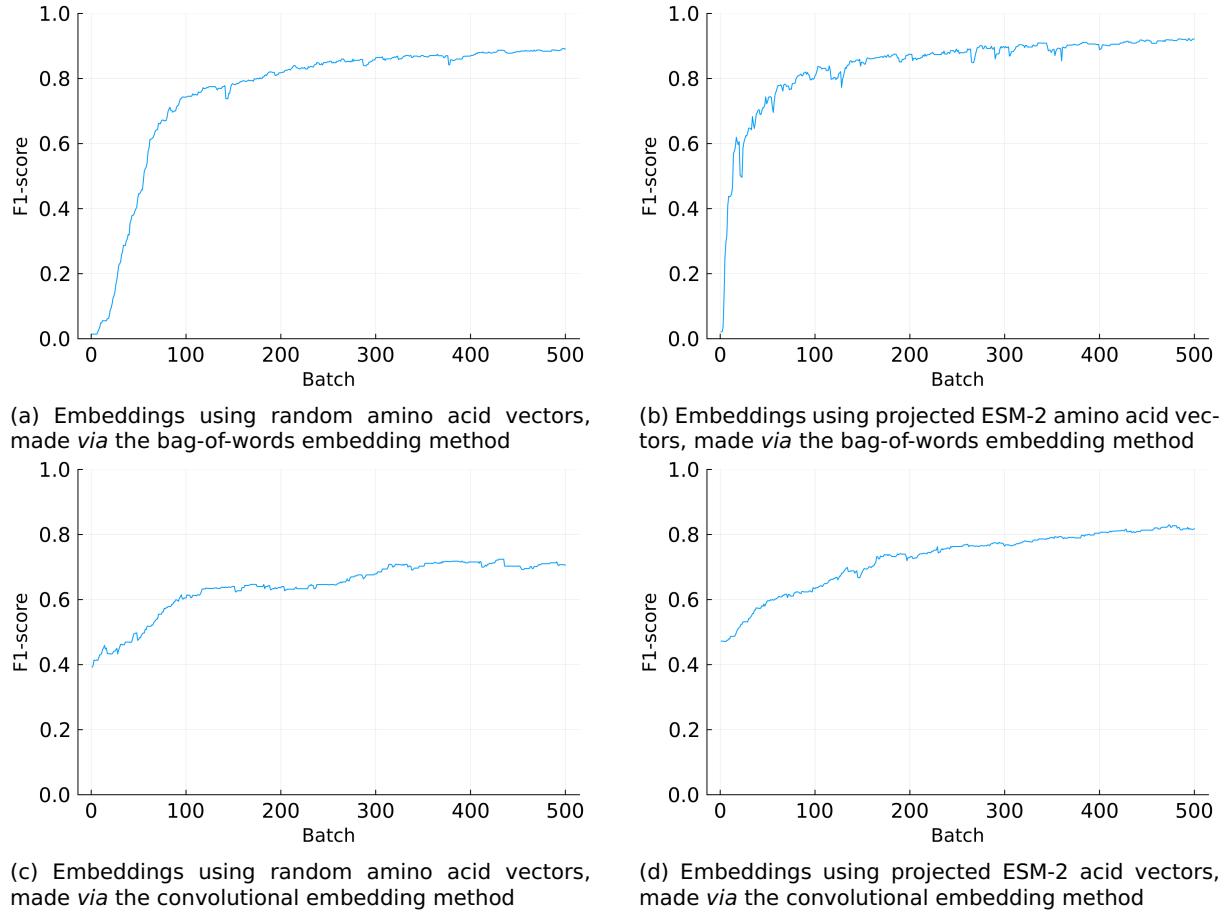


Figure 4.5: The F1-score of the OnlineHD single-pass model with a validation set throughout its training procedure. Trained in batches using several kinds of hyperdimensional embeddings from all protein sequences in the Phalp dataset whose types were manually annotated.

Upon initial observation, it is apparent that the models trained with embeddings generated using the convolutional method exhibit superior performance from the outset. This higher starting point might be partially attributed to the random initialization of the model. However, a common characteristic observed across all models is that their performance tends to stagnate after roughly 100 batches, which corresponds to roughly 870 sequences in our case. This observation suggests that hyperdimensional computing-based classification methods may exhibit promising performance and effectively capture essential information, even after being exposed to just a small fraction of the training data, which is an intriguing finding for further exploration.

4.5 Discussion

The low scores of the naive additive methods are likely due to the possibility of oversaturation of the class vectors as discussed in section 3.6. Thus, using the rudimentary model works only for very small datasets, as seen in the examples in chapter 2.3. The ability of this method to learn from a small dataset can be very useful, but backfires in terms of performance when applied to large and diverse datasets as we have seen in the case of classifying protein sequences from the PhaLP database.

This issue is mitigated in the OnlineHD method when used in classifications as it applies weights to class vector updates. This results in a substantial increase in performance, even coming close to the performance of established machine learning methods when paired with an iterative retraining procedure. From analyses of OnlineHD’s training procedure, we noticed that it is able to capture a lot of information with only a small set of the data, which warrants further study.

Upon comparing various methods for amino acid encoding, we find that dimension-reduction methods tend to capture more information from sequence embeddings created with projected ESM-2 vectors. This observation is consistent with our previous findings, where we noted enhanced performance in classification tasks using these vectors. While no visual differences were discernible after applying PCA to the protein sequences, the results varied subtly depending on the specific model used for classification. Therefore, it appears that there may not be a significant difference between the two sequence embedding methods.

5. HYPERDIMENSIONAL COMPUTING METHODS FOR AMINO ACID-LEVEL PREDICTIONS

As discussed in section 1.3.2, state-of-the-art protein language models primarily rely on transformer-based architectures to learn representations of protein sequences and amino acids in the context of other amino acids. These models have demonstrated their effectiveness in various applications, such as protein function prediction and protein-protein interaction prediction, by capturing complex patterns in amino acid sequences. Moreover, these models are also capable of making detailed predictions at the amino acid level, such as predicting secondary structure. This involves determining local three-dimensional structures of regions within a protein, such as alpha-helices and beta-sheets, based on the sequence of amino acids. This capability further extends their utility in understanding protein structure and function. However, these models are computationally intensive and may be less suitable for certain tasks or scenarios where computational resources are limited.

In section 3.4, we explored the idea of encoding information from a residue's neighborhood into a hyperdimensional vector that represents the residue and its surroundings. This approach allows us to capture local sequence information and can serve as an effective encoding scheme for amino acids. To evaluate the effectiveness of this encoding method, we will employ it as an input for several perceptron-based models.

Perceptron-based models, a type of artificial neural network model, have been employed for various machine learning tasks over the years, including classification and regression problems. The perceptron, one of the simplest and earliest types of neural networks, was proposed by Frank Rosenblatt in 1958 [62]. A perceptron consists of a single artificial neuron that receives

multiple input features and computes a weighted sum of these inputs. This weighted sum is then passed through an activation function, which determines the output of the perceptron. The learning process involves adjusting the weights of the input features iteratively, using a rule that minimizes prediction errors on the training data. The perceptron learning algorithm starts with an initial set of weights, typically set to zero or small random values. For each instance in the training data, the perceptron computes the output, compares it with the true label, and updates the weights accordingly. The weight update rule is based on the difference between the predicted output and the true label, multiplied by the learning rate and the input feature value. Single-layer perceptrons consist of an input layer and an output layer of perceptrons, allowing for multiclass classifications. However, they can only solve linearly separable problems, limiting their applicability for more complex tasks.

As single-layer perceptrons can only solve linearly separable problems, researchers developed more complex neural network architectures, such as multi-layer perceptrons (MLP), which consist of multiple layers of artificial neurons connected in a feedforward manner. MLPs can learn non-linear patterns in the data by using non-linear activation functions (e.g., sigmoid, tanh, ReLU) and employing backpropagation for weight updates [63]. This enhanced ability to model non-linear relationships allows MLPs to tackle a wider range of problems.

By combining the residue neighborhood encoding method with perceptron-based models, we could develop more computationally efficient alternatives to state-of-the-art transformer-based models. These models may provide valuable insights into protein sequences and amino acids while being more suitable for scenarios with limited computational resources.

5.1 Methods

Our objective is to build a straightforward perceptron-based neural network that takes as input a hyperdimensional vector, in particular a neighborhood-encoded amino acid vector made as described in section 3.4. It should then output the amino acid’s secondary structure, either 3- or 8-state as shown in figure [?].

5. Methods for amino acid-level predictions

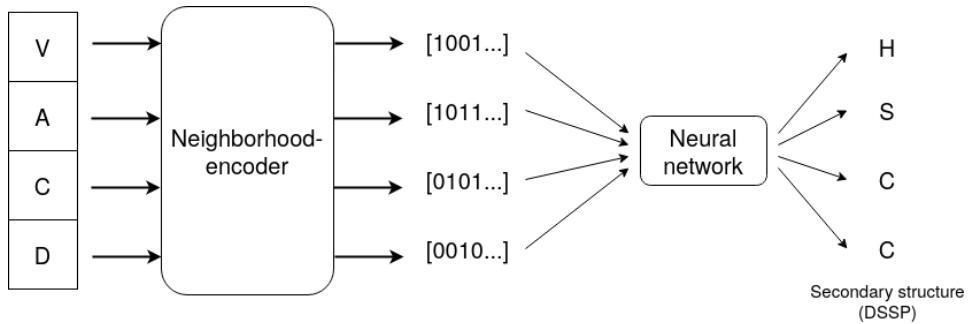


Figure 5.1: A simple demonstration of our pipeline to classify amino acids on their secondary structure based on their neighborhood-encoded vectors. First, a residue is encoded by neighborhood-encoding as discussed in section 3.4. Then, this vector is used as an input for a trained neural network to predict its secondary structure state. This is done for every residue in every given protein sequence.

We used the training dataset compiled by Klausen *et al.* for their model protein language model NetsurfP 2.0 [64] as our training dataset. It contains 10,337 protein sequences obtained from the Protein Data Bank (PDB) [65]. Each residue in these sequences is annotated with its 3- and 8-class secondary structure classification according to *Define Secondary Structure of Proteins* (DSSP) [66], providing a comprehensive description of the protein’s structural properties. All residues have been encoded using our neighborhood-encoder as described in section ?? with $n = 25$ and $n = 100$ using random hyperdimensional vectors for each amino acid. Due to computational memory constraints, we randomly selected 60 % of the sequences from the original dataset for training and testing (80 % and 20 % respectively), ensuring a representative sample of the data.

In summary, we have four distinct supervised training sets. The first model is trained on neighborhood-encodings of range $n = 25$ with a 3-state secondary level. The second model also uses neighborhood-encodings of range $n = 25$, but with an 8-state secondary level. The third model is trained on neighborhood-encodings of range $n = 100$, using a 3-state secondary level. Finally, the fourth model uses neighborhood-encodings of range $n = 100$ and an 8-state secondary level.

As for the perceptron-based methods, to decrease the computational time, we opted to build, train and evaluate the model in Python due to its support for computations on GPUs on our systems. We employed PyTorch-Lightning v1.8.4 for model construction and training and CUDA v11.7.0 on a high-performance computing (HPC) cluster equipped with an NVIDIA V100 GPU. The model comprised fully-connected layers with an input layer size equal to the dimension of the hyperdimensional vectors (10,000 in this case). De-

pending on the secondary structure classification desired, the output layer size was set to either 3 or 8 depending on the secondary structure classification. We used a batch size of 128 during training. Between each layer, we incorporated a ReLU (Rectified Linear Unit) activation function to introduce non-linearity into the model. The training loss was monitored using cross-entropy loss. To optimize the model, we employed the widely-used ADAM optimizer. We evaluated various configurations and hyperparameters to determine the best-performing but still computationally feasible approach:

Table 5.1: Overview of model configurations and hyperparameters tested

	Epochs	Learning rate	Size hidden layer(s)
SLP	100	0.03	/
1-layer MLP	50	0.003	500
10-layer MLP	200	0.0003	8000-5000-2000-1000-800-500-200-100-50-20

As test datasets, we opted to use CB513 [67] and CASP12 [68] (compiled by the developers of NetSurfP 2.0 [64]) since they are easily attainable and commonly used by researchers as benchmark datasets for protein language models. The MLP model with 10 hidden layers was trained using 60 % of the training data and then evaluated using these test datasets. For extra validation, random dummy data was generated.

5.2 Results

Figure B.1 shows the training progress of the SLP-based models. They show notable issues with the SLP models' ability to train effectively on our dataset. This is signaled by the substantial fluctuations in both loss and validation performance, a trend suggesting an inherent instability in the learning process of this model. The 1-layer MLP model also exhibits problematic behavior, as it does not learn as seen in its loss-curves in figure B.2. On top of that, it predicts only one single class during validation (the class with the highest proportion), resulting in a large discrepancy between the accuracy and F1-score. All of this could be attributed to the simplicity of these models, which might not have the capacity to capture the complex patterns in the data. These limitations lead to our decision to discontinue further investigations and experimentation with these particular models.

5. Methods for amino acid-level predictions

Figure 5.2 presents the outcome of the training procedure. In contrast to the SLP and the 1-layer MLP models, the performance of the much deeper 10-layer MLP model demonstrates much more stable and consistent behavior throughout the training process, as evidenced by the performance and loss metrics, even though the resulting performance remains generally low. Although this stability may partially result from the lower learning rate, the model’s computational feasibility made it an option for further investigation, prompting us to select this 133,343,733-parameter model for our subsequent tests with various datasets and comparisons with other state-of-the-art methods.

In order to benchmark our study against the capabilities of state-of-the-art protein language models, we collected the available results of other protein language models on similar experiments. The models include ESM-1b [69], NetSurfP 2.0 [64], its successor NetSurfP 3.0 [70] and ProtTrans’ best-performing model; ProtT5-XL-U50 [37]. ESM-1b is a model by Meta AI, trained on a large corpus of approximately 86 billion amino acid residues. It consists of around 670 million parameters and utilizes a transformer architecture together with evolutionary data. NetSurfP 2.0 and its successor NetSurf 3.0 are models comprising convolutional neural network (CNN) and bidirectional long short-term (biLSTM) layers. Despite not relying on a transformer-based architecture and having been trained on a comparatively smaller dataset, they exhibit state-of-the-art performance. The distinction between these two methods lies in their input configurations. ProtT5-XL-U50 is a large transformer-based model comprising 3 billion parameters, trained on UniRef50 which comprises roughly 45 million sequences [71].

Table 5.2: Accuracies of secondary structure classification of state-of-the-art protein language models. Tested on 3 and 8 secondary structure classifications of the CASP12 and CB513 dataset.

Accuracy	CASP12		CB513	
	ss3	ss8	ss3	ss8
ESM-1b	76.9	83.9	66.0	70.2
NetSurfP 2.0	82.0	84.5	66.9	71.3
NetSurfP 3.0	79.1	84.5	66.9	71.1
ProtT5-XL-U50	77.5	86.2	70.5	74.5

The fully trained 10-layer MLP models were evaluated on the test datasets with the results summarized in table 5.3 and table 5.4. To validate our findings, we also generated a randomly generated dataset for comparison. Disappointingly, the model’s performance on the test data did not surpass its

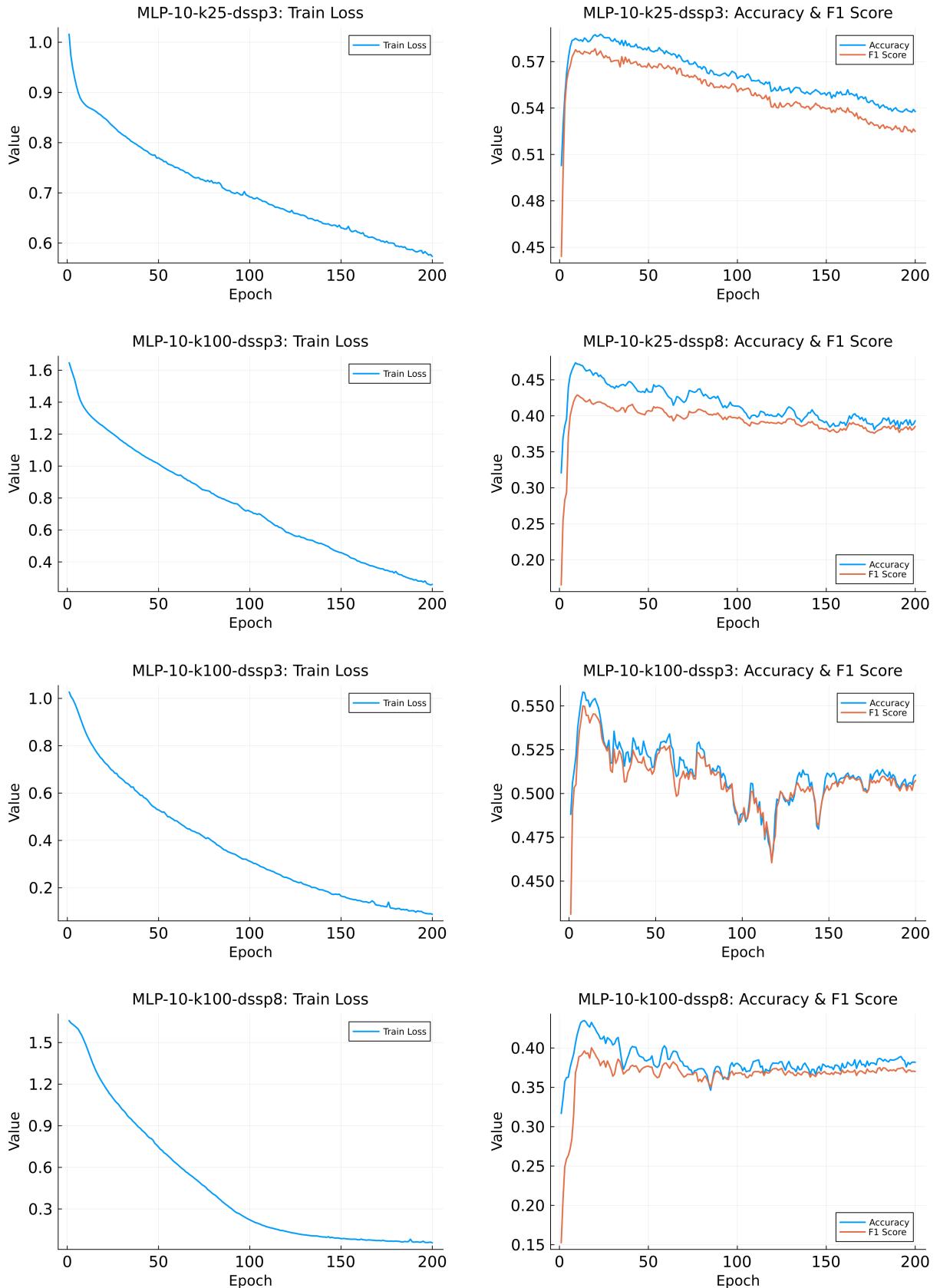


Figure 5.2: Performance of training procedure of the 10-layer MLP model on 60 % of Net-SurfP 2.0's training data, 0.8-0.2 training-test data split. Tested on 3 and 8 secondary structure classifications (dssp3 and dssp8 respectively) and with different kinds of hyperdimensional embeddings made with neighborhood-encoding ($n = 25$ and $n = 100$).

5. Methods for amino acid-level predictions

own performance on the randomly generated data, meaning that our model is not able to accurately predict the secondary structure of an amino acid based on its neighborhood-encoding.

Table 5.3: Accuracies of secondary structure classification of a 10-layer MLP model trained on 60 % of NetSurfP 2.0’s training data encoded into different kinds of hyperdimensional embeddings made with neighborhood-encoding ($n = 25$ and $n = 100$). Tested on the CASP12 and CB513 dataset. Performance with randomly generated data was also assessed.

Accuracy	CASP12		CB513		Random	
	ss3	ss8	ss3	ss8	ss3	ss8
n25	36.9	24.88	35.48	21.01	34.11	21.09
n100	39.33	23.13	37.61	20.86	38.74	20.53

Table 5.4: F1-scores of secondary structure classification of a 10-layer MLP model trained on 60 % of NetSurfP 2.0’s training data encoded into different kinds of hyperdimensional embeddings made with neighborhood-encoding ($n = 25$ and $n = 100$). Tested on the CASP12 and CB513 dataset. Performance with randomly generated data was also assessed.

F1	CASP12		CB513		Random	
	ss3	ss8	ss3	ss8	ss3	ss8
n25	37.50	18.61	35.52	15.30	34.35	14.79
n100	37.19	20.25	35.68	18.03	36.07	17.80

5.3 Discussion

Our exploration into the use of various perceptron-based neural network models for protein secondary structure prediction has yielded several insights. The overarching observation from our results is that our models have not effectively learned from the data. This suggests a couple of potential areas for improvement.

Firstly, the relative simplicity of our neural network models may be a limiting factor. The high-dimensional sparse vectors that represent our data may require a more complex model to capture the intricate patterns within the data effectively. This could involve exploring more sophisticated architectures, such as deeper networks or alternative structures.

Secondly, the neighborhood-encoding algorithm we employed may require further refinement. The quality of the embeddings produced by this algorithm directly impacts the ability of the model to learn. Optimizing this algorithm to produce more informative or discriminative embeddings could potentially enhance the learning capability of our models.

Lastly, the size of the training dataset could also be a contributing factor. Given the computational constraints of this study, we were limited in the amount of data we could use for training. However, machine learning models, especially deep neural networks, often benefit from larger datasets. Increasing the size of the training dataset could provide our models with more examples to learn from, potentially improving their performance.

Furthermore, when compared to state-of-the-art protein language models, our models fell short in performance. This underscores the complexity of protein secondary structure prediction and the need for models that can effectively learn from the intricate patterns in the data.

BIBLIOGRAPHY

- [1] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, April 1953. doi: 10.1038/171737a0. URL <https://doi.org/10.1038/171737a0>.
- [2] Francis Crick. Central dogma of molecular biology. *Nature*, 227(5258): 561–563, August 1970. doi: 10.1038/227561a0. URL <https://doi.org/10.1038/227561a0>.
- [3] J. C. Kendrew, G. Bodo, H. M. Dintzis, R. G. Parrish, H. Wyckoff, and D. C. Phillips. A three-dimensional model of the myoglobin molecule obtained by x-ray analysis. *Nature*, 181(4610):662–666, March 1958. doi: 10.1038/181662a0. URL <https://doi.org/10.1038/181662a0>.
- [4] F. Sanger and E. O. P. Thompson. The amino-acid sequence in the gly-cyl chain of insulin. 1. the identification of lower peptides from partial hydrolysates. *Biochemical Journal*, 53(3):353–366, February 1953. doi: 10.1042/bj0530353. URL <https://doi.org/10.1042/bj0530353>.
- [5] JOSEPH S. FRUTON. Early theories of protein structure. *Annals of the New York Academy of Sciences*, 325(1):1–20, 1979. doi: <https://doi.org/10.1111/j.1749-6632.1979.tb14125.x>. URL <https://nyaspubs.onlinelibrary.wiley.com/doi/abs/10.1111/j.1749-6632.1979.tb14125.x>.
- [6] P. Edman and G. Begg. A protein sequenator. *European Journal of Biochemistry*, 1(1):80–91, March 1967. doi: 10.1111/j.1432-1033.1967.tb00047.x. URL <https://doi.org/10.1111/j.1432-1033.1967.tb00047.x>.
- [7] Margaret Oakley Dayhoff and Robert S. Ledley. Comprotein. In *Proceedings of the December 4-6, 1962, fall joint computer conference on - AFIPS '62 (Fall)*. ACM Press, 1962. doi: 10.1145/1461518.1461546. URL <https://doi.org/10.1145/1461518.1461546>.

-
- [8] Robert T. Hersh, Richard V. Eck, and Margaret O. Dayhoff. Atlas of protein sequence and structure, 1966. *Systematic Zoology*, 16(3):262, September 1967. doi: 10.2307/2412074. URL <https://doi.org/10.2307/2412074>.
 - [9] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970. ISSN 0022-2836. doi: [https://doi.org/10.1016/0022-2836\(70\)90057-4](https://doi.org/10.1016/0022-2836(70)90057-4). URL <https://www.sciencedirect.com/science/article/pii/0022283670900574>.
 - [10] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981. ISSN 0022-2836. doi: [https://doi.org/10.1016/0022-2836\(81\)90087-5](https://doi.org/10.1016/0022-2836(81)90087-5). URL <https://www.sciencedirect.com/science/article/pii/0022283681900875>.
 - [11] David J. Lipman and William R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985. doi: 10.1126/science.2983426. URL <https://www.science.org/doi/abs/10.1126/science.2983426>.
 - [12] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990. ISSN 0022-2836. doi: [https://doi.org/10.1016/S0022-2836\(05\)80360-2](https://doi.org/10.1016/S0022-2836(05)80360-2). URL <https://www.sciencedirect.com/science/article/pii/S0022283605803602>.
 - [13] M Murata, J S Richardson, and J L Sussman. Simultaneous comparison of three protein sequences. *Proceedings of the National Academy of Sciences*, 82(10):3073–3077, May 1985. doi: 10.1073/pnas.82.10.3073. URL <https://doi.org/10.1073/pnas.82.10.3073>.
 - [14] Desmond G. Higgins and Paul M. Sharp. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, December 1988. doi: 10.1016/0378-1119(88)90330-7. URL [https://doi.org/10.1016/0378-1119\(88\)90330-7](https://doi.org/10.1016/0378-1119(88)90330-7).
 - [15] R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, March 2004. doi: 10.1093/nar/gkh340. URL <https://doi.org/10.1093/nar/gkh340>.

BIBLIOGRAPHY

- [16] Phylogeny in multiple sequence alignments. In *Multiple Biological Sequence Alignment: Scoring Functions, Algorithms and Applications*, pages 103–112. John Wiley & Sons, Inc., June 2016. doi: 10.1002/9781119273769.ch6. URL <https://doi.org/10.1002/9781119273769.ch6>.
- [17] Eugene V. Koonin and Artem S. Novozhilov. Origin and evolution of the genetic code: The universal enigma. *IUBMB Life*, 61(2):99–111, February 2009. doi: 10.1002/iub.146. URL <https://doi.org/10.1002/iub.146>.
- [18] F. Sanger, G. M. Air, B. G. Barrell, N. L. Brown, A. R. Coulson, J. C. Fiddes, C. A. Hutchison, P. M. Slocombe, and M. Smith. Nucleotide sequence of bacteriophage x174 DNA. *Nature*, 265(5596):687–695, February 1977. doi: 10.1038/265687a0. URL <https://doi.org/10.1038/265687a0>.
- [19] W. MIN JOU, G. HAEGEMAN, M. YSEBAERT, and W. FIERS. Nucleotide sequence of the gene coding for the bacteriophage MS2 coat protein. *Nature*, 237(5350):82–88, May 1972. doi: 10.1038/237082a0. URL <https://doi.org/10.1038/237082a0>.
- [20] O.B. Ptitsyn. How does protein synthesis give rise to the 3d-structure? *FEBS Letters*, 285(2):176–181, 1991. doi: [https://doi.org/10.1016/0014-5793\(91\)80799-9](https://doi.org/10.1016/0014-5793(91)80799-9). URL <https://febs.onlinelibrary.wiley.com/doi/abs/10.1016/0014-5793%2891%2980799-9>.
- [21] Simone König, Wolfgang M. J. Obermann, and Johannes A. Eble. The current state-of-the-art identification of unknown proteins using mass spectrometry exemplified on de novo sequencing of a venom protease from bothrops moojeni. *Molecules*, 27(15), 2022. ISSN 1420-3049. doi: 10.3390/molecules27154976. URL <https://www.mdpi.com/1420-3049/27/15/4976>.
- [22] S. Altschul. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, September 1997. doi: 10.1093/nar/25.17.3389. URL <https://doi.org/10.1093/nar/25.17.3389>.
- [23] Martin Steinegger, Markus Meier, Milot Mirdita, Harald Vöhringer, Stephan J. Haunsberger, and Johannes Söding. HH-suite3 for fast remote homology detection and deep protein annotation. *BMC Bioinformatics*, 20(1), September 2019. doi: 10.1186/s12859-019-3019-7. URL <https://doi.org/10.1186/s12859-019-3019-7>.

-
- [24] Martin Steinegger and Johannes Söding. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*, 35(11):1026–1028, October 2017. doi: 10.1038/nbt.3988. URL <https://doi.org/10.1038/nbt.3988>.
- [25] Ka Man Yip, Niels Fischer, Elham Paknia, Ashwin Chari, and Holger Stark. Atomic-resolution protein structure determination by cryo-EM. *Nature*, 587(7832):157–161, October 2020. doi: 10.1038/s41586-020-2833-4. URL <https://doi.org/10.1038/s41586-020-2833-4>.
- [26] Andrew Leaver-Fay, Michael Tyka, Steven M. Lewis, Oliver F. Lange, James Thompson, Ron Jacak, Kristian W. Kaufman, P. Douglas Renfrew, Colin A. Smith, Will Sheffler, Ian W. Davis, Seth Cooper, Adrien Treuille, Daniel J. Mandell, Florian Richter, Yih-En Andrew Ban, Sarel J. Fleishman, Jacob E. Corn, David E. Kim, Sergey Lyskov, Monica Berrondo, Stuart Mentzer, Zoran Popović, James J. Havranek, John Karanicolas, Rhiju Das, Jens Meiler, Tanja Kortemme, Jeffrey J. Gray, Brian Kuhlman, David Baker, and Philip Bradley. Rosetta3. In *Computer Methods, Part C*, pages 545–574. Elsevier, 2011. doi: 10.1016/b978-0-12-381270-4.00019-6. URL <https://doi.org/10.1016/b978-0-12-381270-4.00019-6>.
- [27] Tristan Bepler and Bonnie Berger. Learning the protein language: Evolution, structure, and function. *Cell Systems*, 12(6):654–669.e3, June 2021. doi: 10.1016/j.cels.2021.05.017. URL <https://doi.org/10.1016/j.cels.2021.05.017>.
- [28] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishabh Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michał Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, July 2021. doi: 10.1038/s41586-021-03819-2. URL <https://doi.org/10.1038/s41586-021-03819-2>.

BIBLIOGRAPHY

- [29] Ewen Callaway. After AlphaFold: protein-folding contest seeks next big breakthrough. *Nature*, 613(7942):13–14, December 2022. doi: 10.1038/d41586-022-04438-1. URL <https://doi.org/10.1038/d41586-022-04438-1>.
- [30] Dan Ofer, Nadav Brandes, and Michal Linial. The language of proteins: Nlp, machine learning and protein sequences. *Computational and Structural Biotechnology Journal*, 19:1750–1758, 2021. doi: 10.1016/j.csbj.2021.03.022. URL <https://doi.org/10.1016/j.csbj.2021.03.022>.
- [31] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. 2018. doi: 10.48550/ARXIV.1802.05365. URL <https://arxiv.org/abs/1802.05365>.
- [32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018. doi: 10.48550/ARXIV.1810.04805. URL <https://arxiv.org/abs/1810.04805>.
- [33] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020. doi: 10.48550/ARXIV.2005.14165. URL <https://arxiv.org/abs/2005.14165>.
- [34] OpenAI. Gpt-4 technical report. 2023. doi: 10.48550/ARXIV.2303.08774. URL <https://arxiv.org/abs/2303.08774>.
- [35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017. doi: 10.48550/ARXIV.1706.03762. URL <https://arxiv.org/abs/1706.03762>.
- [36] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Xi Chen, John Canny, Pieter Abbeel, and Yun S. Song. Evaluating protein transfer learning with tape. 2019. doi: 10.48550/ARXIV.1906.08230. URL <https://arxiv.org/abs/1906.08230>.

-
- [37] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rihawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, Debsindhu Bhowmik, and Burkhard Rost. Prottrans: Towards cracking the language of life’s code through self-supervised deep learning and high performance computing. 2020. doi: 10.48550/ARXIV.2007.06225. URL <https://arxiv.org/abs/2007.06225>.
 - [38] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. Evolutionary-scale prediction of atomic level protein structure with a language model. July 2022. doi: 10.1101/2022.07.20.500902. URL <https://doi.org/10.1101/2022.07.20.500902>.
 - [39] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. 2021. doi: 10.48550/ARXIV.2104.04473. URL <https://arxiv.org/abs/2104.04473>.
 - [40] Pentti Kanerva. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation*, 1(2):139–159, jun 2009. ISSN 1866-9956. doi: 10.1007/s12559-009-9009-8. URL <http://link.springer.com/10.1007/s12559-009-9009-8>.
 - [41] T.A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641, 1995. doi: 10.1109/72.377968.
 - [42] Pentti Kanerva. The spatter code for encoding concepts at many levels. pages 226–229, 1994.
 - [43] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi: 10.1137/141000671. URL <https://pubs.siam.org/doi/10.1137/141000671>.
 - [44] Manuel Schmuck, Luca Benini, and Abbas Rahimi. Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory. *J. Emerg. Technol. Comput. Syst.*, 15(4), oct 2019. ISSN

BIBLIOGRAPHY

- 1550-4832. doi: 10.1145/3314326. URL <https://doi.org/10.1145/3314326>.
- [45] Lulu Ge and Keshab K. Parhi. Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine*, 20(2):30–47, 2020. doi: 10.1109/mcas.2020.2988388. URL <https://doi.org/10.1109%2Fmcas.2020.2988388>.
- [46] UCI Machine Learning. Zoo animal classification, 2016. URL <https://www.kaggle.com/datasets/uciml/zoo-animal-classification>.
- [47] Francesca Grisoni, Claudia S. Neuhaus, Miyabi Hishinuma, Gisela Gabernet, Jan A. Hiss, Masaaki Kotera, and Gisbert Schneider. 'de novo design of anticancer peptides by ensemble artificial neural networks. *Journal of Molecular Modeling*, 25(5):122, 2019. ISSN 0948-5023. doi: 10.1007/s00894-019-4007-6.
- [48] Zhuowen Zou, Hanning Chen, Prathyush Poduval, Yeseong Kim, Mahdi Imani, Elaheh Sadredini, Rosario Cammarota, and Mohsen Imani. BioHD. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ACM, June 2022. doi: 10.1145/3470496.3527422. URL <https://doi.org/10.1145/3470496.3527422>.
- [49] P.H.A. Sneath. Relations between chemical structure and biological activity in peptides. *Journal of Theoretical Biology*, 12(2):157–195, November 1966. doi: 10.1016/0022-5193(66)90112-3. URL [https://doi.org/10.1016/0022-5193\(66\)90112-3](https://doi.org/10.1016/0022-5193(66)90112-3).
- [50] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1998. doi: 10.7551/mitpress/3927.001.0001. URL <https://doi.org/10.7551/mitpress/3927.001.0001>.
- [51] Evolutionary.jl: a Julia package for evolutionary & genetic algorithms. URL <https://github.com/wildart/Evolutionary.jl>.
- [52] S Henikoff and J G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, November 1992. doi: 10.1073/pnas.89.22.10915. URL <https://doi.org/10.1073/pnas.89.22.10915>.
- [53] R. Grantham. Amino acid difference formula to help explain protein evolution. *Science*, 185(4154):862–864, September 1974. doi: 10.1126/science.185.4154.862. URL <https://doi.org/10.1126/science.185.4154.862>.

-
- [54] Tao Yu, Yichi Zhang, Zhiru Zhang, and Christopher De Sa. Understanding hyperdimensional computing for parallel single-pass learning. 2022. doi: 10.48550/ARXIV.2202.04805. URL <https://arxiv.org/abs/2202.04805>.
 - [55] Bjorn Criel, Steff Taelman, Wim Van Criekinge, Michiel Stock, and Yves Briers. Phalp: A database for the study of phage lytic proteins and their evolution. *Viruses*, 13(7):1240, jun 2021. doi: 10.3390/v13071240. URL <https://doi.org/10.3390/v13071240>.
 - [56] Lorena Rodríguez-Rubio, Diana Gutiérrez, David M. Donovan, Beatriz Martínez, Ana Rodríguez, and Pilar García. Phage lytic proteins: biotechnological applications beyond clinical antimicrobials. *Critical Reviews in Biotechnology*, pages 1–11, January 2015. doi: 10.3109/07388551.2014.993587. URL <https://doi.org/10.3109/07388551.2014.993587>.
 - [57] Elizabeth M. Darby, Eleftheria Trampari, Pauline Siasat, Maria Solsona Gaya, Ilyas Alav, Mark A. Webber, and Jessica M. A. Blair. Molecular mechanisms of antibiotic resistance revisited. *Nature Reviews Microbiology*, 21(5):280–295, November 2022. doi: 10.1038/s41579-022-00820-y. URL <https://doi.org/10.1038/s41579-022-00820-y>.
 - [58] Michael Heinzinger, Ahmed Elnaggar, Yu Wang, Christian Dallago, Dmitrii Nechaev, Florian Matthes, and Burkhard Rost. Modeling aspects of the language of life through transfer-learning protein sequences. *BMC Bioinformatics*, 20(1), December 2019. doi: 10.1186/s12859-019-3220-8. URL <https://doi.org/10.1186/s12859-019-3220-8>.
 - [59] Leo Breiman. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/a:1010933404324. URL <https://doi.org/10.1023/a:1010933404324>.
 - [60] Alejandro Hernández-Cane, Namiko Matsumoto, Er Jwee Ping, and Mohsen Imani. Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system. *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 56–61, 2021.
 - [61] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016. URL <http://arxiv.org/abs/1603.02754>.

BIBLIOGRAPHY

- [62] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. doi: 10.1037/h0042519. URL <https://doi.org/10.1037/h0042519>.
- [63] Juergen Schmidhuber. Annotated history of modern ai and deep learning. 2022. doi: 10.48550/ARXIV.2212.11279. URL <https://arxiv.org/abs/2212.11279>.
- [64] Michael Schantz Klausen, Martin Closter Jespersen, Henrik Nielsen, Kamilla Kjærgaard Jensen, Vanessa Isabell Jurtz, Casper Kaae Sønderby, Morten Otto Alexander Sommer, Ole Winther, Morten Nielsen, Bent Petersen, and Paolo Marcatili. NetSurfP-2.0: Improved prediction of protein structural features by integrated deep learning. *Proteins: Structure, Function, and Bioinformatics*, 87(6):520–527, March 2019. doi: 10.1002/prot.25674. URL <https://doi.org/10.1002/prot.25674>.
- [65] H. M. Berman. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, January 2000. doi: 10.1093/nar/28.1.235. URL <https://doi.org/10.1093/nar/28.1.235>.
- [66] Wolfgang Kabsch and Christian Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, December 1983. doi: 10.1002/bip.360221211. URL <https://doi.org/10.1002/bip.360221211>.
- [67] Yuedong Yang, Jianzhao Gao, Jihua Wang, Rhys Heffernan, Jack Hanson, Kuldip Paliwal, and Yaoqi Zhou. Sixty-five years of the long march in protein secondary structure prediction: the final stretch? *Briefings in Bioinformatics*, page bbw129, December 2016. doi: 10.1093/bib/bbw129. URL <https://doi.org/10.1093/bib/bbw129>.
- [68] John Moult, Krzysztof Fidelis, Andriy Kryshtafovych, Torsten Schwede, and Anna Tramontano. Critical assessment of methods of protein structure prediction (CASP)-round XII. *Proteins: Structure, Function, and Bioinformatics*, 86:7–15, December 2017. doi: 10.1002/prot.25415. URL <https://doi.org/10.1002/prot.25415>.
- [69] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, and

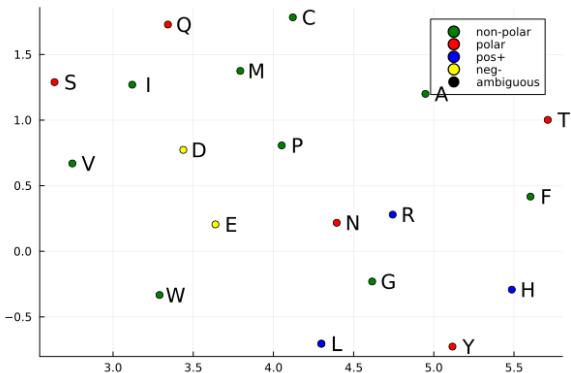
Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. April 2019. doi: 10.1101/622803. URL <https://doi.org/10.1101/622803>.

- [70] Magnus Haraldson Høie, Erik Nicolas Kiehl, Bent Petersen, Morten Nielsen, Ole Winther, Henrik Nielsen, Jeppe Hallgren, and Paolo Marcatili. NetSurfP-3.0: accurate and fast prediction of protein structural features by protein language models and deep learning. *Nucleic Acids Research*, 50(W1):W510–W515, June 2022. doi: 10.1093/nar/gkac439. URL <https://doi.org/10.1093/nar/gkac439>.
- [71] Baris E. Suzek, Hongzhan Huang, Peter McGarvey, Raja Mazumder, and Cathy H. Wu. UniRef: comprehensive and non-redundant UniProt reference clusters. *Bioinformatics*, 23(10):1282–1288, March 2007. doi: 10.1093/bioinformatics/btm098. URL <https://doi.org/10.1093/bioinformatics/btm098>.

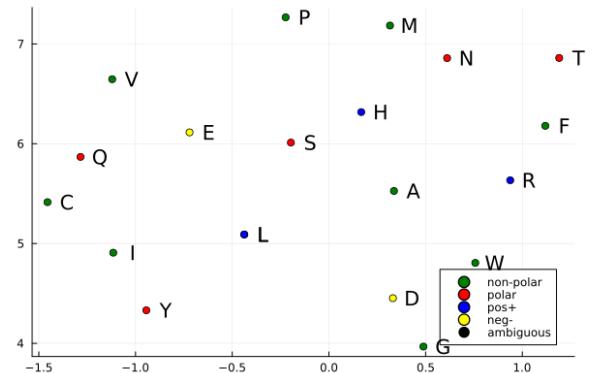
APPENDIX A

ADDITIONAL INFORMATION ON

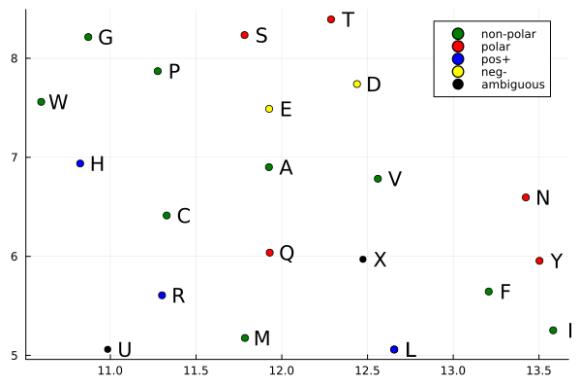
CHAPTER 3



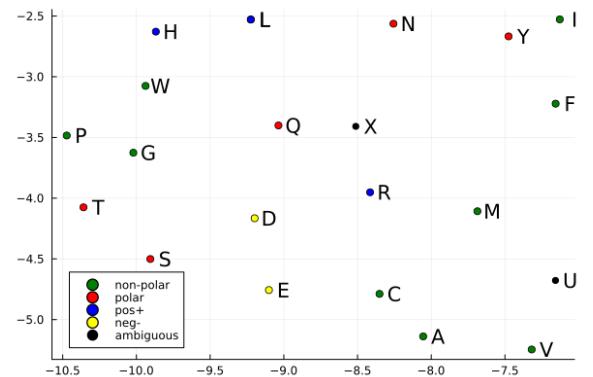
(a) Made starting from random hyperdimensional vectors for each amino acid, $k = 4$.



(b) Made starting from random hyperdimensional vectors for each amino acid, $k = 50$.



(c) Made starting from extended ESM-2 embeddings for each amino acid, $k = 4$.



(d) Made starting from extended ESM-2 embeddings for each amino acid, $k = 50$.

Figure A.1: Scatter plot of a two-dimensional UMAP projection of the average amino acid HDVs with neighborhood-information of k encoded. Learned from the human reference proteome. The amino acids are annotated and colored based on their chemical property of polarity.

Table A.1: Target similarities made from Grantham's distance matrix

0.0	0.73	0.87	0.8	0.87	0.73	0.87	0.8	0.8	0.8	0.87	0.8	0.8	0.8	0.8	0.67	0.73	0.73	0.93	0.87
0.73	0.0	0.93	1.0	0.87	0.93	0.93	0.8	0.93	0.8	0.8	0.93	0.93	0.93	0.93	0.8	0.8	0.8	0.8	0.87
0.87	0.93	0.0	0.6	0.93	0.8	0.8	0.93	0.8	1.0	0.93	0.67	0.8	0.73	0.87	0.73	0.8	0.93	1.0	0.93
0.8	1.0	0.6	0.0	0.93	0.87	0.73	0.93	0.67	0.93	0.87	0.73	0.8	0.6	0.73	0.73	0.8	0.87	0.93	0.87
0.87	0.87	0.93	0.93	0.0	0.93	0.8	0.73	0.93	0.73	0.73	0.93	1.0	0.93	0.93	0.87	0.87	0.8	0.67	0.53
0.73	0.93	0.8	0.87	0.93	0.0	0.87	1.0	0.87	1.0	0.93	0.73	0.87	0.87	0.73	0.87	0.93	0.87	0.93	0.93
0.87	0.93	0.8	0.73	0.8	0.87	0.0	0.93	0.8	0.93	0.87	0.67	0.87	0.73	0.73	0.8	0.87	0.93	0.87	0.6
0.8	0.8	0.93	0.93	0.73	1.0	0.93	0.0	0.93	0.6	0.67	0.93	0.93	0.93	0.93	0.87	0.8	0.53	0.93	0.8
0.8	0.93	0.8	0.67	0.93	0.87	0.8	0.93	0.0	0.87	0.8	0.73	0.8	0.67	0.6	0.73	0.8	0.87	0.93	0.87
0.8	0.8	1.0	0.93	0.73	1.0	0.93	0.6	0.87	0.0	0.6	0.93	0.93	0.87	0.87	0.8	0.67	0.87	0.8	0.87
0.8	0.8	0.93	0.87	0.73	0.93	0.87	0.67	0.8	0.6	0.0	0.87	0.87	0.73	0.8	0.8	0.8	0.67	0.8	0.8
0.87	0.93	0.67	0.73	0.93	0.73	0.67	0.93	0.73	0.93	0.87	0.0	0.87	0.73	0.73	0.67	0.73	0.93	1.0	0.87
0.8	0.93	0.8	0.8	1.0	0.87	0.87	0.93	0.8	0.93	0.87	0.0	0.8	0.87	0.8	0.8	0.87	0.1	0.93	0.93
0.8	0.93	0.73	0.6	0.93	0.87	0.73	0.93	0.67	0.87	0.73	0.73	0.8	0.0	0.67	0.73	0.8	0.87	0.87	0.8
0.8	0.93	0.87	0.73	0.93	0.87	0.73	0.93	0.6	0.87	0.8	0.73	0.87	0.67	0.0	0.8	0.8	0.93	0.93	0.87
0.67	0.8	0.73	0.73	0.87	0.73	0.8	0.87	0.73	0.87	0.8	0.67	0.8	0.73	0.8	0.0	0.67	0.87	0.93	0.87
0.73	0.8	0.8	0.8	0.87	0.87	0.87	0.87	0.8	0.8	0.8	0.73	0.8	0.8	0.8	0.67	0.0	0.73	0.87	0.87
0.73	0.8	0.93	0.87	0.8	0.93	0.93	0.53	0.87	0.67	0.67	0.93	0.87	0.87	0.93	0.87	0.73	0.0	0.93	0.8
0.93	0.87	1.0	0.93	0.67	0.87	0.87	0.93	0.93	0.87	0.8	1.0	1.0	0.87	0.93	0.93	0.87	0.93	0.0	0.6
0.87	0.87	0.93	0.87	0.53	0.93	0.6	0.8	0.87	0.8	0.8	0.87	0.93	0.8	0.87	0.87	0.87	0.8	0.6	0.0

Table A.2: Achieved similarities targeting Grantham's distance matrix

0.0	0.49	0.49	0.48	0.48	0.49	0.5	0.49	0.49	0.5	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.51	0.49	0.49
0.49	0.0	0.49	0.5	0.49	0.49	0.49	0.49	0.49	0.5	0.48	0.5	0.5	0.49	0.48	0.49	0.49	0.49	0.49	0.49
0.49	0.49	0.0	0.49	0.49	0.49	0.49	0.49	0.49	0.5	0.49	0.49	0.49	0.5	0.49	0.49	0.5	0.49	0.49	0.49
0.48	0.5	0.49	0.0	0.49	0.48	0.5	0.5	0.49	0.49	0.49	0.49	0.5	0.5	0.49	0.49	0.5	0.49	0.49	0.49
0.48	0.49	0.49	0.49	0.0	0.5	0.5	0.49	0.49	0.5	0.49	0.5	0.5	0.49	0.49	0.5	0.5	0.51	0.5	0.49
0.49	0.49	0.49	0.48	0.5	0.0	0.5	0.49	0.5	0.49	0.49	0.5	0.5	0.49	0.49	0.49	0.5	0.5	0.49	0.5
0.5	0.49	0.49	0.5	0.5	0.5	0.0	0.5	0.51	0.5	0.5	0.49	0.49	0.49	0.49	0.49	0.49	0.5	0.48	0.49
0.49	0.5	0.49	0.5	0.5	0.49	0.5	0.0	0.48	0.49	0.49	0.51	0.5	0.5	0.49	0.49	0.5	0.5	0.5	0.5
0.49	0.5	0.49	0.49	0.49	0.5	0.51	0.48	0.0	0.5	0.49	0.5	0.5	0.49	0.49	0.5	0.5	0.48	0.49	0.49
0.5	0.48	0.5	0.49	0.49	0.49	0.5	0.49	0.5	0.0	0.5	0.5	0.49	0.49	0.49	0.49	0.49	0.5	0.5	0.48
0.49	0.5	0.49	0.49	0.5	0.49	0.5	0.49	0.49	0.5	0.0	0.49	0.49	0.51	0.5	0.49	0.5	0.5	0.49	0.49
0.49	0.5	0.49	0.5	0.5	0.5	0.5	0.51	0.5	0.5	0.49	0.0	0.49	0.49	0.51	0.5	0.49	0.5	0.49	0.49
0.49	0.5	0.49	0.5	0.5	0.5	0.5	0.51	0.5	0.5	0.49	0.0	0.49	0.49	0.51	0.5	0.49	0.5	0.49	0.49
0.49	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.51	0.5	0.5	0.49	0.0	0.49	0.49	0.51	0.5	0.49	0.5	0.48
0.49	0.49	0.49	0.5	0.49	0.49	0.49	0.5	0.49	0.49	0.51	0.49	0.5	0.49	0.49	0.49	0.5	0.49	0.49	0.49
0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.5	0.49	0.49	0.49	0.49	0.5	0.49	0.49	0.49
0.49	0.49	0.49	0.49	0.51	0.49	0.49	0.49	0.49	0.49	0.49	0.5	0.49	0.49	0.49	0.49	0.49	0.5	0.49	0.49
0.51	0.49	0.5	0.49	0.51	0.5	0.49	0.49	0.48	0.49	0.49	0.49	0.5	0.5	0.49	0.49	0.49	0.5	0.5	0.49
0.49	0.49	0.49	0.49	0.51	0.49	0.5	0.49	0.49	0.5	0.49	0.49	0.5	0.49	0.49	0.5	0.5	0.0	0.49	0.48
0.49	0.49	0.49	0.49	0.5	0.5	0.48	0.5	0.49	0.5	0.49	0.5	0.49	0.5	0.49	0.5	0.49	0.5	0.49	0.5
0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49

Table A.3: Target similarities made from BLOSUM62 matrix

0.0	0.73	0.87	0.8	0.87	0.73	0.87	0.8	0.8	0.8	0.87	0.8	0.8	0.8	0.8	0.67	0.73	0.73	0.93	0.87
0.73	0.0	0.93	1.0	0.87	0.93	0.93	0.8	0.93	0.8	0.8	0.93	0.93	0.93	0.93	0.8	0.8	0.8	0.87	0.87
0.87	0.93	0.0	0.6	0.93	0.8	0.8	0.93	0.8	1.0	0.93	0.67	0.8	0.73	0.87	0.73	0.8	0.93	1.0	0.93
0.8	1.0	0.6	0.0	0.93	0.87	0.73	0.93	0.67	0.93	0.87	0.73	0.8	0.6	0.73	0.73	0.8	0.87	0.93	0.87
0.87	0.87	0.93	0.93	0.0	0.93	0.8	0.73	0.93	0.73	0.73	0.93	1.0	0.93	0.93	0.87	0.87	0.8	0.67	0.53
0.73	0.93	0.8	0.87	0.93	0.0	0.87	1.0	0.87	1.0	0.93	0.73	0.87	0.87	0.73	0.87	0.93	0.87	0.93	0.93
0.87	0.93	0.8	0.73	0.8	0.87	0.0	0.93	0.8	0.93	0.87	0.67	0.87	0.73	0.87	0.73	0.8	0.87	0.93	0.87
0.87	0.93	0.8	0.73	0.8	0.87	0.0	0.93	0.8	0.93	0.87	0.67	0.87	0.73	0.87	0.73	0.8	0.87	0.93	0.87
0.8	0.8	0.93	0.93	0.73	1.0	0.93	0.0	0.93	0.6	0.67	0.93	0.93	0.93	0.93	0.87	0.8	0.87	0.93	0.87
0.8	0.8	0.93	0.87	0.73	0.93	0.0	0.87	0.8	0.73	0.8	0.67	0.6	0.73	0.8	0.87	0.93	0.87	0.93	0.87
0.8	0.8	0.93	0.87	0.73	0.93	0.0	0.87	0.8	0.73	0.8	0.67	0.6	0.73	0.8	0.87	0.93	0.87	0.93	0.87
0.8	0.8	0.93	0.87	0.73	0.93	0.0	0.87	0.8	0.73	0.8	0.67	0.6	0.73	0.8	0.87	0.93	0.87	0.93	0.87
0.87	0.93	0.67	0.73	0.93	0.73	0.67	0.93	0.73	0.93	0.87	0.0	0.87	0.73	0.73	0.67	0.73	0.93	1.0	0.87
0.8	0.93	0.8	0.8	1.0	0.87	0.87	0.93	0.8	0.93	0.87	0.0	0.8	0.87	0.8	0.8	0.87	1.0	0.93	0.93
0.8	0.93	0.73	0.6	0.93	0.87	0.73													

A. Additional information on chapter 3

Table A.4: Achieved similarities targeting BLOSUM62 matrix

0.0	0.49	0.49	0.49	0.5	0.49	0.48	0.49	0.5	0.49	0.5	0.49	0.5	0.5	0.5	0.49	0.49	0.49	0.49	0.49
0.49	0.0	0.5	0.5	0.5	0.49	0.49	0.49	0.49	0.49	0.48	0.49	0.48	0.49	0.49	0.49	0.49	0.49	0.49	0.49
0.49	0.5	0.0	0.49	0.5	0.5	0.49	0.51	0.48	0.49	0.5	0.49	0.49	0.5	0.5	0.48	0.49	0.49	0.49	0.49
0.49	0.5	0.49	0.0	0.49	0.5	0.49	0.5	0.5	0.5	0.5	0.49	0.49	0.51	0.49	0.49	0.49	0.49	0.48	0.49
0.5	0.5	0.5	0.49	0.0	0.5	0.5	0.51	0.5	0.5	0.5	0.5	0.5	0.5	0.49	0.5	0.49	0.5	0.49	0.5
0.49	0.49	0.5	0.5	0.5	0.0	0.49	0.49	0.5	0.5	0.49	0.49	0.49	0.49	0.5	0.49	0.5	0.49	0.49	0.5
0.48	0.49	0.5	0.49	0.5	0.49	0.0	0.5	0.49	0.49	0.5	0.5	0.5	0.5	0.49	0.5	0.49	0.49	0.49	0.49
0.49	0.49	0.49	0.5	0.51	0.49	0.5	0.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.49	0.49	0.49	0.49	0.5
0.5	0.49	0.51	0.5	0.5	0.5	0.49	0.5	0.0	0.49	0.5	0.49	0.49	0.5	0.5	0.49	0.49	0.5	0.49	0.48
0.49	0.49	0.48	0.5	0.5	0.5	0.49	0.5	0.49	0.0	0.5	0.48	0.49	0.49	0.49	0.5	0.49	0.5	0.48	0.49
0.5	0.48	0.49	0.5	0.5	0.5	0.5	0.5	0.5	0.0	0.5	0.5	0.5	0.5	0.5	0.49	0.49	0.49	0.49	0.49
0.49	0.49	0.5	0.49	0.5	0.49	0.5	0.5	0.49	0.48	0.5	0.0	0.51	0.5	0.49	0.49	0.49	0.5	0.49	0.5
0.5	0.48	0.49	0.49	0.5	0.49	0.5	0.5	0.49	0.49	0.5	0.51	0.0	0.5	0.49	0.49	0.49	0.49	0.49	0.49
0.5	0.49	0.49	0.51	0.49	0.49	0.5	0.5	0.49	0.5	0.5	0.5	0.5	0.0	0.5	0.5	0.49	0.5	0.49	0.49
0.5	0.49	0.5	0.49	0.5	0.5	0.49	0.5	0.5	0.49	0.5	0.49	0.49	0.5	0.0	0.5	0.5	0.49	0.49	0.49
0.49	0.49	0.5	0.49	0.49	0.49	0.5	0.5	0.49	0.5	0.49	0.49	0.49	0.5	0.5	0.0	0.49	0.49	0.49	0.49
0.49	0.49	0.48	0.49	0.5	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.5	0.5	0.49	0.0	0.49	0.49	0.49
0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.5	0.49	0.5	0.49	0.49	0.49	0.49	0.49	0.0	0.49	0.49	0.49
0.49	0.49	0.49	0.48	0.5	0.49	0.49	0.5	0.48	0.49	0.49	0.49	0.49	0.5	0.49	0.49	0.49	0.49	0.49	0.0

APPENDIX B

ADDITIONAL INFORMATION ON

CHAPTER 5

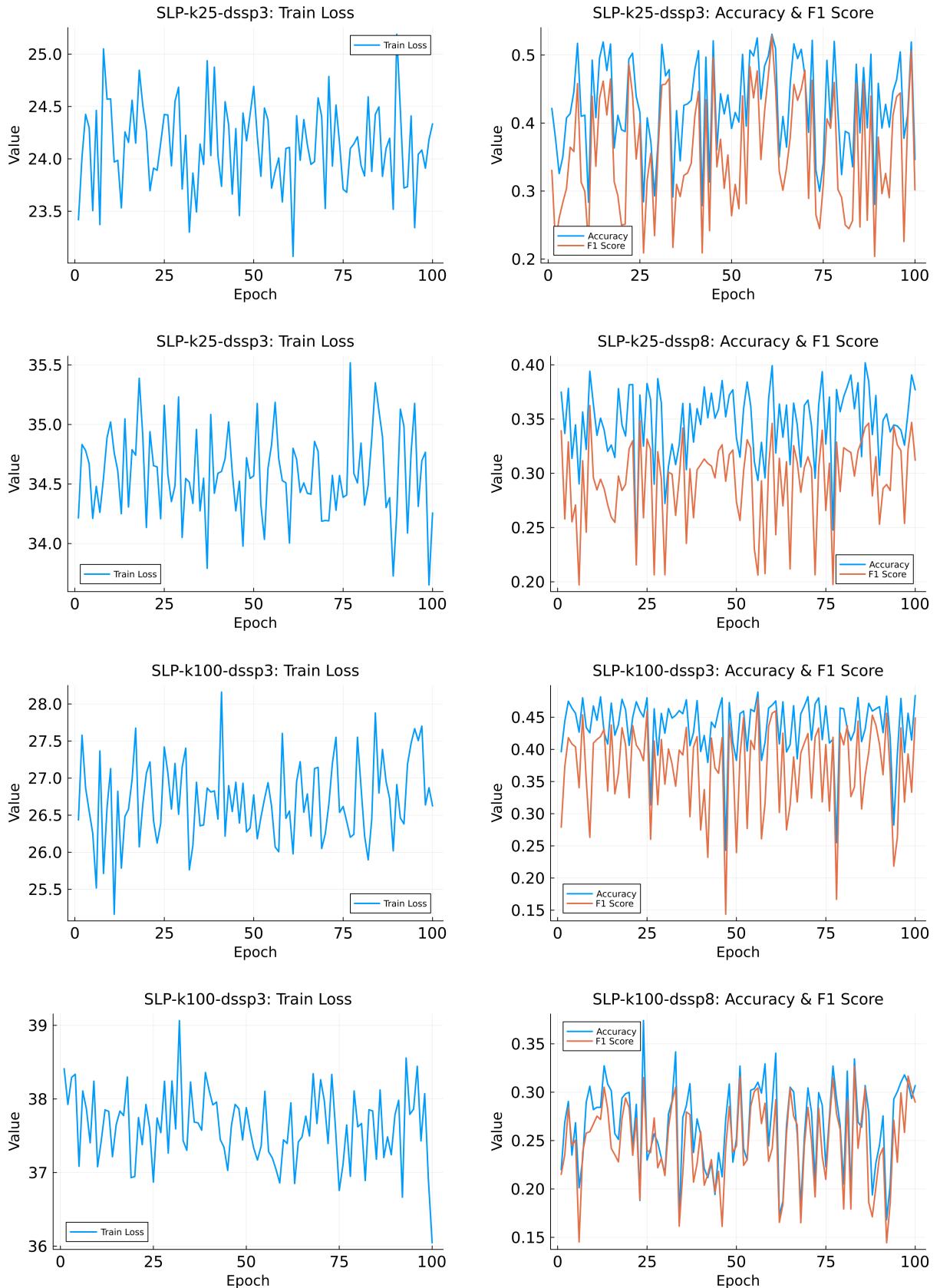


Figure B.1: Performance of training procedure of the SLP on 60 % of NetSurf 2.0's training data, 0.8-0.2 training-test data split. Tested on 3 and 8 secondary structure classifications (dssp3 and dssp8 respectively) and with different kinds of hyperdimensional embeddings made with neighborhood-encoding ($n = 25$ and $n = 100$).

B. Additional information on chapter 5

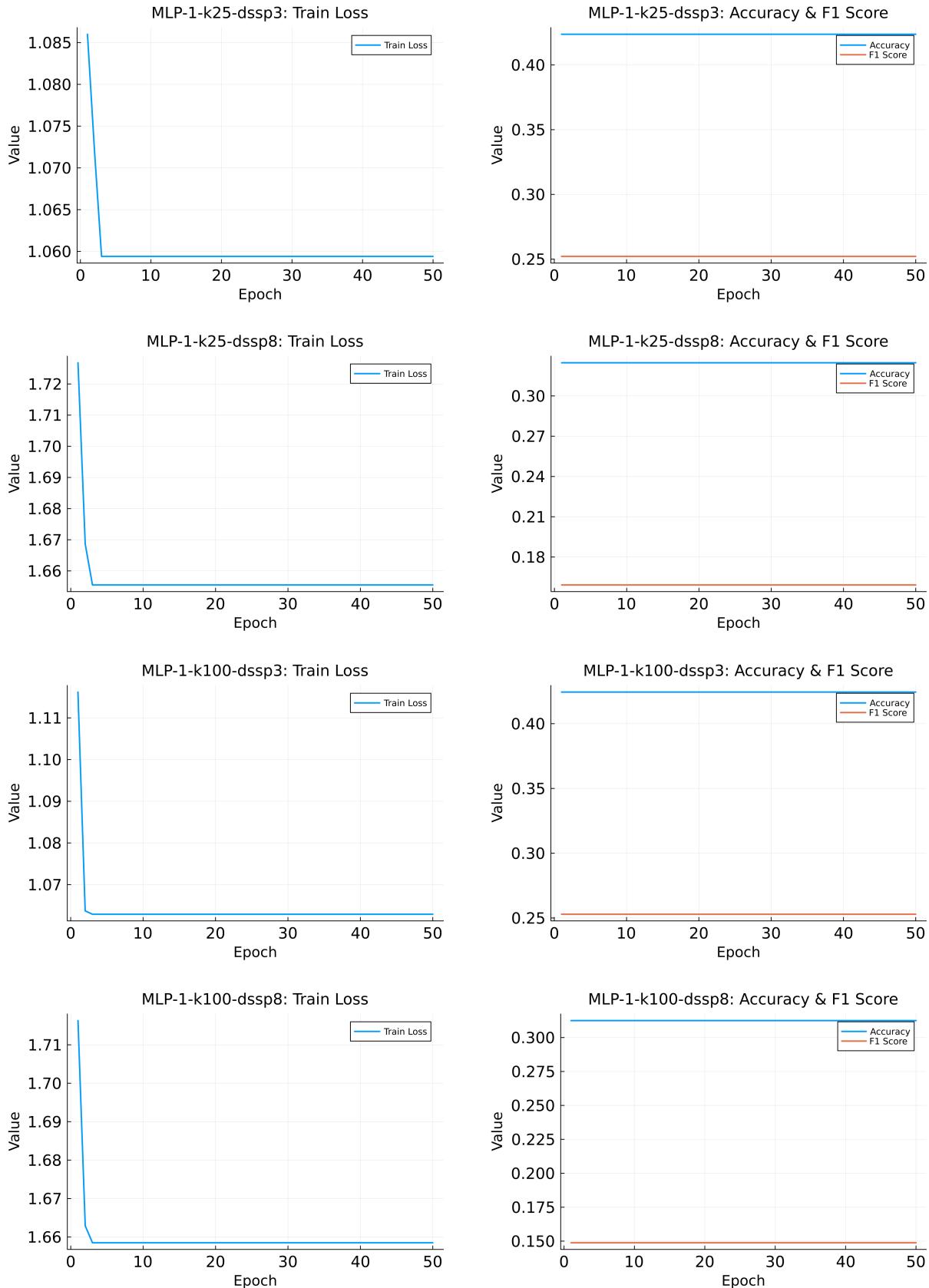


Figure B.2: Performance of training procedure of the 1-layer MLP on 60 % of NetSurf 2.0's training data, 0.8-0.2 training-test data split. Tested on 3 and 8 secondary structure classifications (dssp3 and dssp8 respectively) and with different kinds of hyperdimensional embeddings made with neighborhood-encoding ($n = 25$ and $n = 100$).