FACULTY OF BIOSCIENCE ENGINEERING

HYPERDIMENSIONAL COMPUTING FOR PROTEIN LANGUAGE MODELING

...

Michael Fatjanov
Student ID: ...

Supervisor(s): Prof. Dr. Bernard De Baets

A dissertation submitted to Ghent University in partial fulfilment of the requirements for the degree of master in Bioinformatics.
Academic year: 2022-2023

GHENT UNIVERSITY

Gent, FILL IN THE DATE

The promotor,

Prof. Dr. Bernard De Baets

The author,

Michael Fatjanov

ACKNOWLEDGEMENTS

Thank you, all of you!

CONTENTS

SAMENVATTING

nederlandse samenvatting

SUMMARY

insert english summary here...

1. INTRODUCTION

1.1 outline

1.2 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 cytodine 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

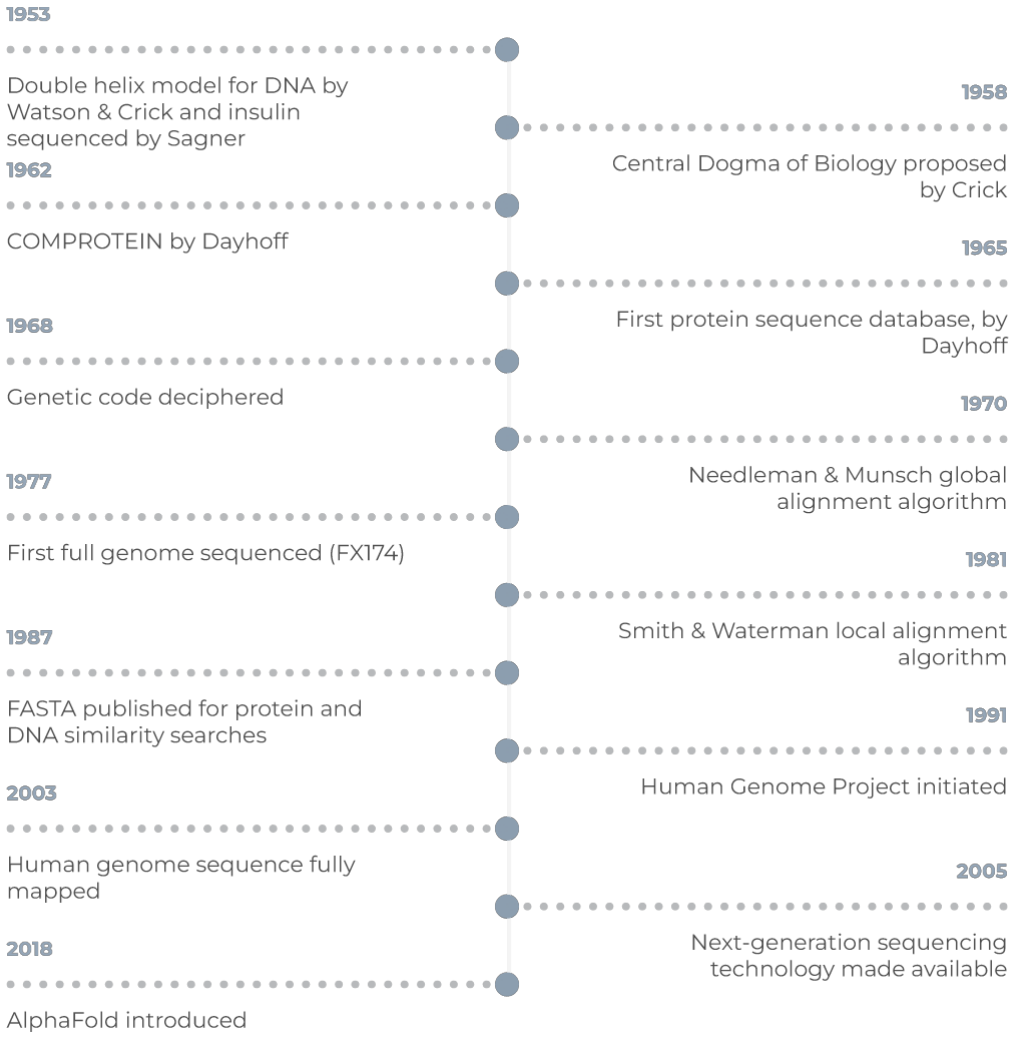


Figure 1.1: A brief timeline of important developments in bioinformatics.

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 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, 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.

Further in 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 & Wun-

sch [9] for global alignments and that of Smith & Waterman [10] for local alignments from the 1970s solved this issue and alignment 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 and so 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.

1.3 Protein biology

1.3.1 Protein structure levels

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.

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.

1.3.2 Computational methods for protein research

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 succes 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 representations which computers can learn from. This is now known as the field of protein language modeling which is more thoroughly discussed in section 1.4.

1.4 State-of-the-art 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. Well-known NLP methods of this kind 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]. Despite the simplicity of these tasks, it is found to develop interesting capabilities as the scale of the model increases together with very little training on a specific task, now mostly referred to as *few-shot learning*.

These deep learning methods also show promise in the field of protein biology with the most notable latest projects being TAPE [34], ProtTrans [35] and Meta AI's ESM-2 [36], one of the most recent and largest protein language models to ever have been developed at the time of writing. ESM-2 has shown to accurately capture evolutionary information and to perform well on structure prediction tasks. It consists of transformer protein lan-

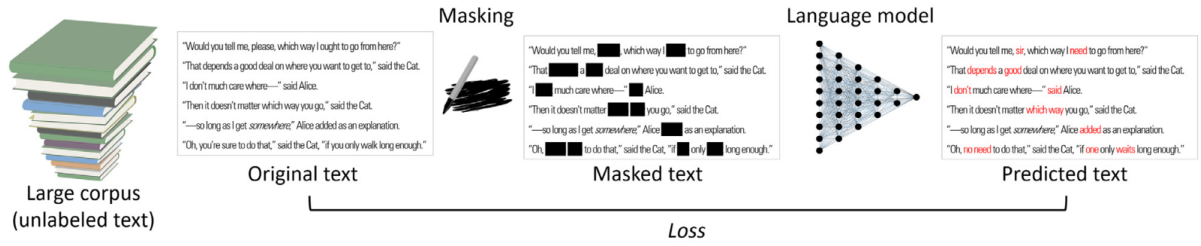


Figure 1.2: Demonstration of a self-supervised masked language learning model. A fraction of the unlabeled text is masked randomly and the language model will attempt to predict these masked tokens. The loss function is a method to asses the performance of the model on its predictions

guage models with up to 15 billion parameters trained on 65 million unique sequences. Like with natural language models, it has been observed that larger models yield consistent improvement and that the performance does not seems to saturate with even the largest models [30]. The amount of computing power needed to train such models and to perform predictions with them rises exponentially with the number of parameters, close to the point that modern computers cannot sustainably handle anymore. For example, it is estimated that it would take at least 34 days with 1024 NVIDIA A100 GPUs to train the 175 billion parameter GPT-3 model with 300 billion tokens [37]. And although, these developments in computational modeling are very impressive, they are very costly too and cause a substantial strain on our environment in terms of the energy and materials needed to sustain these systems.

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 [38], 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 [39], Spatter Code [40] 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 *brocolli*. 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 *brocolli* 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.

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

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.

Addition

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

is similar to the input vectors.

$$X = X_1 + X_2 + \cdots + X_n \tag{2.2}$$

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.

$X_1 =$	$+1$	-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	$+1$
$X_1 + X_2 + X_3 + X_4 =$	0	-1	$+1$	$+1$	-1	0

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. [41] 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.

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

$$X_1 + X_2 = X = X_2 + X_1 \tag{2.3}$$

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.

$$Z = X * Y \tag{2.4}$$

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

$X =$	1	0	1	1	0	0
$Y =$	1	1	0	1	0	1
$X * Y =$	0	1	1	0	0	1

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

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

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.

$$A = Z * (X + Y) = XZ + YZ \tag{2.6}$$

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 [42] 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.

$X =$	1	0	1	1	0	0
$\Pi(X) =$	0	1	0	1	1	0

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.

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

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

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

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. We shall illustrate some applications and examples. From here, all implementations are written in the programming language *Julia* [43] unless noted otherwise. *Julia* makes it possible to write incredibly efficient programs whilst still being a high-level and interpreter-based programming language with many packages suited for mathematical operations included. Also note that all examples have been implemented with binary hyperdimensional vectors unless stated otherwise. However with minimal changes, bipolar vectors could also be applied. All source code is provided in the appendices.

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

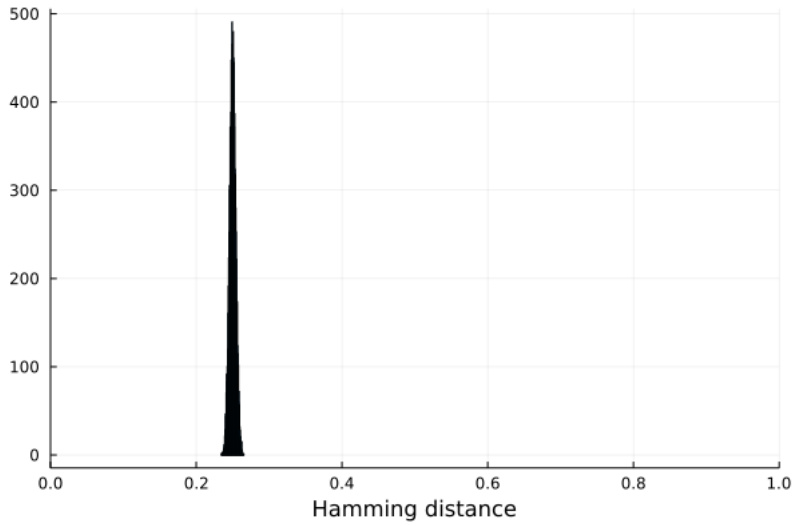


Figure 2.1: 10,000 cases of random 10,000-dimensional binary vectors are made and each time implemented following equation 2.9. The resulting Hamming distances between A and A' are then plotted in a histogram.

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.

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[44] 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 descricptors. 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:

$$C = E + F + T \tag{2.10}$$

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:

$$C = G * E + S * F + L * T \tag{2.11}$$

After all these procedures, there are 101 animals with each a 10,000-dimensional vector as a feature. To effectively analyze this data, a principal component analysis (PCA) has been applied to essentially reduce the 10,000 features into 2. This projection can then be easily shown in a 2D plot, resulting in figure 2.2. This shows 3 distinguishable clusters of animal classes that make sense from an evolutionary standpoint. The model could be easily improved on this part by including more features that should show more separation of these classes.

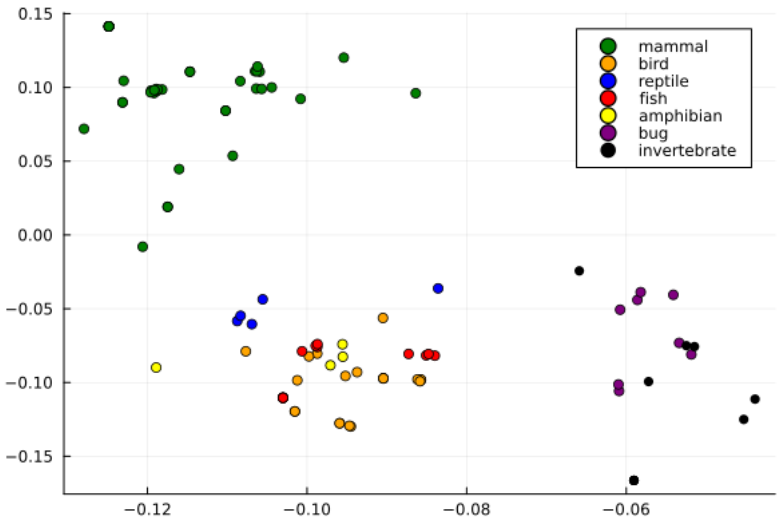


Figure 2.2: Scatter-plot of the first two principal components (PCs) of a 101×10,000 matrix containing hyperdimensional vectors for every animal in the zoo dataset after a PCA procedure. These PCs account for roughly 48 % of the variance.

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:

$$B = P + C + K \tag{2.12}$$

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 invertebrate 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[45]. 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.

$$ABC = A * \Pi(B) * \Pi(\Pi(C)) \tag{2.13}$$

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

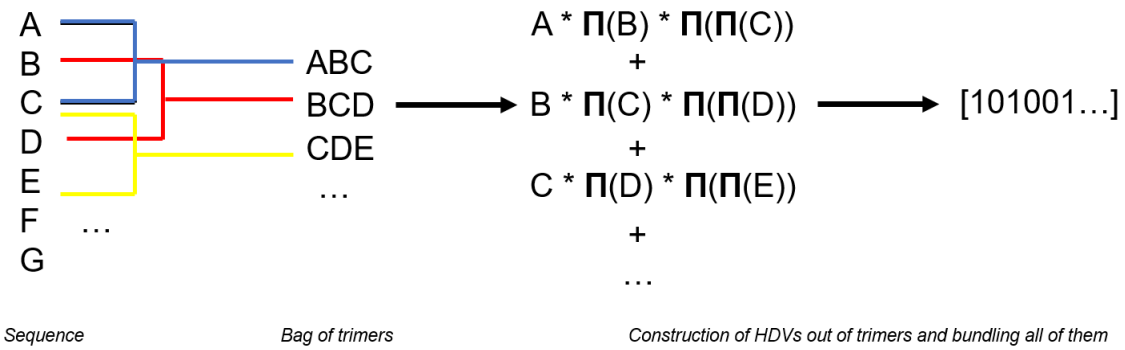


Figure 2.3: Overview of operations done to obtain an HDV of a protein sequence for the example concerning the real peptide dataset. First, a random HDV is generated for every amino acid. Next, a peptide sequence is to be considered as a bag of trimers. A vector representing a trimer is generated by binding 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.

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.

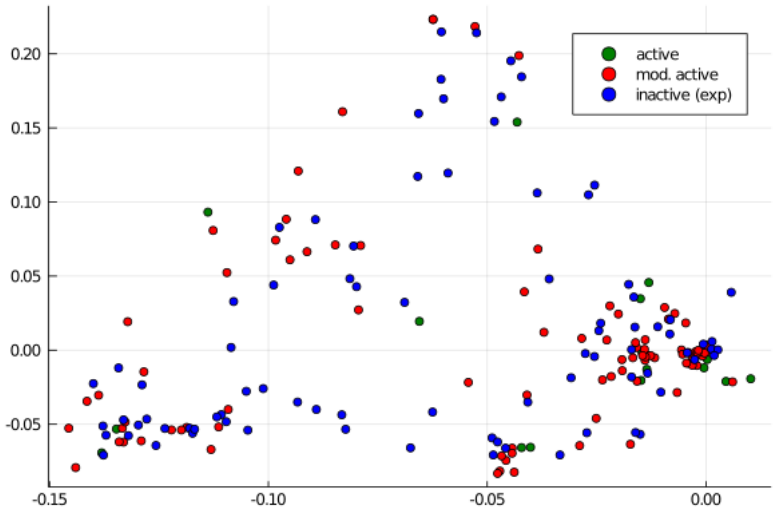


Figure 2.4: Scatter-plot of the first two principal components, decomposing the 10,000-dimensional vectors for every peptide into 2. These PCs account for roughly 5 % of the total variance

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 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 FOR AMINO ACID ENCODING

3.1 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 cases such as natural language processing, where it is usually assumed that a letter does not have varying degrees of similarities to other letters in the alphabet. For protein language modeling, however, this assumption may be suboptimal since some amino acids are chemically more similar to each other than to others. We can already estimate *via* many kinds of distance measures physicochemical distances between amino acids based on their physicochemical properties [46] such as volume, polarity, chemical groups etc. 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 the physicochemical information, we encoded biological information of amino acids into hyperdimensional space using amino acid embeddings from ESM-2 [36] by simple matrix multiplications.

Instead of relying on 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 ($2^{10,000x20}$), making exhaustive search methods computationally infeasible. Thus, more efficient algorithms are needed to solve this problem such as genetic algorithms [47]. GAs, a subfield of evolutionary algortihms, 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. 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.

3.2 Encoding proteins into hyperdimensional vectors

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 embeddings on the level of amino acids, we propose a novel encoding technique within the hyperdimensional computing framework. It encodes interactions of a given amino acid in a sequence to other amino acids in its neighborhood n for a given length n. This method thus tries to learn and encode information about an amino acid within a sequence in an unsupervised manner.

3.3 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

The last layer of the 3 billion-parameter ESM-2 model [36] of every amino acid was extracted, resulting in 1024-dimensional real-valued embeddings for every amino acid. To extend these into hyperdimensionality, a simple matrix multiplication has been employed: $A_{1x1024} \times B_{1024x10,000} = C_{1x10,000}$ 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 acids are reduced in dimensionality *via* PCA into 2 dimensions and then plotted as seen in figure 3.2.

A *BLOSUM62* substitution matrix [48] and Grantham's distance matrix [49] were considered as 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. A genetic algorithm was implemented with the *Evolutionary.jl v0.11.1* [50] package in *Julia*. MIGHT CHANGE 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 respectively unusually high and low because we want to emphasize the bit-flipping and avoid recombination of vectors. HOW TO ANALYZE

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 window are made *via* binding and then all encoded into one vector *via* bundling as shown figure 3.1. Our neighborhood-encoder was tested on the human reference proteome in UniProt, entry *UP000005640*, containing 20591 proteins starting with both random vectors and extended ESM-2 vectors. 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 **??**.

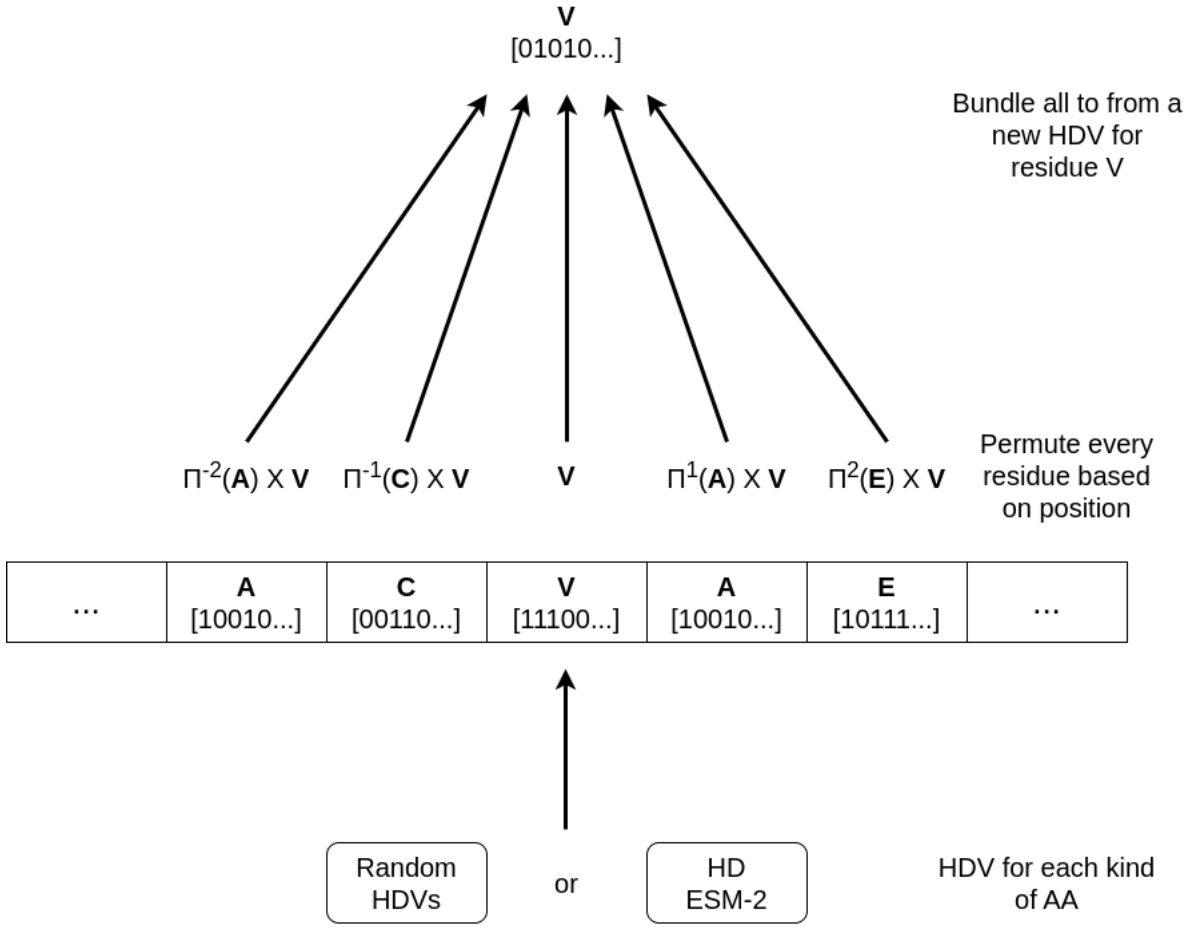


Figure 3.1: A simple demonstration of our amino acid encoder. First, HDVs are generated for every kind of amino acid, randomly or by extending ESM-2 embeddings. It considers an amino acid and all amino acids in a predetermined neigbourhood (here $n = 2$). It produces all possible interactions of the 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.

3.4 Results

At first glance, there is not much to spot in the PCA decomposition of the extended 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. This may be shown more clearly when used and compared in real-world problems.

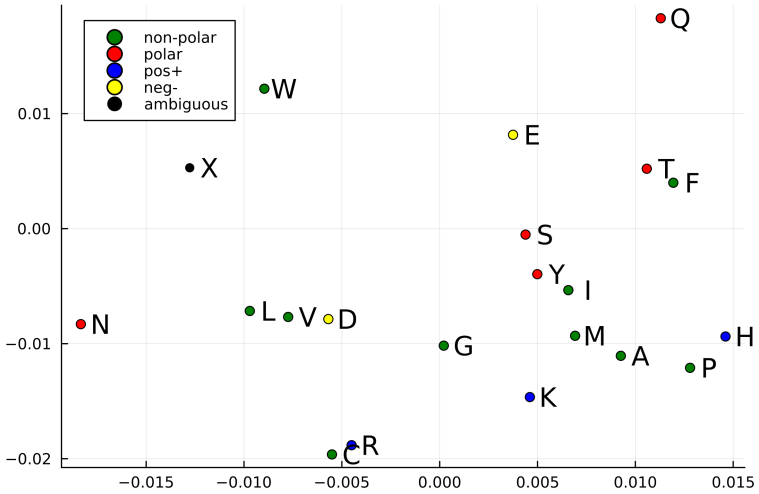
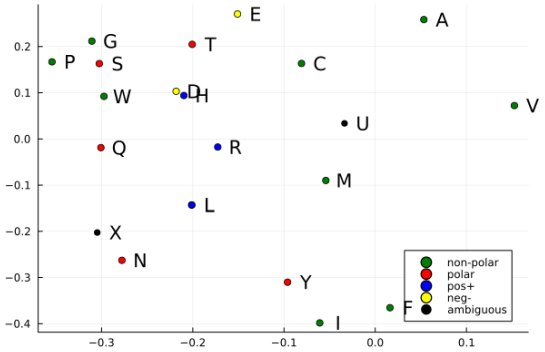


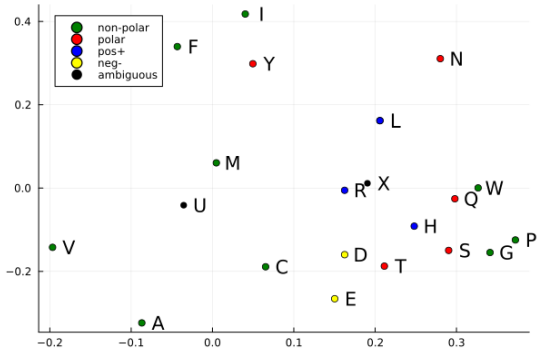
Figure 3.2: Scatter-plot of the first two principal components of ESM embeddings extended into hyperdimensionality. These PCs account for roughly 22 % of the total variance. The amino acids are annotated and colored based on their chemical property of polarity.

Looking at the average neighborhood-encoded amino acids starting from ESM-2 embeddings (figures 3.3a and 3.3b), 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 could indicate that the first four amino acids before and after a residue are the most crucial.

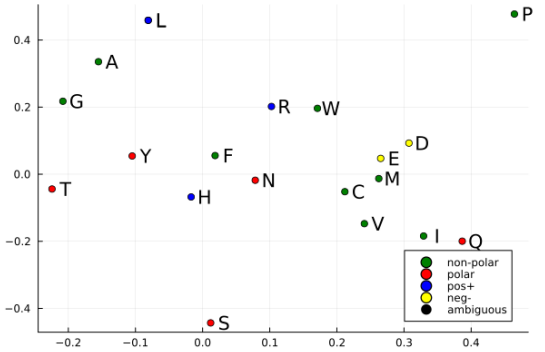
The PCA scatter plots generated from random vectors (figures 3.3c and 3.3d) slightly different results as compared to the ones starting from ESM embeddings. These PCAs also capture less variance which may indicate that encoding biological information/similarities into the vectors beforehand might be necessary. For $k = 4$, some typicalities are noticed such as the close grouping of the negatively charged amino acids and the triangular grouping of H, L and R. Nonetheless, for $k = 50$ we see results deviating from all the PCA scatter plots of other neighborhood-encoded amino acids.



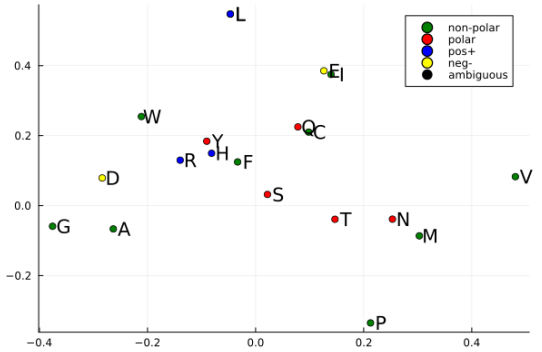
(a) Average neighborhood-endcoded amino acids starting from extended ESM-2 embeddings. These PCs account for roughly 21 % of the total variance. $n = 4$



(b) Average neighborhood-endcoded amino acids starting from extended ESM-2 embeddings. These PCs account for roughly 21 % of the total variance. $n = 50$



(c) Average neighborhood-endcoded amino acids starting from random hyperdiemnsional vectors. These PCs account for roughly 11 % of the total variance. $n = 4$



(d) Average neighborhood-endcoded amino acids starting from random hyperdiemnsional vectors. These PCs account for roughly 11 % of the total variance. $n = 50$

Figure 3.3: 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 extended 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.

In Rives *et al.* (2011) [51], 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. First, the amount of data we used is not comparable to theirs: our method 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, which is more thoroughly discussed in section 4.1, meaning long-range dependencies of a residue

will saturate the hyperdimensional vector depending on the range. Due to the intrinsic randomness of hyperdimensional computing, it is also prone to capture some amount of noise. UMAP projections were also made, shown in section A in the appendices, but these did not reveal any new information. Applying these HDVs to other real-world problems should reveal the performance and usefulness of the neighborhood-encoding.

4. CASE STUDY: PHALP DATASET

To implement and evaluate hyperdimensional computing in real-life problems, the potential of hyperdimensional computing will be evaluated on the PhaLP dataset [52] for 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).

All 17356 unique protein sequences were embedded into hyperdimensional vectors. This took only a few minutes for every method on a consumer laptop.

4.1 Type classifcation

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. They embedded each protein sequence *via* SeqVec [53] and trained a random forest classifier 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 types based on their sequence using several methods. As of March 2023, the latest version of the PhaLP database, *v2021_04*, has been used to test our models.

Embedding of sequences into hyperdimensional vectors

First, we used several sequence encoding techniques tested on several kinds of base vectors. In chapter 2.3, a method of embedding sequences of amino acids has already been discussed. Here, a sequence of amino acids is considered to be a bag of k-mers. Within a k-mer, the amino acids (presented as randomly generated hyperdimensional vectors) are bonded together with sequential information included. All possible k-mers are all then bundled together, the result is then a hyperdimensional vector representing the whole sequence. We also introduce a novel sequence embedding method within the framework of 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 before bundling. Insert figure The resulting sequences are then visually assessed *via* PCA.

There was no visual difference between the PCA plots for the bag-of-words method and the convolutional method, but there is a clear difference between the different starting embeddings. The sequence embeddings from both the bag-of-words method and the convolutional method made with ESM AA embeddings seem to capture more of the variance between the sequences. To demonstrate hyperdimensional computing as an option for machine learning applications, several methods using this framework have been developed and tested. 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.

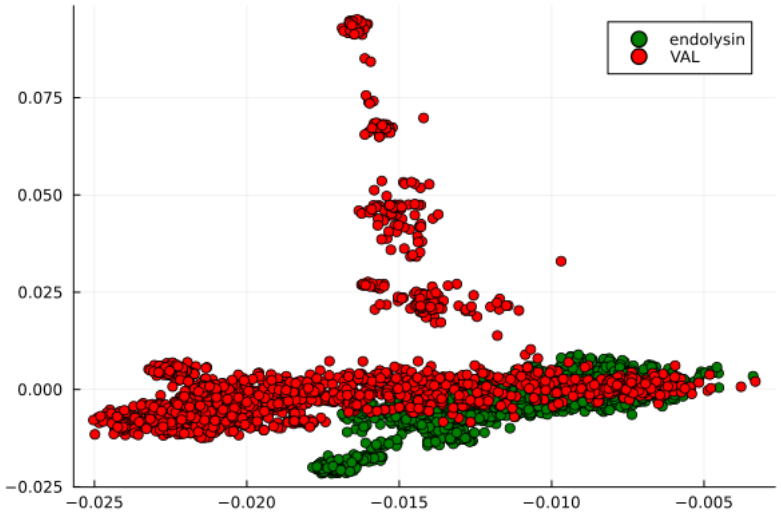


Figure 4.1: Scatter-plot of the first two principal components of the encoded phage lytic proteins. The sequences were encoded via the bag-of-words method starting from random hyperdimensional vectors. Only manually annotated phage lytic proteins were considered and are color-coded based on their type. These PCs account for roughly 7 % of the total variance in the system.

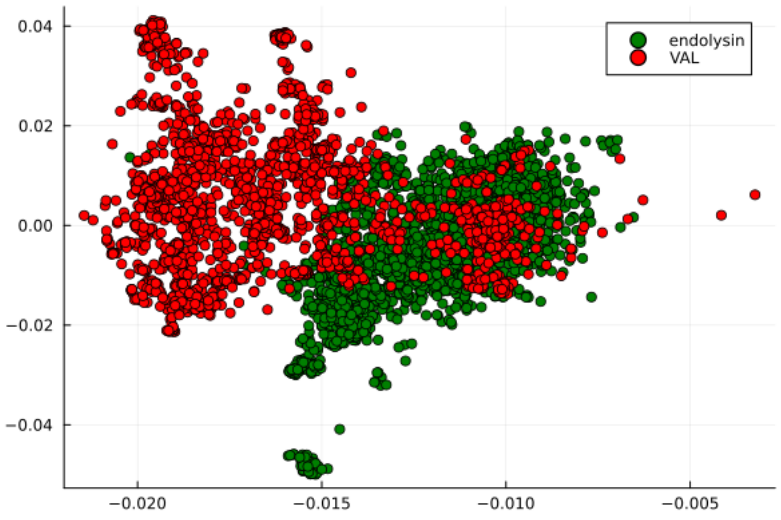


Figure 4.2: Scatter-plot of the first two principal components the encoded phage lytic protein. The sequences were encoded via the bag-of-words method starting from hyperdimensionally-extended ESM embeddings. Only manually annotated phage lytic proteins were considered and are color-coded based on their type. These PCs account for roughly 15.5 % of the total variance in the system.

Naive hyperdimensional addition

As a baseline level, we use the rudimentary HDV classification technique as seen in chapter 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.

Table 4.1: Results of type classifications using the principal classification technique of hyperdimensional computing and an XGBoost classifier 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

It is feasible to learn the classes of every sequence using only operations within the hyperdimensional computing framework. This is done by using the same techniques as in chapter 2.3. Evaluating our model using stratified 10-fold cross-validation results in F1-scores of around 0.14 for every kind of hyperdimensional embedding. This low result is likely due to the possibility of oversaturation of the class vectors.

We can predict the angle between a class vector and a randomly selected vector from said class by $\Theta = \arccos(\binom{2k}{k}/2^{2k})$ with $2k + 1$ equal to the number of sequences in the class [54]. This approximation is valid for bipolar 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 class, 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 class vector becomes to its components. This results in the class vectors not being representative anymore of a given dataset. This equation assumes that the class vector is a bundle of purely random vectors which is not the case for our embeddings; however, it provides us a rough idea about the bundling capacity of a hyperdimensional vector. Thus, using the rudimentary model works only for very small datasets, as seen in the examples in chapter 2.3. So to encode larger datasets, the training algorithm has to be more refined.

Machine learning models with binary hyperdimensional embeddings

The baseline hyperdimensional classification model has been compared to a more established model, the XGBoost classifier. The classification with an XGBoost classifier is done via the default XGBoost classifier from *XGBoost.jl v2.2.5* and is evaluated *via MLJ.jl v0.19.5* with also a stratified 10-fold cross validation. The results with this model for every embedding (provided in table 4.1) are much more comparable to the results of the experiment in the PhaLP paper. This is an indication that hyperdimensional computing can provide 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.

OnlineHD implementation

As an answer for the unsatisfactory results of the rudimentary additive approach in part 4.1, another hyperdimensional computing approach has been assessed for our use case. OnlineHD by A. Hernandez-Cano *et al.* [55] 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 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):

$$\vec{C}_l \leftarrow \vec{C}_l + \eta(1 - cos_l) * \vec{V} \tag{4.1}$$
$$\vec{C}_{l'} \leftarrow \vec{C}_{l'} - \eta(1 - cos_{l'}) * \vec{V} \tag{4.2}$$

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 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. Due to the nature of this model, we cannot constrict our hyperdimensional embeddings to a bipolar or binary nature anymore and the embeddings are then allowed to be real-numbered. Mathematical operations such as multiplications and additions are then assumed to be element-wise.

On top of single-pass model 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 all is then iterated for a given amount of cycles.

To test these algorithms, real-numbered embeddings have to be made from our subject sequences. The same bag-of-words and convolutional approaches as well as the random and ESM base vectors are also applied here, but all starting from random vectors with values in $[-1, 1]$. These embeddings were assessed *via* a scatter-plot of the two first principal components of their PCA projection. (insert firgures)

Since these algorithms are only available as PyTorch implementations, implementations in Julia have been made here. A stratified 10-fold cross validation of these models with our subject sequences has been performed. The learning rate is set at 0.035 and the amount of retraining iterations is set to 120. The results are provided in table 4.2. At first, there is generally a substantial increase in the performance of this model compared to the naive additive model. The single-pass model seems to have widely varying results depending on the type of embeddings used, with the bag-of-words embeddings generally performing better than the convolutional embeddings

and also the ESM-based embeddings performing better than random base vectors. Iterative retraining of the model also 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 a 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 language modeling, whilst still being much more efficient than the commonly used machine learning models. The drawback of the model is that we cannot use hyper-efficient bit-operations anymore, which limits its efficiency compared to the binary nature of the additive model.

Table 4.2: Results of type classifications using implementations of OnlineHD with several kinds of embeddings

F1-scores	**BoW/random**	**BoW/ESM**	**Convolutional/random**	**Convolutional/ESM**
Single-pass OnlineHD	0.8901	0.9214	0.7793	0.8400
Iterative OnlineHD	0.9487	0.9757	0.9486	0.9670

4.2 Domain classification

possible domain classification implementation, may or may not be interesting, but would be again a sequence classification (bit more complex perhaps) that we already tackled

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.

[2] FRANCIS CRICK. Central dogma of molecular biology. *Nature*, 227(5258):561–563, August 1970.

[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.

[4] F. Sanger and E. O. P. Thompson. The amino-acid sequence in the glycyl chain of insulin. 1. the identification of lower peptides from partial hydrolysates. *Biochemical Journal*, 53(3):353–366, February 1953.

[5] JOSEPH S. FRUTON. Early theories of protein structure. *Annals of the New York Academy of Sciences*, 325(1):1–20, 1979.

[6] P. Edman and G. Begg. A protein sequenator. *European Journal of Biochemistry*, 1(1):80–91, March 1967.

[7] J Wesley Leas. *Proceedings of the December 4-6, 1962, Fall Joint Computer Conference*. ACM, 1962.

[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.

[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.

[10] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[11] David J. Lipman and William R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[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.

[13] Michio Murata, Jane S Richardson, and Joel L Sussman. Simultaneous comparison of three protein sequences. *Proceedings of the National Academy of Sciences*, 82(10):3073–3077, 1985.

[14] Desmond G Higgins and Paul M Sharp. Clustal: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, 1988.

[15] Robert C Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5):1792–1797, 2004.

[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.

[17] Francis HC Crick. The origin of the genetic code. *Journal of molecular biology*, 38(3):367–379, 1968.

[18] F Sanger, G M Air, B G Barrell, N L Brown, A R Coulson, C A Fiddes, C A Hutchison, P M Slocombe, and M Smith. Nucleotide sequence of bacteriophage phi X174 DNA. *Nature*, 265(5596):687–695, February 1977.

[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.

[20] O.B. Ptitsyn. How does protein synthesis give rise to the 3d-structure? *FEBS Letters*, 285(2):176–181, 1991.

[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.

[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.

[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.

[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.

[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.

[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.

[27] Tristan Bepler and Bonnie Berger. Learning the protein language: Evolution, structure, and function. *Cell Systems*, 12(6):654–669.e3, June 2021.

[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, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal 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.

[29] Ewen Callaway. After AlphaFold: protein-folding contest seeks next big breakthrough. *Nature*, 613(7942):13–14, December 2022.

[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.

[31] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Pentti Kanerva. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation*, 1(2):139–159, jun 2009.

[39] T.A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641, 1995.

[40] Pentti Kanerva. The spatter code for encoding concepts at many levels. pages 226–229, 1994.

[41] 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.

[42] Lulu Ge and Keshab K. Parhi. Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine*, 20(2):30–47, 2020.

[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.

[44] UCI Machine Learning. Zoo animal classification, 2016.

[45] 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.

[46] P.H.A. Sneath. Relations between chemical structure and biological activity in peptides. *Journal of Theoretical Biology*, 12(2):157–195, November 1966.

[47] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1998.

[48] 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.

[49] R. Grantham. Amino acid difference formula to help explain protein evolution. *Science*, 185(4154):862–864, September 1974.

[50] Evolutionary.jl: a Julia package for evolutionary & genetic algorithms.

[51] 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.

[52] 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.

[53] 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.

[54] Tao Yu, Yichi Zhang, Zhiru Zhang, and Christopher De Sa. Understanding hyperdimensional computing for parallel single-pass learning, 2022.

[55] 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.

APPENDIX A

ADDITIONAL INFORMATION ON CHAPTER 3

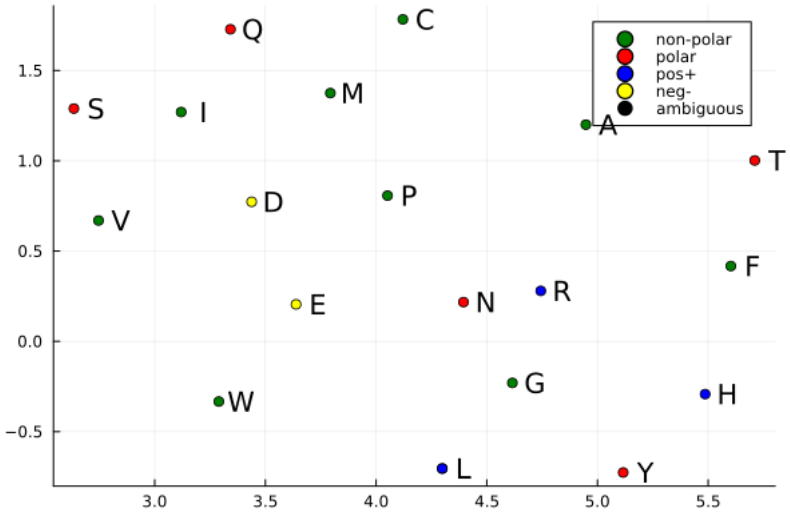


Figure A.1: Scatter plot of a two-dimensional UMAP projection of the average amino acid HDVs with neighborhood-information of $k = 4$ encoded, trained on the human reference proteome. The amino acids are annotated and colored based on their chemical property of polarity. These were made starting from random hyperdimensional vectors.

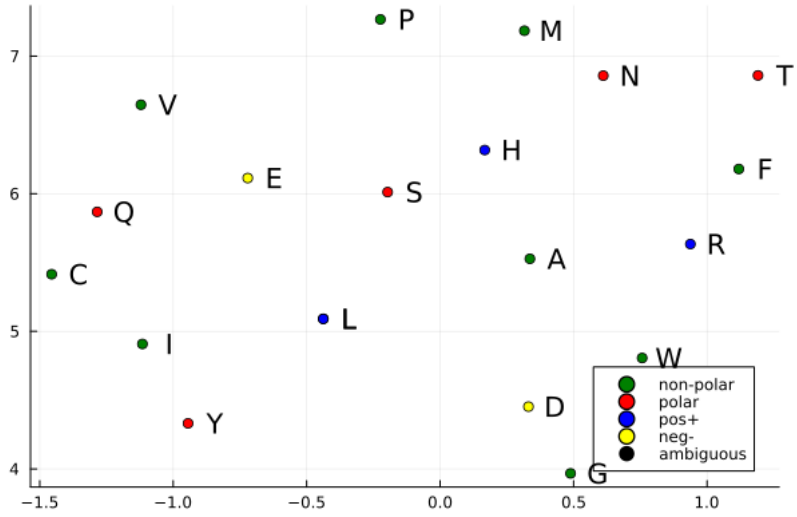


Figure A.2: Scatter plot of a two-dimensional UMAP projection of the average amino acid HDVs with neighborhood-information of $k = 50$ encoded, trained on the human reference proteome. The amino acids are annotated and colored based on their chemical property of polarity. These were made starting from random hyperdimensional vectors.

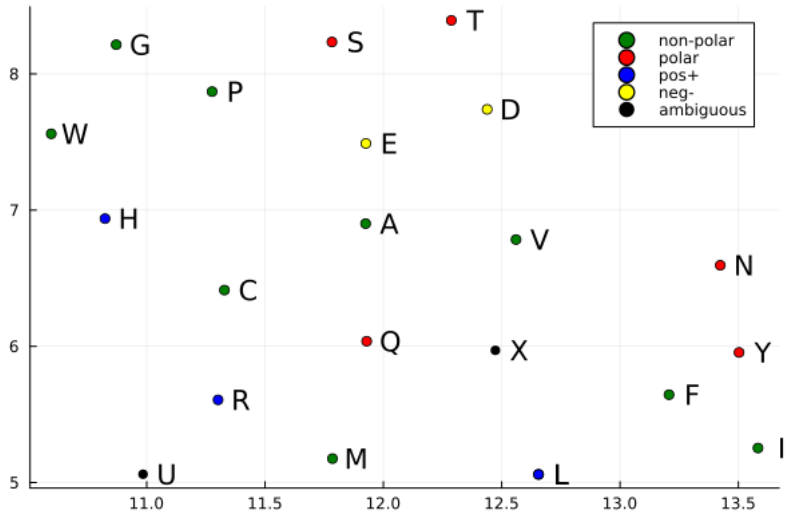


Figure A.3: Scatter plot of a two-dimensional UMAP projection of the average amino acid HDVs with neighborhood-information of $k = 4$ encoded, trained on the human reference proteome. The amino acids are annotated and colored based on their chemical property of polarity. These were made starting from extended ESM embeddings.

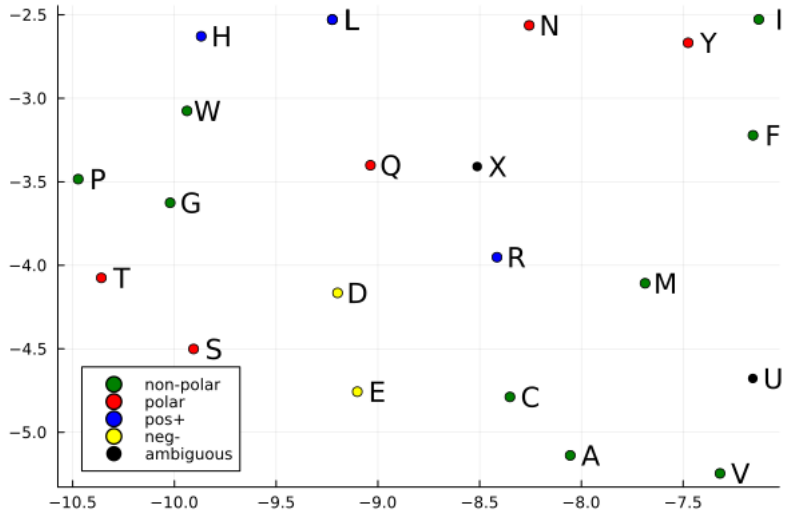


Figure A.4: Scatter plot of a two-dimensional UMAP projection of the average amino acid HDVs with neighborhood-information of $k = 50$ encoded, trained on the human reference proteome. The amino acids are annotated and colored based on their chemical property of polarity. These were made starting from extended ESM embeddings.

APPENDIX B

ADDITIONAL INFORMATION ON CHAPTER 4