Department of Mathematics
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Mathematics in Data Science

Improve peptide-MHC binding prediction with machine learning by leveraging pre-trained protein language models

Théomé Borck

Department of Mathematics

TECHNISCHE UNIVERSITÄT MÜNCHEN

–

Master's Thesis in Mathematics in Data Science

Improve peptide-MHC binding prediction with machine learning by leveraging pre-trained protein language models

–

Vorhersage der Peptid-MHC-Bindung mit maschinellem Lernen durch Nutzung vorab trainierter Protein-sprachmodelle

Author	Théomé Borck (MSc. Mathematics in Data Science)
Supervisors	Prof. Dr. Massimo Fornasier (TUM Faculty of Mathematics) Martin Perier-Dulhoste, ML Research Engineer (InstaDeep)
Submission date	1st November, 2021

I hereby declare that this master's thesis is my own work and that no other sources have been used except those clearly indicated and referenced.

Munich, 1st November, 2021 Théomé Borck

Théomé Borck

Acknowledgements

I would first like to deeply thank my thesis advisor, Martin Perier-Dulhoste, Machine Learning Research Engineer at InstaDeep Paris, for his assistance and dedicated involvement in every step throughout this master's thesis. He has been of great advice and expertise, from software development to modeling. His constant feedback and insights helped me serenely conduct my research over the last six months.

I would also like to thank Dr. Massimo Fornasier, professor in the Department of Mathematics at Technische Universität München, for his supervision and for giving me the opportunity to pursue this master's thesis at InstaDeep. His door was always open whenever I had a question about my research or writing.

On a final note, I would like to thank all the members of my team at InstaDeep. I am grateful for their passionate participation and continuous input, which pushed me to sharpen my thinking, brought my work to a higher level and successfully led this research initiative to its term.

Abstract

Localized on the cell surface, the major histocompatibility complex (MHC) presents peptides to T-cells in order to develop specific immune responses. MHC classes I and II play a key role in identifying the internal (i.e. cancer) and external (i.e. pathogen) threats in one's body. Previous work has shown that the binding between an antigenic peptide and its MHC molecule is one of the most crucial steps towards effective antigen presentation and the immune system activation. Hence, an accurate binding predictor will enable the development of therapies such as personalized cancer vaccines. Several works have recently used computational ML-based approaches to tackle this binding problem. In parallel, language models pre-trained on vast amounts of unsupervised protein data have succeeded in capturing contextual embeddings of protein sequences. These embeddings encode valuable information about biochemical and biophysical properties of the different amino acids, as well as structure information. Using open-source datasets, the goal of this thesis is twofold. On one hand, this work explores linear modeling methods and more elaborated deep learning methods to predict the peptide-MHC binding from amino acids' sequences. On the other hand, it aims at leveraging pre-trained protein language models on this given problem, by bringing a consistent proxy for structural protein features into models.

Der auf der Zelloberfläche lokalisierte Haupthistokompatibilitätskomplex (MHC) präsentiert den T-Zellen Peptide, um eine spezifische Immunantwort zu entwickeln. Die MHC-Klassen I und II spielen eine Schlüsselrolle bei der Erkennung von inneren (z.B. Krebs) und äußeren (z.B. Krankheitserreger) Bedrohungen im Körper. Frühere Arbeiten haben gezeigt, dass die Bindung zwischen einem antigenen Peptid und dem zugehörigen MHC-Molekül einer der wichtigsten Schritte für eine effektive Antigenpräsentation und die Aktivierung des Immunsystems ist. Ein genauer Bindungsprädiktor wird daher die Entwicklung von Therapien wie personalisierte Krebsimpfstoffe ermöglichen. In mehreren Arbeiten wurden kürzlich computergestützte ML-Ansätze verwendet, um dieses Bindungsproblem zu lösen. Parallel dazu haben Sprachmodelle, die auf großen Mengen unüberwachter Proteindaten trainiert wurden erfolgreich kontextuelle Einbettungen von Proteinsequenzen zu erfassen. Diese Einbettungen kodieren wertvolle Informationen über biochemische und biophysikalische Eigenschaften der verschiedenen Aminosäuren sowie Strukturinformationen. Mit der Verwendung von Open-Source-Datensätzen verfolgt diese Arbeit ein zweifaches Ziel. Einerseits erforscht diese Arbeit lineare Modellierungsmethoden und ausgefeiltere Deep-Learning-Methoden zur Vorhersage der Peptid-MHC-Bindung aus Aminosäuresequenzen. Andererseits zielt sie darauf ab, bereits trainierte Protein-Sprachmodelle für dieses Problem zu nutzen, indem wir einen konsistenten Proxy für strukturelle Proteinmerkmale in Modelle einbringen.

Contents

1 Introduction

In 1972, in his acceptance speech for the Nobel Prize in Chemistry, Christian Anfinsen made a prediction: it should, in principle, be possible to determine a protein's three-dimensional shape based solely on the one-dimensional string of molecules that define it. A solution to this puzzle, known as the "protein folding problem", has very recently been found with AlphaFold, an algorithm notably built with a cutting-edge neural network architecture: the Transformer. This breakthrough emerged from intensive research around a class of statistical methods, machine learning. Over the last decade, the latter brought outstanding performances in natural language processing and vision, mainly thanks to advances in software, hardware and data collection. Proteins, as a sequence of amino acids, can be viewed precisely as a language. As such, language models, notably based on Transformer and pre-trained on vast amounts of unsupervised protein data have succeeded in capturing biochemical and biophysical properties from the amino acids into contextual embeddings of protein sequences.

The problem at hand in this thesis slightly differs from the protein folding prediction above. It consists in modeling the interaction between two proteins: a peptide and a major histopatibility complex (MHC), based on their uni-dimensional sequences. Previous work has shown that this binding interaction is a fundamental step in antigen presentation, a process that triggers an immune response in the human body. Boosted by the recent success of mRNA, personalized vaccine's applications against cancer attempt to activate the immune response through the antigen presentation pathway. Therefore, accurate antigen presentation prediction plays a major role in their synthesis. This thesis aims at exploring how computational methods from natural language processing, as well as transfer learning, can help build predictive models towards peptide-MHC binding and peptide presentation, based on high-resolution mass spectrometry data. First, the biological and mathematical motivation behind the problem at hand are introduced (2). In particular, related works are explored to raise research hypotheses. Secondly, the required tools to conduct this work are presented (3). Then, adequate modeling techniques from the fields of machine and deep learning are explained (4). Specifically, the Transformer architecture and concepts around transfer learning are deeply investigated. Finally, results from the application of these methods are shown and discussed (5), and future initiatives are presented (6).

1.1 Description of the company

This master's thesis has been carried out in the context of a 6-months internship at InstaDeep [2021], in the Paris-based offices. Founded in 2014 and with expertise in both machine learning research and business solutions' deployments, **InstaDeep** is now an EMEA leader in decision-making systems for the Enterprise, powered by artificial intelligence (AI). It tackles complex challenges across a wide range of sectors and industries, such as mobility, biology, energy or electronic design. For instance, concrete

applications vary from optimized pattern-recognition in biology to self-learning decision making systems in transport, and some systems are adapted into products such as DeepChain™ or DeepPCB™. Recently named in the 2021 CB Insights' 100 most innovative AI startups, the company has developed collaborations with global leaders in the Artificial intelligence ecosystem, such as Google DeepMind, Nvidia and Intel, and now employs over 130 employees located in London, Paris, Tunis, Lagos, Dubai and Cape Town.

1.2 Scope of the project

The project I have been working for is part of a strategic collaboration [BioNTech, 2020] between the German Maintz-based biotechnology company **BioNTech SE** and InstaDeep. This collaboration is meant to form an AI innovation lab to develop the next generation vaccines and biopharmaceuticals for the treatment of cancer and prevention and therapy of infectious diseases, including Covid-19. More precisely, my team was working on generating insights from public and proprietary meta datasets, as well as anonymized patient data, through the use of machine learning. The final purpose is to identify novel biological targets and predictive biomarkers for the BioNTech's cancer vaccine pipeline development, known as the iNeST (**i**ndividualized **Ne**oantigen **S**pecific **T**herapies against cancer) program.

2 Motivation

2.1 The problem at hand

While this work focuses on mathematics and machine learning applications, precise terms and concepts are needed to correctly understand and consider the problem raised in this thesis and its given challenges. The following sections are in that regard intended to provide a minimal introduction to the biological notions and processes behind the matter in question, from the formation of proteins to what a cancer vaccine might look like in practice. The given definitions and explanations are inspired from general public resources [Khan Academy, 2021; Clancy and Brown, 2008; Koshland and Haurowitz, 2020; O'Connor and Adams, 2010; Osmosis, 2021].

2.1.1 A quick reminder on proteins

Proteins are present in all living organisms and include many essential biological compounds such as enzymes, hormones, and antibodies. They are literally the workhorses in cells and serve as catalysts in every biochemical reaction that occurs in living things. Overall, proteins are the end products of multiple decoding processes that convert the information from the cellular DNA.

DNA In every cell of a living organism, a chemical compound called deoxyribonucleic acid (DNA) encodes the instruction needed for he organism to develop. The DNA molecule is composed of two paired strands – often referred as the double helix –, each of these strands being made of 4 chemical units called nucleotides: adenine (A), thymine (T), guanine (G), and cytosine (C). These units go in two possible pairs: C-G and A-T, and their ordered arrangement encodes the genetic information. The set of all of these bases is called the genome and a human being contains roughly 3 billion base pairs. Much like bits in a computer, it is the lowest level of information encoding in the organism. The pairs can be grouped together in meaningful units of information called genes. The genes contain information on how to create proteins thanks to the gene expression process.

Transcription: from DNA to mRNA Transcription is the first step in gene expression, it occurs in the nucleus and copies out the DNA sequence of a gene in the similar alphabet of RNA – T being replaced by uracile (U). It is performed by enzymes called RNA polymerases, which link nucleotides to form an RNA strand. While DNA cannot leave the nucleus of the cell, the resulting ribonucleic acid (RNA) transcript exits the cell and carries the information needed to build a polypeptide, i.e. a protein or a protein chunk. In eukaryotes like human beings, the transcript of a protein-coding gene then goes through extra processing steps to form messenger-RNA (mRNA). Cells carefully control the transcription and genes are thereby more or less expressed by dif-

ferent cells and different organisms. As a result, this regulates the quantity of produced proteins in a given cell.

Translation: from mRNA to the protein In mRNA, the instructions for building a polypeptide are RNA nucleotides read in group of three, called codons. Each codon (except START and STOP codons) is a specific instruction that encodes one of the 20 amino acids (in the human body), small organic molecules that are the building blocks of a protein. The translation process occurs in the cytoplasm: a ribosome reads the mRNA, acting like a giant clamp which holds all of the players in position, and facilitates both the pairing of bases between the messenger and transfer RNAs, and the chemical bonding between the amino acids. This results in a built polypeptide, which can range from a few dozen to several thousand amino acids in length.

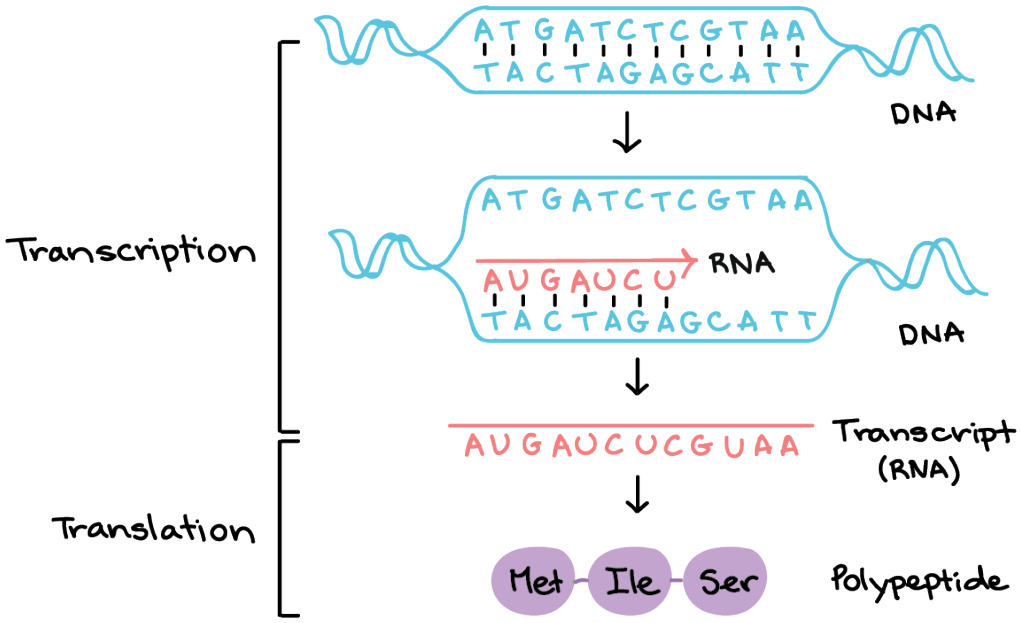

Figure 1: Gene expression.

Source: Khan Academy

In the long run, the central dogma of life can be summarized in a fairly simple way: DNA makes RNA, which in turn makes proteins, as illustrated in Figure 1. Proteins, or polypeptides, are thus chains of amino acids (AAs, or residues), which then spontaneously fold into unique three-dimensional shapes. This 3-D structure and the optional interactions with other proteins carry out the final biological function of the protein in the organism. It is worth noticing that from an information-theory perspective, the protein structure's information is hence contained within its sequence. Four levels of

protein structure can be considered. The ordered sequence of AAs is known as the protein primary structure and the problem in this thesis will mostly consider proteins at this level – statistical methods and machine learning approaches considered in this work are applied to the sequence of AAs. The local interactions between stretches of a polypeptide chain form α-helix and β-pleated sheet structures and this level is called the secondary structure. Tertiary structure is the overall three-dimension folding. One way to characterize this 3-D structure is by a contact map, which describes the pairs of amino acids that are in contact. Eventually, quarternary structure is the orientation and arrangement of sub-units when the interaction between multiple proteins is considered.

2.1.2 The immune response

The immune response consists in the way the body defends itself against foreign or harmful substances. On a large scale, it includes organs, tissues, cells and molecules, and protects from microorganisms, removes toxins or destroys tumor cells. Two different immune responses are considered: the innate (or inductive) and the adaptative. Besides the natural physical and chemical barriers such as the skin or cilia in airways, the former response triggers inflammation and uses nonspecific cells like phagocytes or natural killer cells to kill pathogens. This response is fast and is associated to the pain, heat and redness that hit the body. On the other hand, the adaptative immune response employs highly specific cells, builds immunologic memory and is significantly slower (a couple of weeks).

Humoral and cell-mediated immunity A broad scope of different cells among the leukocytes (white blood cells) are involved in the adaptative response. Among them, two types are of considerable importance:

- B-cells, that bind to specific antigens and exercise phagocytosis, i.e. the process by which the cell ingests other cells or particles, will then load antigens on MHCII proteins (see section 5.1) and display the latter to T-cells at the surface of the cell. B cells mature into plasma cells and will secrete lots of antibodies that will mark future similar pathogens for destruction. This is called the humoral immunity.
- T-cells are, on the other hand, responsible for cell-mediated immunity. Naive T-cells get primed by binding to antigen presenting cells (usually dentitric cells) and can be then split into two categories:
  - CD8+ (cytotoxic) T-cells, that only see antigens on MHCI, will directly kill the target cells. In that regard, they attack virus-infected or cancerous cells.
  - CD4+ (helper) T-cells bind antigens presented with MHCII and secret cytokines to help in the formation of CD8+ T-cells and B-cells.

2.1.3 Peptide-MHC binding and antigen presentation

We can see that the activation of T-cells relies on a crucial set of biologic processes: the antigen presentation pathway, that makes use of major histocompatibility complexes (MHC). The latter is a group of genes on chromosome 6 that code for cell-surface proteins responsible to bring antigens at the surface of the cell. In human beings, it is also referred as human leukocyte antigen (HLA) and is split in three groups: I, II, III.

Class-I MHC Found on all nucleated cells, MHCI presents endogeneous (i.e. coming from the human body, as opposed to virus for instance) antigens located in the cytoplasm. For a given protein, the different processes are involved:

- the degradation and cleavage of the protein into peptides in the proteasome,
- the transport of the peptide in the endoplasmic reticulum (ER) through the TAP complex,
- the binding with one of the several MHC class I proteins expressed by the cell, the overall complex being then presented at the surface of the cell.

In the MHCI structure, the alpha chain, encoded by HLA-A, HLA-B and HLA-C genes, consists in the binding groove that binds peptides roughly 8-10 AAs long.

Class-II MHC Found on antigen-presenting cells, MHCII present exogeneous antigens from foreign bodies, after engulfment and destruction of the pathogens. HLA-DP, HLA-DQ and HLA-DR encode MHCII proteins and the latter bind to peptides 14-20 AAs long.

A detailed version of both of these pathways can be found on Figure 2. Binding affinity between the peptide and the MHC is therefore one of the important steps in the antigen presentation pathway. It is worth noting that while a high-affinity interaction with MHC is the most selective requirement for a peptide to be presented, other processes such as gene expression, RNA splicing, proteosomal processing and transport have important secondary effects.

Class I and II MHC alleles are encoded by genes with a very high level of polyphormism between humans, e.g. there are at least 350 alleles (i.e. heritable gene variants) for HLA-A genes and 400 alleles for HLA-DR. This enables the immune system to reach against a very large number of pathogens at the individual level. Each human cell expresses six MHC class I alleles (one HLA-A, -B, and -C allele from each parent) and six to eight MHC class II alleles (one HLA-DP and -DQ, and one or two HLA-DR from each parent). Coupled with the high variability, this implies that each human being will present very different peptides and this is critical for vaccine development as we will see in the following section.

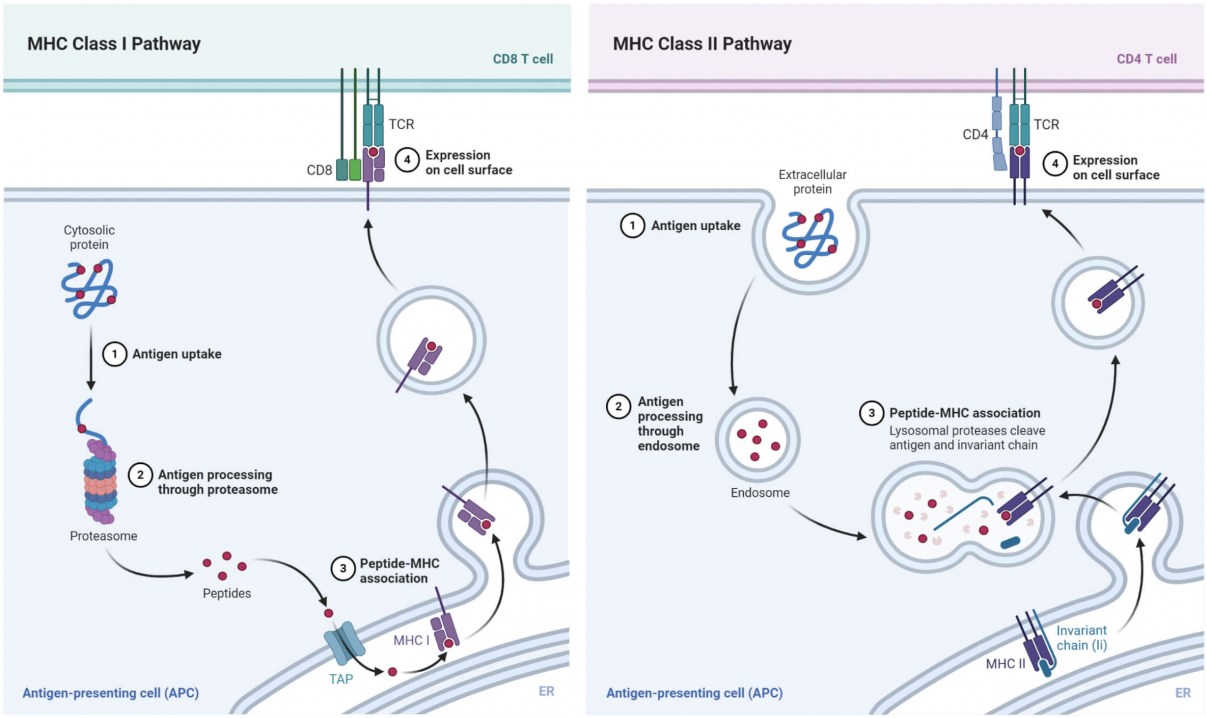

Figure 2: Detailed version of the antigen processing pathway.

Source: thebiologynotes.com

2.1.4 Cancer vaccine through neo-antigen

Personalized therapeutic cancer vaccines using neoantigens have been shown to be feasible, safe and demonstrated promising results in their early stages for patients with melanoma and glioblastoma [Blass and Ott, 2021]. The basic mechanism behind cancer involves the DNA in a patient's cell that becomes corrupted, leading to a cancerous cell. The patient immune system then struggles to distinguish between this cancerous cell and a healthy one. Only certain tumor-associated peptides will bind with the MHC alleles from a specific patient with high affinity, consequently helping the immune system to recognize the cancerous cells. However, this process is often too slow compared to the cancerous cells' growth. The goal of personalized cancer vaccine is therefore to boost this immune response, through the following pipeline, illustrated in Figure 3:

- First, the genome of the cancerous cells is collected thanks to RNA sequencing, in order to identify peptides that contains cancerous genetic modifications, i.e. mutations. These peptides are called **neoepitopes**. This is done by comparing this genome with health cells' genome and the human reference genome.
- Then, up to 20 of these potential neoepitopes are selected, depending on their likelihood to bind to the patient's MHCs and be presented to the cell's surface.

- Then, these neoepitopes are combined with adjuvants and injected to the body. Processed through the antigen processing pathway (2.1.3), it helps the body better identify those cancerous cells and destroy them with T-cells. In a nutshell, the patient learns to kill cancerous cells on its own.

Both MHCI and MHCII help Recent work [Hu et al., 2017; Sahin et al., 2017; Blass and Ott, 2021] have shown that class I MHC alleles play a crucial role in identification of cancerous cells, to help the CD8+ T-cells target the cancerous cells. On the other hand, it has also been recognized [Blass and Ott, 2021] that CD4+ T cells (and MHCII pathway) can orchestrate anti-tumour immunity and, via their classical "helper" functions, have a crucial role in generating and sustaining CD8+ T cell responses – similar to what is done with the mRNA-based vaccines against the coronavirus [WHO, 2020].

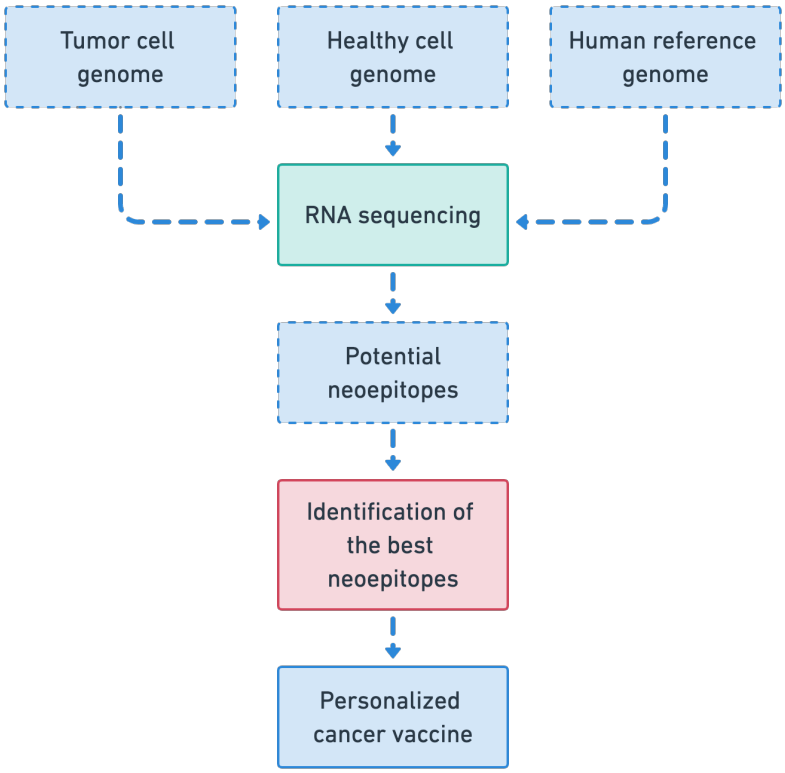

Figure 3: Cancer vaccine pipeline.

This work focuses on the red step: identification of the best MHC-I neoantigens.

Neoantigen target discovery In this manner, a crucial aspect of the efficiency of a vaccine targeting neoantigens is the selection of epitopes that can be presented in vivo by tumour or antigen-presenting cells. Owing to the high cost and technical challenges

of experimentally testing all the of possible neoantigen candidates, recent research have attempted to computationally identify those peptides with machine-learning algorithms (see section 2.4 on related work). This master's thesis is part of this research initiative, looking for **specific axes to improve the peptide-MHCI binding and presentation prediction** and hence enable a better selection for final neoepitopes in the cancer vaccine pipeline.

2.2 Datasets

While my team was working on proprietary datasets from BioNTech, this data was not available for disclosure for privacy reasons. In order to complete this master's thesis, it was hence needed to research open-source datasets from previous related work and open databases. At first glance, this task may look bundersome, but it came out to be really useful. Indeed, it allowed me to explore the related literature, confirm our main modeling take-aways on open-source data and, last but not least, enable a fair comparison and a correct benchmark with previous binding and presentation prediction frameworks.

Furthermore, as a mathematician with a background in computer science, it is tempting to leave the biology completely on the side and focus solely on machine learning challenges. However, the data is this thesis is not as straightforward as in common ML pipelines like image classification or numerical/categorical prediction. That is why understanding what the data is and how it has been collected is, in my opinion, particularly crucial to understand the problem at hand and elaborate ideas and research directions.

2.2.1 Data types

Binding / presentation data Over the last years, experimental assays have been conducted to extract data from the binding and presentation processes in the antigen presentation pathway. Data for the former can be derived from *in vitro* peptide-MHC binding assays. **Binding affinity (BA)** is thus measured with a proxy continuous value: IC_50 (nM), which quantifies the way binding supports a temperature increase. The lower the latter is, the stronger the binding is. More recent experiments, using the mass spectrometry technology, have enabled the detection of **eluted ligands (EL)**: peptides that are presented at the surface of the cell for a given person. Despite some inherent limitations of MS methods, peptide ligand sequences identified by such antigen presentation profiling currently provide the closest sample population to the true presented ligands. For both binding and presentation data, the negative instances, i.e. peptides that do not bind to or are not presented by a given MHC, are called "decoys".

Single-allelic / multi-allelic data Due to the fact that each person produces multiple MHCs (6 for MHC-I), mass spectrometry experiments often cannot precisely identify

the exact MHC molecule a peptide was bound for. The EL data can hence be split in two data types:

- single-allelic (SA) data, for peptides assigned to a single MHC,
- multi-allelic (MA) data, for peptides assigned to multiple MHCs.

2.2.2 Data collection and curation from open-source studies

NetMHCpan-4.1 presentation set Most of the work in this thesis has been conducted around the presentation set used to train the NetMHCpan4.1 framework [Reynisson et al., 2020]. This raw data has been gathered from multiple studies and consists in SA EL data, covering 142 different MHCs. The dataset has been enriched with additional negative peptides (or decoys) randomly sampled from the UniProt [The UniProt Consortium, 2020] database, 5 times the amount of peptides of the most abundant peptide length. The peptides have been filtered to only include 8 to 14 amino acid long peptides (distribution displayed in Figure 5) and only human MHCs (HLA) alleles were selected. An overview of the data can be found in Table 1 and the number of peptides and positives instances per allele is shown in Figure 4. One can observe that the dataset is quite imbalanced: this will be significant for the different losses and classification metrics of the machine learning methods used later in this work. Moreover, the distribution of peptides per allele is largely uneven – for instance, 28 alleles have less than 100 peptides in the dataset. Throughout this work, we will refer to this dataset as **netmhcpan4.1**. Used for most of this thesis' development, this set has been split into train/val/test subsets. This split is fixed for all the experiments and metrics are reported on the test set.

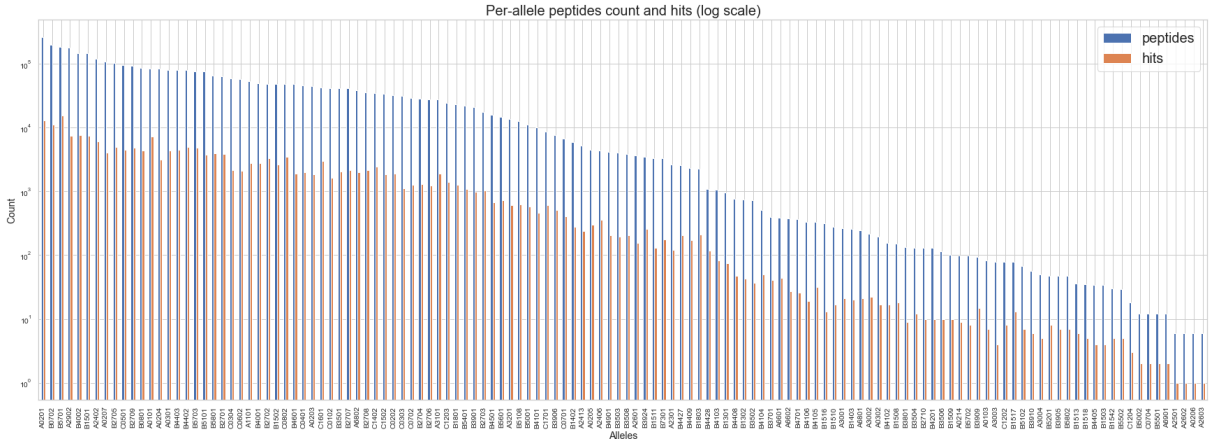

Figure 4: Number of peptides and hits (positive) per-allele for *netmhcpan4.1.*

MHCAttnNet binding set This dataset is extracted from [Venkatesh et al., 2020] and consists in BA SA data. Initially coming from the IEDB database [Sahin et al., 2017], the targeted binding affinity is continuous and values has been gathered in 5 buckets: "Positive", "Positive High", "Positive Intermediate", "Positive Low" and "Negative". The three first categories correspond to binding affinity IC_50 values <500nm and the data points within these classes have been labeled as binding (positive). All of the remaining instances are labeled as negatives. Peptides' length ranges from 3 to 43 and its distribution plot can be found in Figure 6. The original train/val/test splits are kept and this dataset is referred as **mhcattnnet**.

Name	Data type	Positives	Negatives	Positive rate	MHCs
netmhcpan4.1	SA EL	197 547	3 481 858	5.7 %	130
mhcattnnet	SA BA	379 783	111 235	77.0 %	161

Table 1: Overview of the datasets.

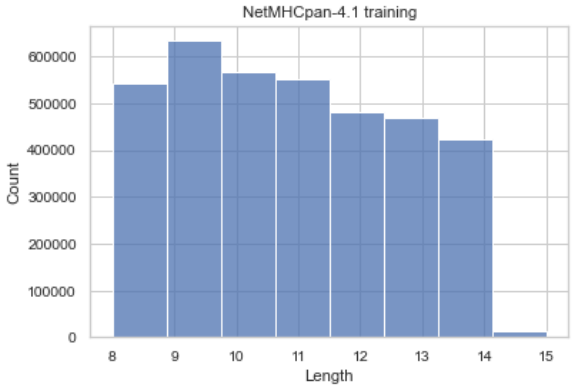

Figure 5: Peptide length distribution for *netmhcpan4.1*.

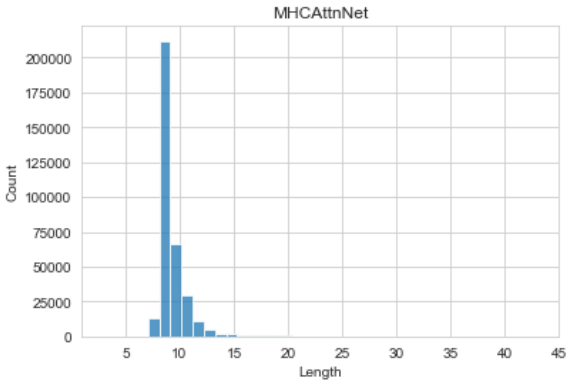

Figure 6: Peptide length distribution for *mhcattnnet*.

UniRef50 protein dataset This database [Suzek et al., 2015], named **UniRef50** in this work, contains clustered protein sequences from the UniProt database [The UniProt Consortium, 2020]. Compared to UniProt, it hides redundant sequences and enables the complete coverage of the sequence space, by clustering the original UniRef dataset at three different resolutions, UniRef50 being the last one. In short, this dataset is a curated set of 45 million protein sequences that will be used as unsupervised data to build a language model in this work.

2.3 Mathematical formulation and metrics

2.3.1 Problem formulation

In all of the given datasets, we consider the study of N samples. In their raw format, each sample i has two features (peptide and MHC) and one target.

Peptide data For each sample i, the peptide information is encoded into a variable-length string sequence $\mathbf{s_p} = (AA_{p,1}, ..., AA_{p,L_p})$, with L_p being the length of the sequence. Each sequence token takes its values into the fixed-sized vocabulary V of amino acids (20 single-letter codes that encode AAs + "X" for unknown AAs). Hence, $AA_{p,i} \in V = \{A, R, N, ...Y, V, X\}, |V| = 21$.

MHC data For each sample i, the MHC information is encoded into an ID (e.g. "A0201") that identifies a specific HLA allele. The latter is composed of:

- a letter (A, B or C for MHC-I) which identifies the gene class,
- two digits that identify the allele type (or *major* type),
- two additional digits that identify the allele subtype (or *minor* type).

This data can thus be considered either as categorical or as a sequence of three tokens (e.g. "A0201" $\rightarrow$ ["A", "02", "01"]), if we want to integrate the hierarchical structure of the allele nomenclature. Furthermore, and it will be highly used in this thesis, an MHC allele can also be encoded as a sequence of AAs, since the MHC gene encodes a protein, as seen in 2.1.1. MHC encoding AA sequences are between 300 and 400-AAs long, but many AAs do not differ between alleles. That is why we would more specifically consider a MHC pseudo-sequence instead. The latter contains 34 AAs on certain fixed positions. These fixed positions have been determined in [Hoof et al., 2008] and consist of polymorphic residues of the MHC in contact with the peptide ($< 4.0\mathring{A}$). Consequently, MHC allele information can also be represented as an aligned sequence $\mathbf{s_a} = (AA_{a,1}, ..., AA_{a,L_a})$, with L_a being the length of the MHC sequence and $AA_{a,i} \in V$.

> In this thesis, $X \in \mathcal{R}^{N \times d}$ refers to the peptide and MHC features data, d being the dimension which depends on how the sequence or categorical features are encoded.

Target binding/presentation data Each sample i has a discrete binary label, either specifying the binding or the antigen presentation.

> In this thesis, $Y \in \mathcal{R}^{N \times \{0,1\}}$ refers to the target data.

Data snapshot In order to illustrate the last paragraphs, a snapshot of a 1000-length random subset of *netmhcpan4.1* data is provided in Figure 7. With the previous notations, X refers to "peptide" and "allele" (or "allele_sequence") columns whereas Y corresponds to the "hit" column.

	peptide	allele	allele_sequence	hit
0	EHAVQIGFQ	B5108	YYATYRNIFTNTYENIAYWTYNYYTWAVDAYLWH	0
1	LNGVVGGK	A2402	YSAMYEEKVAHTDENIAYLMFHYYTWAVQAYTGY	0
2	GRIGSIGKLYWFMP	B0702	YYSEYRNIYAQTDESNLYLSYDYYTWAERAYEWY	0
3	EVMKIKAEI	A6802	YYAMYRNNVAQTDVDTLYIRYHYYTWAVWAYTWY	1
4	DPHKVYAL	B3501	YYATYRNIFTNTYESNLYIRYDSYTWAVLAYLWY	1
...	...	...	...	...
995	VKPGPPLAP	A0204	YFAMYGEKVAHTHVDTLYVMYHYYTWAVLAYTWY	0
996	ECVAIGMVK	A0201	YFAMYGEKVAHTHVDTLYVRYHYYTWAVLAYTWY	0
997	DDERIKRHLG	A2902	YTAMYLQNVAQTDANTLYIMYRDYTWAVLAYTWY	0
998	DRPDEEADQEC	B0801	YDSEYRNIFTNTDESNLYLSYNYYTWAVDAYTWY	0
999	PGIKELLAS	B1501	YYAMYREISTNTYESNLYLRYDSYTWAEWAYLWY	0

1000 rows × 4 columns

Figure 7: Snapshot of presentation data from *netmhcpan4.1*.

Binary classification The overall mathematical problem explored in this thesis is hence a binary classification problem. Given the N independant random copies $((x_1, y_1), ..., (x_n, y_n))$ of (X, Y), we aim at building a rule to predict y based on x. Such a rule is a function $h : x \to \{0, 1\}$ called a classifier. Since $Y \in \{0, 1\}$, we consider a Bernoulli distribution for Y. We write, for $x \in X$ and $y \in Y$, $y|x \sim Ber(f(x))$, where $f(x) = P(y = 1|x) = E[y|x]$ is called the regression function of y onto x, with P the probability measure and E the expectation over this probability. This means that the different approaches in this thesis are conducted to find a suitable f modeling the likelihood of a sample being a binder/hit ($y = 1$).

Note that $y = f(x) + \epsilon$, where $\epsilon = y - f(x)$ is a "noise" random variable that satisfies $E[\epsilon|x] = 0$. In particular, this noise accounts for the fact that X may not contain enough information to predict Y perfectly.

2.3.2 Classification metrics

Some classifiers are better than others and the classification approaches explored in this master's thesis will be favored in light of specific classification metrics. First of all, in such binary classification scenario, we divide the predicted samples in 4 buckets:

- True positives (TP), for which the label is positive and is correctly predicted,
- True negatives (TN), for which the label is negative and is correctly predicted,
- False negatives (FN), for which the label is positive and is incorrectly predicted,
- False positives (FP), for which the label is negative and is incorrectly predicted.

Based on these four quantities, we can derive three important metrics: the precision, the recall (or true positive rate) and the false positive rate, for which we provide the formulas below (1). The precision tells us how many predicted samples are relevant and the recall highlights how many relevant samples are selected.

$$precision = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN} \tag{1}$$

F1-score As the datasets are quite imbalanced, the accuracy is a misleading metric. For example, a dummy classifier that classifies every sample as negative will have an accuracy of 94.3% on *netmhcpan4.1* dataset. That is why we will have a focus on the F1-score instead. The latter is an harmonic average of precision and recall and is calculated as below (2). The range of F1 is in $[0, 1]$, where 1 stands for perfect classification, 0 total failure and 0.5 random classification.

$$F1 = \frac{precision * recall}{precision + recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{2}$$

ROC-AUC In fine, the goal is to select the best binders or presented neo-antigens. Therefore, we are interested in ranking predictions. The ROC-AUC is one metric suited for that purpose. The Receiver Operating Characteristic (ROC) curve represents the tradeoff between the true positive rate (TPR, on y-axis) and the false positive rate (FPR, on x-axis) for different thresholds. The ROC-AUC metric is then computed as the Area Under its Curve (AUC).

PR-AUC Once again, for the datasets heaviliy imbalanced in favor of negative instances such as *netmhcpan4.1*, another metric needs to be added to our set of evaluation tools: the PR-AUC. The latter is calculated as the area under the precision-recall (PR) curve which plots precision against recall for different thresholds. The intuition behind PR-AUC is the following: since it focuses mainly on the positive class (precision and recall), it cares less about the frequent negative class and is suitable for our problem.

Precision@k Eventually, an interesting metric is the precision@k. This shows how many relevant samples are present in the Top-K predicted samples. Technically speaking, you select a subset of the k best predicted samples and computes the ratio of true positives within this subset. In this thesis, and as the datasets are quite imbalanced in favor of negatives (except for *mhcattnnet*), we will take k equals to the initial number of positives. This metric is also called **Top-K** and will be highly considered for the comparison of different approaches/frameworks.

2.4 Related works and research hypotheses

This section aims at relating different research works around the topic at hand, as well as techniques from natural language processing (NLP), which have been explored or used to approach our classification model in this thesis. This literature review enables us to raise some questions and hypotheses that this work will further assess. While the algorithms and methods (e.g. recurrent neural networks (RNNs), Transformers, auto-encoding, etc.) are evoked below, their inner workings, their mathematical derivation and the comparison between them will be developed later in section 4. This also applies for the different concepts brought in this section.

Predicting MHC-peptide binding/presentation has been a long standing problem and many recent research initiatives have considered it. Despite the different ways this problem is approached, it is worth nothing that most of these works use deep learning. *NNalign_MA* [Alvarez et al., 2019], which is the architecture of the *NetMHCpan4.1* [Reynisson et al., 2020] framework, deploys an ensemble of feed-forward neural networks on fixed-length peptide representations of peptide and MHC pseudo-sequences trained with stochastic gradient descent (SGD). With this architecture and a framework able to leverage multi-allelic data, *NetMHCpan-4.1* provides a strong public baseline today, which is often updated with more data. **The evaluation of our approaches, especially with attention-based models, will be hence compared to this framework.** Briefly, this framework outputs a presentation prediction based on a percentile rank, a transformation that normalizes presentation prediction scores across different MHC molecules and enables inter-specific MHC presentation prediction comparisons. In order to obtain a likelihood of a peptide being presented by a MHC, we transform this percentile rank as follows: $P(presentation) = 1 - \frac{\%Rank}{100}$.

Other methods, like *MHCFlurry2.0* [O´ Donnell et al., 2020] or *NeonMHCII* [Abelin et al., 2019], leverage convolutional neural networks (CNNs), a well-tried technique from computer vision. For instance, while the former predicts binding affinity with, once again, a pan-allele feed-forward neural network ensemble on peptide and MHC pseudo-sequences, it models antigen preprocessing, a representation that helps predict the antigen presentation, by learning convolutional filters on top of a fixed-length representation of the peptide and its flanks (sides of the peptide before cleavage). In

a similar manner, the second framework predicts binding prediction with a per-allele ensemble of CNNs.

In parallel, some research initiatives consider deep learning approaches more suitable to sequential data: recurrent neural networks (RNNs). Examples of such methods can be found in the *MHCSeqNet* [Phloyphisut et al., 2019], *MARIA* [Chen et al., 2019] and *MHCAttnNet* [Venkatesh et al., 2020] frameworks. A sequential hidden representation is hence built for each amino acid in the sequences, and this allows to consider variable-length peptide inputs. Moreover, the latter seems to yield stronger final performance. This RNN approach needs to be applied upon embeddings for each amino acid. The latter can either be built with one-hot encoding (like in *MARIA*), learnt through training (like in *MHCSeqNet*), or pre-learnt with techniques such as Word2Vec (like in *MHCAttnNet*).

Given the different approaches considered in these research works, a first task in this thesis consists in exploring these modeling methods on our classification problem and comparing them to more basic NLP approaches. This would enable us to build first baselines and validate, in light of what has been recently seen in the NLP field, the two following hypotheses:

Hypothesis 1: *Deep learning applied on TF-IDF representation or word (non-contextual) embeddings outperform simple linear modeling methods.*

Hypothesis 2: *By considering the sequential nature of the peptide, recurrent neural networks boosts the binding/presentation prediction.*

More recently, a popular deep learning architecture called *Transformer* [Vaswani et al., 2017] has demonstrated great results in many NLP tasks such as language translation or sentence classification. This encoder-decoder architecture uses an important mechanism known as "attention". For a given token, the attention makes the network concentrate on other specific tokens in the text and the resulting token embeddings are contextual. This mechanism takes away the recurrence and consequently allows to process tokens in parallel rather than sequentially. A common derivative of the Transformer architecture is called the BERT [Devlin et al., 2019] architecture and is built upon the encoder. The latter is, for instance, used in *BertMHC* [Cheng et al., 2020] to tackle the binding affinity problem with MHCII. Since this thesis strongly relies on this BERT architecture, its detailed explanation will be developed in section 4.3.2. Given this more elaborated architecture, we can, with good reason, explore the following hypothesis:

Hypothesis 3: *Binding/presentation prediction is improved with attention-based Transformer encoder models.*

Over the last years, the NLP field has been booming with the development of language modeling. The latter stands for models trained to predict tokens based on their surrounding context. A crucial advantage of such models is that they do not require explicit labels – the labels are generated by masking some parts of the inputs. Therefore, they are self-supervised learners and can be used on any corpus at large scale. BERT, for instance, uses a bidirectional masked language modeling (MLM) task to learn representative contextual embeddings from large unlabeled datasets. The pre-trained model (and hence this contextual representation) can then be transferred to tasks such as sentence classification. This transfer can either be done by simply extracting the embeddings and learning linear layers on top (forward pass of the core model, without updating the weights of the core model) or by fine-tuning the entire model on relevant domain data. Largely used in natural language, this technique has naturally been applied to proteins. Research initiatives, like *ProteinBERT* [Brandes et al., 2021], *ESM* [Rives et al., 2020], or *ProtTrans* [Elnaggar et al., 2021], have trained large auto-regressive or auto-encoding (e.g. BERT) models on databases such as UniRef [Suzek et al., 2015] or BFD [Steinegger and Söding, 2018]. These sequence-only models have shown strong performance, reaching or outperforming state-of-the-art methods, on downstream tasks such as per-residue secondary structure prediction (token classification) or per-protein localization and membrane prediction (sentence classification). For instance, *BertMHC* notably explored the binding prediction with a pre-trained BERT model from *TAPE* [Rao et al., 2019], a benchmark of five tasks to assess protein sequence models. We will try to leverage a specific similar and more recent Transformer model from ProtTrans, *prot-bert-bfd*, to our classification problem and assess if this self-supervised pre-training step on large protein data can boost the performance of our classifier. This would consequently verify the following hypothesis:

Hypothesis 4: *Pre-trained protein language models can be transferred to predict binding/presentation, with or without fine-tuning them on this task.*

In *DeepLigand* [Zeng and Gifford, 2019], an interesting approach is to consider the pre-training of a language model on only natural ligands (positive peptides) from all MHCs. The learnt embedding vector for a given peptide is then concatenated to the output of a convolutional network applied on fixed-length representations of peptide sequence and MHC pseudo-sequence (a more classic binding affinity module). Fully-connected layers are then applied on top of this concatenated vector. During training, the weights of the peptide language model are not updated. This research initiative notably shows that the peptide embedding module only is highly predictive of natural ligands, which indicates that meaningful patterns associated with peptides selection are captured in the contextual embeddings. Moreover, this proposed peptide embedding module seems to complete the binding affinity module and model other cellular processes playing a role in the ligand presentation, such as cleavage. For their language model, they use an early contextualized embedding model called ELMo, which derives

representations from the hidden states of bidirectional LSTMs. The idea would be to build a similar unsupervised language modeling approach with a more elaborated technique like BERT on our positive data (peptides only or peptide-MHC complexes) to boost the binding/presentation performance.

More generally, in their attempt to build strong compact language models, [Turc et al., 2019] identified that pre-trained language models on out-of-domain (i.e. not directly relevant to the final task) data benefit from another pre-training step (with masked language modeling objective) on relevant domain data before fine-tuning the model on the same data, using, this time, the labels. This approach, as well as the one from *DeepLigand*, motivates us to explore the following hypothesis:

Hypothesis 5: *An additional MLM pre-training step on relevant domain positive data boosts the performance of pre-trained protein language models.*

In [Rao et al., 2020], they leveraged the learnt attention weights from a pre-trained language model (ESM-1b from [Rives et al., 2020]) to predict the contact maps of proteins. Without fine-tuning the model, they showed that attention maps already contain valuable information for the contact map prediction and allow the model to beat a state-of-the-art method that relies not only on the protein sequence but also on multi-sequence alignments (MSA), which require a retrieval step from a sequence database. A quick supervision task, using sparse logistic regression, is used to extract the appropriate attention maps from the model, but an ablation analysis confirmed that the contacts are learned without supervision (during pre-training). Moreover, they showed that perplexity (Appendix C), the metric used to evaluate the masked language modeling task, is a good proxy for downstream contact prediction. This suggests that improved pre-trained protein language models will drive better downstream prediction on such residue-residue contacts inference.

In parallel and in an effort to boost interpretation in Transformer-based models, [Vig et al., 2021] showed that attention maps in pre-trained BERT models capture the folding structure of proteins, especially connecting amino acids that are far apart in the underlying sequence, but spatially close in the three-dimensional structure. Similarly, functional properties of proteins can be derived from the knowledge of these models in their attention mechanism.

These two research works strongly showed that the attention mechanism captures valuable information for structure prediction and, hopefully, interaction between the MHC and the peptide. It would thus be interesting to explore the following hypothesis:

Hypothesis 6: *Sequence-based pre-trained models can provide a good proxy to structural features .*

3 Organization and technical tools

This section aims to provide a deeper understanding of the tools that have been required to complete this master's thesis project. Apart from the communication tools (3.1) within the team, a list of the technical tools (3.2) is provided for each step of the research work, from computing to data exploration and model analysis. Their overall interconnection in the machine learning workflow is then displayed (3.3).

3.1 Work organization

As a Research Engineer intern at InstaDeep, I was part of a 10-people team, composed of data scientists, machine learning engineers, computational biology scientists and a lead project manager. We were all working in close relation with BioNTech teams to tackle diverse immunology-related problems, including the p-MHC binding and presentation prediction I have been researching within the scope of this master's thesis.

My contribution to this team and these projects have been twofold:

- As part of my master's thesis work, I was investigating approaches to improve p-MHC binding prediction, with a specific focus on pre-trained protein language models. This work mostly included literature review, data exploration, modeling and analysis, which led to this research work.
- In parallel, I was working on improving our custom-made Python-based machine learning framework, mainly to leverage the latter for the experiments that have been conducted in this thesis.

Communication In order to coordinate our research work, what had been achieved so far and our next steps, I participated in daily team meetings (internal), as well as weekly team meetings with the client (external) and with my supervisor (internal). The team was remote and international – we were split between Paris, London, Tunis and Cape Town offices. This is why most of the communication was held online.

Documentation The different projects were managed with sprint boards on the cloud-based code versioning platform **Gitlab** and regular presentations were built with **Google Slides**. We used **Confluence** as a "wiki" platform in order to document our work, the different biology-related concepts, our papers' literature reviews, as well as to share our findings.

3.2 Tools

3.2.1 Cloud storage and computing

The different steps in the machine learning workflow, such as training models or storing and sharing large datasets, require a lot of compute capabilities and storage capacities, often more than what one single laptop could provide. For this purpose, many companies such as Amazon, Google or Microsoft provide a series of management tools and modular cloud services including computing, data storage, data analytics and machine learning.

At InstaDeep, **Google Cloud Platform (GCP)** is leveraged. Built by Google, GCP is a suite of such cloud services that runs on the same infrastructure that Google uses internally for its end-user products, such as YouTube or Google Maps. The services that have been used for this project are the following:

- **Google Cloud Storage**: Similar to AWS (Amazon Web Services) S3 service, Google Cloud Storage is an online object storage web service for storing and accessing unstructured data on the GCP infrastructure. Thanks to its integrated edge caching system, it combines the performance and scalability of Google's cloud with security and sharing capabilities. This service was used to store datasets as well as model artifacts: parameters' weights, configuration, logs, etc.
- **Google Compute Engine**: As an infrastructure as a service (IaaS), Google Compute Engine enables users to launch virtual machines (VMs) on demand – the user only pays per compute time and does not need any DevOps expertise or physical hardware. The VMs can be launched from standard or custom images and are accessible through the command line interface (CLI) with SSH connection. The different machine and deep learning models implemented in this work have been trained on the cloud with VMs launched thanks to the **Deep Learning VMs** marketplace service, the latter offering Intel(R) optimized and GPU-ready images with pre-installed machine learning frameworks. Depending on the needed compute, two different cloud VMs have been used, for which characteristics can be found in table 2. Additional details on the used GPUs are available in appendix A.

Machine instances			
Type	Nb. of vCPUs	RAM	GPU(s)
n1-standard-8	8	30 GB	2 * Nvidia Tesla V100 (16 GB)
a2-highgpu-1g	12	85 GB	1 * Nvidia Tesla A100 (40 GB)

Table 2: Details on cloud virtual machines.

For the sake of completeness, it is worth mentioning that some models have been trained on InstaDeep's on-premise servers, using one A100 GPU with the same characteristics as the one detailed in table 2.

Environmental considerations Modern AI models, such as the GPT-3 or BERT, consume a massive amount of energy, and these energy requirements are growing at a breathtaking rate. Indeed, in today's deep learning-centric research paradigm, advances in artificial intelligence are primarily achieved through substantive scale: bigger datasets, larger models, more compute. This raises questions regarding the environmental impact of such big models. While the considered models in this thesis are not at such a large scale, we can estimate that the trainings and experiments in this master's thesis use a cumulative amount of 100 ($\simeq$ 2 days $\times$ 2 large models) hours on A100 and 300 ($\simeq$ (2 days $\times$ 2 large models $\times$ 2 V100) + (1h $\times$ 100 large models $\times$ 1 V100)) hours on V100. Therefore, with the help of the ML CO_2 Impact tool [Lacoste et al., 2019], the total emissions for this master's thesis are estimated to be 31.05 $kgCO_2eq$, which is equivalent to 125.4 km driven by an average car.

3.2.2 Data exploration

This work required a strong focus on data exploration and the curation of the datasets. Both steps were conducted on **Jupyter Notebook**, a web-based interactive computational Python environment which allows to create notebook documents, i.e. documents split in an ordered list of cells containg code, text, mathematics or plots. Depending on the required memory compute, Jupyter Notebook was used either locally or remotely on VMs.

3.2.3 Data processing, analysis and modeling libraries

NumPy, pandas and Vaex These Python libraries were mostly used to handle data preprocessing and analysis. The former brings support for large, multi-dimensional arrays and matrices with an easy-to-use collection of high-level mathematical functions to operate on these arrays. pandas is a popular library to manipulate, explore and visualize tabular datasets. While Vaex fulfills the same needs, it uses a memory mapping technology to get rid of the RAM constraints and therefore provide pandas-like functions on very large datasets. In our case, the latter are converted from `.csv` files to `.hdf5` ones.

scikit-learn This popular Python machine learning library offers lots of predictive algorithms such as naïve bayes, logistic regression or principal component analysis. It leverages NumPy for its under-the-hood high-performance linear algebra and array operations, and has been used to model machine learning parts of this thesis.

TensorFlow and transformers TensorFlow [Abadi et al., 2015] is an open-source deep learning library built by the Google Brain team. It specifically provides an high-level API to build and train deep neural networks, and it was used to train the deep learning models in this research work. As TensorFlow quickly evolves, it is worth mentioning

that we used the version 2.2.0. In order to leverage pre-trained Transformer-like models, we also took advantage of the HuggingFace transformers [Wolf et al., 2020] library. It offers state-of-the-art NLP algorithms in TensorFlow and previously trained models are openly shared on the HuggingFace Hub, from which we, for instance, leveraged the ProtTrans [Elnaggar et al., 2021] models.

matplotlib and seaborn Both are Python data visualization libraries – the latter is based on the former –, that have been used to build the plots in this thesis, whether it be to display insights from the data or to properly show computed metrics.

3.2.4 Custom-made python-based framework

Developed in Python 3.6 and based on TensorFlow, a machine-learning purpose framework was used to train deep learning experiments shown throughout this thesis. The code base was constantly improved by multiple engineers at InstaDeep and I developed the additional features that were required to launch my experiments.

In a nutshell, this framework consists in a Python package that allows to train machine learning algorithms without the need to implement them directly. Working in the command line interface (CLI), the user uses the `train` command while providing some configuration (a `.yml` file) on the model definition and paths to the training and validation datasets, and the framework builds and trains the model by itself. As a consequence, the framework can be used by people without any strong expertise in deep learning such as biologists or bio-statisticians. An example of such a configuration file can be found in appendix F. Other commands such as `predict` or `compute-metrics` were available to the user. The toolkit is deployed thanks to Docker (see 3.2.6) and leverages the libraries defined 3.2.3. Unit and integration tests are implemented in **pytest** and ensure its proper functioning.

Data input pipeline For deep learning models, the data is preprocessed (features are transformed and standardized according to the type of the feature). This transformed data is then stored in a special format called **TFRecords**. The latter optimizes the space taken by each file and is particularly convenient to create a **tf.data.Dataset** which is then fed into a model. These TFRecords can be cached locally or on an on-premise AWS S3 bucket. The rest of the pipeline then manages the shuffling, the batching as well as the parsing and encoding of batches.

3.2.5 IDE and code versioning

PyCharm In order to efficiently develop on the framework, I used the integrated development environment (IDE) PyCharm. Among other great features, the latter enables code search and completion, virtual Python environment creation and Git history.

GitLab GitLab is a web-based DevOps tool that provides a Git repository manager to ensure code versioning and code sharing. This software also features wiki, issue-tracking, continuous integration and deployment pipelines (see next section 3.2.6). For instance, the feature implementation in the framework was following a strict but powerful contribution workflow, using issues and merge requests to ensure a clear technical planification and enable collaboration on the feature itself. Hence, merge requests' reviews were a common task in the software development scope of this thesis.

3.2.6 Software deployment

The packaged framework was delivered across different devices, i.e. VMs or local computers, using the **Docker** technology, a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. A container is locally created from a remotely pulled Docker image, the characteristics of the latter being defined in a `Dockerfile`. An image is based on an official TensorFlow image and contains all the python dependencies. This images is then built thanks to the **GitLab CI** pipeline and stored in a GitLab container registry.

Kubernetes Additionally, for the trainings that have been launched on on-premise InstaDeep compute cluster, we used Kubernetes, which is an open-source container-orchestration system for automating application deployment, scaling, and management.

3.2.7 Experiment monitoring

In order to track our deep learning trainings and metrics such as loss, AUC, F1-score, etc., we leveraged **Neptune.ai**, a lightweight experiment management tool that enables the tracking of learning curves, CPU/GPU memory utilization monitoring and comparison between different experiments. For some experiments run without the framework, we also took advantage of an another experiment visualization tool called **TensorBoard**.

3.3 Overall infrastructure

The different tools presented so far can be combined together in two different pipelines that range from data storage to model training, as illustrated in Figure 8. On the left is shown the workflow when the framework was leveraged, while on the right is streamlined the workflow when an experiment was run in a Jupyter notebook – this being useful for rapid experiments that the framework did not support yet.

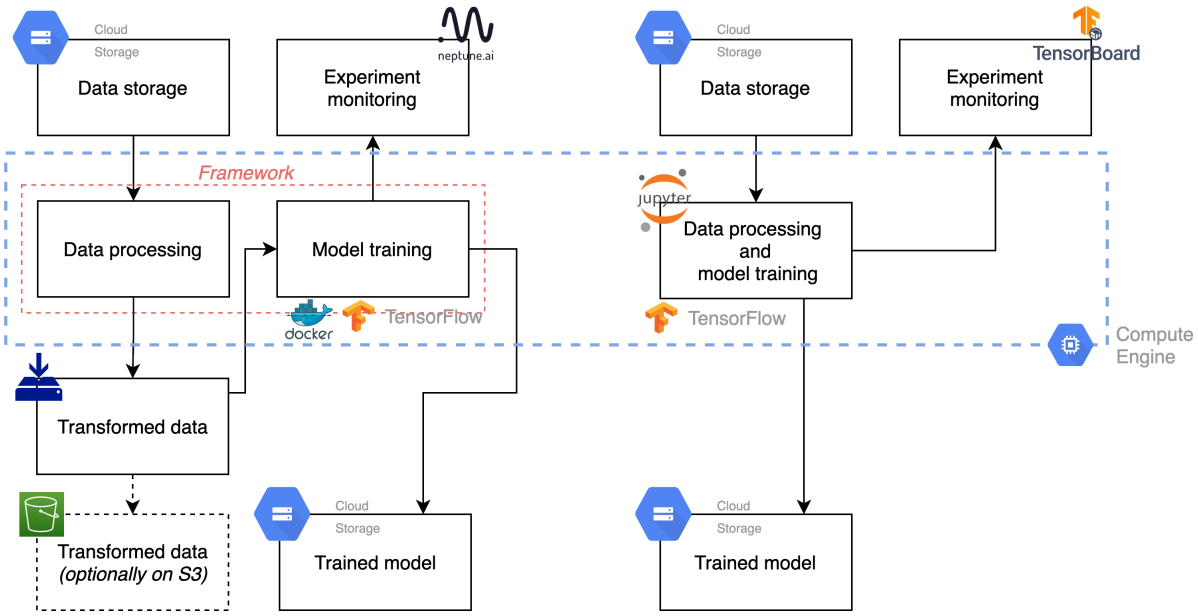

Figure 8: Machine learning workflow pipelines.

4 Methods and approaches

In this section, we present the methods required to develop this master's thesis. While these methods are developed in great detail in the dedicated external resources linked throughout this section, we aim at providing instead a succinct presentation and a minimal mathematical derivation to fully understand the inner workings of these techniques and grasp our classification problem. The first section explores techniques such as TF-IDF, Word2Vec and the application of linear modeling and deep learning techniques on top of these data representations, in order to solve hypothesis 1. Then, recurrent neural networks, and especially bidirectional LSTMS (BiLSTMs), are briefly presented in order to tackle the hypothesis 2. In the third section, we develop multiple concepts, architectures and techniques around Transformer. This will enable to assess the hypotheses 3, 4, 5. Eventually, dimensionality reduction techniques are explored in order to visualize and clusterize attention maps and apprehend the hypothesis 6. Overall, as the Transformer and the BERT architectures constitute the core upon which this master's thesis is built, it will be developed in detail.

What is at stake in NLP ? Natural language processing is mainly applied to string of letters which are sentences of multiple words. These sentences are processed through a tokenization step, which enables the representation of theses sentences as ordered lists of tokens – the most basic tokenization technique takes words as tokens. We can consider proteins in a similar manner: **the protein sequence stands for a given sentence while the amino acids stand for the tokens**. Given this similarity, it appears natural to apply the wide range of NLP techniques developed over the last decades to protein sequences. One should also remark that, with proteins, the vocabulary is the 21-length alphabet of the 20 common AAs and 1 unknown AA, and is therefore far smaller than the size of vocabularies considered in natural languages like English. Below, we will make no distinction between a token and a word w, which corresponds to a given AA. Similarly, sequences will be often assimilated to sentences s.

4.1 TF-IDF, Word2Vec and machine learning

One-hot encoding for MHCs For all the modeling techniques introduced in this section, the information from the MHC is considered as a categorical feature and is one-hot encoded. The MHC feature for *netmhcpan4.1* is thus transformed to a sparse representation of 130 one-hot vectors.

Peptide sequence representation In order to apply linear modeling methods or simple deep learning on these protein sequences, a first step consists in building a representation out of these lists of AA tokens. Indeed, most machine-learning algorithms (e.g. logistic regression, naïve bayes, neural networks) require a fixed-size input vector of features.

4.1.1 Data representation

The following concepts and techniques developed below are mainly derived from their detailed explanation from this Speech and Language Processing course [Jurafsky and Martin, 2021].

Bag-of-words (BoW) This is one of the simplest feature extraction technique used in NLP. The basic idea of BoW is to count the frequency (term-frequencies (TF)) of each token in the sentence, without taking into account their order. The assumption behind this technique is that these token frequencies can help to compare protein sequences by estimating their similarity, which enables the ultimate prediction of global properties of a given protein sequence. BoW can also count multi-token combinations, called n-grams (e.g. "WS" (2-gram) or "PYC" (3-gram)). However, this could lead to computational problems since the dimension of the input space would explode – AA-level 3-grams would result in $21^3 = 9261$ features.

TF-IDF The naive BoW approach favors highly frequent tokens. In natural language, frequent words like "the" would be given high importance while not being very useful for most classification tasks. Similar behavior can be expected with proteins and this can be misleading. Therefore, these term-frequencies need to be normalized with the counts of AAs in all protein sequences. This produces a Term Frequency-Inverse Document Frequency (TF-IDF) vector for which the formula (3) is given below:

$$TF\text{-}IDF(x,s) = TF(x,s) * IDF(x) = TF(x,s) * log(\frac{1+N}{1+DF(x)}) \tag{3}$$

with $TF(x,s)$ the term frequency of token x in sequence s, N the number of sequences, $DF(x)$ the number of times x appears in a sequence.

Implementation In order to build a first representation of our peptides' data, we apply this feature extraction step to the peptide sequences, using either 1-grams or 1 and 2-grams, resulting in a 21-length or 431-length vocabulary, and hence a space of either 21 or 431 features for the peptides. As an illustration, we can observe the inverse-document frequencies for 1-grams when considering the *netmhcpan4.1* dataset in Figure 9. For instance, one can remark a low frequency for the cysteine (C).

Word2Vec Despite its simplicity, TF-IDF proved to be an efficient way to represent sequences and often holds as the feature engineering step in first NLP baselines. However, one can see some of its limitations: the data representation built by TF-IDF is long (especially for high n-grams) and sparse. In machine learning, short vectors are easier to use as features and dense vectors may generalize better than storing explicit normalized counts. Furthermore, the TF-IDF technique does not handle the context around

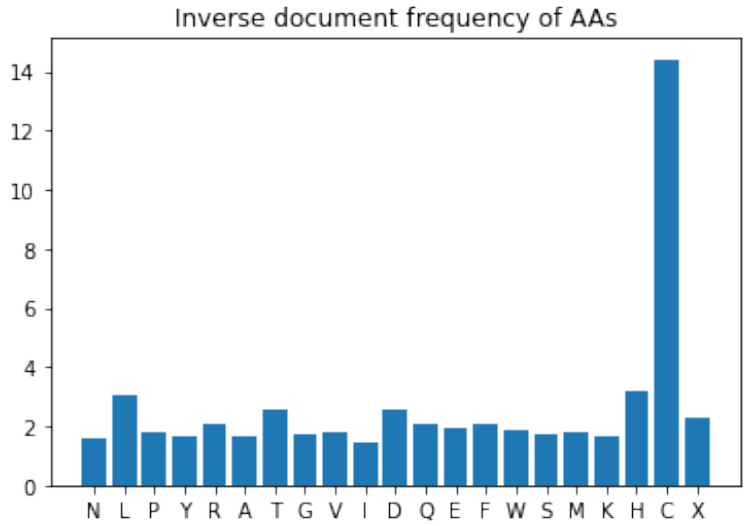

Figure 9: Inverse-document frequencies for AAs on *netmhcpan4.1*.

a given word and does not capture any sense of similarity between word representations. This is exactly what Word2Vec seeks to do. This popular embedding method, very fast to train, tries to determine the meaning of a word based on its context, i.e. its neighboring words. The algorithm exists in various flavors and we will focus on the "Skip-gram with negative sampling" approach. The idea behind this algorithm (1) is to model if a given context word c, taken from a window around a target word w, is indeed a context word for this pair of words w and c.

Algorithm 1 Skip-gram with negative sampling.

1: Treat the target word w and a neighboring context word c as positive examples.
2: Randomly sample k other words in the lexicon to get negative samples.
3: Train a classifier using (stochastic) gradient descent to learn weights W and C.
4: Use these weights as embeddings.

We will make no distinction in notation between a word (like w) and its corresponding embedding vector. As we want to capture a kind of similarity between w and c, the probabilities of a pair being positive or negative can be noted as the following:

$$\begin{aligned} P(+|w,c) &= \frac{1}{1+e^{-w^T c}} = \sigma(w^T c) \\ P(-|w,c) = 1 - P(+|w,c) &= \frac{e^{-w^T c}}{1+e^{-w^T c}} = 1 - \sigma(w^T c) \end{aligned} \tag{4}$$

For a given target word w, one of its context word c_{pos} and the k selected negative samples c_{neg_i}, assuming that all context words are independent, we obtain the following cross-entropy loss which we want to minimize:

$$\begin{aligned}\mathcal{L}_{skipgram} &= -log[P(+|w, c_{pos}) \prod_{i=1}^{k} P(-|w, c_{neg_i})] \\ &= -[log(\sigma(w^T c_{pos}) + \sum_{i=1}^{k} log(\sigma(-w^T c_{neg_i}))]\end{aligned} \tag{5}$$

We can derive (5) with respect to c_{pos}, c_{neg} and w and use gradient descent (GD) (B) for which we have the following update equations, given a learning rate η:

$$\begin{aligned}c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma({c_{pos}^{t}}^T w^t) - 1]w^t \\ c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma({c_{neg}^{t}}^T w^t)]w^t \\ w^{t+1} &= w^t - \eta[\sigma({c_{pos}^{t}}^T w^t) - 1]c_{pos}^{t} + \eta \sum_{i=1}^{k}[\sigma({c_{neg_i}^{t}}^T w^t)]c_{neg_i}^{t}\end{aligned} \tag{6}$$

Then, we obtained weights matrices W and C and the final embedding for the i-th word is represented as a vector $w_i + c_i$.

Implementation On *netmhcpan4.1*, we choose an embedding size of 100 and a window size of 5. For a given sequence, we then concatenate the maximum, minimum and average pooling vectors over the word embeddings. This gives us a final representation of 300 dense features for the peptides' sequences.

4.1.2 Linear classification

Once we have such a fixed representation for our peptide and MHC data, we can build a classifier upon it. We will first consider classifiers that build linear boundaries on the feature space.

Let's consider our dataset $D = (X, Y) = \{(x_1, y_1), ..., (x_N, y_N)\}$, with $x \in R^d$ and $y \in \{0, 1\}$, for which we aim at building a distribution $p(y|x)$. By Bayes rule, we have:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

Discriminative models are models that directly estimate $p(y|x)$ while generative classifiers model the distributions $p(x|y)$ and $p(y)$ instead. We will first consider such a generative model: the naïve bayes, and then develop a discriminative one: the logistic regression.

Naïve Bayes With this algorithm, we aim at modeling $p(x|y)$ and $p(y)$ in order to predict the class with $\hat{y} = argmax_y p(y)p(x|y)$. We can use Maximum A Posteriori (MAP) estimation to estimate $p(y)$ and $p(x|y)$. In our binary classification case, estimating can be done by counting how many times we observe each outcome. For $c \in \{0, 1\}$:

$$p(y = c) = \frac{\sum_{i=1}^{N} I(y_i = c)}{n} = \hat{\pi}_c \tag{7}$$

In order to estimate $p(x|y)$, Naïve Bayes makes the strong assumption that feature values are independent given the label: $p(x|y) = \prod_{i=1}^{d} p(x_i|y)$, where x_i is the value for feature i. Then, the estimation depends on the assumption we make on the distribution of x given y. Given our TF-IDF (peptide) and categorical (MHC) feature space, we make the assumption that it follows a multinomial distribution, parametrized by vectors $\theta_y = (\theta_{y1}, ..., \theta_{yd})$ for each class y where d is the number of features and θ_{yi} is the probability $p(x_i|y)$ of a feature i appearing in a sample belonging to class y. Then, the parameters can be estimated by a smooth version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha d} \tag{8}$$

where $N_{yi} = \sum_{x \in T}$ is the number of times feature i appears in a sample of class in the training set T, and $N_y = \sum_{i=1}^{d} N_{yi}$ is the total count of all features for class y. The smoothing prior α accounts for features not present in the learning samples and prevents zero probabilities in further computations – it is often chosen equal to 1, which corresponds to Laplace smoothing.

Implementation Therefore, we train such a Naïve Bayes on the TF-IDF representation of peptides and the one-hot encoding of MHCs, using a class weight to support the unbalanced dataset: positive data points are given higher importance while training through subsampling.

In this case of binary classification and multinomial features, it can be shown [Weinberger, 2018] that Naïve Bayes is a linear classifier. In NLP, this approach is widely used to build first baselines. Despite being surprisingly good for sequence classification and requiring a few amount of data, this technique has one main downside: its naïve conditional independance assumption is pretty unrealistic in most cases. Another technique which can robustly deal with correlated features by just distributing weight elements accordingly is the logistic regression.

Logistic Regression As a discriminative model, logistic regression aims at finding parameters w, b such that:

$$p_{w,b}(y|x) = \hat{y}^y(1-\hat{y})^{1-y}, \qquad \hat{y} = \sigma(w^T x + b) \tag{9}$$

This is done by minimizing the cross-entropy loss defined in the following equation:

$$\begin{aligned} \mathcal{L}_{CE}(\hat{y}, y) &= -\log p_{w,b}(y|x) + C\|w\| \\ &= -[y \log \sigma(w^T x + b) + (1-y)\log(\sigma(1 - w^T x + b)] + C\|w\| \end{aligned} \tag{10}$$

In the equation above (10), C is an hyper-parameter to more or less regularize the norm of the parameters' vector, using the L2-regularization (Gaussian prior) or L1-regularization (Laplace prior). This is quite useful to avoid over-fitting.

Implementation Many solvers are available for the logistic regression. Since our dataset is quite large, we will use a solver based on stochastic gradient descent (SGD). You can find details on this technique in Appendix B. For the regularization, we use *elasticnet*, which combines L1 and L2 priors as regularizers. Multiple regularization hyper-parameters C are tested in the range $[0.1; 10]$ through grid search hyper-parameter optimization. Once again, this logistic regression is applied on our TF-IDF-based representation of peptides.

4.1.3 Non-linear classification

Feed-forward neural networks The previously developed methods assume that the data can be linearly separated. However, the latter is quite a strong assumption, and non-linear feature interactions might be useful to better model the binding/presentation prediction. Nowadays, deep learning is a popular technique used to tackle non-linearities. Some detailed insights of this modeling technique can be found in [Günnemann and Shchur, 2020], and we will briefly recall the basic foundations of deep learning, since it is at the core of this master's thesis – all the remaining approaches are based on deep learning. In its most basic consideration, deep learning can be seen as modeling a function which is built by layers that first linearly transform an input data into a new feature space and then apply non-linear activation functions, such as the sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$) or ReLU ($ReLU(x) = max(0, x)$), on this transformation. Feed-forward neural networks consists in this basic architecture, for which the modeled function can be written as $f(x, W) = W_L\sigma_L(W_{L-1}\sigma_{L-1}(...\sigma_0(W_0 x)))$, with σ as activation function. This technique is initially based on the Universal Approximation Theorem stated below:

Universal Approximation Theorem *An MLP with a linear output layer and one hidden layer can approximate any continuous function defined over a closed and bounded subset of R^D, under mild assumptions on the activation function and given the number of hidden units is large enough. [Cybenko 1989; Funahashi 1989; Hornik et al 1989, 1991; Hartman et al 1990]*

According to the universal approximation theorem, a two-layer feed-forward network can represent any function. However, in practice, the learning algorithm does not guarantee to find the true parameters. This is why, a large amount of more elaborated architectures have been developed over time, such as the recurrent neural networks or attention-based networks, that we will develop in the following sections 4.2, 4.3. Similar to the feed-forward neural networks (FFNN), each operation, activation or loss in these architectures are differentiable, so that local gradients can be computed. Indeed, in order to learn the parameters of a given deep learning architecture, the latter relies on a learning paradigm: automatic differentiation, or more commonly called "backpropagation".

Backpropagation Each architecture can be seen as a computational graph. At each learning step, we first do a forward pass with a given input. The output of this forward pass is then compared to the ground truth, thanks to a loss function differentiable with respect to the parameters W of the model – in this very case, we will use once again the cross-entropy loss $CE(W) = -\sum_{i=1}^{N}(y_i \log f(x_i, W))$. Using the local gradients of the loss function with respect to the parameter at each node or activation, we can perform a backward pass to "back-propagate" and update the parameters with a gradient update scheme such as gradient descent, stochastic gradient descent, or more elaborated schemes such as the Adam optimizer (see Appendix B). Even if the function loss is often non-convex, this learning scheme will allows us to optimize over it and find local (and potentially global) minima.

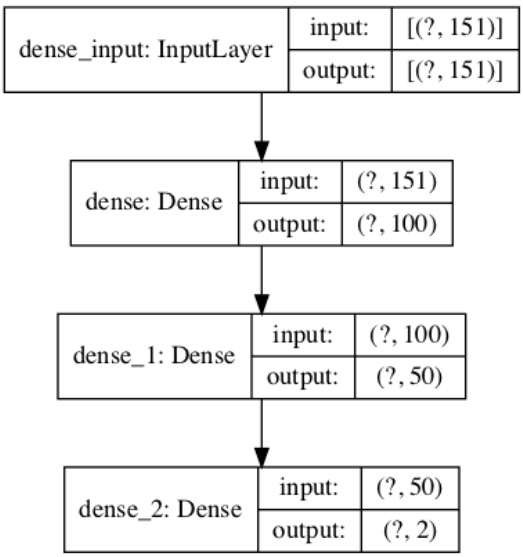

Figure 10: FFNN architecture on TF-IDF representation – *netmhcpan4.1*.

Implementation On *netmhcpan4.1* data, we will build a simple neural network with two-hidden dense (linear multiplications) layers (respectively with 100 and 50 units,

ReLU activations) and will train it on the 1-gram TF-IDF representation (151 features), the 1 and 2-gram TF-IDF representation (561 features), and the Word2Vec representation (430 features). An illustration of this basic architecture can be found in Figure 10. Once again, negative samples are sub-sampled in order to balance the classes and we use *elasticnet* to regularize the weights. The loss is the binary cross-entropy, optimized with Adam (see details in Appendix B). ROC-AUC on the validation split is monitored and we use early stopping based on this metric to stop when validation performance does not increase.

Deep learning architectures Generally, more complicated architectures such as RNNs (4.2) or Transformers (4.3) can be seen as simple FFNNs for which some weights are shared among certain nodes and interactions are limited to certain nodes. This mainly brings two advantages:

- the network needs fewer learnable parameters and, by implication, the risk of overfitting is reduced,
- it introduces an inductive bias from the chosen architecture, i.e. "forces" the model to exploit the characteristics of the data (e.g., recurrent information in sequences) and allows the model to prioritize one solution (or interpretation) over another.

4.2 Recurrent neural networks and BiLSTMs

In the last approaches, we applied linear and non-linear modeling techniques to a feature space that were previously built (4.1.1). However, such a data representation obtained from TF-IDF or a pooling over word embeddings has several drawbacks. First, finding an appropriate size for this input feature space can be cumbersome and if the latter becomes too large, we might suffer from the curse of dimensionality. Secondly, while Word2Vec builds word embeddings, the latter do not take local word order into account and do not consider the surrounding context of words. Hence, such approach might be limited given the intrinsic sequential nature of proteins. Recurrent neural networks [Rumelhart and McClelland, 1987] are a type of deep learning approach that integrates the recursion of sequences in its architecture. The explanations of this section are built upon the following resources: Günnemann and Shchur [2020]; Goodfellow et al. [2016].

RNNs The initial setup of recurrent neural networks is, given a sequence of input tokens $(x_1, ..., x_L)$ and outputs $(y_1, ..., y_L)$, L being the length of the token sequences, to model the probability $p(y_t|x_1, ...x_t)$. As such, the model represents the sequence up until the $(t-1)$-th token $(x_1, ..., x_{t-1})$ with a hidden state h^{t-1}. In practice, the forward pass is done sequentially and the update equations at each token t can be found in

(11). In these equations, W, U and b are shared parameters over all token steps. An illustration is also given in Figure 11.

$$\begin{aligned} z_t &= Wh_{t-1} + Ux_t + b \\ h_t &= tanh(z_t) \\ o_t &= Vh_t \\ \hat{y}_t &= softmax(o_t) \end{aligned} \tag{11}$$

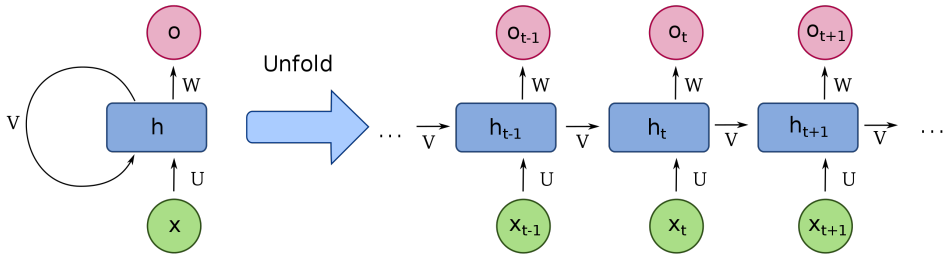

Figure 11: Recurrent neural networks.

Source: Wikipedia.

In our case, we aim at extracting a representation of the entire peptide sequence. It is a common practice to use the final latent state of a recurrent model as a representation for the sequence. The clear intuition is that by the time the model has processed all of the sequence elements through the recursive chaining, the last hidden state is a latent representation that encodes information for the entire sequence. For that reason, we will use the last token output $\hat{y}_L$, as an output representation that can be merged in a more broad deep learning architecture which integrates other features, such as the allele category in our case.

Backpropagation through time The hidden state h_t recursively depends on all previous hidden states $(h_0, ..., h_{t-1})$. This implies that, during backpropagation, the gradient of a given loss with respect to h_t will depend on future times $(h_{t+1}, ..., h_L)$ for $t \in [1; L-1]$. In order to illustrate this statement, consider that we apply a binary cross-entropy loss directly on the last output, comparing it to a binary ground truth of the entire sequence: $\mathcal{L} = -[y \log \hat{y}_L + (1-y)\log(1-\hat{y}_L)]$. Then, derivation leads to the given formula:

$$\frac{\partial \mathcal{L}}{\partial h_t} = W^T diag(1 - (h_{t+1})^2) \frac{\partial L}{\partial h^{t+1}} \tag{12}$$

Impact of future times might vanish or explode so that native RNNs cannot retain information for many steps. It becomes hard to retain long-range dependencies, which can largely occurs in protein sequences (as the protein sequence of AAs folds into a complex 3-D shape). Long-short-term-memory (LSTM) provides an adaptation of the native RNN update cell and provides a good first contribution towards solving this issue.

LSTM This technique [Hochreiter and Schmidhuber, 1997] introduces a cell state c_t in addition to h_t and gating mechanisms (forget f_t, input i_t and output o_t "gates") that can be mathematically derived in the following equations (13). Like for the RNN update cell, an illustration of the LSTM unit is given in Figure 12.

$$\begin{aligned} f_t &= \sigma(W_f[h_{t-1}, x_t]) \\ i_t &= \sigma(W_i[h_{t-1}, x_t]) \\ o_t &= \sigma(W_o[h_{t-1}, x_t]) \\ c_t &= f_t \odot c_{t-1} + i_t \odot tanh(W[h_{t-1}, x_t]) \\ h_t &= o_t \odot tanh(c_t) \end{aligned} \tag{13}$$

Above, $\odot$ stands for the Hadamard product (component-wise product). With this gating mechanism, LSTM units decide how much of the new input should be written to the memory cell, and how much of the current content of the memory cell should be forgotten. Hence, some inputs can be skipped to better capture long-term dependencies.

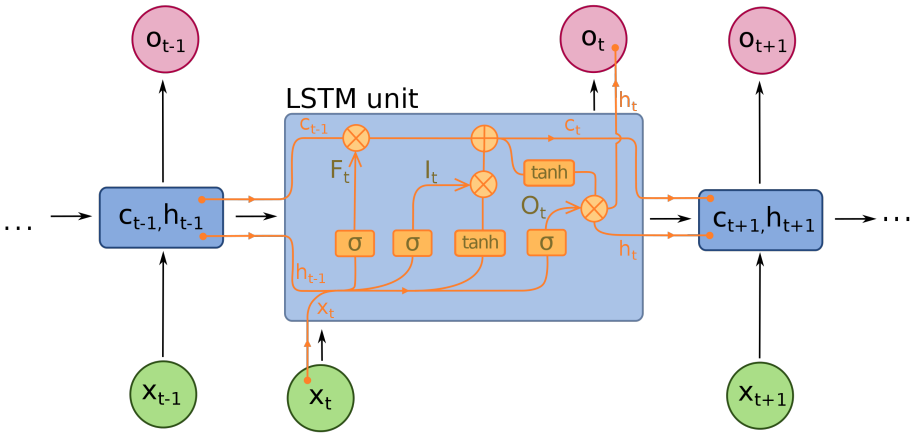

Figure 12: Long-short-term-memory unit representation.

Source: Wikipedia.

Because of their internal memory, RNNs, and more specifically RNN with LSTMs, can remember important things about the input they received, which allows them to

be very precise in representing valuable sequential information. This is why they are a popular algorithm for sequential data like time series, text, audio, or proteins. Recurrent neural networks can form a much deeper understanding of a sequence and its context compared to other algorithms. Specifically, for language modelling, the hidden states learnt through LSTM layers better represent the interaction of words rather than traditional n-gram based models. The representation is contextual (e.g. ”left” in ”I left my phone” will have a different embeddings representation rather than in ”the phone is on the left of the table”) and happens inside the algorithm and not as a first preprocessing step, as we did in previous section 4.1.

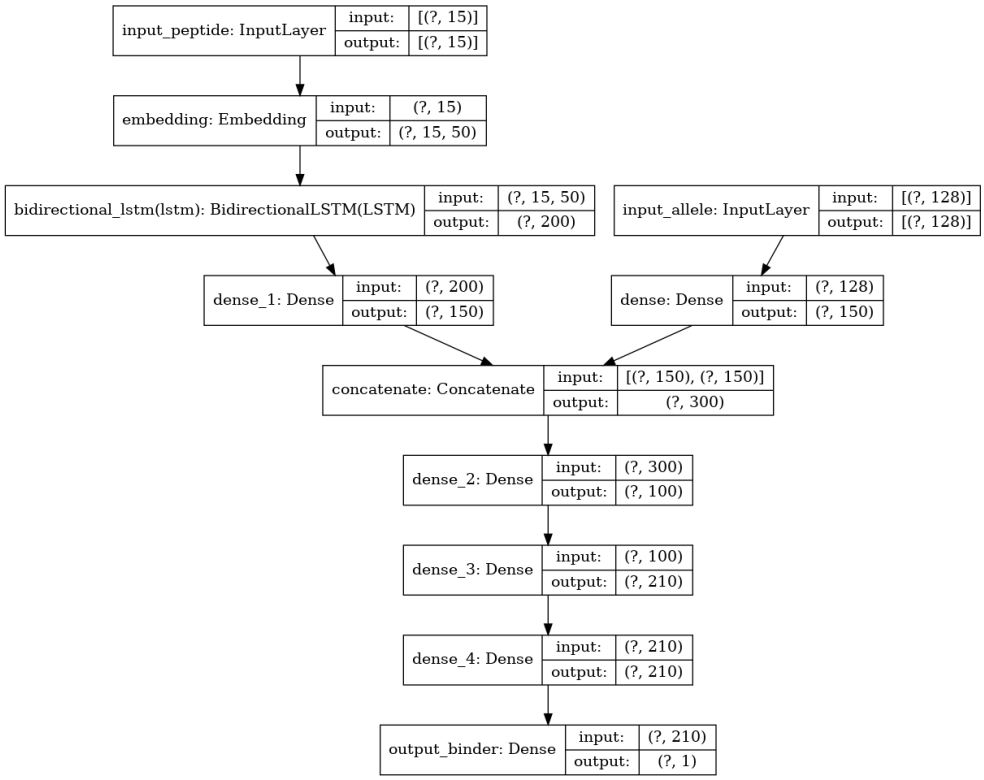

Figure 13: Architecture of our BiLSTM approach.

Scaling Let us take a look back to the feed-forward neural networks parametrization technique (4.1.3). Through the use of fully connected layers, they parametrize functions that scale linearly with the dimensionality of the input sequence x. This makes FFNN prone to overfitting. In contrary, RNNs attempt to mitigate this overfitting tendency by controlling the expressivity of the paramatrized function class: the number of parameters in a RNN is independant of the number of inputs L

Implementation First of all, one can observe that, as we take the last token output of the sequence, the LSTM is uni-directional. In order to capture bi-directionality which

is inherent to protein sequences, we stack two LSTM layers in what we call a BiLSTM layer. Therefore, in order to compare the value brought by BiLSTMs on the *netmhcpan4.1* dataset, we will embed the AAs from the peptide into a learnable 50-length embedding, on which we apply a BiLSTM layer of $2 * 100 = 200$ units (output dimension). This output is then transformed by a Dense layer and concatenated to the dense representation of the categorical MHC feature. Eventually, Dense layers are added on the top of this concatenated input in order to predict the presentation. This architecture is illustrated in Figure 13, the model has roughly 260k parameters. It is trained using the Adam optimizer (explained in Appendix B) for 30 epochs using early stopping. Instead of sub-sampling, the focal loss, which is explained below, is used to handle the less frequent positive class. As an example, Appendix F provides the configuration used to train such a model in the framework.

Focal loss This loss [Lin et al., 2018] refines the original binary cross-entropy loss $CE(p(y|x)) = -\log(p(y|x))$ by adding a scaling factor: $FL(p(y|x) = (1 - p(y|x))^{\gamma} \times CE(p(y|x)))$, thus reducing the impact of well-classified instances on the loss. These "easy" examples are down-weighted during training, so that learning focuses on hard negative examples, as seen in Figure 14. The scaling factor decays to zero as confidence in the correct class increases: the cross-entropy loss is consequently dynamically scaled.

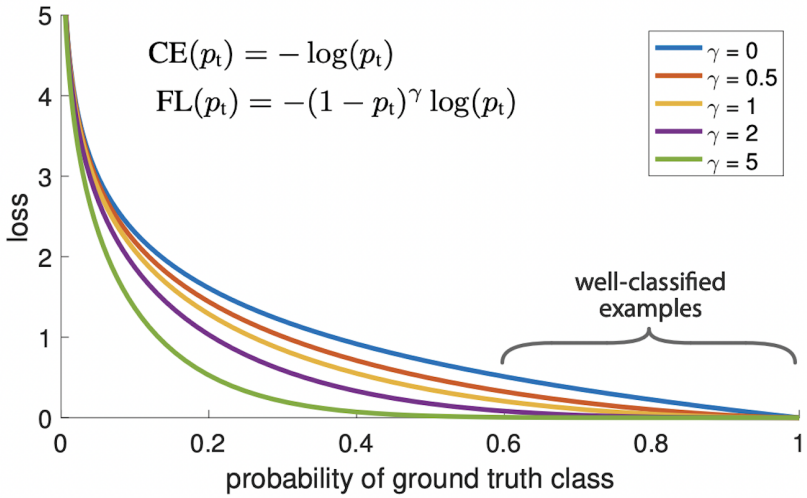

Figure 14: Focal loss for multiple γ factors.

Source: Lin et al. [2018]

4.3 Transformer

By definition, the sequential nature of the RNN architecture (4.2) prevents parallelization – each RNN layer needs the output of the previous hidden state to com-

pute the next hidden state. As such, RNN layers are processed sequentially. Despite more elaborated techniques such as LSTM, RNN-based representation are still not suitable for parallelization. From this observation and preceding intensive research on sequence transduction, a new paradigm emerged recently: the Transformer [Vaswani et al., 2017]. In this paper, the authors addressed many of the pitfalls of RNN-based models by replacing the reliance on RNNs and their hidden states with attention-based operations (4.3.1). In doing so, the Transformer architecture eliminated the most significant bottleneck in the training process, and also brought new state-of-the-art results on tasks such as sentence classification or machine translation. The following details and mathematical intuitions have been constructed with the original paper and publicly available resources [Manning and Hewitt, 2021; Thickstun, 2019].

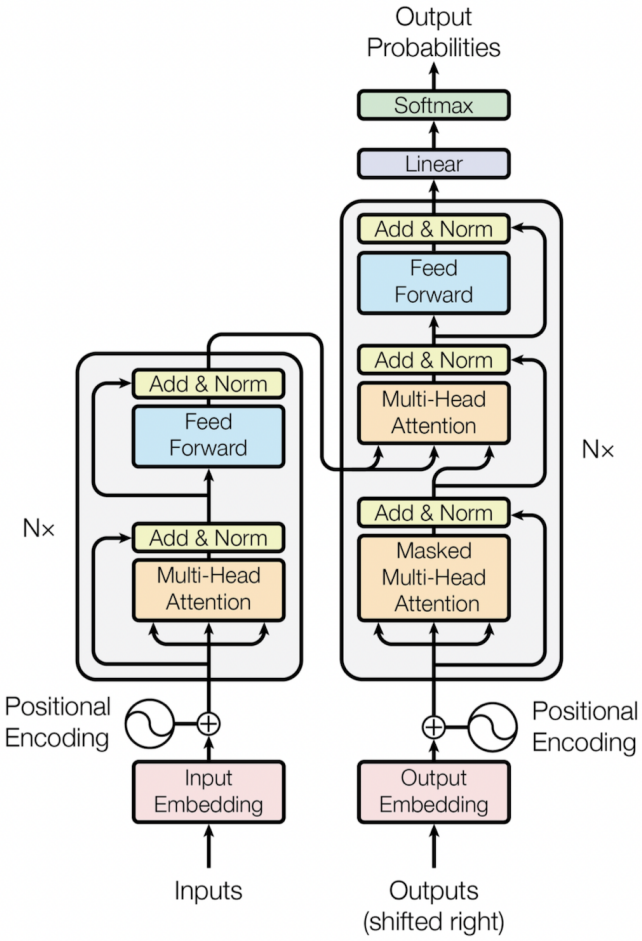

Figure 15: Transformer encoder-decoder blocks.

Source: [Vaswani et al., 2017]

At first look (Figure 15), the Transformer is an encoder-decoder architecture with attention (4.3.1) at its core. The encoder and the decoder stacks multiple Transformer blocks. Take an input sequence $s = (AA_1, ..., AA_L)$, with L being the maximum length over all input sequences and $AA_i \in R^{d_{in}}$. Similar to previous methods such as BiLSTM, this input sequence is first embedded into a dense representation $x = (x_1, ..., x_L)$, $x_i \in R^d$, where words might form clusters given their similarity. This embedding linear projection is parametrized by learnable weights. We no longer consider this input as a sequence but as a set of elements (tokens), hence getting rid of their order and dependencies. As such, a Transformer block can be seen as an automorphic parametrization function class $f_\theta : R^{L\times d} \to R^{L\times d}$, $z = f_\theta(x)$. In the following sections, we will expand on each of transformations steps of the function f_θ and detail the intuition behind.

4.3.1 The attention mechanism

Self attention The first transformations applied to our input x, illustrated by the orange blocks in Figure 15, can be wrapped up into the following equations, for $i \in L$ and h such that $d = k \times h$:

$$Q^{(h)}(x_i) = W_{q,h}x_i, K^{(h)}(x_i) = W_{k,h}x_i, V^{(h)}(x_i) = W_{v,h}x_i \qquad W_{\cdot,\cdot} \in R^{k\times d} \tag{14}$$

$$\alpha_{i,j}^{(h)} = softmax_j(\frac{\langle Q^{(h)}(x_i), K^{(h)}(x_i)\rangle}{\sqrt{k}}) \tag{15}$$

$$u'_i = W_o \times concat_{(h)}(\sum_{j=1}^{L} \alpha_{i,j}^{(h)} V^{(h)}(x_j)) \qquad W_o \in R^{d\times d} \tag{16}$$

First, observe that we consider H sets of equations, indexed by h. Each of this set of equations and parameters is an attention head. In a given head h, elements of x attend to each other. This can be seen with the attention weight $\alpha_{i,j}^{(h)}$ that controls how much element x_i attends to x_j. Interaction between objects x_i and x_j only occurs in equation 16, which has no prior or constraints: attention can happen equally to all tokens. One interpretation of the self-attention layer is as a learned and differentiable lookup table. The query functions queries Q, keys K, and V are can be seen as "queries", "keys" and "values" respectively. Each query $Q^{(h)}(x_i)$ will compute a similarity score with each key $K^{(h)}(x_j)$ for all the elements x_j, defined by the cosine similarity between both:

$$cos(Q^{(h)}(x_i), K^{(h)}(x_j)) = \frac{Q^{(h)}(x_i) \cdot K^{(h)}(x_j)}{|Q^{(h)}(x_i)||K^{(h)}(x_j)|}$$

This similarity score enables to select some values $V^{(h)}(x_j)$ which will be weighted in the end with equation 16. We construct a soft lookup of values compatible with x_i: an attention mechanism thus condenses a set of hidden states into a weighted sum of its constituent vectors, and the weighting would be based in part on the contents of

the vectors. In other words, we find correlations between different words of the input, indicating the syntactic and contextual structure of the sequence.

Multi-head attention In practice, we run through an attention head h several times, in order to build several representations at different layers. These attention heads are concatenated in a multi-head attention layer. Its representation; as well as the inner attention mechanism, can be seen in Figure 16. The intuition behind multi-head attention is that it allows us to attend to different parts of the sequence differently each time. Therefore, the model can:

- better capture positional information because each head will attend to different segments of the input. The combination of them will give us a more robust representation.
- Each head will capture different contextual information as well, by correlating words in a unique manner.

In the original paper, they used $d = 512$ and 8 concatenated attention heads which results in output vector size of $k = 64$.

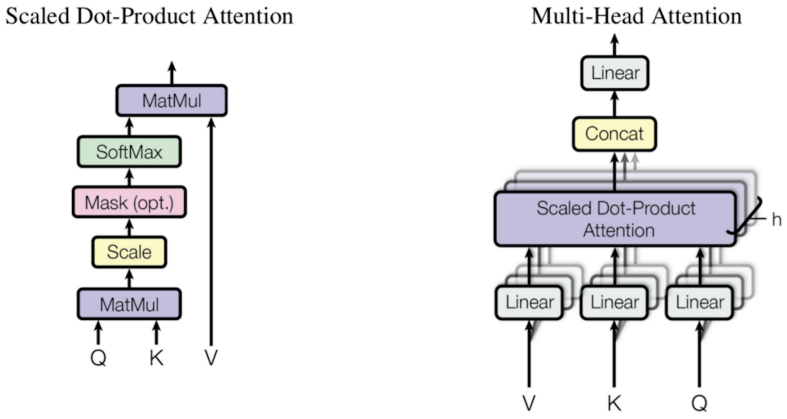

Figure 16: (left) Scaled dot-product attention. (right) Multi-head attention block, several attention layers running in parallel.

Source: [Vaswani et al., 2017]

Multi-head attention is the core of the Transformer architecture and is used in both the encoder and the decoder, in a slighly different manner. Here, since we will focus on using only the encoder part in our approach, we consider the self-attention in its decoder interpretation: tokens attend to other tokens before and after, potentially far

from their immediate surrounding context, in order to build a meaningful representation of the sequence.

Layer normalization, residual connections and feed-forward layers Then, the representation u'_i is added to the initial representation x_i (residual, or "skip", connection – this has been extensively used in vision with algorithms such as *Resnet*), followed by a batch normalization step, a per-object fully connected layer and a final residual connection with batch normalization, as described in the following equations:

$$u_i = LayerNorm(x_i + u'_i; \gamma_1, \beta_1) \qquad \gamma_1, \beta_1 \in R^d \tag{17}$$

$$z'_i = W_2 \times ReLU(W_1 u_i) \qquad W_1 \in R^{m \times d}, W_2 \in R^{d \times m} \tag{18}$$

$$z_i = LayerNorm(u_i + z'_i; \gamma_2, \beta_2) \qquad \gamma_2, \beta_2 \in R^d \tag{19}$$

where, the $LayerNorm$ function is defined for $y \in R^d$:

$$LayerNorm(y; \gamma, \beta) = \gamma \frac{y - \mu_y}{\sigma_y} + \beta \qquad \gamma, \beta \in R^d \tag{20}$$

$$\mu_y = \frac{1}{d} \sum_{i=1} dy_i, \quad \sigma_y = \sqrt{\frac{1}{k} \sum_{i=1}^{d} (y_i - \mu_y)^2} \tag{21}$$

These different transformations, displayed as the yellow and blue blocks in Figure 15, have been mainly motivated by effective optimization of deep learning models and are quite important in practice.

Parametrization Overall, with all the previous equations, we can see that the parameters θ consist of the entries of the weight matrices W, alongside the $LayerNorm$ parameters γ and β. The hyper-parameters of the Transformer are d, k, m, H, as well as the number of such Transformer blocks. Common settings of these hyper-parameters are $d = 512, k = 64, m = 2048, H = 8$. The original paper stacked 6 Transformer blocks but more recent work consider a much deeper stacking.

Position encodings As we lost the notion of order, we need to find a way to represent it. Indeed, Transformer processes sequences as sets and they are, in theory, permutation invariant. To overcome this issue, the authors of [Vaswani et al., 2017] added small constants to the initial word embeddings, so that a given word representation would be slightly different depending on where it was initially in the sentence. In the original paper, a sinusoidal function is used, to pay attention to a wave length and, in turn, pay attention quite far.

The decoder Part of the original Transformer architecture, where they use similar concepts but where attention is employed through masked multi-head attention in which future tokens are masked and connections are made to encoder attention outputs. This was for a sequence-to-sequence modeling. While it was part of the original paper, we do not expand on this, since our considered Transformer architecture is based on the encoder only.

Scaling and interpretation Once again and similarly to the RNNs (4.2), from a parameter counting perspective, the number of parameters in a Transformer is independent of the number of input tokens L. Also, observe that the values of the self-attention weights are computed on the fly. They are data-dependent dynamic weights because they change dynamically in response to the data (fast weights). Furthermore, deep learning are often seen as non-interpretable "black box". Transformer, in contrast, provides a good way of understanding AAs interactions through the attention maps. Eventually, an interesting connection between LSTMs and the attention can be further explored in [Thickstun, 2019].

4.3.2 An encoder-based architecture

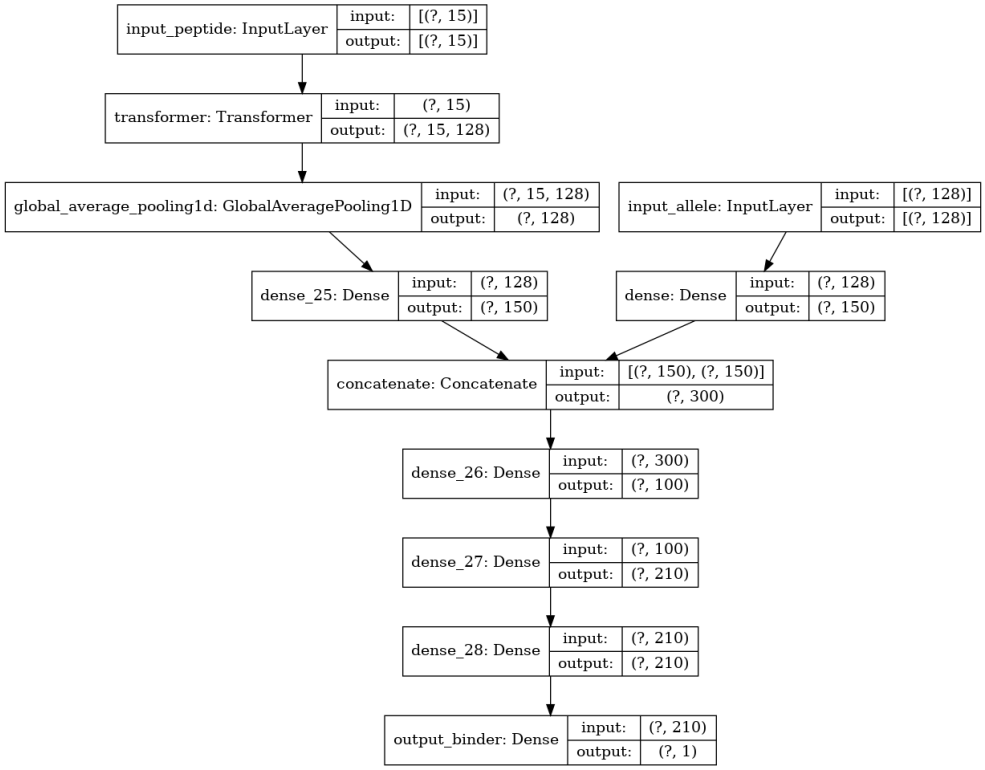

Figure 17: Architecture of our Transformer-based approach.

Our considered architecture One of our very first approach with the Transformer is to incorporate the bidirectional attention-based architecture of its encoder into our architecture, and compare it to our BiLSTM baseline. As such, we transform the previous BiLSTM block in our previous considered model (13) with multiple Transformer blocks as displayed in Figure 17. We process the output $z \in R^{L\times d}$ with a global averaging pooling $g : R^{L\times d} \to R^d,\ a = g(z) = \frac{1}{L}\sum_{i=1}^{L} z_i$ and input it to Dense layers. This is similar to language applications where they typically use a single encoded output, relying either on the last n-th element of the transformed data or on a pooling of this data. Some hyper-parameter optimization led us to consider 4 Transformer blocks with a hidden size of $d = 512$. The allele is categorically encoded but an approach where the allele code is split in three tokens, on which a small Transformer block is applied has been also tested. Similarly to the BiLSTM approach, we used the focal loss in order to tackle the imbalanced nature of our dataset. This approach has been used on both *netmhcpan4.1* and *mhcattnnet*.

4.3.3 Self-supervised learning with BERT

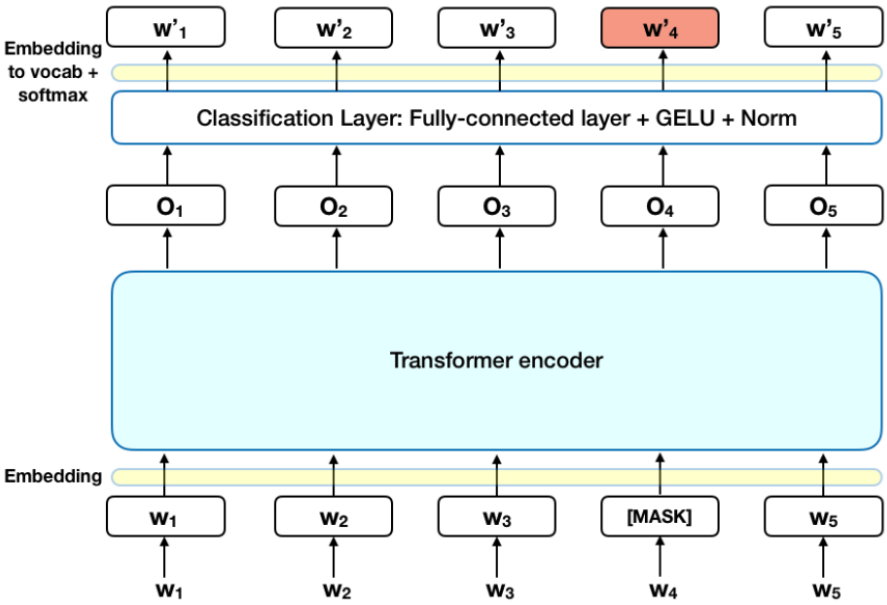

Figure 18: Masked language modeling with BERT.

Source: Rani Horev @ TowardsDataScience

Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2019] is a series of encoder blocks trained on a cleverly designed objective function to account for the encoder's inherent bidirectionality. In short, the BERT's key technical innovation is applying the bidirectional training of Transformer to language mod-

elling. This contrasts to previous language modeling efforts which looked at a text sequence either from left to right (e.g. predict the next word in "The dog came in the ...") or combined left-to-right and right-to-left training, such as ELMO with BiLSTMs. While we have seen that in its vanilla form, Transformer includes two separate mechanisms, an encoder that reads the text input and a decoder that produces a prediction for the task, BERT only uses the encoder mechanism to generate a language model. As such, the researchers provide with two specific tasks to train a model: masked language modeling (MLM) and next sentence prediction (NSP). The latter is more suited to a corpus of ordered sentences in natural language and is not used in the pre-trained models we will consider later in this work. Hence, we will focus on the first task.

Masked language modeling (MLM) The goal of this task is to, given an input set of tokens x partially and randomly masked (for some indices k), predict the original value of these masked tokens x_k, based on the context provided by the other non-masked tokens in the sequence. Such a process is illustrated in Figure 18, where a classification layer is added on top of the encoder output, from which the output is then projected to the vocabulary space with the embedding matrix, and then processed through softmax to calculate a probability of each word. The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words. Unlike left-to-right language model pre-training, the MLM objective allows the representation to fuse the left and the right context. All the pre-training approaches in this master's thesis follow the original mask scheme:

- 15 % of the tokens (amino acids) are masked.
- In 80 % of the cases, the masked AAs are replaced by [MASK].
- In 10 % of the cases, the masked AAs are replaced by a random different AA from the one they replace.
- In the 10 % remaining cases, the masked AAs are left as is.

Leveraging unlabeled protein data The real advantage of BERT is therefore its capacity to be trained in a self-supervised fashion. Hence, it can leverage unlabelled corpus of text (here proteins data) rather than labeled data, that is far more tedious to gather and curate through manual labeling. BERT learns protein language from understanding AAs' cohesion from this large body of protein content and is then educated further by fine-tuning on smaller, more specific tasks such as our classification binding/presentation task. In that respect, large databases, such as BFD [Steinegger and Söding, 2018], UniProt [The UniProt Consortium, 2020] or its UniRef derivations [Suzek et al., 2015], can be used to build a representative language model that can be then adapted to downstream tasks such as our binding/presentation task.

Perplexity A language model assigns probabilities $P(s)$ to sequences of arbitrary symbols such that the more likely a sequence $s = (AA_1, AA_2, ..., AA_L)$ is to exist in that language, the higher the probability. Most language models estimate this probability as a product of each token's probability given its preceding tokens: $P(AA_1, AA_2, ..., AA_L) = \prod_{i=1}^{L} P(AA_i|AA_1, ..., AA_{i-1})$. Alternatively, some language models estimate the probability of each symbol given its neighboring symbols, and this is how the MLM BERT modeling task is trained, i.e. the latter is optimized to find the probability of masked tokens given their surrounding context. In fine, a language model can be evaluated in two ways:

- extrinsically, i.e. the pre-trained language model is "evaluated" by reporting its performance when employed on a downstream task, e.g. how this model performs on a sentence classification task such as our problem,
- intrinsically, i.e. with some metric to evaluate the language model itself, not taking into account the specific tasks it will be used for. It it s a useful way of quickly comparing models and the perplexity is such an intrinsic evaluation metric.

While perplexity might sometimes not reflect the true performance of a LM, it has been shown in some research initiatives (such as in [Rao et al., 2020]) that it is a good proxy metric for further performance on downstream tasks. Given a probability of a sequence $P(AA_1, AA_2, ..., AA_L)$, e.g. the softmax output for mask tokens from a MLM pre-trained, the perplexity is computed as the inverse probability of a given set, normalised by the number of words:

$$PPL(S) = \sqrt[L]{\frac{1}{P(S)}} = \sqrt[L]{\frac{1}{P(AA_1, AA_2, ..., AA_L)}} \tag{22}$$

Alternatively, the perplexity can be defined from the cross-entropy $H(S)$ (Equation 23) and the link between both is detailed in Appendix C. The cross-entropy indicates the average number of bits needed to encode one token, and perplexity is the number of tokens that can be encoded with those bits. Intuitively, if a model assigns a high probability to a set, it means that it is not "perplexed" by it, which means that it has a good understanding of how the language works. A lower perplexity is thus a better performance.

$$PPL(S) = 2^{H(s)} = 2^{-\frac{1}{N} log_2(P(AA_1, AA_2, ..., AA_L))} \tag{23}$$

***prot-bert-bfd*: a pre-trained protein language model (PLM)** This model, publicly available on the HuggingFace hub [Wolf et al., 2020] trained within the ProtTrans research initiative [Elnaggar et al., 2021], is a BERT model which was pre-trained on a

large corpus of protein sequences (BFD [Steinegger and Söding, 2018], a dataset containing 2.1 billion protein sequences) in a self-supervised fashion. In their paper, the researchers demonstrated that the features extracted from this model (last embeddings of the model) captured important biophysical properties governing protein shape. This implies that the model learned some of the grammar of the language of life realized in protein sequences. The architecture of this model has an embeddings size of 1024 and contains 30 Transformer blocks, each one built with 16 attention heads. The model was trained for 800k steps using sequence length 512 (batch size 32k), and 200K steps using sequence length 2048 (batch size 6k). The optimizer used was Lamb with a learning rate of 0.002, a weight decay of 0.01, a learning rate warmup for 140k steps and a linear decay of the learning rate after.

A custom-made tiny pre-trained PLM While we will assess the value of pre-trained models on our downstream classification task with *prot-bert-bfd*, we will see in the next section (4.3.4) that this model results to a large-size architecture. To quickly iterate and analyse the training of such a transferred model – a 420M model requires a lot of compute to be trained, we will herewith also consider a far smaller pre-trained model that we train on UniRef50. The architecture of this model is similar to *esm1_t12_85M_UR50S* from [Rives et al., 2020], with 12 Transformer layers and an embeddings' dimension of 768. Trained on 55×10^9 tokens, it reached a perplexity of 11.05. This is quite close to the one from *esm1_t12_85M_UR50S*, for which further hyper-optimisation has been conducted.

These two pre-trained PLMs will enable us to conduct a precise analysis on how such a PLM can be further used/fine-tuned on our downstream classification task, as explained in the next section 4.3.4. This aims at transferring and leveraging the learnt biological language knowledge on our presentation problem.

4.3.4 Transfer learning with pre-trained models

Data representation First of all, in our attempt to leverage pre-trained PLMs, we change our data representation in order to have a single sequence in input. Accordingly, we build a representation of our peptide-MHC complex as a concatenated sequence $s = [CLS], AA_{a,1}, ..., AA_{a,L_a}, [SEP], AA_{p,1}, ..., AA_{p,L_p}, ([PAD], ...), [SEP]$ where $AA_{a,i}$ is the i-th amino acid in the allele pseudo-sequence, $AA_{p,i}$ the i-th amino acid in the peptide sequence, $[CLS]$ and $[SEP]$ the special classification and separator tokens required for the BERT model input, and $[PAD]$ some optional padding tokens that will be masked through the Transformer layers. We pad sequences in order to have a fixed length input sequence, whose size will be 52 on *netmhcpan4.1* (15 (maximum) peptide' AAs + 34 allele pseudo-sequence' AAs + 3 special tokens). Notably, this representation is similar to the one considered in BertMHC [Cheng et al., 2020].

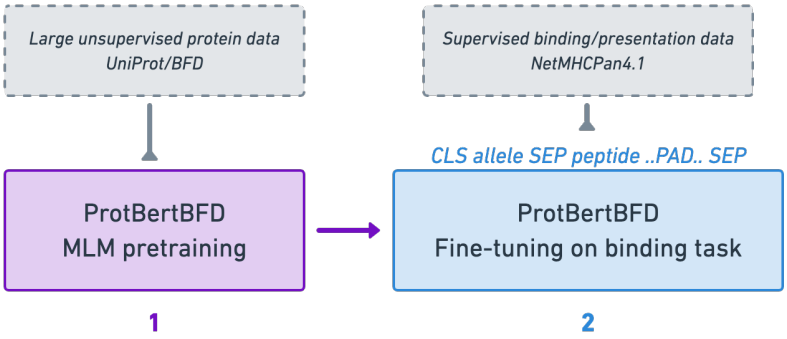

Figure 19: MLM pre-training on general data and fine-tuning.

Direct transfer Our first consideration is to take the pre-trained PLM, such as *prot-bert-bfd*, load a multi-layer perceptron (MLP) classification head (feed-forward layers with a softmax output) on top of the average pooling of the last hidden layer embeddings (1024-dim size), and directly train this architecture on our classification presentation problem, as displayed in Figure 20. As illustrated in Figure 19, this corresponds to a setup where we first pre-train a model on large unsupervised protein data and then we fine-tune on our downstream task with domain peptide-MHC data. While the embedding corresponding to the [CLS] token could be used, we preferred an approach with an average pooling over all the AA embeddings (last layer, excluding special tokens ([PAD], [SEP], [CLS]) is inspired by common practice when considering such a BERT model on downstream tasks.

On the parameters The overall model (Figure 20) contains 420M parameters to optimize, which is quite big. In that respect, we will investigate this model in three different flavors:

- **ProtBertBFD**: a model where all the parameters (embedded BERT + classification MLP head) are fine-tuned and where the parameters of the embedded BERT model has been pre-trained on protein unsupervised data,
- **ProtBertBFD_Frozen**: a model where only the MLP classification head parameters are tuned and where the parameters of the embedded BERT model has been pre-trained on protein unsupervised data,
- **ProtBertBFD_Scratch**: a model where all the parameters (embedded BERT + classification MLP head) are fine-tuned and where the parameters of the embedded BERT model has been randomly initialized (no step 1 in Figure 19).

Frozen/Fine-tuned The embeddings from the last hidden layer of a pre-trained model have proven to contain valuable information for further downstream classifi-

cation at the sequence level without the need to fine-tune the pre-trained model's parameters [Rives et al., 2020; Elnaggar et al., 2021]. Consequently, we hope to be able to reach decent performance without the need of fine-tuning the entire model, which requires a lot of compute, and thus time and money.

Pre-trained/Trained from scratch Furthermore, we have seen that the model is quite big. Thus, should this model attains great performance, it could be because of the vector space size of the parametrized functions it can model: a not pre-trained model with the same architecture could learn the same representation. This is why we also train a similar model for which the parameters have been randomly initialized (ProtBertBFD_Scratch).

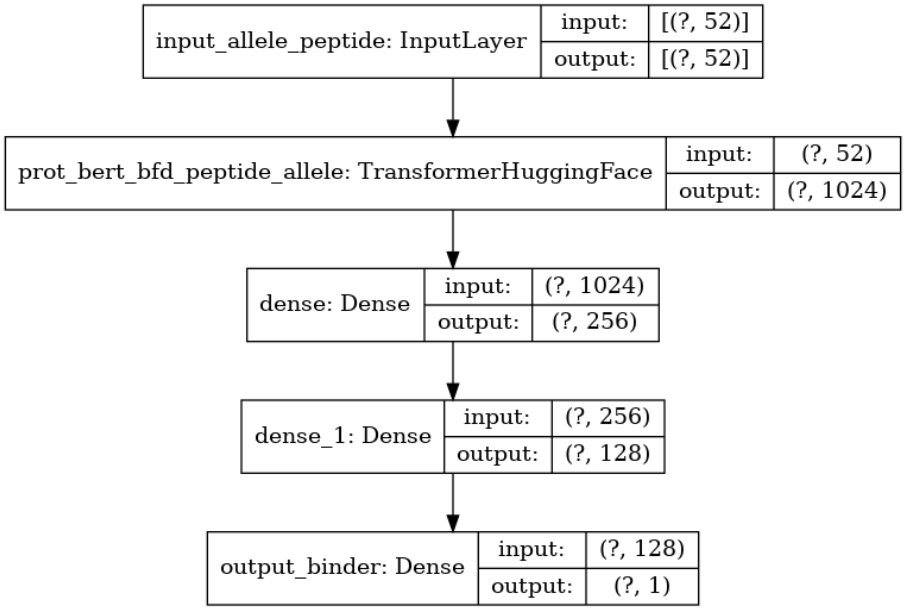

Figure 20: Architecture for the fine-tuning of a PLM on a binding task - *netmhcpan4.1* data

Training ProtBertBFD has been trained for 30 epochs on a A100 GPU with a batch size of 128. The latter was required to be quite small so that the models fits in the 40G memory of the A100. We used a learning rate scheduler with a warmup (for half an epoch) and a linear decay from the maximal learning rate $lr = 5e^{-6}$. This learning rate has been found with some cyclical learning rate strategies described in Appendix D. As with previous methods, the Adam optimizer and the focal loss are used.

Similar approach with the tinier pre-trained PLM In order to analyse the training curves, a similar approach has been conducted with the custom-made tiny pre-trained PLM. The fine-tuned, trained-from-scratch and frozen setups have been trained by

replacing only the embedded pre-trained PLM. The latter being quite smaller, we used a batch size of 256 and trained on a 32GB V100 GPU with similar other parameters described for the other methods.

Other considerations Throughout this 6-months work, other methods have also been tested. As an example, we tried to integrate the pretrained frozen/fine-tuned model to other state-of-the-art approaches incorporating more features and learning over non-public data. These methods won't developed in this master's thesis, because they use proprietary approaches from InstaDeep and BioNTech, that can be disclosed in this thesis. However, this does not prevent us to investigate the value brought by pre-trained models.

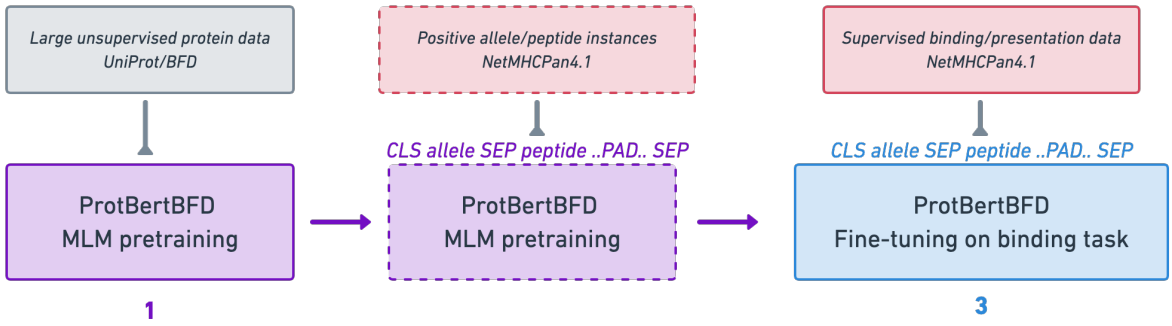

Figure 21: Additional MLM pre-training on domain positive data.

An additional MLM pre-training task on positive data Eventually, the reasoning around the fifth hypothesis (2.4) led us to consider an additional pre-training step (step 2 in Figure 21) where the pre-trained PLM is trained on relevant domain data before fine-tuning onto the classification task. The relevant domain data consists in the positive peptide-MHC instances here. As such, we will therefore explore if such a pre-training step could help boost the final downstream performance. In order to additionally pre-train ProtBertBFD, an hyper-parameter search over the learning rate has been conducted, for which the learning loss curves can be found in Appendix E. The resulting model reaches a perplexity of 10.82, and will be fine-tuned on the *netmhcpan4.1* dataset, using the same architecture as before (Figure 20). We refer to this transferred model as **ProtBertBFD_MLM**.

4.4 Dimensionality reduction with t-SNE

In this section, we briefly present a machine learning technique for dimensionality reduction that identifies relevant local patterns in high-dimensional data, such as embeddings or attention maps. This will be useful to visualize the latter in 2-D. This technique is called t-distributed stochastic neighbor embedding (t-SNE) [van der Maaten

and Hinton, 2008]. This algorithm models the probability distribution of neighbors around each point, i.e. the set of points close to each point. In the original high-dimensional space, this is modeled as a Gaussian distribution. As such, the conditional probability of point x_j to be next to point x_i is given by:

$$p_{i|j} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)} \tag{24}$$

The symmetric joint probability is then obtained with $p_{i,j} = \frac{p_{i|j}+p_{j|i}}{2n}$, while in the 2-dimensional output space, the probability distribution of neighbors is modeled as a t-distribution:

$$q_{i,j} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}} \tag{25}$$

The goal of the procedure is to find a mapping onto the 2-dimensional space that minimizes the differences between these two distributions p and q over all points. This is done by minimizing the Kullback-Leibler divergence $KL(p||q)$ with gradient descent.

Besides, the main parameter controlling the fitting is called the perplexity, which corresponds to the standard deviation σ_i in the gaussian distribution. Perplexity can be intuitively seen as the number of nearest neighbors considered when matching the original and fitted distributions for each point. A low perplexity will strongly focus on local scale. For our visualizations, we will use a perplexity equals to 40. Overall, the distributions are distance based, so t-SNE should be applied on numeric data.

5 Results and discussion

5.1 Incremental improvements on presentation prediction

5.1.1 Comparison between linear and non-linear modeling techniques

In this section, we compare the performance of the different methods introduced in 4.1 on the *netmhcpan4.1* dataset, that we split into train/val/test splits. As a common way to experiment with machine learning, training and tuning is done with the first two sets while performance is reported on the test split.

Classifiers As a first assessment, we consider different learning methods on the simple 1-gram TF-IDF representation of peptides and categorical representation of alleles for the *netmhcpan4.1* dataset. By looking into the first three entries in the following performance's table (3), one can observe that the metrics for classifiers that linearly separate data (Logistic Regression and Naive Bayes (4.1.2)) are quite equivalently performing, while a non-liner approach with a FFNN steps up upon this performance (**Top-K improvement = x2,54**). This can be visually confirmed with the ROC- and PR-curves displayed in Figure 22. This is the very first conclusion of this master's thesis: deep learning (and thus non-linearity) helps to boost the presentation prediction.

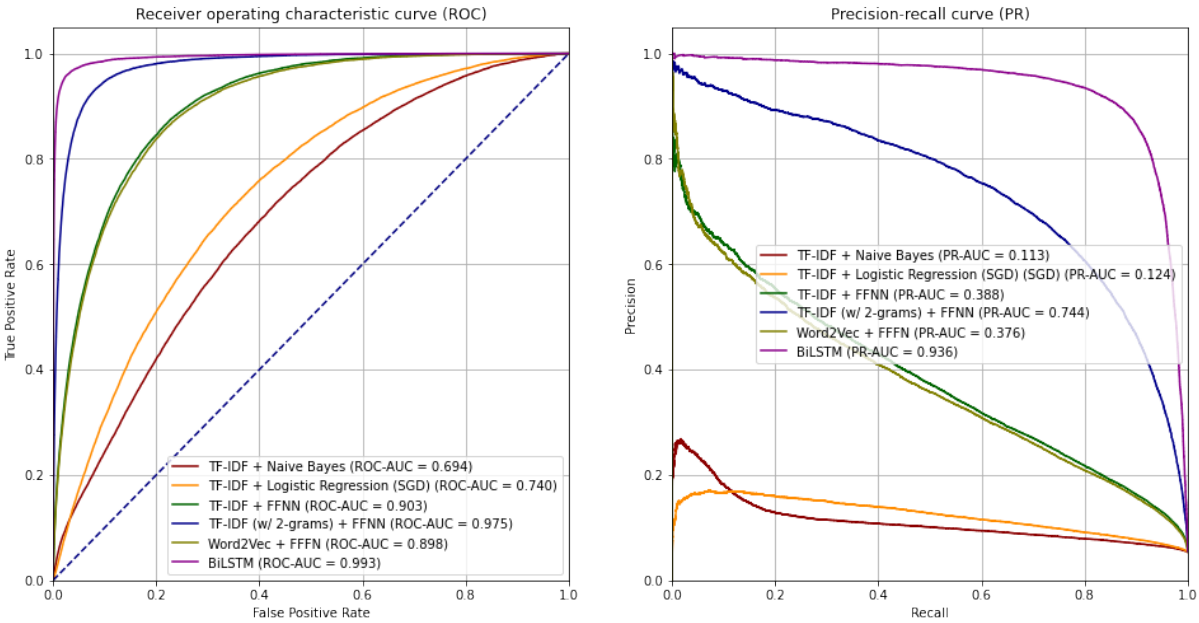

Figure 22: ROC and PR curves for baselines on *netmhcpan4.1*.

Data representation Then, a comparison regarding the pre-processing step of data representation is performed, taking the previously best performing modeling technique: the FFNN (4.1.3). As explained in 4.1.1, three data preprocessing steps are

Model	F1	ROC-AUC	PR-AUC	Top-K
TF-IDF + Naive Bayes	0.161	0.694	0.113	0.148
TF-IDF + Logistic Regression	0.178	0.740	0.124	0.165
TF-IDF + FFNN	0.316	0.903	0.388	0.419
Word2Vec + FFNN	0.324	0.898	0.376	0.407
TF-IDF (1&2-grams) + FFNN	**0.556**	**0.975**	**0.744**	**0.697**

Table 3: Comparison between 1. data representation techniques, and 2. linear and non-linear modeling techniques.

tested: both TF-IDF with either 1- or 1-&2-grams, as well as a Word2Vec (with 100-dim feature space and max/min/mean pooling over AA embeddings). The three last rows of Table 3 highlights the fact that: 1. Word2Vec does not enable a boost in performance compared to a 1-gram approach, 2. using 2-grams helps make the FFNN a better classifier (**Top-K improvement = x1,66**). The latter can be expected, given the fact that this approach enables to increase the input feature space and leads to a stronger representation. However, using more n-grams is not entirely feasible in practice since we cannot increase indefinitely the input space dimension. This is why we need to consider other approaches such as BiLSTMs.

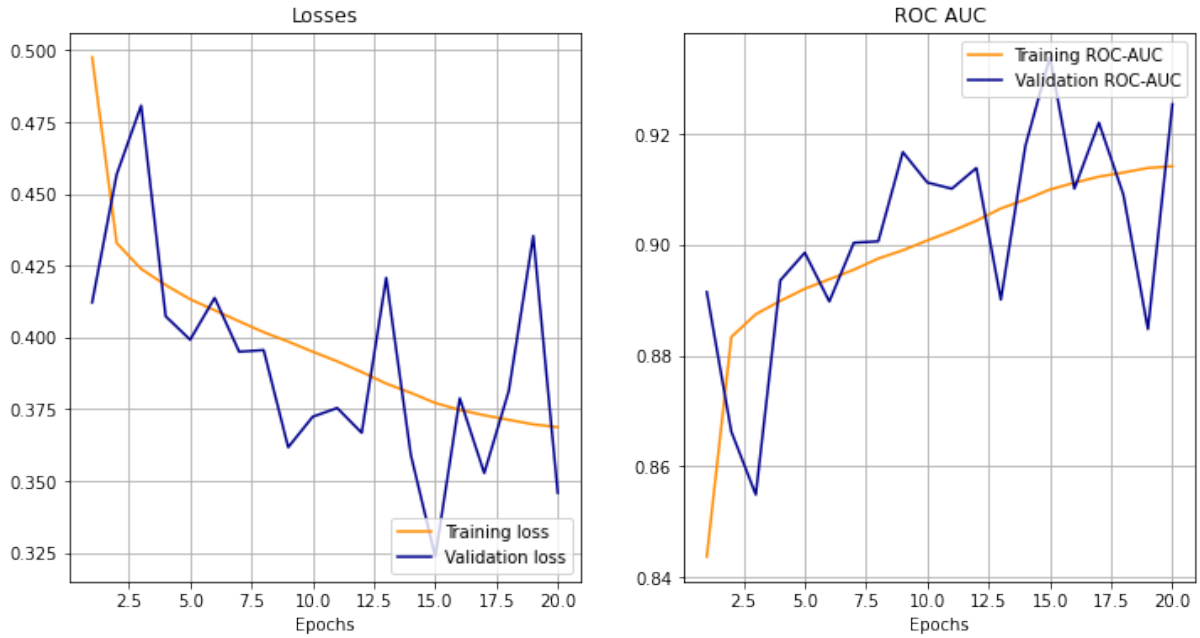

Figure 23: Learning curves for FFNN on TF-IDF representation with *netmhcpan4.1*.

Model	F1	ROC-AUC	PR-AUC	Top-K
TF-IDF (1&2-grams) + FFNN	0.556	0.898	0.744	0.697
BiLSTMs	**0.862**	**0.993**	**0.937**	**0.884**

Table 4: Metrics - simple deep learning and BiLSTMs.

In light of the two previous comparisons, we can consequently validate the first hypothesis:

> **A non-linear modeling method such as FFNN (deep learning) with a TF-IDF approach using 1- and 2-grams outperforms more simple data representations and/or linear modeling methods.**

For the sake of completeness, one can find the learning curves used to train the FFNN on TF-IDF data representation on Figure 23. We can see that the model converges towards the performance reported in the Table 3 and that no over-fitting (i.e. large performance drop between train and validation split) is observed.

5.1.2 BiLSTMs-based models enhance presentation prediction

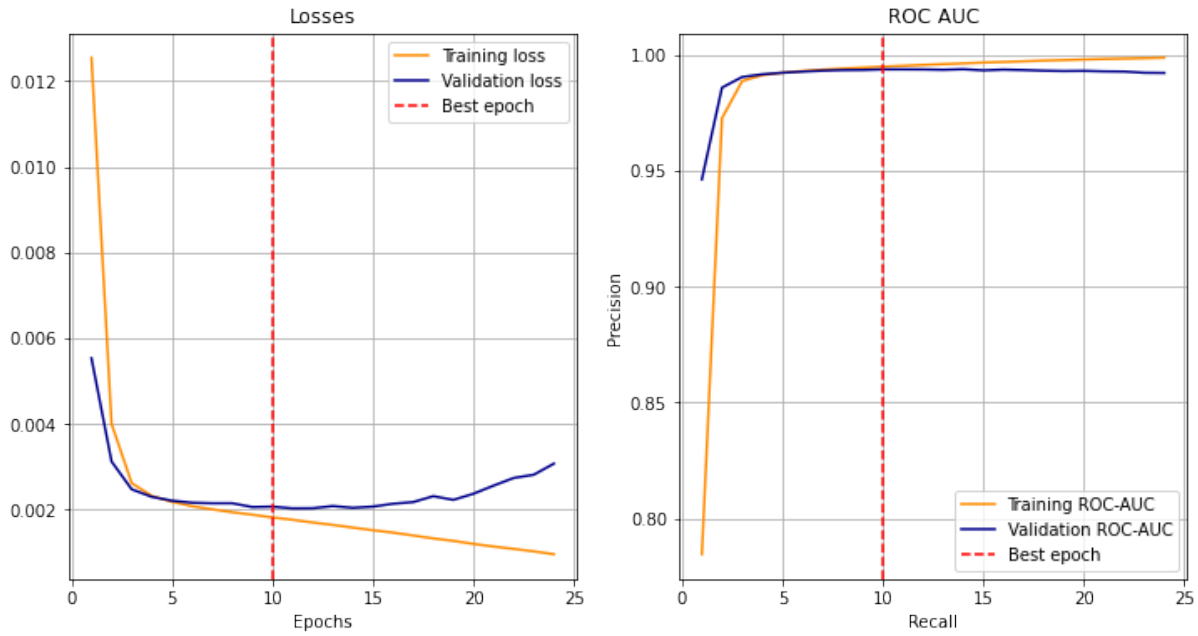

Figure 24: Learning curves for the BiLSTMs approach with *netmhcpan4.1*.

Then, we present in this second section the results regarding the approach with BiLSTMs (4.2), for which the learning curves are plotted in Figure 24. The best epoch before over-fitting is determined by the red vertical line. The performance between the best simple FFNN approach and the RNN approach explored here can be found in Table 4. All the metrics are boosted and **the Top-K improves by 26,8%**. This is also illustrated by the purple curve in Figure 22. Therefore, as expected given the current developements in the NLP field, an RNN approach (especially a bidirectional sequential approach with BiLSTMs) enables the model to better understand the data at hand and integrates the recursive information, which is quite important for protein sequences. This approach is therefore our first strong baseline on our classification presentation problem with the *netmhcpan4.1* dataset. This leads us to the validation of our second hypothesis:

> **Recurrent neural networks better represent the sequential nature of peptides and yield a stronger presentation prediction.**

5.1.3 Attention-based architectures boosts further the performance

Figure 25 illustrates the ROC and PR curves for the NetMHCPan4.1 framework, the BiLSTMs approach and the Transformer encoder approach. The metrics associated to these curves are given in Table 5. First of all, one can observe that an attention-based approach (here using a Transformer encoder, as described in Methods (4.3)), improves the performance regarding our presentation prediction on *netmhcpan4.1* dataset **(Top-K improvement = 1,45 %)**. This performance increase is similar on all the metrics presented (5). This confirms that the contextual representation built by this novel attention mechanism enables a better understanding of the peptide and yields a more predictive peptide presentation. In parallel, we can also see that both the BiLSTM approach and the Transformer encoder approach outperform the public baseline framework NetMHCPan4.1 **(Respectively 1,70 % and 3,12 % improvements on Top-K)**. Knowing that NetMHCPan4.1 has been trained on the test split we use for evaluation – as a consequence, it already knows this data –, it highlights the great capacity of both LSTM-based and attention-based methods to predict the presentation of the peptide by the MHC. Eventually, it is worth noting that the performance increase between BiLSTM and Transformer encoder remains quite small, despite the theoritical and experimental advantages the latter should have compared to the former. However, this is not surprising since the peptide itself is quite a short sequence (maximum 15 AAs) and the real advantage of attention relies on long or very long sequences.

Limitations of metrics Observe that both the BiLSTM baseline and the Transformer yield very strong results on ROC-AUC, and in a slightly lesser extent on PR-AUC – they are both already quite close to 1. The ROC and PR curves (25) now become quite inadequate to validate the tiny improvements that will be observed later. Furthermore, we want to optimize the Top-K (or precision@k) metric, which is still far from

Model	F1	ROC-AUC	PR-AUC	Top-K
NetMHCPan4.1	0.838	0.987	0.923	0.869
BiLSTMs	0.862	0.993	0.937	0.884
Transformer encoder	**0.888**	**0.994**	**0.948**	**0.897**

Table 5: Metrics - NetMHCPan4.1, BiLSTMs and Transformer encoder on *netmhcpan4.1*.

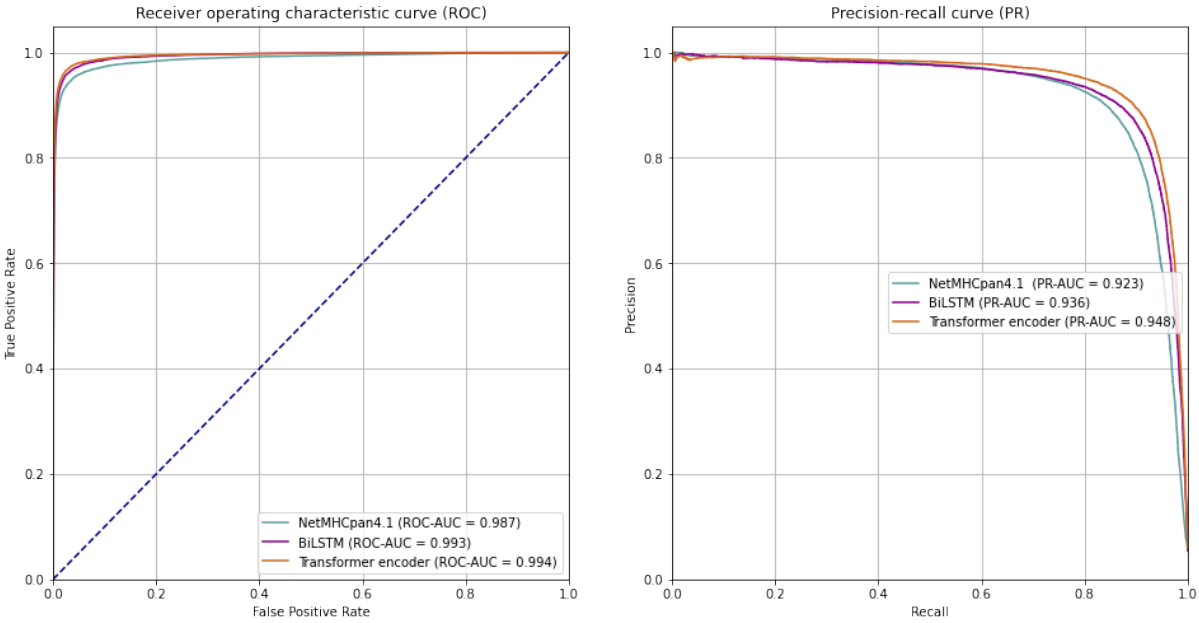

Figure 25: ROC and PR curves for NetMHCPan4.1, BiLSTMs and Transformer on *netmhcpan4.1*.

its maximal value 1. This is why we will now have a strong focus on this metric for the next comparisons. Specifically, we will monitor a per-allele performance, reporting both a global Top-K (over all instances) and the per-allele mean Top-K (average of all per-allele Top-K).

Comparison between BiLSTMs and Transformer encoder on *mhcattnnet* Another comparison is conducted on the *mhcattnnet* binding dataset. It is worth noting that, in this case, we consider the binding affinity (BA) output from the NetMHCPan4.1 framework, processing the percentile rank in the same manner as with the EL percentile rank. The metrics can be found in Table 6. In terms of global metrics, we substantially retrieve the findings as with *netmhcpan4.1*, i.e., a transformer-based modeling approach outperforms a BiLSTM one **(Global Top-K improvement = 0,8 %)**, the latter being once again stronger than the public baseline NetMHCPan4.1 **(Global Top-K improvement = 3,0 %)**. However, one can observe that, per-allele, NetMHCPan4.1

Model	ROC-AUC	Global Top-K	Per-allele Mean Top-K
NetMHCPan4.1	0.933	0.926	**0.897**
BiLSTMs	0.971	0.955	0.869
Transformer encoder	**0.982**	**0.963**	0.896

Table 6: Metrics - NetMHCPan4.1, BiLSTMs and Transformer encoder on *mhcattnnet*.

slightly outperforms our methods. This is visually confirmed with the per-allele mean Top-K distributions on Figure 26. This is mainly due to the fact that this framework has been trained on other large datasets, in which more instances are available for the low-respresented alleles in the *mhcattnnet* dataset. Once again, we can see that these results are quite high and pretty decent and this becomes more and more complex to compare newly methods on such data.

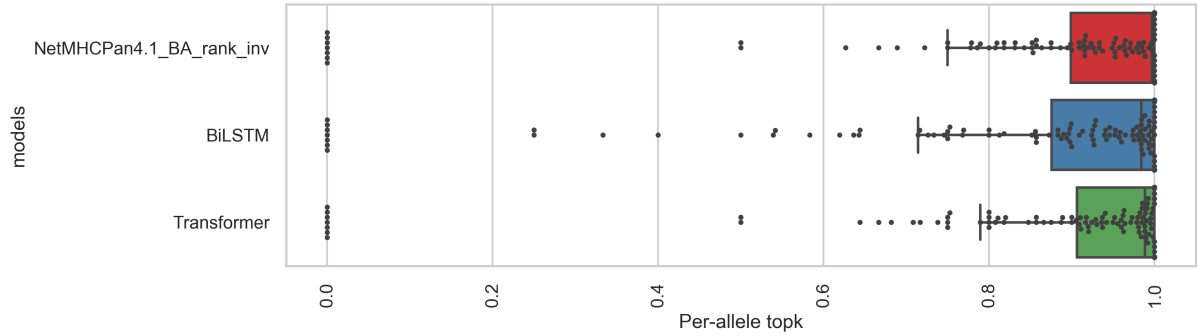

Figure 26: Distributions of the per-allele Top-K performance for NetMHCPan4.1 (BA), BiLSTM and Transformer on *mhcattnnet*.

Despite quite limited, in light of the above results on both a presentation dataset (*netmhcpan4.1*) and a binding dataset (*mhcattnnet*), the third hypothesis can be verified:

> **Binding/presentation prediction is improved with attention-based Transformer encoder models.**

5.2 The value brought by pre-trained models

5.2.1 An analysis with *prot-bert-bfd*

Extensive comparison and analysis are conducted between the different considered ProtBertBFD settings, defined in the section 4.3.4. The metrics are reported on Table 7. Per-allele Top-K performance is displayed for each allele in Figure 27 and the distribution for this metric is illustrated in Figure 28.

First, we can observe that our considered transferred model ProtBertBFD reaches state-of-the-art performance: while it stays slighlty behind our Transformer encoder approach in terms of global Top-K and ROC-AUC, it outperforms it on per-allele mean Top-K **(2,42 % improvement)**. This could indicate that the pre-training allows better generalization on under-represented alleles, which in turn leads to a better mean top-k over alleles. However, we must acknowledge that this better generalization could also be due to the different incorporation of the allele in the input. Overall, compared to previous methods, our transfer learning approach with a pre-trained model only boosts by a tiny margin the performance, and it is hard to conclude it improves with high confidence the presentation prediction on *netmhcpan4.1*.

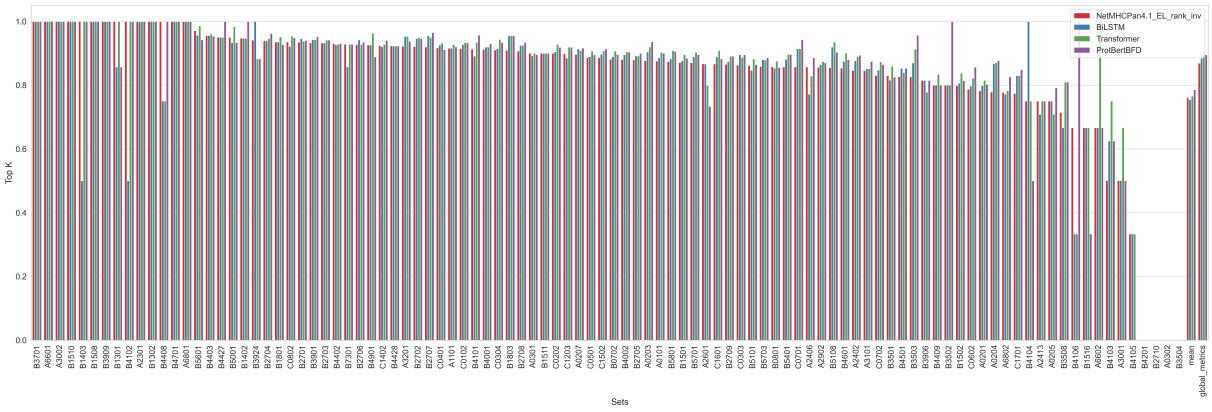

Figure 27: Per-allele Top-K performance for NetMHCPan4.1, BiLSTM, Transformer and ProtBertBFD.

This bar plot shows the performance on the test split of *netmhcpan4.1* data.

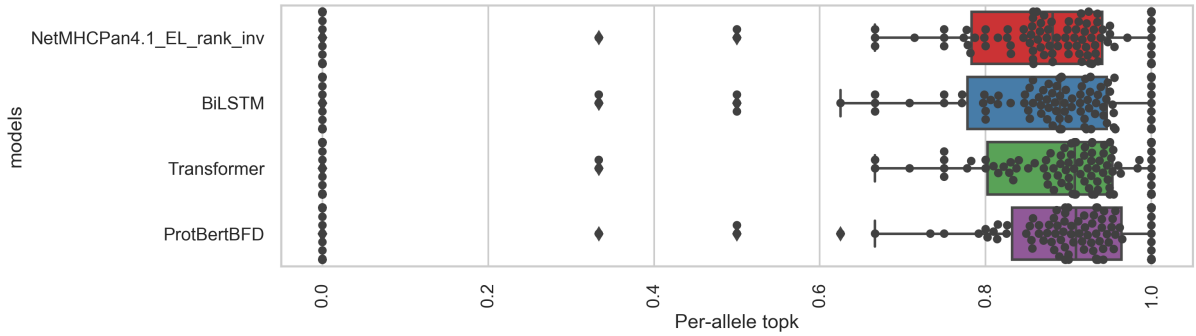

Figure 28: Distributions of the per-allele Top-K performance for NetMHCPan4.1, BiLSTM, Transformer and ProtBertBFD.

This box-and-whisker plot shows the distribution of per-allele Top-K on the test split of *netmhcpan4.1* data.

Feature extraction through a frozen incorporated model In recent pre-training approaches [Rives et al., 2020; Elnaggar et al., 2021], it has been shown that the output embeddings of such pre-trained models could be valuable for downstream prediction such as sequence classification. This led us to consider the model as a frozen feature extractor, without updating the pre-trained weigths but only those from the classification head. As described in the section 4.3.4, this approach is named ProtBertBFD_Frozen and its per-allele Top-K distribution is displayed in Figure 29. In comparison with ProtBertBFD, we can see that ProtBertBFD_Frozen does not learn to discriminate peptide-MHC instances on their presentation nature, and by a large margin **(Top-K margin = 31, 1%)**. This leads us to the conclusion that, without fine-tuning, using an average pooling over the last layer's token embeddings is not valuable. This confirms the BertMHC [Cheng et al., 2020] findings on MHCII: recent pre-trained models still need to be entirely fine-tuned, at least for task similar to our given problem. It entails that, in order to leverage such models, the compute to train them is still required.

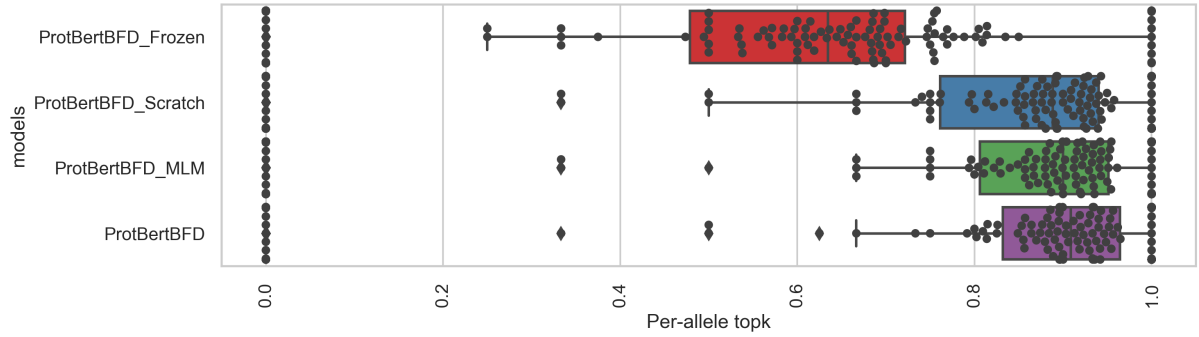

Figure 29: Distributions of the per-allele Top-K performance for ProtBertBFD with/without fine-tuning or additional pre-training.

This box-and-whisker plot shows the distribution of per-allele Top-K on the test split of *netmhcpan4.1* data.

Training from scratch Another required comparison was to assess if the final performance of ProtBertBFD was due to transfer learning or merely just the size of its parameters' space. As such, we built an approach, ProtBertBFD_Scratch, for which performance is also reported in Figure 29, where the similar architecture of ProtBertBFD was used but all the weights were randomly initialized. Trained from scratch in the same conditions, we see that the final performance of this model lies behind ProtBertBFD's performance. This implies that the pre-training step on protein data considerably helped the model to reach this state-of-the-art performance. One could also notice, that while this model has far more parameters than the Transformer encoder and relies on the same attention mechanism, its performance is still below the latter approach. This could also imply that the representation of the allele as integrated pseudo-sequences is not evidently the best way to consider these. A way to assess this

Model	ROC-AUC	Global Top-K	Per-allele Mean Top-K
NetMHCPan4.1	0.987	0.869	0.760
BiLSTMs	0.993	0.884	0.755
Transformer encoder	**0.994**	**0.897**	0.766
ProtBertBFD_Frozen	0.968	0.681	0.541
ProtBertBFD_Scratch	0.993	0.883	0.756
ProtBertBFD_MLM	0.993	0.887	0.768
ProtBertBFD	0.989	0.895	**0.785**

Table 7: Metrics - NetMHCPan4.1, BiLSTMs, Transformer encoder, ProtBertBFD Frozen/Finetune MLM/From scratch on *netmhcpan4.1*.

implication could be to test such a pre-trained model only on the peptides' data while keeping the allele categorically encoded. We let this research idea to further work.

Given the last findings and explorations, and with regard to the fourth hypothesis, we conclude with two main assessments:

> **Pre-trained protein language models can be transferred to predict presentation but need to be entirely fine-tuned on this precise downstream task.**

> **The initial pre-training step on unsupervised protein data makes the protein language model reach better final performance on the presentation downstream task.**

5.2.2 A training analysis with a tinier custom PLM

In order to analysis the value brought by pre-training over a randomly initialized language model, we performed a precise comparison with the 12-layer PLM trained on UniRef50. In Figure 30, four different approaches are considered, similarly to what have been explored with ProtBertBFD (same architecture as in Figure 20). The first approach "Random & Frozen" considers a model randomly initialized for which the BERT pre-trained model is frozen, a second one "Pre-trained & Frozen" where the pre-trained model remains frozen but has been fine-tuned, a third one "Random & Fine-tuned" where all parameters are fine-tuned but the BERT model is randomly initialized, i.e. trained from scratch like with ProtBertBFD, and eventually a last approach "Pre-trained & Fine-tuned" where the pre-trained model is fine-tuned, as with ProtBertBFD. For all these training experiments, the exact same hyper-parameters have been used (batch_size of 256, maximal learning rate of $5e^{-5}$ with warmup and linear decay, Adam optimizer (Appendix B)). In such experimental settings, we can see that, contrary to ProtBert_Scratch, the model entirely fine-tuned from scratch did not reach a decent performance. It even diverged and led to a deep learning model no longer

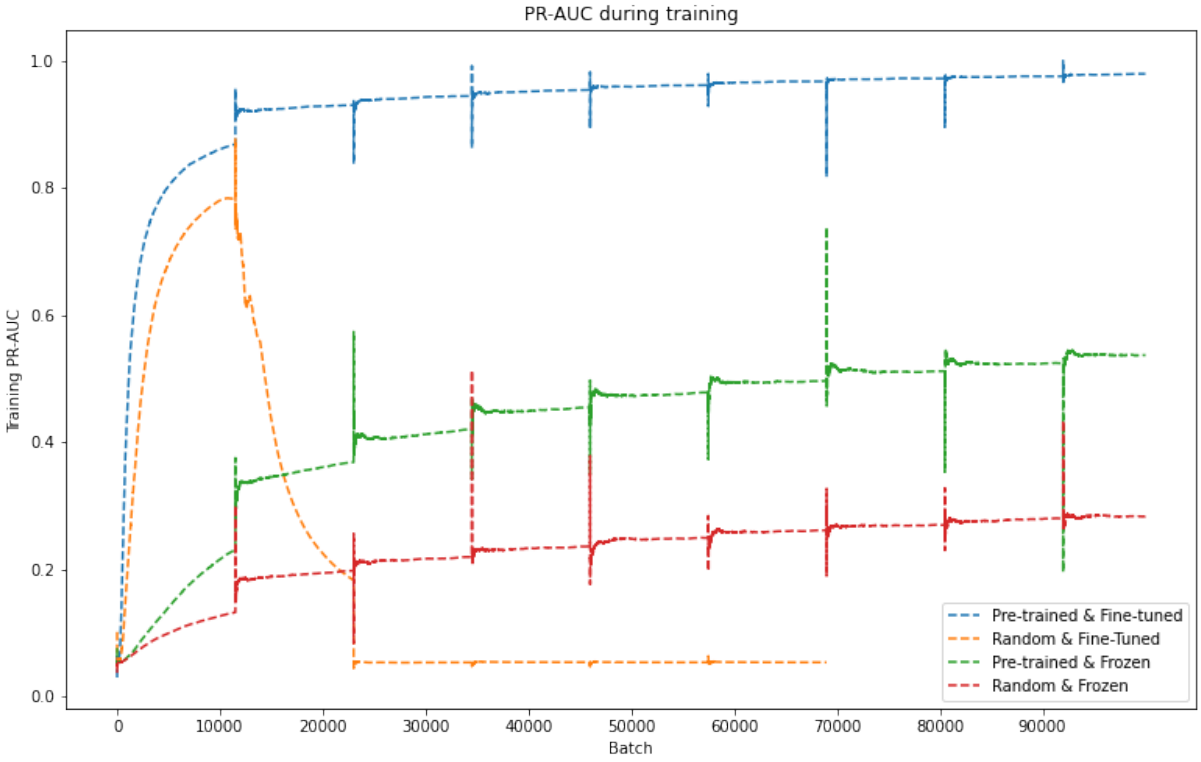

Figure 30: Training PR-AUC curves during training of the 12-layer PLM on *netmhcpan4.1*.

The 4 different models use the same parameters and hyper-parameters. The only difference is regarding the random initialization and the fine-tuning of the BERT model.

able to learn – the learning rate is too high. This means that the pre-training helps to construct a representation of the data that enables a more stable fine-tuning of the entire model during downstream classification training. With respect to the fourth hypothesis, we can in consequence bring the following take-away:

> **The initial pre-training step on unsupervised protein data makes, in addition, the protein language model more stable on downstream classification task.**

5.2.3 Additional MLM pre-training step on positive data

MLM Pre-training As explained in section 4.3.4 and described in Figure 21, we pre-trained ProtBertBFD with masked language modeling on relevant positive data from the presentation task. This is referred as ProtBertBFD_MLM. In Figure 31, given the data from only one allele (A2301), we performed a t-SNE projection on two dimensions of the last embeddings' average pooling. We can see that after the additional MLM pre-

Figure 31: t-SNE 2-D projection of the last layer's embeddings average pooling with ProtBertBFD on allele A2301 from *netmhcpan4.1*.

training step on positive data, the clustering of presented instances ("binder" on the plot) suggests that this step made the PLM learn a distribution of the positive data under the hood. However, the metrics in Table 7 demonstrate that it did not lead to a boost on the final presentation prediction task. It is confirmed with the training curves on Figure 32 and 33 where the final fine-tuning of ProtBertBFD_MLM is neither outperforming ProtBertBFD nor being quicker to converge. Moreover, ProtBertBFD_MLM used as feature extractor (frozen) does not lead to decent performance or convergence, compared to its counterpart ProtBertBFD_Frozen.

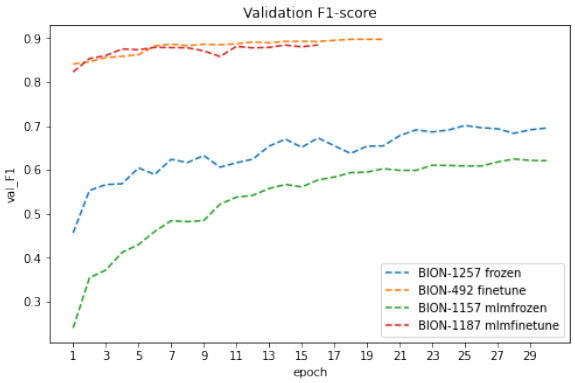

Figure 32: Validation F1 during training of ProtBertBFD with/without fine-tuning/MLM pre-training on *netmhcpan4.1*.

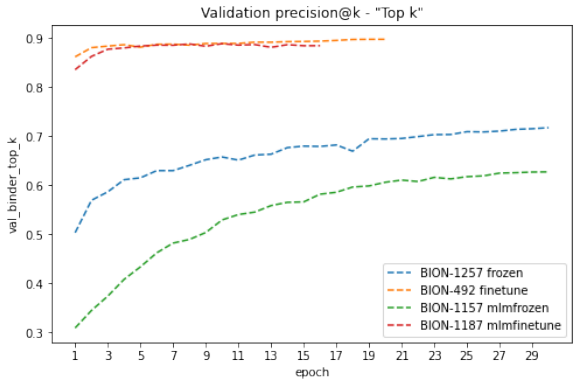

Figure 33: Validation Top-K during training of ProtBertBFD with/without fine-tuning/MLM pre-training on *netmhcpan4.1*.

With regard to the hypothesis 5, this makes us conclude:

> **An additional MLM pre-training step on relevant domain positive data neither boosts the performance of the final presentation task nor helps to increase the convergence of the model on the final downstream task.**

5.3 Dimensionality reduction and visualization of attention maps

In order to quickly visualize if the attention maps contain some valuable information, we performed, as in the last section, a t-SNE 2-D projection using the attention maps of the first and last layers of ProtBertBFD (pre-trained on protein data) was performed on data from allele A2301. The projections before and after the additional MLM pre-training on positive data step are displayed in Figure 34 and 35. In the latter, we observe a similar pattern as with the embeddings (Figure 31): the additional MLM pre-training step on positive data seems to help clusterize the data in a representation more suitable to discriminate if a peptide-MHC complex is presented or not. Moreover, on the left plot (before any other pre-training of fine-tuning) of the last layer's attention maps (Figure 35), the positive instances seem to be more grouped, compared to the left plot of the embeddings average pooling (Figure 31). It may imply that the prior we made on using only the last embeddings' layer could not be the best way to leverage pre-trained embeddings, and extracting directly the attention maps as in [Rao et al., 2020] could improve the final performance. Given these limited results, we cannot conclude on the sixth hypothesis and these attention maps will be further investigated in further research work (6.3).

Figure 34: t-SNE 2-D projection of the first layer's attentions with ProtBertBFD on allele A2301 from *netmhcpan4.1*.

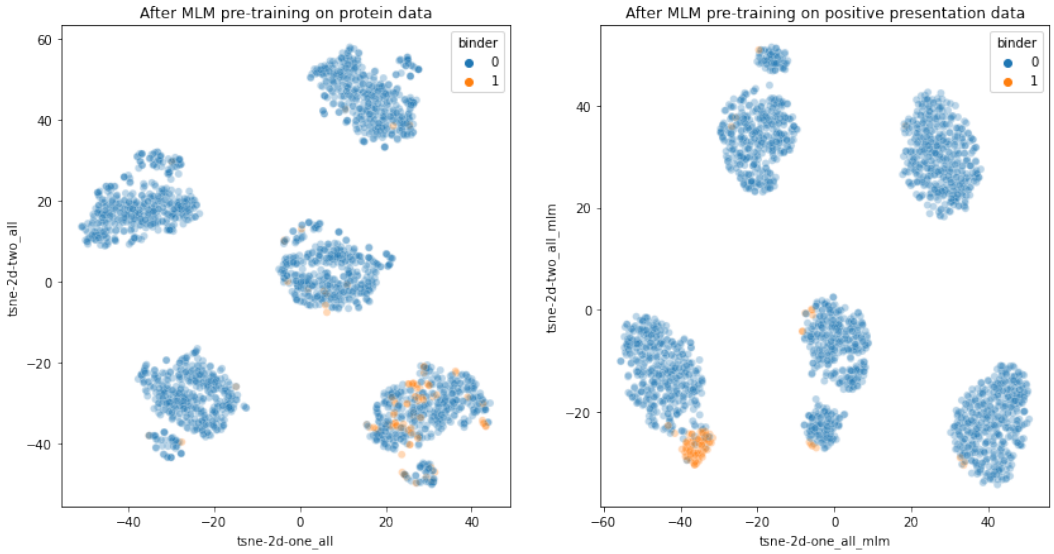

Figure 35: t-SNE 2-D projection of the last layer's attentions with ProtBertBFD on allele A2301 from *netmhcpan4.1*.

6 Future work

In light of what have been explored in this master's thesis as well as techniques reviewed in related works (2.4), this section extends on further research initiatives that could be investigated on top of this master's thesis. Notably, I will conduct part of these ideas in my upcoming months at InstaDeep, in order to improve presentation prediction using proprietary datasets.

6.1 Extension of presentation features

While we mainly use a presentation dataset in this work, we focus on modeling the binding interaction between a peptide and a MHC, using only these features to predict further peptide presentation by the MHC. Although peptide-MHC binding is arguably the most selective step in the MHC antigen presentation pathway, other properties contribute to determining immunogenicity of T-cell epitopes. For instance, gene expression levels, proteasome cleavage preference or translational efficiency are features that have been proven to correlate with immunogenicity. They could be combined to the peptide and allele data to predict the presentation with more confidence. Such incorporation with additional potential correlating features will be investigated.

6.2 Anomaly detection

On another note, the masked language modeling training could be leveraged to build a representation of positive data. Take the positive peptides of a given allele. Per se, a BERT language model, or any other language modeling approach, could be trained on positive peptides, in a quite similar manner as with the additional MLM pre-training step investigated in this master's thesis. At inference, the language model could process peptide sequences, for which some tokens have been randomly masked. Then, we could compute a "score" for the sequence by comparing masked tokens predictions with their ground truth. This score would look like a global likelihood for the peptide sequence, giving a sort of proxy for the binding likelihood. By thresholding this score, we could then predict the binding/presentation. This is quite similar to the anomaly detection framework explored in [Guo et al., 2021] and such a model could be trained for each allele. This approach is attractive since the decoys (negative data) won't be considered. Given the fact that the generation process of decoys diverges between different studies, we could consequently take away the bias induced by this process from our model.

6.3 Feature extraction from frozen pre-trained models

Eventually, another approach would be to iterate on the extraction of attention maps from a frozen pre-trained protein language model (PLM), as investigated in [Vig et al.,

2021; Rao et al., 2020]. Attention maps from frozen PLMs proved to contain some valuable information regarding the structure and the folding of proteins. We could try to extract and stack these attention maps and perform approaches similar to vision-based algorithms such as convolutional neural networks (CNN) on top. Today, some approaches are based on molecular dynamics experiments where we generate structure features for a given peptide-MHC complex. These approaches are computationally intensive. In a given method based on attention maps, we would thus consider the latter as contact maps of the complex, therefore using the PLM in place of a structural features' generator.

7 Conclusion

Natural language processing approaches have been definitely booming over the last recent years and their application to the domain of biology generated strong findings, as developed in [Ofer et al., 2021]. As such, this master's thesis work addressed a number of questions regarding the application of mathematical methods and techniques from the natural language processing field to a precise biological classification problem. In that purpose, we mainly used data generated from mass-spectrometry experiments (*netmhcpan4.1*). Consequently, such data models the distribution of presentation prediction. However, this work strongly relates to the binding prediction, since it builds predictive models based only on peptide and MHC features, whose binding leads, to a certain extent, to antigen presentation. That is notably why binding and presentation were often interchanged throughout this work.

Building on top of the straightforward analogy between the natural and protein language, different modeling techniques have been derived, explored, implemented and evaluated. Through this thorough investigation, the constant goal was to improve the binding/presentation prediction. In brief, using count-based techniques to represent the peptide's sequence and in comparison to methods building linear boundaries in the input space, more intricated deep-learning approaches have shown to better discriminate presented peptides from decoys. Then, another deep learning approach using the recursive information from the sequence drove the presentation performance to a higher level. Eventually, this led to consider a non-sequential Transformer-based method that yielded the strongest results, notably beating a widely used public baseline (NetMHCPan4.1).

Furthermore, the second main purpose of this work aimed at leveraging some information from pre-trained protein language models on the specific downstream binding/presentation prediction task. The considered pre-trained model reached state-of-the-art performance, but failed in providing an overall improvement, compared to the more basic Transformer-based approach without transfer learning. All things considered, while the gain of the fine-tuning of such a model has been limited, it has still been shown that the transfer of such models benefit from pre-training on both the performance – on the downstream classification task, ProtBertBFD was stronger than its counterpart model trained from scratch –, and the training stability. Besides, with strong expectations from the masked language modeling training, ProtBertBFD has been additionally pre-trained on relevant positive data. Unfortunately this approach proved unsuccessful as most of our assumptions were not confirmed by our available data. On a final note, the modest analysis regarding the attention maps, the inner mechanism behind the BERT model, brought new questions and ideas that would be worth being investigated following this work.

List of Figures

List of Tables

Bibliography

[1] InstaDeep. Company website, 2021. URL https://www.instadeep.com/.

[2] BioNTech. BioNTech and InstaDeep announce strategic collaboration and form AI innovation lab to develop novel immunotherapies. *Globe Newswire*, Nov 2020. URL https://www.globenewswire.com/news-release/2020/11/25/2133666/0/en/BioNTech-and-InstaDeep-Announce-Strategic-Collaboration-and-Form-AI-Innovation-Lab-to-Develop-Novel-Immunotherapies.html.

[3] Khan Academy. Biology: Gene expression and regulation, 2021. URL https://www.khanacademy.org/science/ap-biology/gene-expression-and-regulation.

[4] Ph.D. Suzanne Clancy and Ph.D. William Brown. Translation: DNA to mRNA to Protein. *Nature Education*, 2008. URL https://www.nature.com/scitable/topicpage/translation-dna-to-mrna-to-protein-393/.

[5] Daniel E. Koshland and Felix. Haurowitz. "protein". *Encyclopedia Britannica*, 2020. URL https://www.britannica.com/science/protein.

[6] Ph.D. C. M. O'Connor and Ph.D. J. U. Adams Adams. Essentials of cell biology. *Cambridge, MA: NPG Education*, 2010. URL https://www.nature.com/scitable/ebooks/essentials-of-cell-biology-14749010/.

[7] Osmosis. Immune system, 2021. URL https://www.osmosis.org/library/do/organ-systems/immune-system.

[8] Eryn Blass and Patrick Ott. Advances in the development of personalized neoantigen-based therapeutic cancer vaccines. *Nature Reviews Clinical Oncology*, 18, 01 2021. doi: 10.1038/s41571-020-00460-2.

[9] Zhuting Hu, Patrick Ott, and Catherine Wu. Towards personalized, tumour-specific, therapeutic vaccines for cancer. *Nature Reviews Immunology*, 18, 12 2017. doi: 10.1038/nri.2017.131.

[10] Ugur Sahin, Evelyna Derhovanessian, Matthias Miller, Björn-Philipp Kloke, Petra Simon, Martin Löwer, Valesca Bukur, Arbel Tadmor, Ulrich Luxemburger, Barbara Schrörs, Tana Omokoko, Mathias Vormehr, Christian Albrecht, Anna Paruzynski, Andreas Kuhn, Janina Buck, Sandra Heesch, Katharina Schreeb, Felicitas Müller, and Özlem Türeci. Personalized rna mutanome vaccines mobilize poly-specific therapeutic immunity against cancer. *Nature*, 547, 07 2017. doi: 10.1038/nature23003.

[11] WHO. mrna vaccines against covid-19: Pfizer-biontech covid-19 vaccine bnt162b2: prepared by the strategic advisory group of experts (sage) on immunization working group on covid-19 vaccines, 22 december 2020. Technical documents, 2020.

[12] Birkir Reynisson, Bruno Alvarez, Sinu Paul, Bjoern Peters, and Morten Nielsen. NetMHCpan-4.1 and NetMHCIIpan-4.0: improved predictions of MHC antigen presentation by concurrent motif deconvolution and integration of MS MHC eluted ligand data. *Nucleic Acids Research*, 48(W1):W449–W454, 05 2020. ISSN 0305-1048. doi: 10.1093/nar/gkaa379. URL https://doi.org/10.1093/nar/gkaa379.

[13] The UniProt Consortium. UniProt: the universal protein knowledgebase in 2021. *Nucleic Acids Research*, 49(D1):D480–D489, 11 2020. ISSN 0305-1048. doi: 10.1093/nar/gkaa1100. URL https://doi.org/10.1093/nar/gkaa1100.

[14] Gopalakrishnan Venkatesh, Aayush Grover, G Srinivasaraghavan, and Shrisha Rao. MHCAttnNet: predicting MHC-peptide bindings for MHC alleles classes I and II using an attention-based deep neural model. *Bioinformatics*, 36 (Supplement_1):i399–i406, 07 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btaa479. URL https://doi.org/10.1093/bioinformatics/btaa479.

[15] Baris E. Suzek, Yuqi Wang, Hongzhan Huang, Peter B. McGarvey, and Cathy H. Wu. Uniref clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics*, 31:926 – 932, 2015.

[16] Ilka Hoof, Bjoern Peters, John Sidney, Lasse Pedersen, Alessandro Sette, Ole Lund, Søren Buus, and Morten Nielsen. Netmhcpan, a method for mhc class i binding prediction beyond humans. *Immunogenetics*, 61:1–13, 12 2008. doi: 10.1007/s00251-008-0341-z.

[17] Bruno Alvarez, Birkir Reynisson, Carolina Barra, Søren Buus, Nicola Ternette, Tim Connelley, Massimo Andreatta, and Morten Nielsen. NNAlign_MA; MHC peptidome deconvolution for accurate MHC binding motif characterization and improved T cell epitope predictions. 06 2019.

[18] Timothy O´ Donnell, Alex Rubinsteyn, and Uri Laserson. MHCflurry 2.0: Improved Pan-Allele Prediction of MHC Class I-Presented Peptides by Incorporating Antigen Processing. *Cell Systems*, 11, 07 2020. doi: 10.1016/j.cels.2020.06.010.

[19] Jennifer G. Abelin, Dewi Harjanto, Matthew Malloy, Prerna Suri, Tyler Colson, Scott P. Goulding, Amanda L. Creech, Lia R. Serrano, Gibran Nasir, Yusuf Nasrullah, Christopher D. McGann, Diana Velez, Ying S. Ting, Asaf Poran, Daniel A. Rothenberg, Sagar Chhangawala, Alex Rubinsteyn, Jeff Hammerbacher, Richard B. Gaynor, Edward F. Fritsch, Joel Greshock, Rob C. Oslund, Dominik Barthelme, Terri A. Addona, Christina M. Arieta, and Michael S. Rooney.

Defining hla-ii ligand processing and binding rules with mass spectrometry enhances cancer epitope prediction. *Immunity*, 51(4):766–779.e17, 2019. ISSN 1074-7613. doi: https://doi.org/10.1016/j.immuni.2019.08.012. URL https://www.sciencedirect.com/science/article/pii/S1074761319303632.

[20] Poomarin Phloyphisut, Natapol Pornputtapong, Sira Sriswasdi, and Ekapol Chuangsuwanich. MHCSeqNet: a deep neural network model for universal MHC binding prediction. *BMC Bioinformatics*, 20, 05 2019. doi: 10.1186/s12859-019-2892-4.

[21] Binbin Chen, Michael S. Khodadoust, Niclas E Olsson, Lisa E. Wagar, Ethan Fast, Chih Long Liu, Yagmur Muftuoglu, Brian J. Sworder, Maximilian Diehn, Ronald Levy, Mark M. Davis, Joshua E. Elias, Russ B. Altman, and Ash A. Alizadeh. Predicting HLA class II antigen presentation through integrated deep learning. *Nature Biotechnology*, 37:1332–1343, 2019.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.

[24] Jun Cheng, Kaïdre Bendjama, Karola Rittner, and Brandon Malone. BERTMHC: Improves MHC-peptide class II interaction prediction with transformer and multiple instance learning. *bioRxiv*, 2020. doi: 10.1101/2020.11.24.396101. URL https://www.biorxiv.org/content/early/2020/11/24/2020.11.24.396101.

[25] Nadav Brandes, Dan Ofer, Yam Peleg, Nadav Rappoport, and Michal Linial. ProteinBERT: A universal deep-learning model of protein sequence and function. *bioRxiv*, 2021. doi: 10.1101/2021.05.24.445464. URL https://www.biorxiv.org/content/early/2021/05/25/2021.05.24.445464.

[26] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, 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. *bioRxiv*, 2020. doi: 10.1101/622803. URL https://www.biorxiv.org/content/early/2020/08/31/622803.

[27] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, Wang Yu, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, Debsindhu Bhowmik, and Burkhard Rost. ProtTrans: Towards Cracking the Language of Lifes Code Through Self-Supervised Deep Learning and High Performance Computing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. doi: 10.1109/TPAMI.2021.3095381.

[28] Martin Steinegger and Johannes Söding. Clustering huge protein sequence sets in linear time. *Nature Communications*, 9, 06 2018. doi: 10.1038/s41467-018-04964-5.

[29] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Xi Chen, John Canny, Pieter Abbeel, and Yun S Song. Evaluating protein transfer learning with tape. In *Advances in Neural Information Processing Systems*, 2019.

[30] Haoyang Zeng and David K Gifford. DeepLigand: accurate prediction of MHC class I ligands using peptide embedding. *Bioinformatics*, 35(14):i278–i283, 07 2019. ISSN 1367-4803. doi: 10.1093/bioinformatics/btz330. URL `https://doi.org/10.1093/bioinformatics/btz330`.

[31] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models. 2019.

[32] Roshan Rao, Joshua Meier, Tom Sercu, Sergey Ovchinnikov, and Alexander Rives. Transformer protein language models are unsupervised structure learners. *bioRxiv*, 2020. doi: 10.1101/2020.12.15.422761. URL `https://www.biorxiv.org/content/early/2020/12/15/2020.12.15.422761`.

[33] Jesse Vig, Ali Madani, Lav R. Varshney, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. BERTology Meets Biology: Interpreting Attention in Protein Language Models, 2021.

[34] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

[35] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[36] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association

for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[37] Dan Jurafsky and James H. Martin. Speech and language processing, 2021. URL https://web.stanford.edu/jurafsky/slp3/.

[38] Kilian Weinberger. Bayes classifier and naive bayes, 2018. URL https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote05.html.

[39] Stephan Günnemann and Oleksandr Shchur. Lecture 9: Deep Learning I. *WS19 Machine Learning I, TUM, Munich*, 2020.

[40] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[41] Stephan Günnemann and Oleksandr Shchur. Sequential Data â€“ Neural Network Approaches. *SS20 Machine Learning for Graphs and Sequential Data, TUM, Munich*, 2020.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[43] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[44] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr DollÃ¡r. Focal loss for dense object detection, 2018.

[45] Chris Manning and John Hewitt. Natural language processing with deep learning, 2021. URL http://web.stanford.edu/class/cs224n/.

[46] John Thickstun. The transformer model in equations, 2019. URL https://homes.cs.washington.edu/~thickstn/docs/transformers.pdf.

[47] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[48] Haixuan Guo, Shuhan Yuan, and Xintao Wu. LogBERT: Log Anomaly Detection via BERT. 2021.

[49] Dan Ofer, Nadav Brandes, and Michal Linial. The language of proteins: Nlp, machine learning & protein sequences. *Computational and Structural Biotechnology Journal*, 19:1750–1758, 2021. ISSN 2001-0370. doi: https://doi.org/10.1016/j.csbj.2021.03.022. URL https://www.sciencedirect.com/science/article/pii/S2001037021000945.

[50] Chip Huyen. Evaluation metrics for language modeling. *The Gradient*, 2019.

Appendices

A Characteristics of V100 and A100 NVIDIA Tesla GPUs

Unlike **Central Processing Units (CPUs)** which are optimized for low latency and serial instruction processing, **Graphics Processing Units (GPUs)** emphasise on high throughput and parallel processing. By using multiple cores, they can process far more operations in a given time than CPUs and are particularly efficient for matrix multiplication, which is the core of deep learning. Consequently, in the last decade, they have become key to this technology – the latter pouring vast quantities of data through neural networks –, and have accelerated the successful development of computer vision and natural language processing applications.

We used two different GPU technologies developed by NVIDIA: **Tesla V100 GPU** and **A100 Tensor Core GPU**. The latter is based on the new NVIDIA Ampere GPU architecture, and builds upon the capabilities of the former. It adds many new features and delivers significantly faster performance for AI and data analytics workloads. A benchmark on NLP between both technologies can be found in Figure 36.

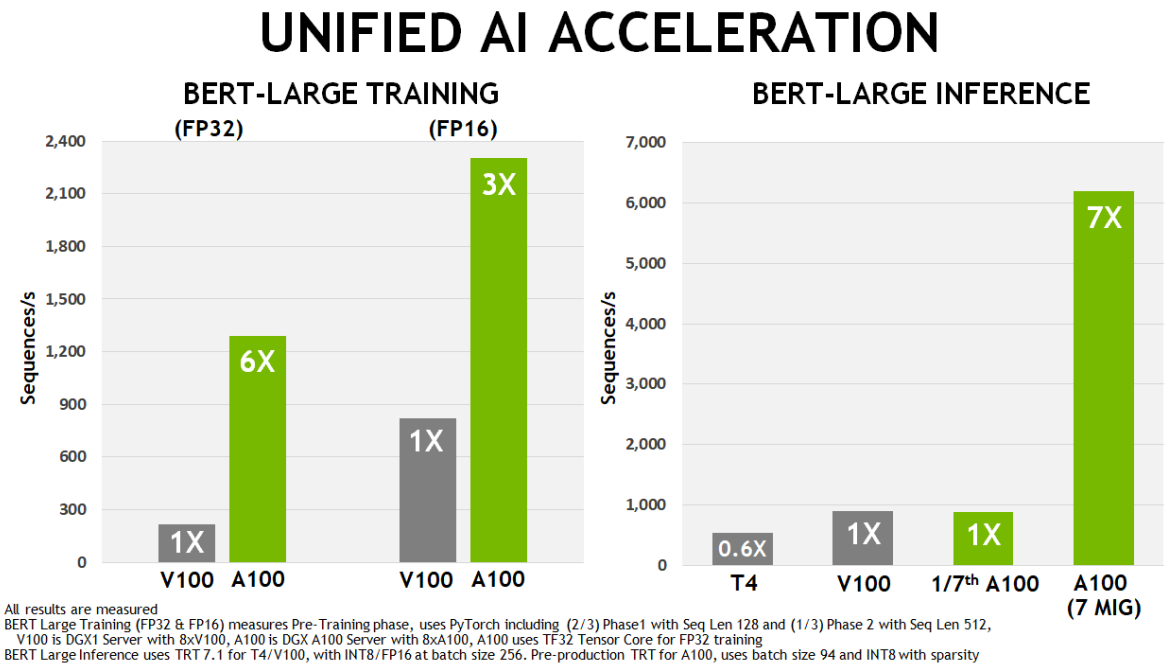

Figure 36: A100 GPU performance in BERT deep learning training and inference scenarios compared to Tesla V100 and Tesla T4.

Source: NVIDIA

B Stochastic gradient descent and Adam optimizer

Algorithm 2 Mini-batch Stochastic Gradient Descent

Input:
L: loss function
f: function parametrized by θ
X: set of training inputs $x^{(i)}$
Y: set of labels $y^{(i)}$
T: number of iterations
η: learning rate, that can be updated over iterations
Output: θ: function parameters

function SGD(L, f, X, Y, T, η)
  $\theta \leftarrow 0$
  for $k = 1 \rightarrow T$ **do**
    Sample a mini-batch of m examples $\{(x^{(1)}, y^{(1)}, (x^{(m)}, y^{(m)}\}$
    $g \leftarrow 0$
    for $i = 1 \rightarrow m$ **do**
      Compute $\hat{y}^{(i)} = f(x^{(i)}, \theta)$
      Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$
      $g \leftarrow g + \nabla_\theta L(\hat{y}^{(i)}, y^{(i)})$
    end for
    $\theta \leftarrow \theta - \eta_k * g$
  end for
  return η
end function

Above is given the algorithm (2) for the mini-batch SGD optimizer. The gradients are computed over all the training samples recursively (per mini-batch and in random order) and a complete pass over the training set is called an epoch. Thus, one epoch contains multiple iterations where gradients are updated.

However, choosing a proper learning rate η_k (and its schedule) with SGD can be difficult in practice. Also, minimizing highly non-convex error functions is highly challenging and SGD proved to make these functions be trapped in a sub-optimal local minima. Momentum is a method that helps accelerate SGD in the relevant direction. Recently, new methods using momentum, have been developed as optimizers for deep learning. The *Adaptive Moment Estimation* (Adam) optimizer is one of them and tweaks the SGD update step $\theta_{k+1} = \theta_k - \eta_k * g_k$ as follows:

$$m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k \tag{26}$$
$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2 \tag{27}$$
$$\hat{m}_k = \frac{m_k}{1 - (\beta_1)^k} \tag{28}$$
$$\hat{v}_k = \frac{v_k}{1 - (\beta_2)^k} \tag{29}$$
$$\theta_{k+1} = \theta_k - \frac{\eta_k}{\sqrt{\hat{v_k}} + \epsilon} \hat{m}_k \tag{30}$$

In addition to storing an exponentially decaying average of past squared gradients v_k, Adam also keeps an exponentially decaying average of past gradients m_k, similar to momentum. In practice, $\beta_1 = 0.9, \beta_2 = 0.999, \eta = 10^{-8}$, and it has been show that Adam makes loss minimization converge better compared to other adaptive learning-method algorithms.

C Considerations around the perplexity

These considerations as well as the formal connection between the perplexity and the cross-entropy have been derived with the help of [50].

The true distribution of a language P is unknown. In that respect, a language model aims to learn a distribution Q close to the empirical distribution P of the language, thanks to data. In order to measure the closeness of two distributions, cross entropy of Q with respect to P can be used and is defined in the discrete case as follows:

$$H(P, Q) = -\sum_x (P(x) log(Q(x)) \tag{31}$$
$$= -\sum_x (P(x) log(P(x)) - \sum_x (P(x) log(\frac{Q(x)}{P(x)})) \tag{32}$$
$$= H(P) + D_{KL}(P||Q) \tag{33}$$

where $H(P)$ is the empirical entropy of P, and $D_{KL}(P||Q)$ the Kullback-Leibler (KL) divergence of Q from P. As $H(P)$ is unoptimizable, the true objective when training a model with cross-entropy is to minimize the KL divergence of the distribution, which was learned by the language model from the empirical distribution of the language. Similarly, the KL divergence is positive, so that we have $H(P, Q) \geq h(P)$. This means that the cross-entropy will be at least equal to the empirical entropy $H(P)$ of the true distribution P. The latter is also limited by the vocabulary size. Indeed, let us consider a vocabulary V – in our AA case: $|V| = 21$ (or 23 if we consider [CLS] and [SEP]

tokens). We know that the entropy of a probability distribution is maximized when it is uniform. Hence, it follows:

$$H(P) \leq H(P,Q) \leq -|V| * \frac{1}{|V|} \log(\frac{1}{|V|}) = \log(|V|) \tag{34}$$

For $V = 23$, the entropy will be at most $H(P) = 4,5236$.

Link to cross-entropy In our case, P is the real distribution of our language, while Q is the distribution estimated by our model on the training set. We cannot know the real P, but given a long enough sequence of tokens AA_i (so a large L), we can approximate the per-word cross-entropy using the Shannon-McMillan-Breiman theorem:

$$H(P,Q) \simeq -\frac{1}{L} \log Q(s) \tag{35}$$

Considering our initial cross-entropy $H(s)$ for a given set (sequence) s, we can retrieve the link to the perplexity as follows:

$$H(s) = -\frac{1}{L} \log_2 p(AA_1, ..., AA_L) \tag{36}$$

$$\iff 2^{H(s)} = 2^{-\frac{1}{L} \log_2 p(AA_1,...,AA_L)} \tag{37}$$

$$\iff PPL(s) = \sqrt[L]{\frac{1}{P(AA_1, AA_2, ..., AA_L)}} \tag{38}$$

D Cyclical learning rate strategies

Figure 37: Multiple (cyclical) learning rate schedulers with different range.

Figure 38: AUC for these different schedulers.

In Figure 37, you can find different learning rate scheduling strategies. Most of them are cyclical. Observe that if the learning rate is too high (purple curve), the performance (here ROC-AUC displayed in Figure 38) will drop since the model cannot converge anymore to a local optimum. In contrary, if the learning rate is too low (yellow curve), the model slowly converges. The goal of these cyclical experiments on first iterations of the model is to precisely figure out the learning rate range to consider for an optimal training, i.e. a deep learning training that converges to an optimal solution. By increasing the learning rate from a very low to a very large one, we can estimate the step where the loss starts to diverge, an optimal maximal learning rate can in that case be taken a few steps before.

In practice, and this is what have been used for most of the methods in this master's thesis, we use a linear decay learning rate scheduler: the learning rate linearly decays from its maximal value until 0. A ramping warm-up is also used for the very first iterations. With pre-trained models, the intuition behind this warm-up is to softly adapt the parameters at the beginning of the training and, as a consequence, avoid to reach *catastrophic forgetting*. The latter is a behavior by which the pre-trained model looses its language modeling knowledge by using a too high learning rate too soon on a downstream task training. These considerations lack a formal mathematical derivation and should be seen as take-aways resulting from practical engineering initiatives with such neural networks.

E Training loss curves for additional MLM pre-training

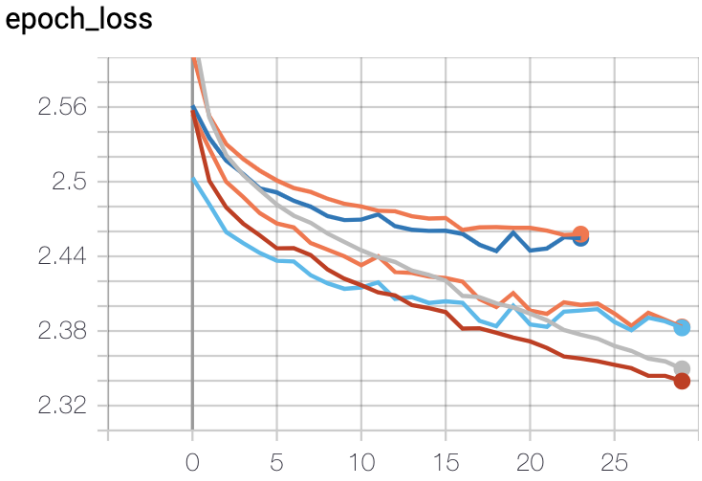

Figure 39: Loss curves during additional MLM pre-training of ProtBertBFD on positive data from *netmhcpan4.1*.

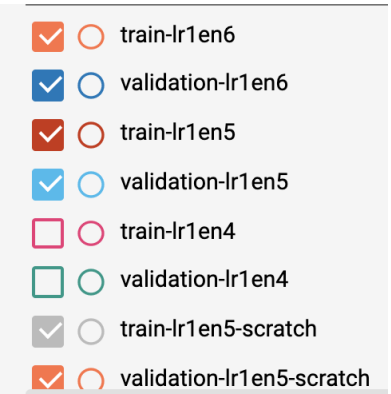

Figure 40: Legend for the MLM loss curves.

Above are displayed the training and validation loss curves, for different learning rates, during the MLM pre-training of ProtBertBFD on positive domain data, i.e. positive instances of peptide and MHC (allele).

F Example of a model configuration file

The following `.yaml` file contains the configuration to train a BiLSTM-based (4.2) presentation model within the framework. The different features and the target are defined in the variables' section (L1) while the model architecture is specified in the model section (L17). All hyper-parameters, losses metrics and attributes required for the training are defined in the training section (L49).

```yaml
variables:
  inputs:
    allele:
      type: MULTICLASS
      default_value: ''
    peptide:
      type: SEQUENTIAL
  outputs:
    binder:
      type: BINARY
preprocessing:
  chunksize: 1000000
  max_workers: 12
  allele:
    steps:
    - clean_allele
model:
  type: PresentationModel
  builder: PresentationModelBuilder
  default_activation: relu
  fusion:
    concat_allele_peptide:
      combiner: concat
      inputs:
      - allele
      - peptide
  encoding:
    allele:
    - class_name: Dense
      units: 150
```

```yaml
      activation: relu
    peptide:
    - class_name: Embedding
      output_dim: 50
      input_dim: 23
    - class_name: BidirectionalLSTM
      units: 100
    - class_name: Dense
      units: 150
    concat_allele_peptide:
    - class_name: Dense
      units: 100
  decoding:
    concatenated:
    - class_name: Dense
      units: 210
    - class_name: Dense
      units: 210
training:
  nb_gpus: 2
  epochs: 80
  input_pipeline:
    batch_size: 4096
    max_workers: -1
    prefetch: -1
    shuffle_buffer_size: perfect
    cache: memory
  neptune_logs: true
  use_mixed_precision: false
  optimizer:
    class_name: Adam
    clipglobalnorm: 1.0
    learning_rate:
      class_name: BertLRSchedule
      num_train_steps: 6720
      init_lr: 0.001778
      warmup_steps: 168
  callbacks:
  - class_name: EarlyStoppingByTopK
    target_name: binder
    patience: 20
    min_delta: 0
    restore_best_weights: true
```

```
74    metrics:
75    - class_name: BinaryAccuracy
76      name: accuracy
77    - class_name: AUC
78      name: AUC
79    - class_name: Precision
80      name: precision
81    - class_name: Recall
82      name: recall
83    - class_name: F1Score
84      threshold: 0.5
85      name: F1
86    - class_name: BinaryCrossentropy
87      from_logits: false
88      label_smoothing: 0.1
89    loss:
90    - class_name: SigmoidFocalCrossEntropy
91      from_logits: false
92      alpha: 0.25
93      gamma: 2.0
94      reduction: sum_over_batch_size
```