Predicting Epitope Specificity of T Cell Receptors with Deep Learning

Submitted by: Justin Barton
Email: jbarto02@mail.bbk.ac.uk

First supervisor: Dr. Adrian Shepherd
Affiliation / Position: Professor of Computational Biology, Birkbeck
Email: a.shepherd@mail.cryst.bbk.ac.uk

Second supervisor: Pejvak Moghimi
Affiliation / Position: PhD student, LIDo CASE studentship
Email: pmoghi01@mail.bbk.ac.uk

Birkbeck, University of London
Malet Street, Bloomsbury
London WC1E 7HX

MRes Bioinformatics

Abstract

With the development of high-throughput TCR sequencing techniques, an increasing amount of data is starting to be accumulated on TCRs and the antigens that they bind. Unfortunately, at present antigen targets are known for only a small subset of TCRs. These antigen targets must be investigated experimentally, and the process is expensive, time-consuming, and low throughput.

Attempts have been made to tackle the prediction of TCR antigen specificity *in silico*, including models such as TCRex (Gielis *et al.*, 2018), NetTCR (Jurtz *et al.*, 2018), and TCRGP (Jokinen *et al.*, 2019). TCRex reported a mean AUC of 0.849, NetTCR a mean AUC of 0.727, and TCRGP a mean AUC of 0.863. However, these models were variously limited in their scope and generalisability.

Leveraging recent advances in deep learning, we were able to construct a deep neural network model which has the ability to predict TCR-epitope binding with an accuracy of 0.986 and a mean AUC of 0.994. Unlike previous models, this model is pan-epitope, pan-HLA, and generalises well to unseen TCRs.

We believe that this model has potential applications in pathogen diagnostics, immunotherapy treatments, and treatment of autoimmune diseases.

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor Dr. Adrian Shepherd for his invaluable guidance and support throughout this project. I would also like to thank Pejvak Moghimi and William Lees for their time and expertise.

Molecular graphics and analyses were performed with UCSF Chimera, developed by the Resource for Biocomputing, Visualization, and Informatics at the University of California, San Francisco, with support from NIH P41-GM103311 (Pettersen *et al.*, 2004).

Abbreviations

AA: Amino Acid

AUC: Area Under the (Receiver Operating Characteristic) Curve

BLOSUM: BLOcks SUbstitution Matrix

CDR: Complementarity Determining Region

CNN: Convolutional Neural Network

DFF: Deep Feed-Forward network

FFN: Feed-Forward Network

GP: Gaussian Process

HLA: Human Leukocyte Antigen

HMM: Hidden Markov Model

LSTM: Long Short-Term Memory

MHC: Major Histocompatibility Complex

MLP: Multi-Layered Perceptron

OHE: One-Hot Encoded

PBMC: Peripheral Blood Mononuclear Cell

PBT: Population-Based Training

pMHC: A complex of peptide and MHC

PR: Precision-Recall

ReLU: Rectified Linear Unit

Rep-Seq: Repertoire Sequencing

RF: Random Forest

RNN: Recurrent Neural Network

ROC: Receiver Operating Characteristic

TCR: T-Cell Receptor

1. Introduction

The aim of this project was to create a deep learning model capable of accurately predicting the binding behaviour of T cells to epitopes. The following introduction serves to provide a cursory overview, of both the aspects of the human immune system and the concepts in deep learning, that will elucidate the context of this work and the challenges that it addresses.

1.1. Adaptive immunity

The human immune system is divided into two functional parts: the innate immune system and the adaptive immune system. The innate immune system is present within all multicellular organisms. It responds to pathogens immediately or within a few hours and is nonspecific, meaning that the response is not targeted to any particular pathogen.

The adaptive immune system, by contrast, is only found in vertebrates. Its response can take days to weeks and it has a long-lasting memory that can span decades. It is specific, meaning that its response is targeted to specific pathogens or antigens. This response is mediated by cells called lymphocytes.

Broadly speaking, the subset of lymphocytes that respond to humoral (extracellular) antigens are called B lymphocytes or B cells and lymphocytes that respond to intracellular antigens are called T lymphocytes or T cells.

1.2. TCR structure

T cells have proteins on their surface called T-cell receptors (TCRs) that bind with specific peptides presented by major histocompatibility complex (MHC) proteins, as shown in *Figure 1*. T-cell receptors are heterodimeric proteins, composed of an α chain and a β chain. Each chain has three loops that interact with the peptide-MHC (pMHC) complex. Owing to their role in binding the pMHC, these six loops are referred to as complementarity determining regions (CDRs). Of these six CDRs, it is believed that the third loop on the β chain, referred to as CDR3β, has the greatest impact on the interaction of TCR and pMHC (Lanzarotti *et al.*, 2019) (see *Figure 3*).

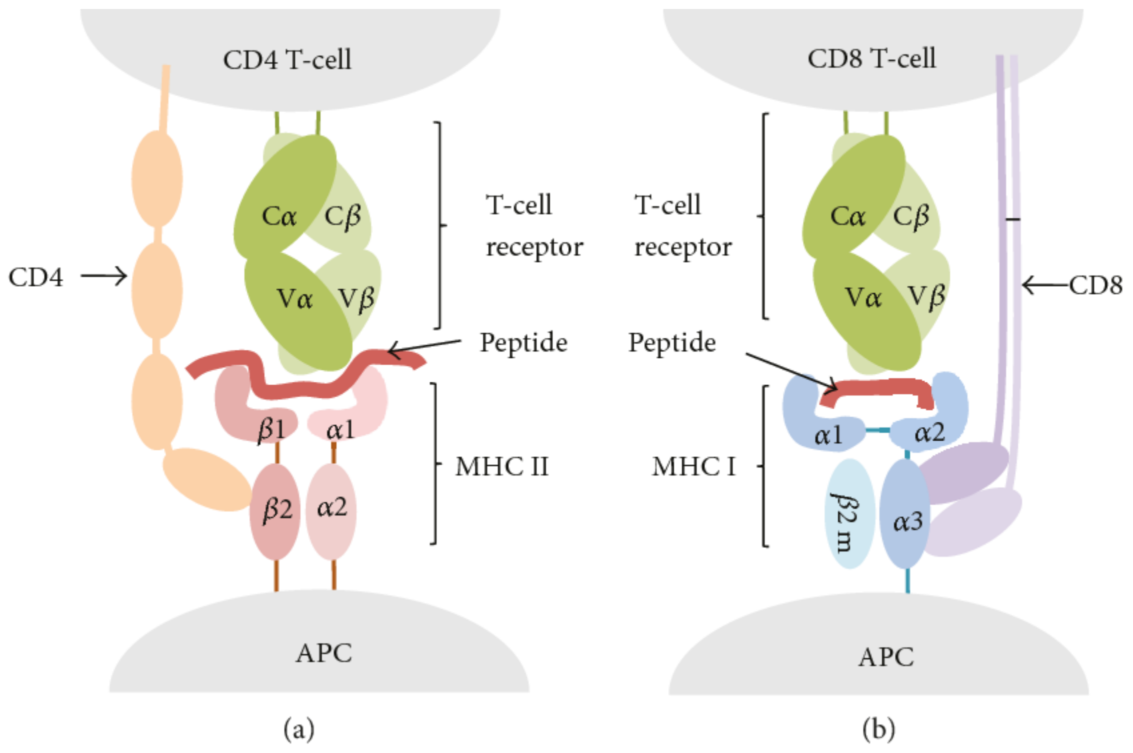


Figure 1: Illustrations of the TCR-pMHC complexes of CD4+ and CD8+ T cells. (a) The CD4+ T cell receptor binds to peptides presented by MHC Class II glycoproteins. The MHC Class II proteins present peptides of 14-18 amino acids from exogenous antigens found in the extracellular environment. (b) The CD8+ T cell receptor binds to peptides presented by MHC Class I glycoproteins. The MHC Class I proteins present peptides of 8-10 amino acids from endogenous antigens found within the cytoplasm. Notice how the presented peptide sits within the MHC Class I groove and the residues at each end of the peptide are anchored in the groove. By contrast, the residues at each end of the longer peptide presented by the MHC Class II protein overhang the binding groove. (adapted from Sanchez-Trincado et al., 2017)

Recombination of variable (V), diversity (D), and joining (J) gene segments during T cell maturation in the thymus leads to a vast space of possible TCRs, with estimates of 10^{15} distinct possible combinations (Sewell, 2012; Zarnitsyna *et al.*, 2013). Not all combinations are observed with equal frequency, however. Biases in the VDJ recombination process mean that some combinations are much more likely than others (Madi *et al.*, 2017).

These biases in the VDJ recombination process give rise to what are called 'public' TCRs -- TCR clonotypes which are shared by a large sub-population (in contrast to private TCRs which are specific to an individual). In practice the public/private dynamic is more of a spectrum than a binary distinction. The set of TCRs within an individual subject is referred to as the TCR repertoire. The TCR repertoire is shaped in part by a selection process that promotes useful TCRs that successfully bind antigens and downregulates potentially harmful TCRs which bind self-antigens. This selection process is often referred to as 'thymic selection' since it takes place in the thymus. The TCR repertoire effectively stores a memory of the pathogens that a host has been exposed to (DeWitt *et al.*, 2018).

1.2.1 a and β chains

Many early TCR sequence data sets provided sequence data only for the β chain of the TCR. Studies reporting paired α and β chain sequences were historically much less common due to the higher cost and lower throughput of the methods required (Meysman *et al.*, 2019). However, there has been a mounting body of evidence suggesting that models that utilise both the α and β chain sequence have improved predictive power over those that use only the β chain (Dash *et al.*, 2017; Lanzarotti *et al.*, 2019; Fischer *et al.*, 2019) (see *1.4* for more detail on why this may be the case). As a result, there has been a dramatic shift in trend towards sequencing paired α and β chains (see *Figure 2*). As can be seen in *Figure 2*, choosing to use paired α and β chains still comes with a material penalty with respect to sample size, since many records still only contain β chains or α chains in isolation.

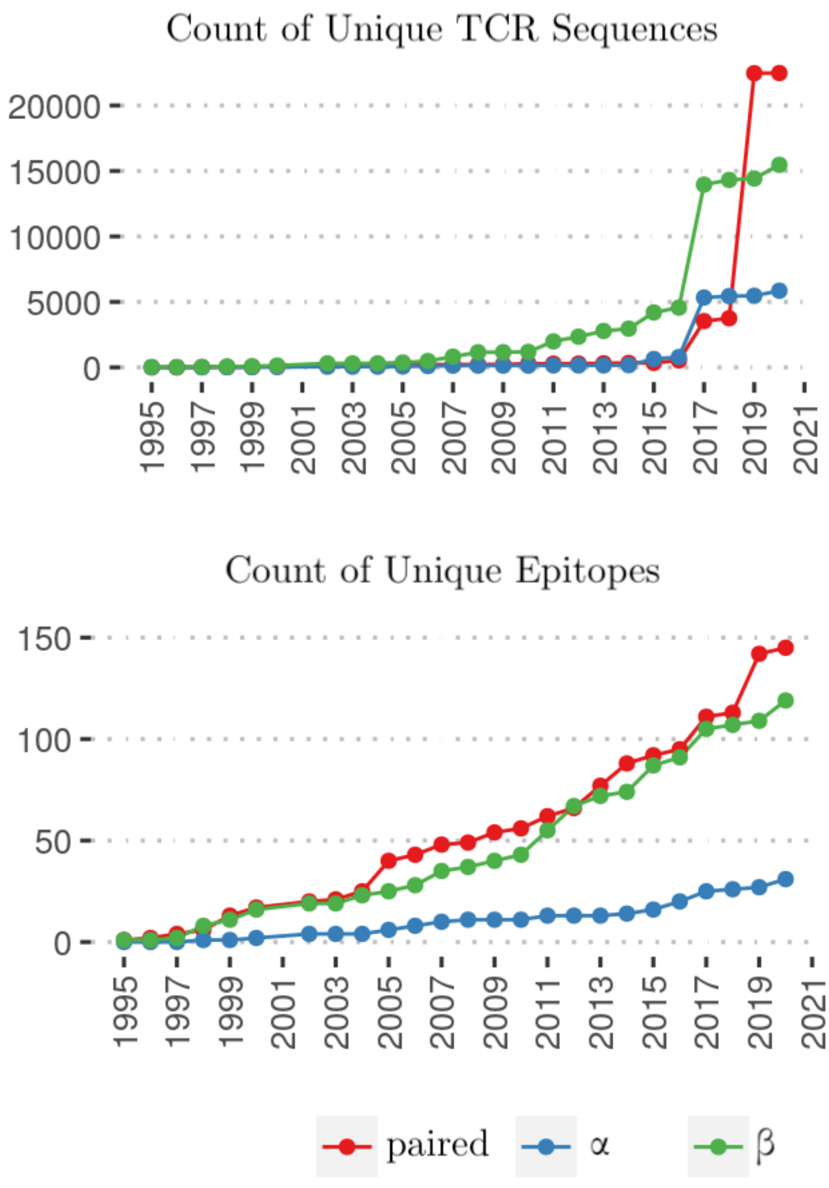
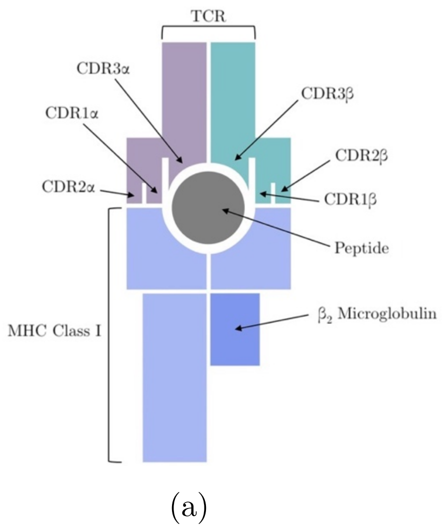


Figure 2: Counts from VDJdb illustrate the trend towards reporting TCRs with paired a and β chains. 'a' refers to records with a chain sequence only, 'β' refers to records with β chain sequence only, and 'paired' refers to records with paired a and β chain sequences. (adapted from: https://vdjdb.cdr3.net/overview)

1.2.2 Complementarity Determining Regions

With respect to the three complementarity determining regions on each of the α and β chains, based on structure analysis, it is believed that CDR3β has the greatest impact on the interaction of TCR and pMHC (Lanzarotti *et al.*, 2019) (see *Figure 3 (a)*).

More recently, there has been evidence suggesting that the CDRs most actively involved with TCR-pMHC binding may vary by epitope. This is likely due to the complex conformational changes in the TCR and pMHC complex which can facilitate binding (Baker *et al.*, 2012; Rossjohn *et al.*, 2015). Through this conformational plasticity a single TCR can potentially recognise many different epitopes (Wooldridge *et al.*, 2012) (see *1.4* for a more in-depth discussion).



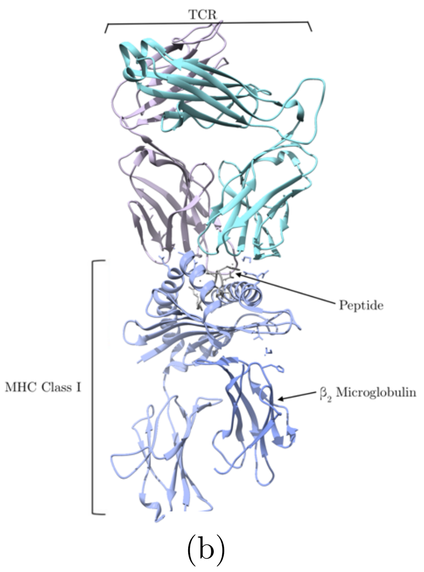


Figure 3: (a) Diagram of a CD8+ T Cell Receptor (TCR) bound to a peptide-MHC (pMHC) complex (class I) showing the locations of the Complementarity Determining Regions (CDRs) on the a and β chains (adapted from Mösch et al., 2019). (b) 3D structure of a CD8+ TCR-pMHC complex showing the secondary structure and conformation (from PDB structure 5ivx, modelled in UCSF Chimera). Note that the same colors are used in both diagrams to distinguish the TCR a and β chains, peptide, MHC, and β_2 microglobulin (a domain of the MHC Class I molecule).

1.3. MHC haplotype

On the ligand side of the binding problem, there are different variants of the MHC proteins coded for in genes located on chromosome six in humans. For the MHC Class I protein (MHC-I or MHCI, called the Human Leukocyte Antigen or HLA in humans) that present antigen peptides to CD8+ T cells, there are three loci referred to as HLA-A, HLA-B, and HLA-C. Because these genes are so polymorphic, most humans are heterozygous at these loci, which means that each person may have up to six HLA alleles and correspondingly up to six MHC-I isoforms on the surface of cells. These alleles are collectively grouped into a haplotype. Accounting for MHC haplotype in the model is essential for accurately predicting TCR-pMHC binding (Pogorelyy and Shugay, 2019). This is intuitive since the MHC haplotype will significantly affect peptide presentation and the pMHC complex. For example, different MHC haplotypes can present different sets of peptides, or the same peptides in different conformations, or even just slightly different orientations which expose different side chains of key residues in the peptide. Taking into account MHC-restriction allows the model to avoid false positives by not predicting invalid pMHC combinations (Zvyagin *et al.*, 2020).

HLA supertypes, which group HLA haplotypes with similar peptide binding specificities, have been developed for use in MHC binding prediction (Sanchez-Trincado *et al.*, 2017). These HLA supertypes may be more performant as input features than standard HLA haplotypes under certain conditions. For example, HLA supertypes can provide a reduction in input dimensions by grouping inputs into larger categories, which may be useful for models which are suffering from statistical power issues due to a small number of training examples and a large number of model coefficients. HLA supertypes may also be able to provide the model with additional input information about functional similarity which may not be included otherwise.

A number of pan-HLA MHC-epitope binding prediction models have been created (V. Jurtz *et al.*, 2017; Sanchez-Trincado *et al.*, 2017), suggesting that such an approach should also be viable for TCR-pMHC binding prediction models, given sufficient data.

1.4. Cross-reactivity

As mentioned in *1.2.2*, a single TCR can potentially recognise more than a million different epitopes (Wooldridge *et al.*, 2012). Conversely, an epitope can elicit a response from millions of TCRs (Sewell, 2012). This behaviour is referred to as 'cross-reactivity', or alternatively as 'degeneracy'.

Cross-reactivity is driven in large part by conformational flexibility in the TCR, peptide, and MHC, leading to a high diversity in the interfaces between the TCR and pMHC (Borbulevych *et al.*, 2009). These different interfaces can have very different contact residues and conformational geometries. *Figure 4* illustrates the known mechanisms that can lead to these varied interfaces.

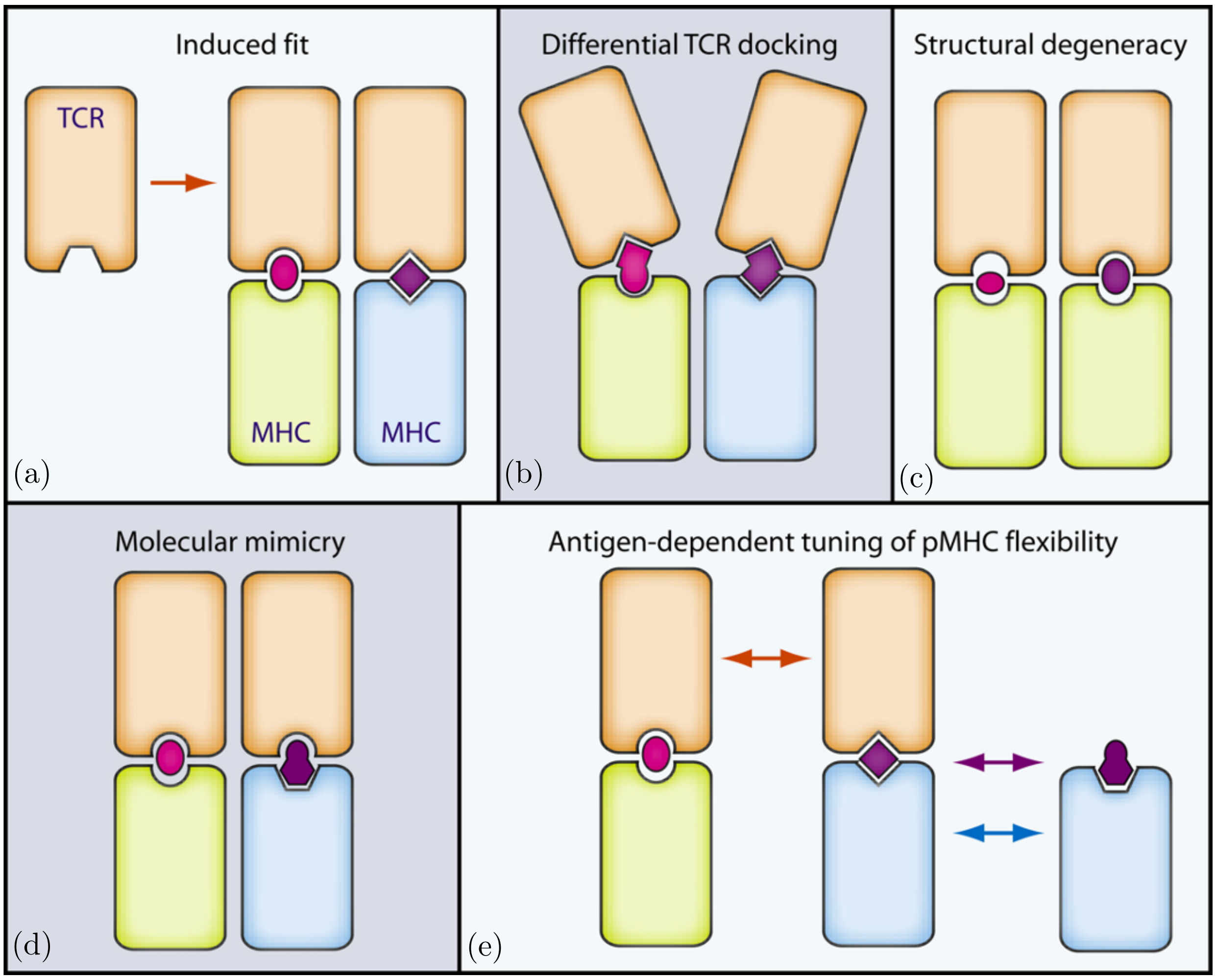


Figure 4: The mechanisms of cross-reactivity in TCR-pMHC binding. (a) Conformational changes in the TCR, typically in the CDR3 loops, allow the TCR to accommodate different pMHC complexes without changing the docking orientation. (b) A TCR is able to bind different pMHC complexes by employing different docking orientations. (c) In some instances, TCRs are able to bind with only a subset of the binding mechanisms that might otherwise be present, including some cases with no hydrogen bonds or salt bridges and poor shape complementarity. This allows the TCR to be tolerant to substitutions in contact residues of related peptides. (d) pMHC complexes that are unrelated can share key structural and chemical features which allow the TCR to recognise both with the same structure. (e) Varying degrees of conformational flexibility of the pMHC complex can allow for substantial rearrangement at the time of TCR binding. (adapted from (Yin and Mariuzza, 2009))

Perhaps the most broadly discussed of these mechanisms is the conformational flexibility of the TCR, particularly in the CDR3 loops (referred to in *Figure 4 (a)* as 'induced fit'). This flexibility allows the TCR to accommodate different pMHC complexes without changing the docking orientation. *Figure 5* gives a sense for just how variable the conformations of these CDR3 loops can be, superimposing the structures from 20 different TCR-pMHC structures from the PDB.

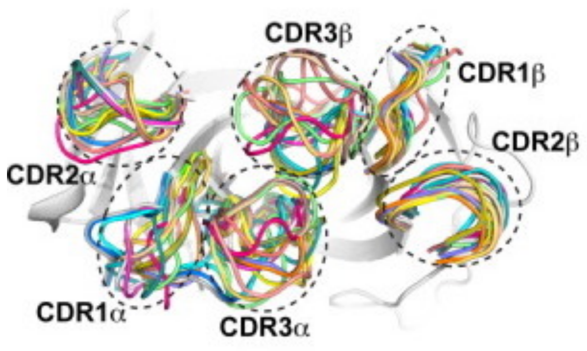


Figure 5: A superimposition of the CDR loops from 20 TCR-pMHC complex structures found in the PDB. Note the exceptional conformational variability in the CDR3β and CDR3a regions. (adapted from (Pierce and Weng, 2013))

In a similar fashion, the pMHC complex can exhibit varying degrees of conformational flexibility, which can likewise allow for substantial rearrangement at the time of TCR binding (Yin and Mariuzza, 2009). This phenomenon is referred to in *Figure 4(e)* as 'antigen-dependent tuning of pMHC flexibility'. At the present time this mechanism has been less studied and is less well-understood, though anecdotal examples are presented in (Fodor *et al.*, 2018; Pierce and Weng, 2013).

Variable docking orientation is another mechanism by which a TCR is able to bind different pMHC complexes. As seen in *Figure 6*, these docking angles can be quite variable in both the vertical (TCR tilt, see *Figure 6(a)*) and horizontal (TCR twist, see *Figure 6(b)*) planes. To-date incident angles (TCR tilt) have mostly been observed to fall within a 0° to 30° range (relative to the MHC normal vector) and crossing angles (TCR twist) within a 22° to 69° range (relative to the MHC groove vector) though outliers have been observed substantially outside of these ranges and the sample size of solved structures remains relatively small (Pierce and Weng, 2013; Rudolph *et al.*, 2006).

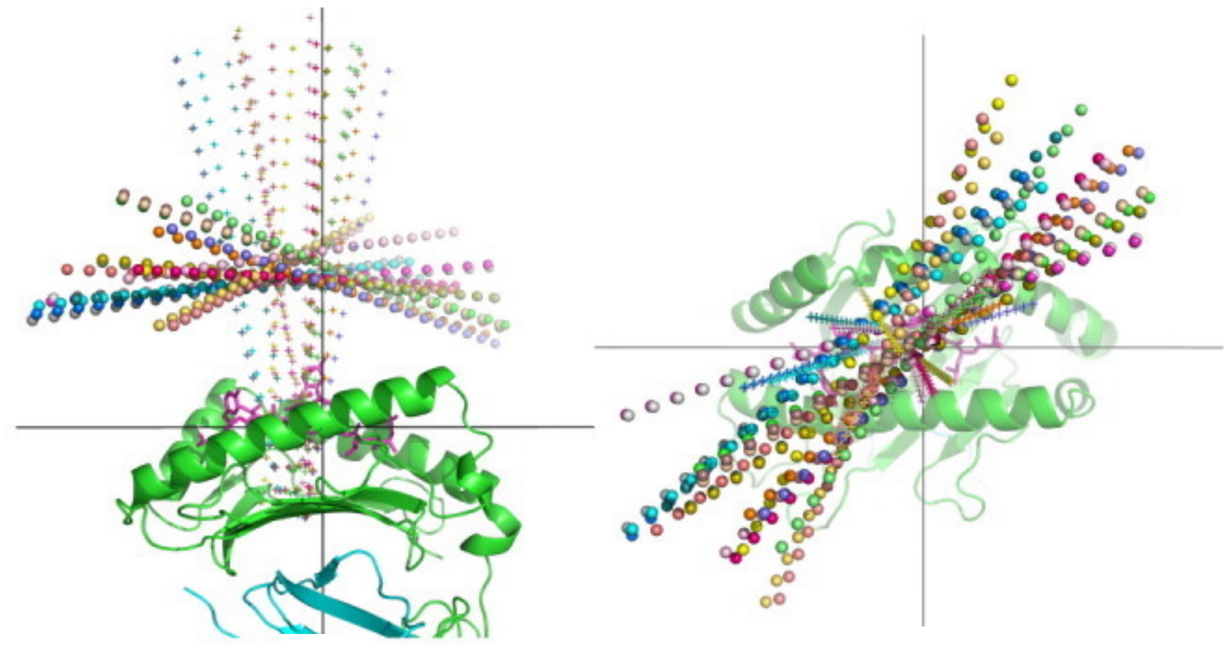


Figure 6: A superimposition of the docking angles calculated from 20 TCR-pMHC complex structures found in the PDB. (a) A side view of the TCR docking orientations showing the diversity in incident angles (TCR tilt). (b) A top view of the TCR docking orientations showing the diversity in crossing angles (TCR twist). Note that TCR interdomain vectors are shown with sphere lines and TCR interdomain axes are shown with '+' lines. The pMHC from PDB structure 1AO7 is shown for reference. (adapted from (Pierce and Weng, 2013))

'Molecular mimicry' (see *Figure 4(d)*) refers to instances where pMHC complexes that are unrelated can share key structural and chemical features, which allow a TCR to recognise both complexes. This raises an important concern for predictive models. Cross-reactive TCRs are often removed from data before training models (Lanzarotti *et al.*, 2019). In fact, most predictive models created thus far have aimed to predict a single antigen for each unique TCR. However, for improved safety and utility in immunotherapy applications, models should ideally aim to predict all reactive epitopes. Unanticipated cross-reactivity is a major concern since on-target and off-target toxicity can be extremely damaging, and even fatal. On-target toxicity occurs when introduction of an epitope causes unintentional cytotoxic destruction of healthy tissues. Off-target toxicity occurs in situations like molecular mimicry where different antigens are presented as pMHCs with similar key structural and chemical features and bind indifferently with TCRs, potentially leading to unintended collateral damage (Mösch *et al.*, 2019).

In some cases, TCRs are able to bind pMHC complexes with only a subset of the binding mechanisms that might otherwise be present, including some cases with no hydrogen bonds or salt bridges and poor shape complementarity (see *Figure 4(c)*). This allows the TCR binding to be resilient to substitutions in contact residues of related peptides.

Note that the mechanisms of TCR cross-reactivity discussed here are laid out for conceptual understanding and are by no means mutually exclusive. Combinations of these mechanisms have been observed and it is likely that cross-reactivity dynamics include a complex interplay of various mechanisms.

In the complementary context of peptide-MHC binding, it has been shown that cross-reactivity can be predicted with a high degree of accuracy using machine learning approaches trained on feature data including the electrostatic charges and accessible surface area of selected residues in the pMHC-I complex (Mendes *et al.*, 2015).

Despite the complexity of cross-reactivity, TCR sequences binding the same epitope often share similarities (Dash *et al.*, 2017; Madi *et al.*, 2017; Glanville *et al.*, 2017; Meysman *et al.*, 2019). This suggests that it should be possible to apply machine learning methods to predict TCR epitope specificity in silico. Some attempts have previously been made to do so (Gielis *et al.*, 2018; Jurtz *et al.*, 2018; Jokinen *et al.*, 2019) (see *1.7.4* for further discussion of these models).

1.5. Data sources

1.5.1 Databases

Name	Paper	Location
VDJdb	(Shugay et al. 2018)	vdjdb.cdr3.net
IEDB	(Vita *et al.*, 2019)	iedb.org
McPAS-TCR	(Tickotsky et al. 2017)	friedmanlab.weizmann.ac.il/McPAS-TCR/
TBAdb	(Zhang *et al.*, 2020)	db.cngb.org/pird/tbadb/

Table 1: Databases containing TCR sequences with associated epitopes. Each of these databases collate data from published studies, but as shown later in Table 5, they contain substantially different records.

Table 1 shows the main databases which house TCR sequence data with associated epitopes. Of the four, VDJdb currently has the greatest number of TCR sequences, particularly with paired α and β chain sequences. Recent counts and historical trends can be seen in *Figure 2*. Given the current sparsity of TCR specificity data, it is often necessary to aggregate data from multiple sources in order to compile a sufficient training data set.

1.5.2 Data from individual studies

There are a number of TCR and TCR repertoire studies which have provided novel TCR sequence data sets that can be utilised as training data. Two commonly used examples are the Dash data set (Dash *et al.*, 2017) and the MIRA data set (Jurtz *et al.*, 2018). In time these may be added to VDJdb and other databases.

1.6. Data collection

TCR specificity data is typically experimentally collected in one of two ways: proliferation assay methods or epitope-MHC multimer techniques. Proliferation assay methods such as limiting dilution assays had been the standard until recent years. These methods test for reactions to epitopes added to T cell populations, typically in peripheral blood mononuclear cells (PBMCs). Because these methods require adding one peptide at a time, they are time-consuming, low-throughput, and require a specific antigen of interest.

More recently, epitope-MHC multimer techniques have gained traction. These allow many epitopes to be tested simultaneously, but require the input of expensive epitope-MHC libraries (Birnbaum *et al.*, 2014). As a consequence of this, Epitope-MHC multimer techniques are subject to the limitation that the epitope or antigen under investigation must be included in the library, so the breadth of antigens for which this technique is applicable is a limited finite set. In other words, though these methods are significantly higher throughput, they are limited in the scope of epitopes and antigens.

Since these techniques typically use PBMCs, their application can be limited for tissue-specific tumours and localised autoimmune diseases for which specific T cells are rarely found circulating in blood (Mann *et al.*, 2020). There can also be specificity issues for any TCR-epitope combinations with low affinity.

1.7. Existing models

There are four broad types of models that exist today for predicting TCR epitope specificity: database search algorithms, cluster models, structural models, and ML models.

1.7.1 Database search algorithms

Each of the databases listed in *Table 1* have at least one kind of database search algorithm allowing for searching by TCR sequence or epitope sequence. In many cases this is just an implementation of blastp. In addition, VDJdb has a tool called VDJmatch which allows for command line and API searches (Shugay *et al.*, 2018). The VDJmatch algorithm computes a weighted score based on match/mismatch of the V and J germline regions and sequence alignment of the CDR3 regions.

While there is certainly a precedent for using sequence alignment searches with protein databases, for the purpose of making inferences about the antigen specificity of a TCR, blastp and other standard sequence alignment approaches are likely suboptimal. As noted in *1.4*, conformational changes controlled by key residues have a substantial impact on binding dynamics, and as noted in *Figure 4 (c)*, epitope specificity can be in some cases be tolerant of substitutions in contact residues. In other words, the impact of substitutions on binding may vary greatly by residue position.

1.7.2 Structural approaches

There are a number of proposed structural modelling approaches for TCR-pMHC complexes such as TCRmodel (Gowthaman and Pierce, 2018) and LYRA (Klausen *et al.*, 2015). These approaches start with a structural template from a known, similar TCR and refine the structure by replacing CDR loops with other known CDR loop structures, based on sequence similarity. A serious limitation of these types of approaches is that they presume that the TCR has a single, fixed structure. As discussed in *1.4*, TCR binding with pMHC complexes is often associated with substantial conformational changes. *Figure 7* shows just how substantial these conformational changes can be, in the TCR variable domain, by plotting the mean RMSD between the bound and unbound locations for the residues in 20 TCRs which have structures for bound and unbound states in the PDB.

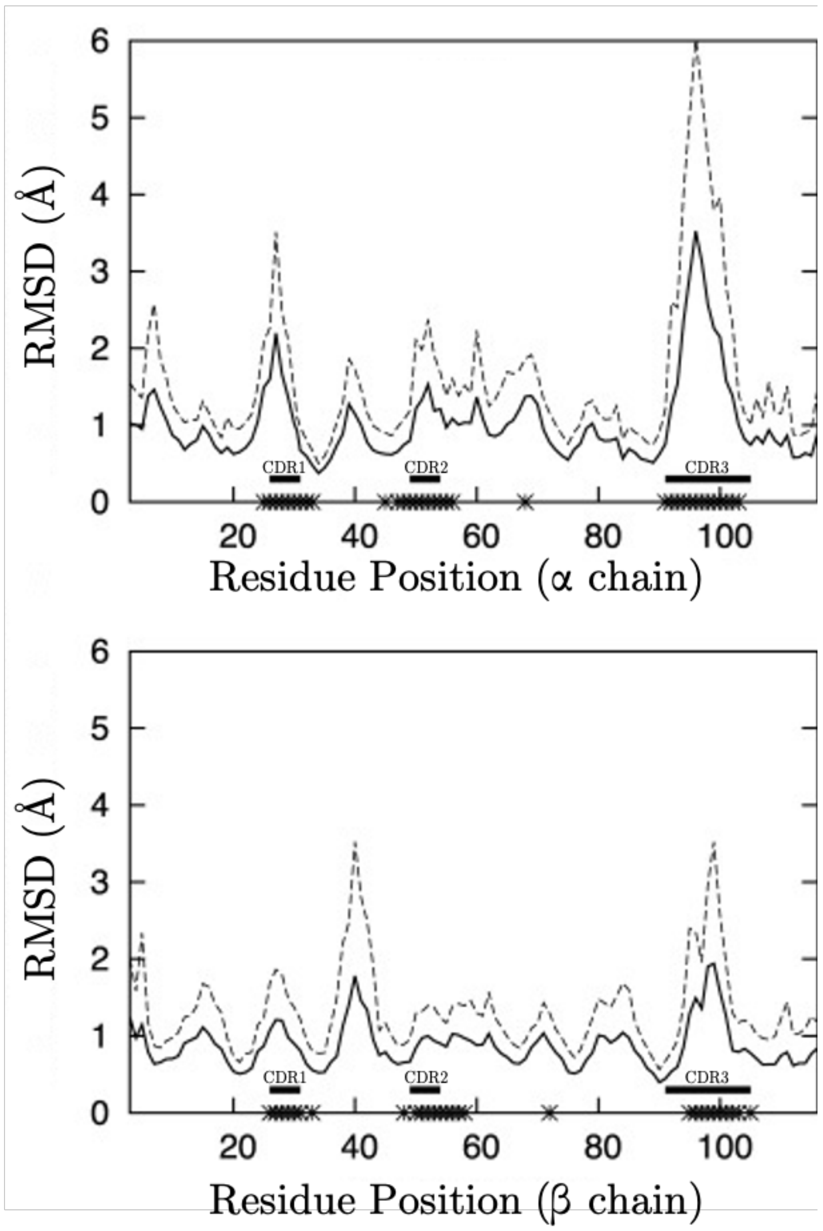


Figure 7: The average distance between the bound and unbound locations of TCR variable domain residues by residue position (index). 20 TCRs were found that had both bound and unbound structures in the PDB. These 20 TCRs were superimposed in their bound and unbound states and the mean backbone RMSD between bound and unbound structures calculated, by residue, for the a chain and β chain. Mean RMSD is plotted as a solid line, with a dotted line representing one standard deviation above the mean. CDR loop regions are noted with horizontal bars (based on indices from the IMGT definition), and asterisks on the x-axis indicate contact residues (within 6.0 Å) for ≥3 of the 20 TCR-pMHC complexes. (adapted from (Pierce and Weng, 2013))

TCRBuilder (Wong *et al.*, 2020) somewhat addresses this issue by returning an ensemble of conformations where structures for multiple CDR conformations are available. The major limitation that remains with this approach is that it still only returns combinations of observed conformations, and just a few hundred TCR-pMHC structures have been resolved (and only 112 human TCR-pMHCI structures can be found in the PDB presently).

TCRFlexDock (Pierce and Weng, 2013) is an interesting example which is not limited to combinations of observed conformations. TCRFlexDock is a docking protocol using a combination of RosettaDock (Gray *et al.*, 2003) and ZRANK (Pierce and Weng, 2008). It uses Monte Carlo methods to simulate a combination of rigid body movements, side chain movements, and CDR loop refinements, to model the docking of TCR and pMHC. 1000 docking predictions are generated per complex and scored using ZRANK to rank the predictions. While an interesting approach, the primary function of this model is to predict structures for known binding TCR-pMHC complexes, not to predict whether they will bind. The applications of this approach to the latter problem are questionable as even in the ideal case when the protocol is given the known structure of the TCR-pMHC complex and asked to predict binding affinities, the correlation between predicted and measured binding affinities is R=0.79.

1.7.3 Clustering

There are a number of clustering models for clustering TCR sequences by specificity groups (ie. clusters of TCRs targeting the same antigen or MHC-restricted epitope), including TCRdist (Dash *et al.*, 2017), GLIPH (Glanville *et al.*, 2017), ALICE (Pogorelyy *et al.*, 2019), and TCRNET (Ritvo *et al.*, 2018).

One key difference between these models is that ALICE uses a probabilistic VDJ recombination model to account for biases that VDJ recombination creates in TCR frequency (Pogorelyy *et al.*, 2019) whereas TCRNET requires user-provided baseline samples as a control (Ritvo *et al.*, 2018).

A strength of clustering approaches is that they are particularly good at accurately labelling de novo TCRs. In one validation experiment, 85% of TCRs correctly annotated by TCRdist were not observed in the training data (Dash *et al.*, 2017). This is important because previously unseen TCR clonotypes frequently show up in repertoire data, and are to be expected due to the dynamics of public and private TCR clonotypes mentioned in *1.2*.

A challenge of clustering approaches is that the set of TCRs that will bind a given epitope are very diverse, often being spread over multiple clusters (Meysman *et al.*, 2019). They are also prone to a high number of type I errors at any threshold (Meysman *et al.*, 2019).

1.7.4 Machine learning models

Name	Paper	Web Service	Model type
TCRex	(Gielis et al. 2018)	tcrex.biodatamining.be	RF
NetTCR	(Jurtz et al. 2018)	cbs.dtu.dk/services/NetTCR/	CNN
TCRGP	(Jokinen et al. 2019)	-	GP

Table 2: Previous machine learning approaches to predicting TCR-epitope interaction. The 'model type' refers to the family of machine learning models used, where 'RF' = Random Forest, 'CNN' = Convolutional Neural Network, and 'GP' = Gaussian Process.

Algorithms using machine learning methods to predict TCR-epitope interaction are listed in *Table 2*. While these algorithms all aim to fulfil the same predictive function, the methods vary substantially.

TCRex (Gielis *et al.*, 2018) is a Random Forest classifier using a profile scoring method. It takes TCR sequence and epitope sequence as joint inputs and returns a binary classifier p-value. At present this model only accepts TCRβ CDR3 sequences, though it could be adapted and retrained to work on more sequence regions. The main concern with this model is how scalable it would be to large numbers of TCRs and epitopes. Since the scoring must be run once for every TCR-epitope pair, run times could become very long when testing large numbers of TCRs and/or epitopes. Further, Random Forest models tend to perform well by specialising trees to subpopulations, which works well for small numbers of categories. As the number of TCRs and epitopes become large, the number of trees required for high performance may be impractical.

NetTCR (Jurtz *et al.*, 2018) is a CNN that, like TCRex, also uses a profile scoring approach with BLOSUM substitution matrix scores. It similarly uses only the TCRβ chain as input, but again this could be adjusted and retrained on more sequence regions. Perhaps the biggest limitation of NetTCR is that it is restricted to HLA-A*02:01 only. Though HLA-A*02:01 is the most common allele and is found in an estimated 19% of the global population (Gonzalez-Galarza *et al.*, 2020), this limits clinical utility and raises questions about generalisability of this model to multiple HLA alleles.

TCRGP (Jokinen *et al.*, 2019) utilises a Gaussian Process model, which is nonparametric. Sequences are trimmed to equal length and then BLOSUM substitution scores are applied via a squared exponential kernel function.

It is worth noting that there exists a mature class of models for the purpose of predicting MHC peptide presentation. One particularly feature-rich example is NetMHCpan (for MHC Class I) and NetMHCIIpan (for MHC Class II), which allows for pan-HLA predictions of MHC peptide presentation (Reynisson *et al.*, 2020). For a comprehensive list and comparison of MHC peptide presentation prediction algorithms, see the review by (Sanchez-Trincado *et al.*, 2017).

1.8. Use of features in models

1.8.1 Physicochemical properties

A common characteristic of the models examined in *1.7.4* is the inclusion of the physicochemical properties of the CDR3 loop(s) and epitope. In particular, vectors of length, charge, helicity, and hydrophobicity are commonly used (Dash *et al.*, 2017; Jurtz *et al.*, 2018; Fischer *et al.*, 2019). There are many indices providing numerical representations of various properties of amino acids, commonly referred to as 'amino acid indices'. The database 'AAindex' currently contains 566 distinct amino acid indices (Kawashima and Kanehisa, 2000). It is not immediately obvious which subset of indices is optimal for any given application, so this must be determined empirically. More recently, some researchers have attempted to incorporate all 566 indices as inputs to k-means clusters (Guo and Chen, 2019), or directly as input features to a convolutional network (Kim *et al.*, 2019).

1.8.2 Structure

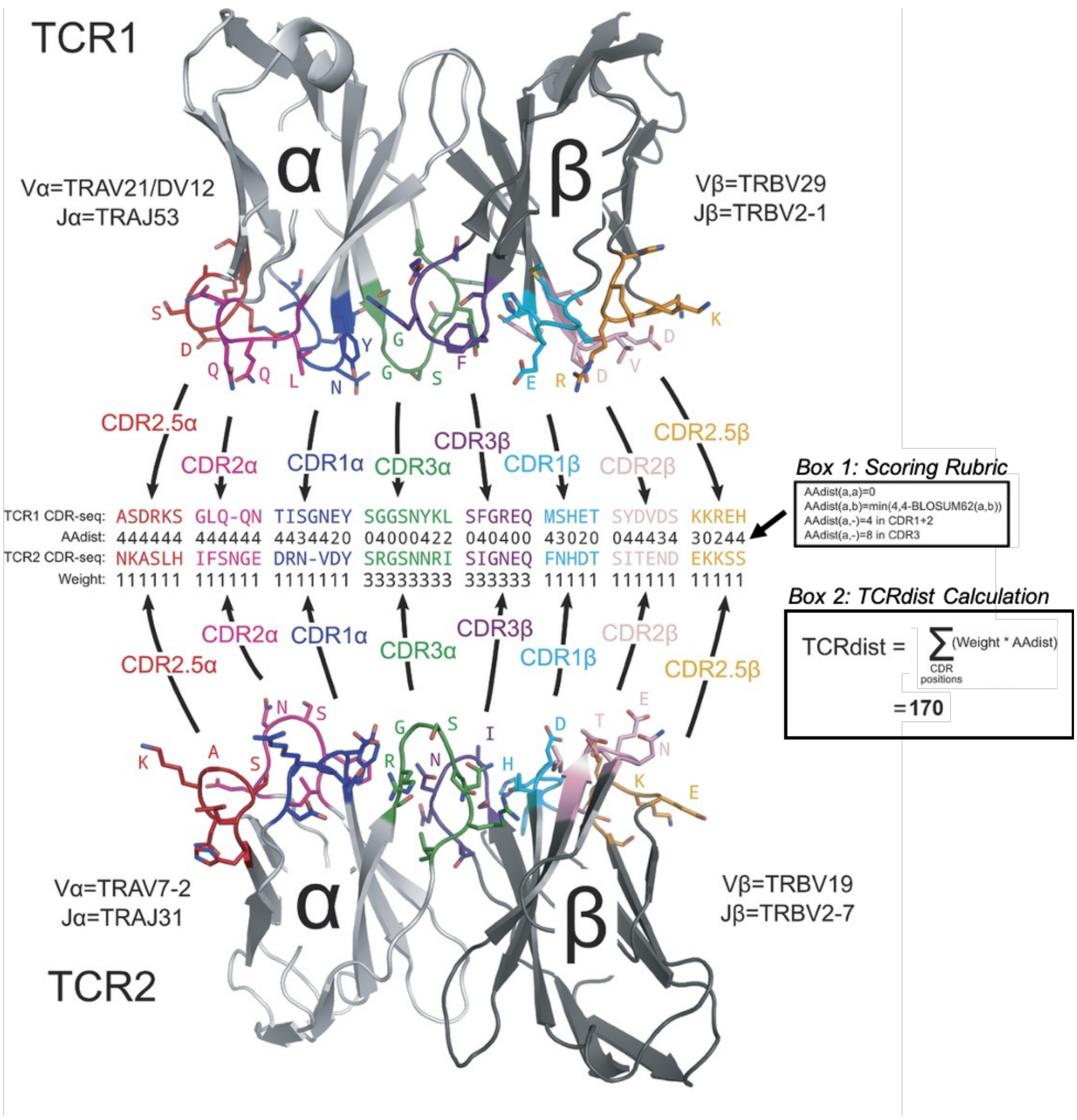
The models discussed in *1.7.2* show that it is possible to encode information about the conformational structure of the TCR and TCR-pMHC complex as input features to a model. For example, the structure of CDR loops has been included via homology modelling (Lanzarotti *et al.*, 2019).

Care should be taken with these approaches as it has been shown that TCR structure models tend to have high inaccuracy in CDR3 loop regions, with median RMSD being ~2x greater in CDR3 loops when compared to CDR1 and CDR2, in both the α and β chains (Gowthaman and Pierce, 2018). This is possibly due to the high variability of the CDR3 regions (see *Figure 5*) and the very small number of solved structures available. Consistent with this hypothesis, when templates with >90% sequence identity in the variable domains were excluded from benchmarking, many more high-RMSD outliers appeared for the CDR3β region (Gowthaman and Pierce, 2018).

1.8.3 Embeddings

Embeddings, which are vector representations of features, have been indirectly used as model inputs a number of times via sequence distance measures like TCRdist (Dash *et al.*, 2017) or physicochemical feature vectors (Jurtz *et al.*, 2018; Gielis *et al.*, 2018).

TCRdist is a pairwise measure of the sequence distance of two TCRs, designed to focus on the subsequence regions which are hypothesised to have the greatest impact on specificity (the CDR3α and CDR3β loops). *Figure 8* illustrates the steps involved in calculating the TCRdist score for an example pair of TCRs. Note that an additional loop is considered here, which Dash et al. refer to as 'CDR2.5'. This loop is found between CDR2 and CDR3 on both the α and β chains and is believed to be involved in the conformational stabilisation of the CDR3 loop. Because TCRdist produces pairwise relative distances rather than vectors in a Cartesian space, it is typically used more for clustering applications than for input features to supervised learning models.



TCR1 CDR-seq:	ASDRKS	GLQ-QN	TISGNEY	SGGSNYKL	SFGREQ	MSHET	SYDVDS	KKREH
AAdist:	444444	444444	4434420	04000422	040400	43020	044434	30244
TCR2 CDR-seq:	NKASLH	IFSNGE	DRN-VDY	SRGSNNRI	SIGNEQ	FNHDT	SITEND	EKKSS
Weight:	111111	111111	1111111	33333333	333333	11111	111111	11111

Box 1: Scoring Rubric

AAdist(a,a)=0
AAdist(a,b)=min(4,4-BLOSUM62(a,b))
AAdist(a,-)=4 in CDR1+2
AAdist(a,-)=8 in CDR3

Box 2: TCRdist Calculation

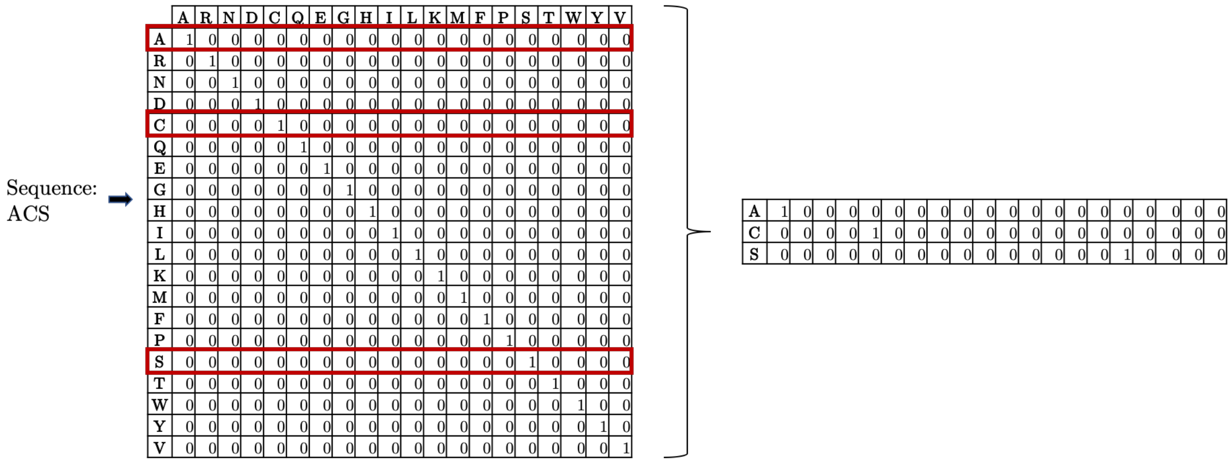
$$\text{TCRdist} = \sum_{\text{CDR positions}} (\text{Weight} * \text{AAdist})$$

$$= 170$$

Figure 8: An illustration of the TCRdist calculation. TCRdist is a pairwise measure of the sequence distance between two TCRs, weighted to the residues which are hypothesised to have the greatest impact on specificity (residues in the CDR3a and CDR3β loops). First, the subsequences of the six CDR loops (and an additional variable loop between CDR2 and CDR3 on both the a and β chains, referred to as 'CDR2.5') are extracted from both TCR sequences. The CDR subsequences are then aligned (based on the IMGT canonical forms) and a distance score (labelled 'AAdist') is computed at each index position using the rubric in Box 1. The weighted sum of AAdist scores is then computed to produce the final TCRdist score (Box 2). (adapted from (Dash et al.*, 2017))*

A naïve embedding strategy that is commonly used in supervised learning applications is one-hot encoding. One-hot encoding represents each amino acid as a vector of length

20, with a value of one in the index position corresponding to that amino acid, and a zero in all other positions. *Figure 9* illustrates an example of one-hot encoding used for the example amino acid sequence 'ACS'.

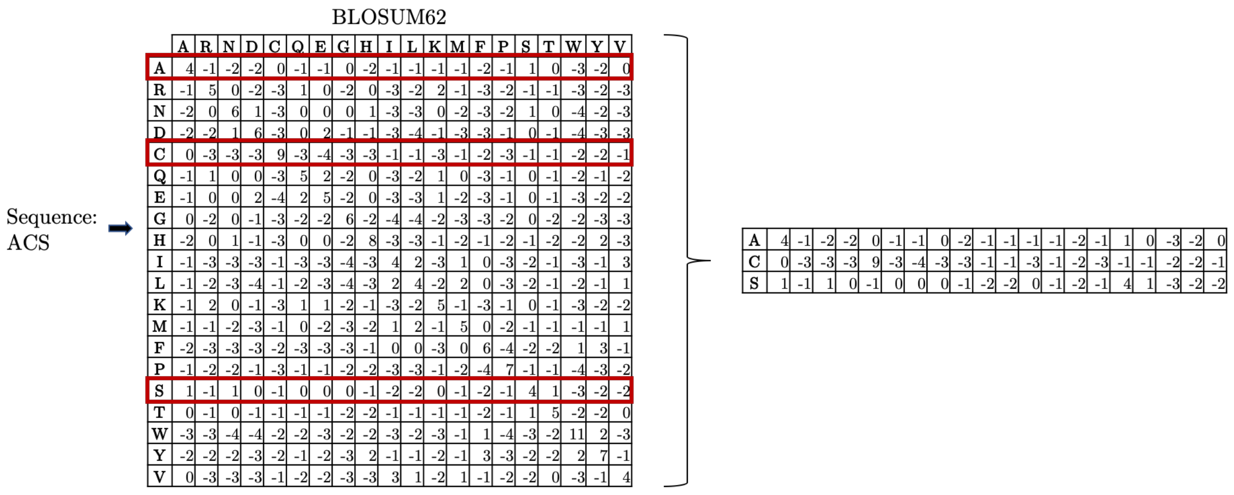


	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

A	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 9: An illustration of one-hot encoding used to encode the amino acid sequence 'ACS'. Each residue in the sequence is mapped to a sparse vector of length 20 with a one in the position corresponding to that residue's amino acid, and zeroes in all other positions. Note that the output of this process is a matrix of size [sequence length, 20].

An alternative approach to one-hot encoding is to replace the one-hot vector with the row corresponding to that amino acid from a substitution matrix, for example one of the BLOSUM matrices. *Figure 10* illustrates an example of this type of embedding using the BLOSUM62 matrix to encode the example amino acid sequence 'ACS'.



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2

Figure 10: An illustration of encoding an amino acid sequence with BLOSUM embedding vectors. Each residue in the sequence is replaced with its corresponding row from the BLOSUM matrix. Note that the output of this process is a matrix of size [sequence length, 20].

In performance comparisons, scoring with BLOSUM matrices tends to outperform naïve embeddings (Dash *et al.*, 2017). For further discussion on this, and details on implementation, see *2.2.1*.

1.8.4 TCR sequence regions

With respect to the three complementarity determining regions on each of the α and β chains of the TCR, there does not seem to be a consensus about which must be included as input features to models. As discussed in *1.8.3*, Dash et al. include a pMHC-facing loop between CDR2 and CDR3 on both the α and β chains which they refer to as CDR2.5, on the basis that this loop is important for CDR3 stabilisation and has been observed making pMHC contacts in solved structures (Dash *et al.*, 2017). They also assign a 3x higher weight to the CDR3α and CDR3β loops (see *Figure 8*), but use a 'trimmed' version with five fewer residues, trimming off the first three and last two residues. As *Figure 7* shows, the structures analysed in (Pierce and Weng, 2013) support the view that the residues contacting the pMHC are concentrated on the six CDR loops with a small number between the CDR2 and CDR3 loops at the position which Dash et al. refer to as the 'CDR2.5' loop.

Table 3 shows the CDRs which have ultimately been chosen for inclusion as input features in each of the models predicting TCR-epitope specificity (see *1.7.4)*.

Name	Paper	Chains	CDRs
TCRdist	(Dash et al. 2017)	α+β	CDR(1-3) + CDR2.5
TCRex	(Gielis et al. 2018)	β	CDR3
NetTCR	(Jurtz et al. 2018)	β	CDR3
TCRGP	(Jokinen et al. 2019)	α+β	CDR(1-3)

Table 3: The TCR sequence regions used as input features in previous ML models predicting TCR-epitope specificity, noting in particular the use of the TCR a and β chains and the subset of CDR loop regions. Note that CDR2.5 is a non-standard CDR created by Dash et al.

1.9. Project objectives

The primary aim of this project was to lay the groundwork for a usable pipeline for in silico prediction of TCR-epitope binding that would support pathogen diagnosis and immunotherapy development.

A model that successfully predicts TCR-epitope binding could be used to functionally annotate the TCR repertoires output by modern immunosequencing techniques. By making binding predictions for each TCR clonotype in the repertoire against a database of known epitopes, repertoires can be annotated with predicted antigen specificity. This antigen specificity data can be combined with relative abundance counts in order to aid in clinical diagnoses or track disease progression over time.

A successful TCR-epitope binding model could also be used in immunotherapy development in order to identify TCRs that will bind with a target antigen, to identify other epitopes that could be used to stimulate TCRs of interest, and to screen for potential off-target toxicity.

A secondary aim of the project was to establish through experimentation the optimal feature representations and architecture to use for deep learning approaches to this problem.

Aside from the aforementioned objectives, it was an aspiration of this project to develop a model that is scalable and applicable to novel TCRs and epitopes. It was not clear at the outset whether this would be achievable given the data available (Fischer *et al.*, 2019).

2. Review of relevant deep learning concepts

This work lies at the intersection of immunogenetics and deep learning, so it is prudent to briefly review some of the relevant deep learning concepts and the motivating choices for the approaches taken in this model.

Biological subfields such as genomics, genetics, and bioinformatics have been using machine learning for decades, and in recent years the explosion of genomic data, broader availability of computing resources, and the release of easy-to-use libraries have drawn attention to the sub-discipline of machine learning known as 'deep learning'. In the last two decades, deep learning has had great successes in computer vision, natural language processing, and speech recognition which have drawn a great deal of research attention.

2.1. Deep neural network architectures

All deep learning models are neural networks, but there is substantial variation in how those networks are structured. What follows is not an exhaustive list of neural network architectures, but highlights of those families of neural network architectures which are most apposite to genomic applications. For a more comprehensive survey of deep neural network architectures, see (Liu *et al.*, 2017).

2.1.1 Feed-Forward Networks (FFNs)

Feed-Forward Networks (FFNs), sometimes called Deep Feed-Forward networks (DFFs), are based on the earliest neural networks and are the simplest in terms of architecture. In Feed-Forward Networks, information moves from the input layer through a number of hidden layers to the output layer, as illustrated in *Figure 11.*

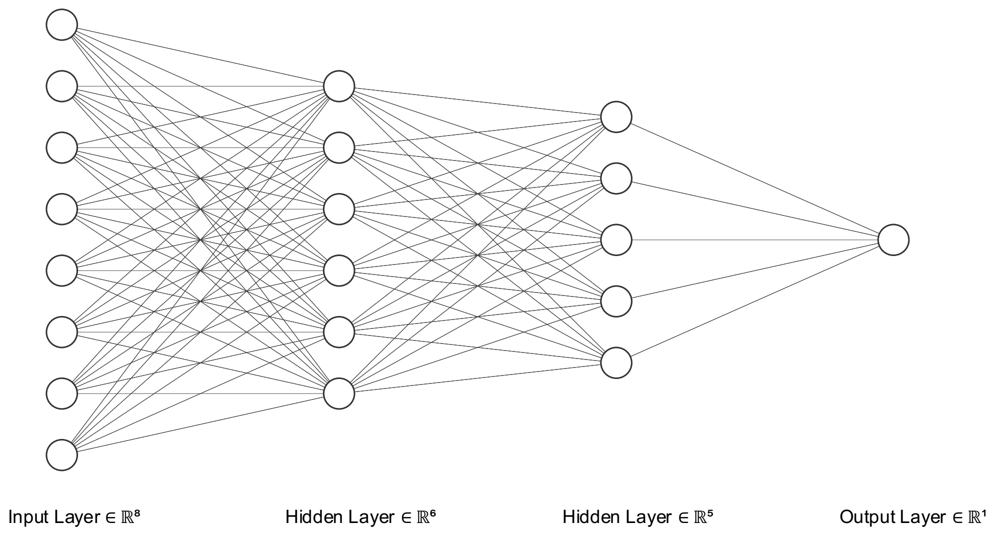


Figure 11: An example of a Feed-Forward Network (FFN). This example is comprised of an input layer with 8 features, 2 hidden layers of 6 and 5 nodes respectively, and a single-valued output layer.

Each layer is made up of a number of nodes, which are fully connected to the nodes of the preceding layer. The output of a hidden layer node is a linear function of the output values of all the nodes in the preceding layer multiplied by a set of node-specific weights which are learned during the model training process, often including an additive bias term which is also learned during the model training process. Before being output, this value is passed through a non-linear 'activation function', such as a sigmoid function. Without this activation step which introduces non-linearity, the network could always be collapsed to an equivalent single layer network since it would just be a series of linear functions.

As pictured in *Figure 11*, Feed-Forward Networks begin with an input layer and end with an output layer. The role of the input layer is simply to output the input features to the first hidden layer of the model. The role of the output layer is to output the final output of the model.

Feed-Forward Networks are sometimes referred to as a Multi-Layered Perceptrons (MLPs), though more accurately MLPs are a special case of FFNs with a particular node type and activation function.

2.1.2 Convolutional Neural Networks (CNNs)

Because all layers in a Feed-Forward Network are fully connected, they can be prone to overfitting the training data. Layers of a Convolutional Neural Network, on the other hand, are not all fully connected. Inspired by the human primary visual cortex, these networks have convolutional layers which generate local features based on a filter of weights that works in an analogous fashion to a receptive field (LeCun, Y. *et al.*, 1990). As illustrated in *Figure 12*, a convolutional filter generates local feature maps by applying a weight matrix to all local areas of the input, typically reducing in dimensions per channel at each step.

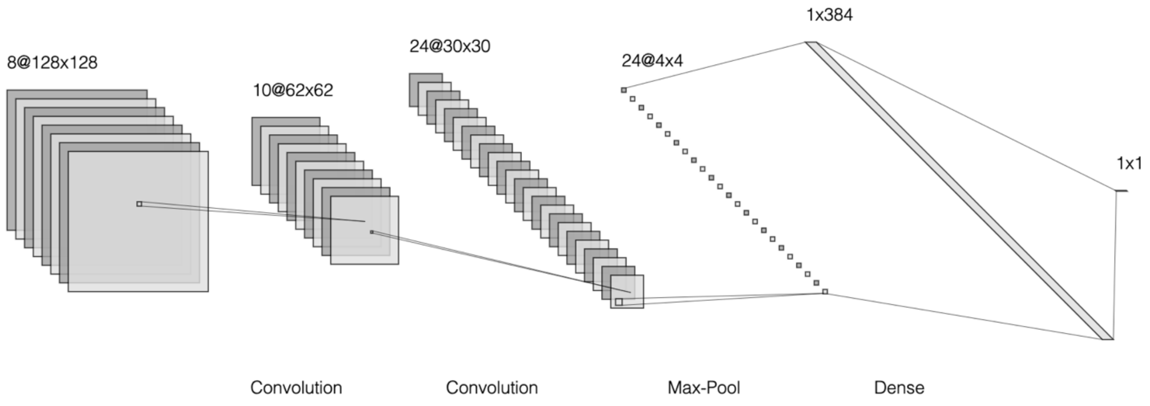


Figure 12: An example Convolutional Neural Network (CNN). This network is comprised of three layer types: Convolutional, Max-Pooling, and Dense. Convolutional layers are local feature maps created by convolving a weight matrix (often called a 'kernel' and pictured here as squares of input mapping to local outputs in the next layer) with the previous layer. In this example, the first convolutional layer is using 10 kernels of size 4x4, taking an 8x128x128 input and outputting 10 channels of 62x62. The second convolutional layer is using 24 kernels of size 2x2, taking a 10x62x62 input and outputting 24 channels of 30x30. Max-Pooling layers are a type of pooling layer typically employed after or between convolutional layers in order to reduce the dimensions of the data. In this example the Max-Pooling layer is taking all the maximum values in 6x6 windows to reduce the data from 30x30 to 4x4. Note that the number of channels (24) is not changed by max pooling. Dense layers are fully connected hidden layers like those seen in the Feed Forward Network example in Figure 11. In this example, the Dense layer is a flattening of the output of the Max-Pooling layer from 24x4x4 to 1x384, which is finally fully connected to the 1x1 output layer.

Convolutional Neural Networks have been very successfully applied to many problems of computer vision. More recently, they have proven useful in biological applications, particularly with sequence data where local features can be important (Jurtz *et al.*, 2018; Kim *et al.*, 2019).

2.1.3 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks are used to model sequences, commonly temporal sequences or sequences in text data. They are composed of 'units' which are repeated at each time step or position in the sequence, as illustrated in *Figure 13.* In this way they are structurally similar to Hidden Markov Models (HMMs). For a comprehensive overview of RNNs see (Sutskever *et al.*, 2011).

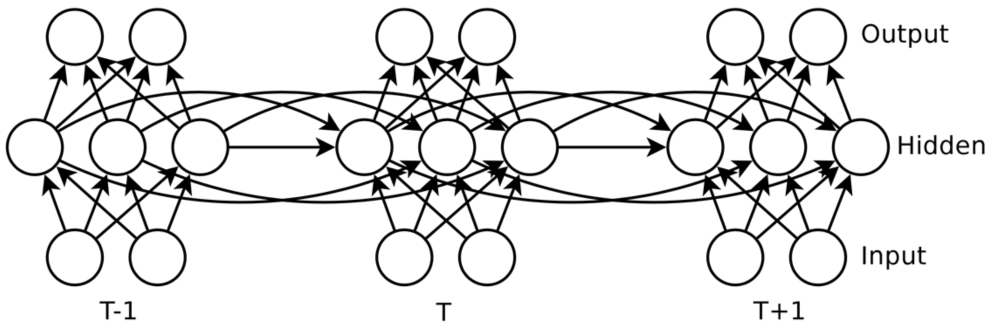


Figure 13: An example Recurrent Neural Network (RNN). Note how the same functional unit is used for each time period and how information is propagated forward through time periods. (adapted from (Sutskever et al.*, 2011))*

Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) units are an example of unit types used in RNN architectures. They expand on the RNN concept of using the output of each time period as an input to the next by adding a long-term memory component. This long-term memory component makes LSTMs particularly well-suited to problems with time lagged effects of unknown duration and makes them relatively insensitive to gap length in text sequence applications (when compared to classic RNNs or HMMs). For this reason, LSTMs have been employed in many biological applications that use sequence data as input, for example, predicting protein function from sequence (Liu, 2017), predicting protein secondary structure from sequence, and protein subcellular localisation (V. I. Jurtz *et al.*, 2017).

2.2. Model inputs

2.2.1 Representing sequence data

There are many ways that amino acid sequence data could be represented for use as input to deep learning models. The common representation as a sequence of single-letter characters is convenient for human readability, but is not appropriate for most deep learning models which require a numeric tensor as input.

As discussed in *1.8.3*, one common approach is to one-hot encode the amino acid residues. One-hot encoding represents each amino acid as a vector of length 20, with a value of one in the index position corresponding to that amino acid, and a zero in all other positions (see *Figure 9*). One-hot encoding an entire amino acid sequence produces a matrix of dimensions [sequence length, 20]. Where sequence lengths vary, in order to achieve matrices of uniform size zero-padding is sometimes added in order to force all sequences into a [max sequence length, 20] shape. Zero-padding refers to adding vectors of all zeros until the desired length is achieved. The drawback of one-hot encoding is that it omits a lot of potentially pertinent information. It serves only to identify each amino acid as being distinct from the others but provides no input about the properties of the amino acids or their relationship to each other.

An alternative strategy that addresses some of the limits of one-hot encoding is to replace the one-hot vectors with something that encodes more information about the amino acid. For example, replacing the one-hot vector with the row corresponding to that amino acid from a substitution matrix like one of the BLOSUM matrices, as described in *1.8.3* and illustrated in *Figure 10.* This is the approach taken by (Jokinen *et al.*, 2019; Jurtz *et al.*, 2018) and is similar to (Dash *et al.*, 2017) except that the substitution vector is preserved rather than used to produce scoring. This approach has the benefit of encoding additional information about the amino acids. By including the BLOSUM data quantifying the relative substitutability of the amino acids, the model may be able to learn something about the physicochemical properties of the amino acids and therefore generalise better.

Yet another approach to the problem is to learn a vector representation of the sequence or the constituent amino acids. This is called a 'learned embedding' or simply 'embedding' and there are a few approaches which have been advanced in this area. Firstly, exploiting techniques that were highly successful in the Natural Language Processing domain, (Asgari and Mofrad, 2015) proposed a ProtVec model as an analogous approach to the popular Word2Vec model (Mikolov *et al.*, 2013). In the ProtVec model, input sequences are decomposed into k-mers and a learned embedding of k-mers is trained using a skip-gram model (Mikolov *et al.*, 2013). A skip-gram model is one that attempts to predict surrounding context words given an input word. In the case of ProtVec, the skip-gram model is trained to predict surrounding k-mers given an input k-mer. The vector representation of the input k-mer in the hidden layer of this trained skip-gram model is the real output of interest and is the learned embedding. Though this type of model could be trained a number of ways, the ProtVec implementation used trimers (k-mers of length k=3) and decomposed each input sequence into three trimer arrays, with frame shifting. For example, the sequence MAFSAEDVLK would be decomposed into the following three trimer arrays: ['MAF', 'SAE', 'DVL'], ['AFS', 'AED', 'VLK'], and ['FSA', 'EDV']. This is sometimes referred to as ‘sequence splitting with overlap’ and is done in order to ensure that all residues are captured and all possible k-mers are included.

(Kimothi *et al.*, 2016) built on this work and proposed Seq2Vec, an analogous approach to the Doc2Vec model. Just as Doc2Vec seeks to embed whole documents rather than words, Seq2Vec seeks to embed whole sequences rather than k-mers.

An ancillary benefit of using learned embeddings is that they can represent sequences of varying length as vectors within the same embedding space, thereby translating from character-encoded, variable-length inputs to fixed-length numeric vector representations. This format is a strict requirement of most deep learning models.

2.3. Hyperparameter optimisation

One challenge with training deep learning models is that there are many hyperparameters involved. By definition, hyperparameters are those parameters which specify the model and training which are not learned via training, for example the number of nodes in a hidden layer, or the learning rate α. Whatever model architecture is chosen, there are a number of hyperparameters which can have a substantial impact on the efficiency and accuracy of the model.

Whereas previously the choice of hyperparameter values may have been largely informed by domain knowledge and rules of thumb, in recent times it has become common practice to employ algorithms to optimise these values. The process of discovering the optimal hyperparameter values for a model is called ‘hyperparameter optimisation’. The design of hyperparameter optimisation algorithms is an entire subfield unto itself, and one that has received a great deal of attention over the past decade. What follows is a high-level summary of the types of approaches used by hyperparameter optimisation algorithms. For a current and in-depth review of the state of the field, see (Yu and Zhu, 2020).

2.3.1 Grid Search

Hyperparameters may be categorical, continuous, or integer valued. In a Grid Search, for each of these hyperparameters a finite search space is defined. For a categorical hyperparameter, this might be the exhaustive set or a subset of the categories. For continuous or integer-valued hyperparameters, this is typically a value range, sometimes with a defined step-size (e.g range(0, 20, 5)). The hyperparameter space is thus populated with a 'grid', a finite set of Cartesian coordinates to test. *Figure 14* illustrates this concept for a 2-dimensional hyperparameter space.

The model performance is then tested for each of the hyperparameter value combinations corresponding to the grid coordinates in the hyperparameter space. This performance is tested at each step with a validation data set that was not included in the training. In the following, we will refer to hyperparameter combinations that yield 'the model with the highest accuracy', but equally this could be the model with the lowest error, or that minimises the cost function, as appropriate to the task at hand.

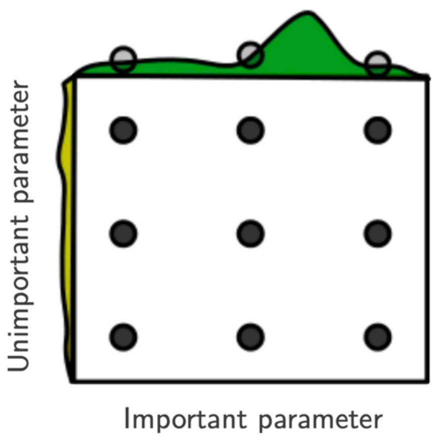


Figure 14: An illustration of hyperparameter optimisation using a Grid Search approach. In this stylised example there are two hyperparameters, one of which has a material impact on model accuracy (important parameter). (adapted from (Bergstra and Bengio, 2012))

The Grid Search approach is embarrassingly parallel (ie. can be parallelised with little to no effort) but suffers from combinatorial explosion of search space as the number of hyperparameters or size of the search spaces increases. As illustrated in *Figure 14*, because Grid Search tests continuous hyperparameters at discrete values, there is no guarantee that it will discover the optimal hyperparameter values.

2.3.2 Random Search

As a principled improvement on Grid Search, the Random Search approach varies by randomly sampling values from each hyperparameter rather than using user-defined discrete values (see *Figure 15*).

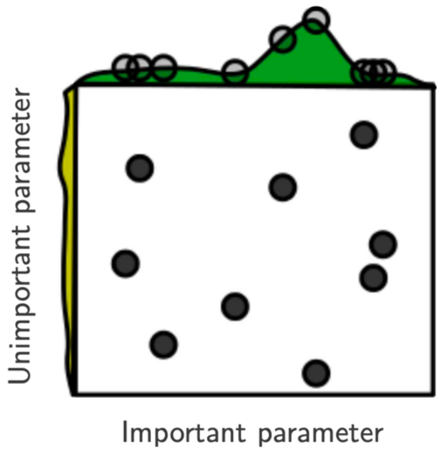


Figure 15: An illustration of hyperparameter optimisation using a Random Search approach. In this stylised example there are two hyperparameters, one of which has a material impact on model accuracy (important parameter). (adapted from (Bergstra and Bengio, 2012)

After randomly sampling from each hyperparameter to create a finite Cartesian grid of coordinates to test, Random Search likewise tests the model performance for each combination of hyperparameter values against a validation dataset.

Like Grid Search, Random Search is embarrassingly parallel, but suffers from the same combinatorial explosion of search space or 'curse of dimensionality'. Random Search has been a successful though computationally expensive approach. While it is reasonable for small hyperparameter spaces, more efficient methods exist for higher dimensional cases. For discussion of the theory behind Random Search algorithms and various implementations, see (Bergstra and Bengio, 2012).

2.3.3 Bayesian Optimisation

In order to reduce the time wasted on unlikely combinations in the hyperparameter space, Bayesian methods can be employed. Bayesian optimisation typically assumes that the unknown function (mapping hyperparameter values to the objective of interest) was sampled from a Gaussian Process, and during iteration a posterior distribution for this function is updated as model accuracy is tested with different hyperparameter combinations. There are many variations on how this is implemented in practice. For more details on Bayesian approaches see (Snoek *et al.*, 2012).

In principle, Bayesian algorithms seek to minimise the number of trials required to explore the search space, but consequently they are sequential, making them difficult to parallelise. While very successful in low dimensional spaces ($d<10$), it is computationally difficult to scale to high-dimensional problems as the computational complexity scales as $O(d^3)$ (see (Kandasamy *et al.*, 2016)).

2.3.4 Adaptive Resource Allocation

Adaptive Resource Allocation methods (sometimes called 'bandit algorithms' in reference to the multi-armed bandit problem) seek to focus resources on promising hyperparameter combinations. In deep learning applications those resources typically refer to training epochs, though in some instances they could instead refer to the size of the training data in terms of the number of records, number of input features, or batch size. Batch size in this context refers to the number of records processed before the model weights are updated during training.

Figure 16 illustrates an example of adaptive resource allocation in action. The figure illustrates an example of the 'successive halving' approach where every n training epochs (in this case n=5) the bottom performing 50% of models are abandoned and training continues for the remaining top 50% until only one model remains.

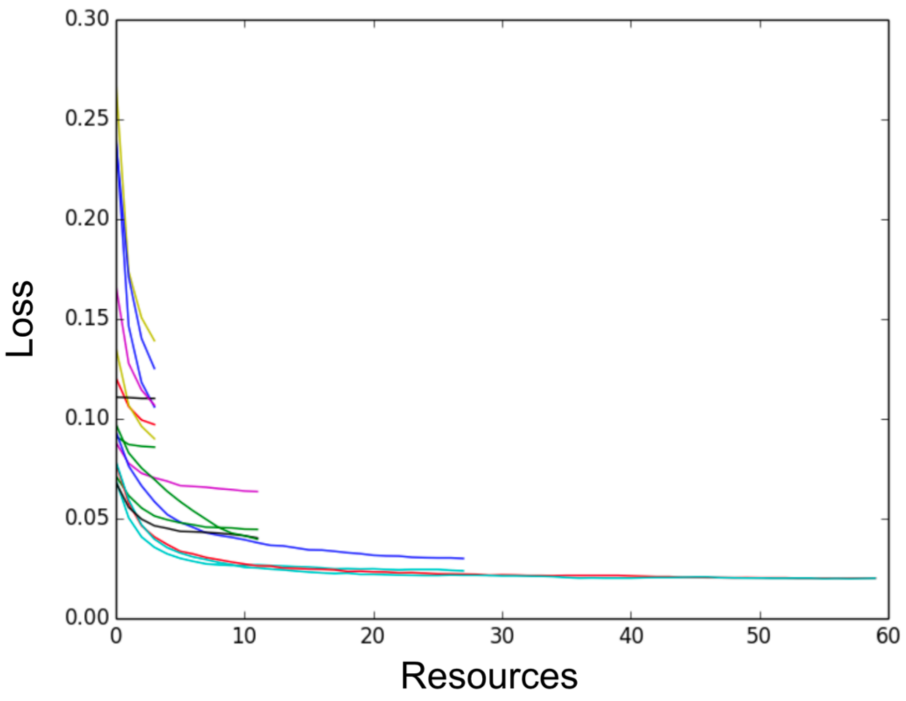


Figure 16: An example of the 'successive halving' approach, which illustrates early stopping of poor-performing models and resource allocation to high-performing models. In this example, every 5 epochs the bottom-performing 50% of models are abandoned. (adapted from (Li et al.*, 2018)*

Though there are adaptive resource allocation algorithms that use Bayesian optimisation, due to the aforementioned complexity of scaling Bayesian optimisation to high-dimensional hyperparameter spaces, methods like Hyperband have seen more frequent use. Hyperband is an infinite-armed bandit algorithm that uses adaptive resource allocation to exponentially increase the number of hyperparameter sets which can be tested from the random search hyperparameter space (Li *et al.*, 2018).

2.3.5 Evolutionary (population-based)

Evolutionary algorithms also rely on eliminating poor-performing models at intervals, but in addition to this they also add new models with a process inspired by biological evolution. The process begins with a set of models generated from combinations of randomly initialised hyperparameter values. These models are trained in parallel and go through an iterative process. At each iteration, as before, some proportion of worst-performing models are eliminated. However, with Evolutionary algorithms, at each iteration new hyperparameter sets are also generated. These new hyperparameter sets are ‘children’ of the most successful hyperparameter sets from the previous iteration, with crossover of particular hyperparameter values and some amount of random mutation. This process is then repeated for many iterations.

Following this methodology with early-stopping, evaluating every n epochs, is called Population-Based Training (PBT) and is the most common methodology in use today see (Jaderberg *et al.*, 2017).

2.4. Software

There are a number of software frameworks available for the development of deep learning algorithms. In choosing a framework, there are some salient criteria to bear in mind:

2.4.1 Open source

Choosing an open source deep learning framework helps to ensure broad reproducibility since the software is freely available to anyone at no cost.

2.4.2 Actively developed

For longevity and ongoing reproducibility, it is advisable to choose a deep learning framework that is still actively developed. Those that are not actively developed are at risk of being abandoned and gradually becoming incompatible with critical dependencies.

2.4.3 Scalability

Scalability is often a consideration when choosing a deep learning framework. Parallelisation can drastically improve the ability to scale with increasing amounts of input data (data parallel), and to scale the number of model variations which can be tested for hyperparameter tuning (model parallel). In modern deep learning frameworks, a common approach to providing parallelisation is to offer GPU support. This GPU support most often comes via the CUDA (Compute Unified Device Architecture) parallel computing platform from Nvidia. This kind of GPU parallelisation has been shown to provide 10x to 50x improvements in speed for biological applications (Boyer *et al.*, 2009). An ancillary benefit of such speedups is that they make models with greater network depth practicable.

Software	Creator	Initial Release	Software License	Written In	Interface	Actively Developed
Caffe	Berkeley Vision and Learning Center	2013	BSD	C++	Python MATLAB C++	No
Deeplearning4j	Adam Gibson	2014	Apache 2.0	C++ Java	Java Scala Clojure Python (Keras) Kotlin	Yes
Keras	François Chollet	2015	MIT license	Python	Python R	Yes
CNTK	Microsoft Research	2016	MIT license	C++	Python (Keras) C++ Command line	No
Apache MXNet	Apache Software Foundation	2015	Apache 2.0	C++	C++ Python Julia Matlab JavaScript Go R Scala Perl Clojure	Yes
PyTorch	A. Paszke S. Gross S. Chintala G. Chanan (Facebook)	2016	BSD	Python C C++ CUDA	Python C++ Julia	Yes
TensorFlow	Google Brain	2015	Apache 2.0	C++ Python CUDA	Python (Keras) C/C++ Java Go JavaScript R Julia Swift	Yes
Theano	Université de Montréal	2007	BSD	Python	Python (Keras)	No

Table 4: Popular deep learning frameworks meeting the base criteria.
Base criteria: open source, supports CUDA, supports parallel execution.
(adapted from https://en.wikipedia.org/wiki/Comparison_of_deep-learning_software)

3. Materials and Methods

For this work the Tensorflow framework (Abadi *et al.*, 2016) was selected. More specifically, Tensorflow 2 was chosen, which is a significant departure from Tensorflow 1.0. From version 2.0 the Keras project was integrated directly into Tensorflow and, for many people, became the primary API for using the framework. The Keras API is a high-level interface that allows for increased productivity by abstracting away many implementation details. Where more direct control is required, the Tensorflow computation graph can still be directly modified.

Tensorflow also offers substantial scalability. The framework supports CPU, GPU, and TPU (Tensorflow Processing Units) acceleration with minimal code changes to implement. In addition, Tensorflow offers distributed training options for parallelising training across multiple GPUs/TPUs on a single machine or multiple machines in a cluster. Distributed processing allows for data-parallel or model-parallel processing of the computation graph. Data-parallel processing allows for training of models on data sets which are too large to fit on any single machine. Model-parallel processing allows for training multiple models concurrently, which allows for much more experimentation with alternative model architectures and enables a much broader search space for hyperparameter optimisation.

In addition to using the Tensorflow framework for model development, this work employed a number of Python packages including Pandas, Numpy, Scikit-learn, and Gensim for data acquisition and preparation. The entire pipeline for this model was laid out in a series of sequential Jupyter notebooks for exposition purposes and is available as a github repository at github.com/justin-barton/tcr-pmhc .

3.1. Data sources

The positive training data for this model comprises 24,037 examples of TCR-pMHC complexes. That is, TCR and epitope pairs which are experimentally confirmed to bind. In order to create the most comprehensive training set possible, this data was collated from VDJdb (Shugay *et al.*, 2018), IEDB (Vita *et al.*, 2019), McPAS-TCR (Tickotsky *et al.*, 2017), and TBAdb (Zhang *et al.*, 2020) (see *Table 1*).

3.2. Data transformation and filtering

Prior to collation, data was filtered to only include examples where the host organism is human. This was in part to make the model useful to applications in human health, and in part because there is very little data available for any other species. Data was further filtered to only include TCR-pMHC examples which contained amino acid sequences for the CDR3 regions of the α and β chains, as well as the amino acid sequence of the epitope.

Data was also filtered to only contain CD8+ TCR-pMHC complexes, filtering out all CD4+ and MHC II complexes. This was done due to the difference in binding dynamics and antigen presentation. It may be possible to train a model that would encompass both CD8+ and CD4+ binding dynamics, but this was not a goal of this work due to the paucity of available data for CD4+.

Finally, CDR3 sequence correction was applied consistent with the methods outlined in (Shugay *et al.*, 2018) in order to address issues with truncation and ensure consistency with the canonical CDR3 format.

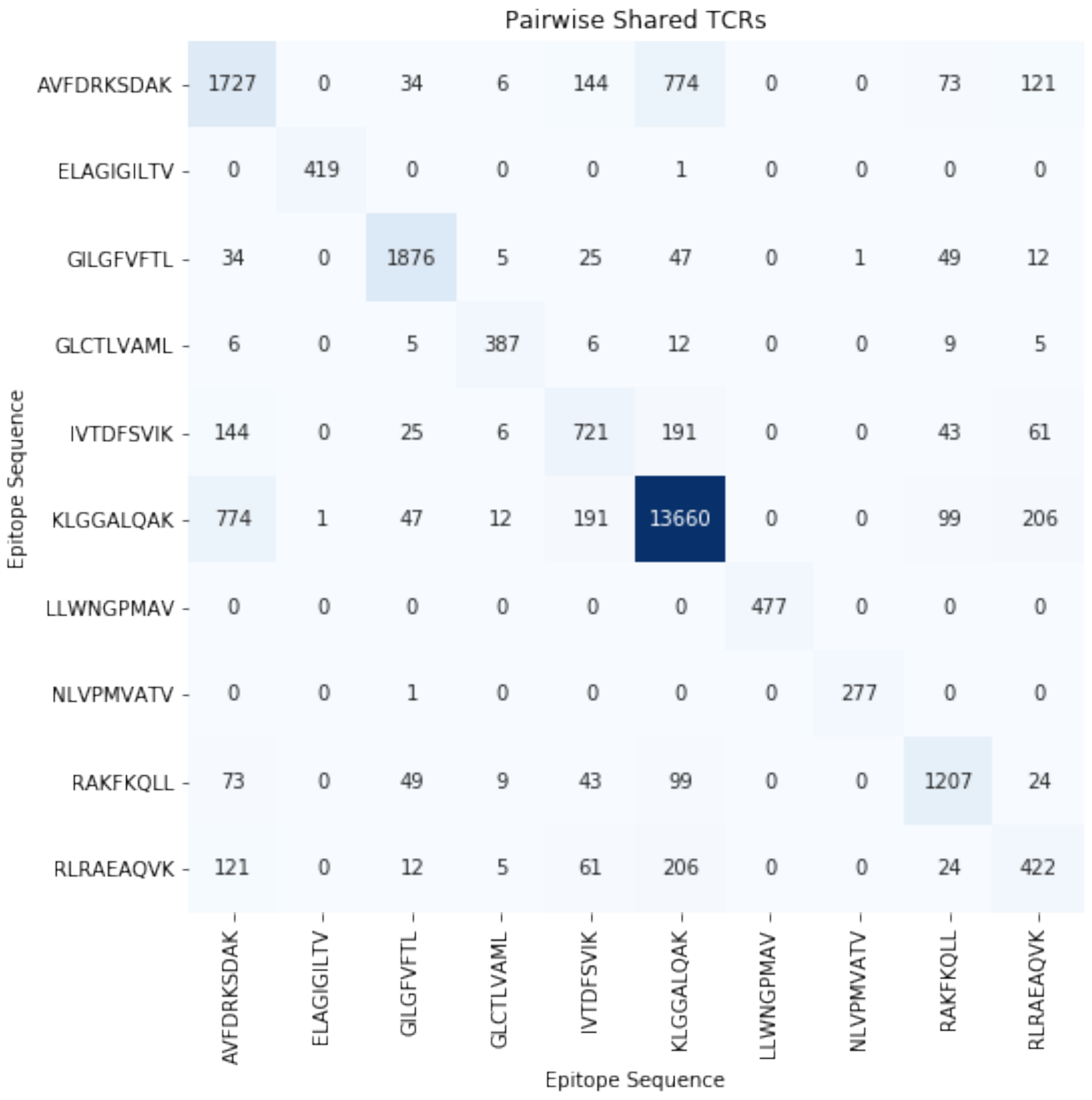
3.2.1 Properties of the data

	IEDB	McPAS-TCR	TBAdb	VDJdb
IEDB	2,590	453	234	580
McPAS-TCR	453	1,061	666	709
TBAdb	234	666	824	495
VDJdb	580	709	495	21,552

Table 5: A pairwise comparison of the count of unique TCR-pMHC complexes found in each of the databases used for this work.

After transformation and filtering, there were 24,037 unique positive training examples remaining. *Table 5* shows which databases these examples originated from, and the overlap between those databases. As *Table 5* shows, the vast majority of positive training examples came from VDJdb, however adding the other databases did provide a significant number of additional positive training examples. For example, of the 2,590 examples taken from IEDB, only 580 were found in the VDJdb data, yielding 2,010 net new positive training examples.

A question that is central to TCR binding behaviour is the cross-reactivity or degeneracy of individual TCRs. Conversely, for repertoire analysis a key question is just how many TCR clonotypes will bind with a particular antigen, and whether those same TCRs will bind other antigens. This type of behaviour has implications for diagnostics using repertoire sequencing data. To that end, we examined the unique TCR counts for the ten epitopes with the most available data (ie. the highest numbers of associated TCRs) in the dataset. *Figure 17* shows the pairwise intersection counts of unique TCRs for these top ten epitopes.



	AVFDRKSDAK	ELAGIGILTV	GILGFVFTL	GLCTLVAML	IVTDFSVIK	KLGGALQAK	LLWNGPMAV	NLVPMVATV	RAKFKQLL	RLRAEAQVK
AVFDRKSDAK	1727	0	34	6	144	774	0	0	73	121
ELAGIGILTV	0	419	0	0	0	1	0	0	0	0
GILGFVFTL	34	0	1876	5	25	47	0	1	49	12
GLCTLVAML	6	0	5	387	6	12	0	0	9	5
IVTDFSVIK	144	0	25	6	721	191	0	0	43	61
KLGGALQAK	774	1	47	12	191	13660	0	0	99	206
LLWNGPMAV	0	0	0	0	0	0	477	0	0	0
NLVPMVATV	0	0	1	0	0	0	0	277	0	0
RAKFKQLL	73	0	49	9	43	99	0	0	1207	24
RLRAEAQVK	121	0	12	5	61	206	0	0	24	422

Figure 17: Shared TCRs between the top 10 epitopes (by count of associated TCRs) from the collated positive TCR-pMHC data.

This analysis suggests that even epitopes with very low sequence similarity can share a large number of TCRs in common. For example, the epitopes AVFDRKSDAK (Epstein-Barr Virus nuclear antigen 4, http://www.iedb.org/epitope/5316) and KLGGALQAK (Human Cytomegalovirus, http://www.iedb.org/epitope/31883) share 774 unique TCRs in the dataset. Unfortunately, there are no TCR-pMHC structures for these epitopes in the PDB, so we cannot make any comparisons of contact residues or CDR conformations to speculate as to why this overlap might be so high.

3.2.2 Generating negative training examples

In order to train a binary classification model which would predict binding or non-binding of a TCR with a given epitope, it was necessary to assemble a set of negative training examples, since the data collated only contained positive examples of TCR-epitope binding. Given the high degree of cross-reactivity seen in the preceding analysis, it seemed prudent to replicate the approach that Jurtz et al. had taken for NetTCR (Jurtz *et al.*, 2018). Jurtz et al. paired TCR sequences from healthy individuals with self-peptides. This method should create true negative examples in the vast majority of cases. Unfortunately, when the model was trained on this data it proved too simple a task, and accuracy consistently exceeded 99% on validation and testing datasets. We hypothesise that the model had too easily learned to discriminate between pathogenic epitopes and self-epitopes. This remained the case even after interventions to ensure similar distributions in epitope sequence length. Ultimately, this approach was abandoned in favour of the more common approach of generating random combinations of TCR and epitope that were not observed in the positive training set from the TCRs and epitopes found in the positive training set (Dash *et al.*, 2017; Jokinen *et al.*, 2019). This of course comes with the complication that it is possible that some randomly paired TCRs and epitopes will be true positives not observed in the training data. While true, it is extremely infrequent, since any particular epitope-specific TCR is rarely observed (Gielis *et al.*, 2019). The true positives and true negatives in the resulting dataset should far outweigh any false negatives.

3.2.3 Train/evaluate/test split

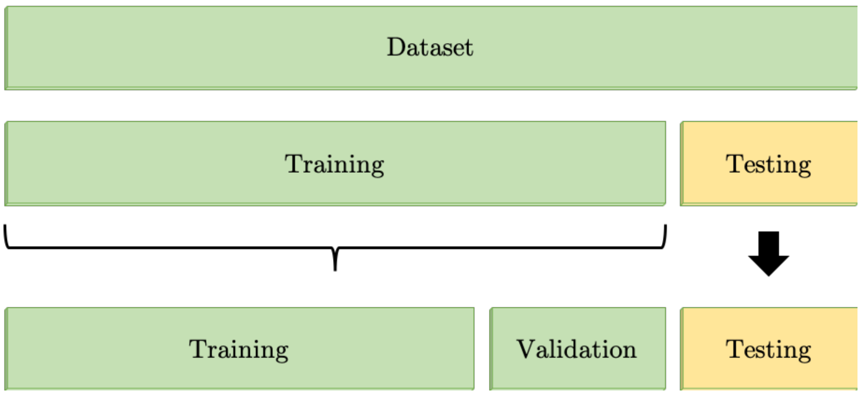


Figure 18: Illustration of the train, validation, test split methodology. The dataset is first randomly divided into 'training' and 'testing' sets with a specified proportion in each. The training set is then further randomly divided into 'training' and 'validation' sets, again with specified proportions.

Once the complete dataset of collated positive and manufactured negative examples had been prepared, it was randomly split into a training dataset and testing dataset as illustrated in *Figure 18*. The testing dataset contained approximately 10% of TCRs and was reserved for evaluating the performance of the completed model. Next the training dataset was randomly split into training and validation datasets, with the validation dataset containing approximately 10% of the TCRs from the training dataset. The purpose of the validation dataset was to provide a holdout that could be used during model development to prevent overfitting during training and for use during hyperparameter optimisation.

Note that while common practice is to randomly split training examples, in this case we chose to randomly split by TCR in order to ensure that the testing and training datasets were mutually exclusive with respect to TCRs. Once a TCR had been randomly assigned to either the training or testing dataset, all training examples which contained that TCR were assigned to the corresponding dataset. The purpose of this was to ensure that none of the TCRs within the testing dataset were also present within the training dataset. This was designed to increase the difficulty and ensure the generalisability of the model to de novo TCRs.

3.3. Feature selection

The input features to the model were selected to be the TCR α chain CDR3 sequence, the TCR β chain CDR3 sequence, the epitope sequence, the HLA allele, and the V and J genes of the α and β chains.

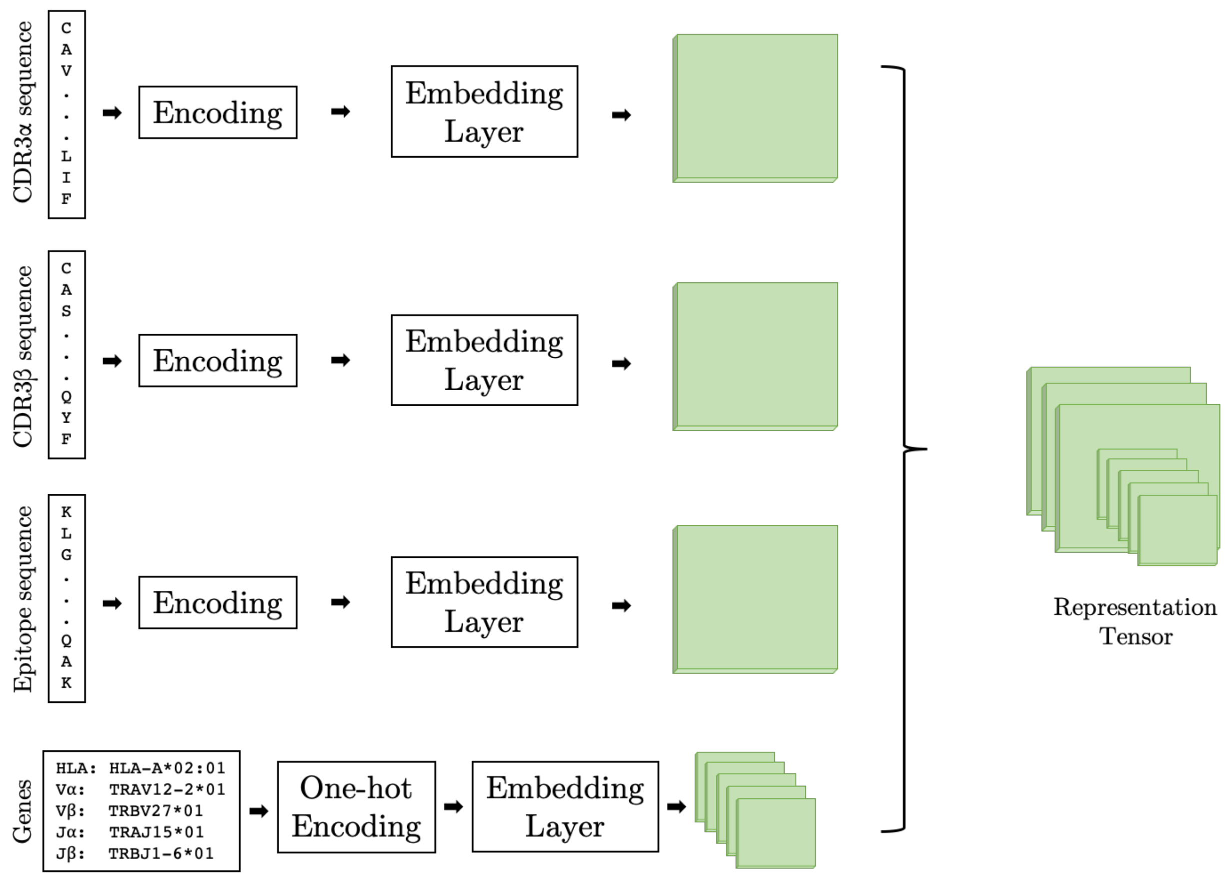
As discussed in *2.2.1*, learned embeddings were chosen to represent sequence data in the model, and trained as described below. As discussed in *1.8.2* and *1.7.2*, structural features were excluded from the model due to insufficient data. Although there is a theoretical basis for each of the input features which were included, the ultimate decisions about which features to include were heavily influenced by the breadth of availability of these features in existing databases and assay methods. This was to ensure a large dataset with which to train the model, and for broader accessibility of the model to future researchers.

3.4. Model

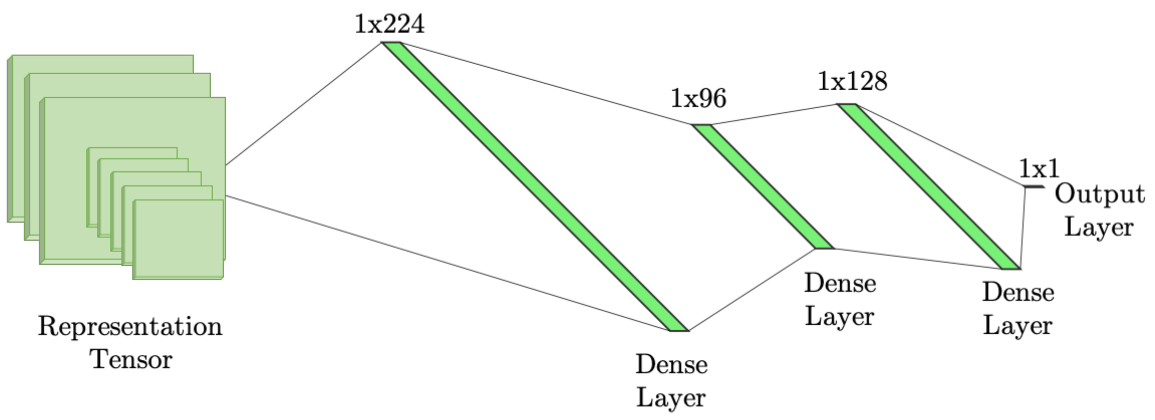
A custom model was built in Tensorflow 2.2.0 using the Keras API. The model is illustrated in *Figure 19*. Conceptually, the model can be divided into two parts: the encoder illustrated in (a) and the decoder illustrated in (b).

The encoder uses embedding techniques to transform the heterogenous input features into a representation tensor. This is important because deep neural networks require numeric-valued tensors of fixed dimensions as input. Each input feature is passed through its own dedicated embedding layer, which produces an n-dimensional vector representation. These feature vectors are then stacked to create the representation tensor. Thus, the embedding approach used here can take inputs of varying length and type and convert them to tensors of uniform dimensions. Because each input feature has its own embedding layer which is trained separately, the embeddings are able to capture the dynamics particular to that specific feature.

The decoder is comprised of three dense layers (of 224, 96, and 128 nodes respectively) with rectified linear unit (ReLU) activation functions, followed by an output layer of a single node with a sigmoid activation function. This output layer serves to output the p-value prediction for the binding of the input TCR and epitope.



(a)



(b)

Figure 19: The deep neural network created for the task of predicting TCR-pMHC binding, illustrated in two parts: (a) The encoder that encodes input amino acid sequences and genes as a numeric, representation tensor. (b) The decoder which transforms the representation tensor into a predicted probability of TCR-pMHC binding.

3.5. Hyperparameter optimisation

The architecture and parameters of the model were not arbitrarily chosen. Hyperparameter optimisation was performed using the Hyperband algorithm (Li *et al.*, 2018) as implemented in the Keras Tuner package (https://keras-team.github.io/keras-tuner/). The hyperparameters optimised included: the number of dense layers, the number of nodes in each dense layer, the number of dimensions in the embedding space each input feature was projected into, the learning rate, and the learning batch size.

The hyperparameters were tuned simultaneously with model training, using the validation AUC of the resultant models as the objective criteria. 'Validation AUC' refers to the AUC statistic achieved by the model when evaluated against the validation dataset. This objective was chosen in order to maximise predictive accuracy whilst preventing overfitting to the training dataset.

In addition to the model architecture seen in *Figure 19 (b)*, the resulting optimised set of hyperparameters included the number of dimensions for the embedding space of each of the sequence and gene input features. *Table 6* provides a summary of the optimised number of dimensions for each input feature.

Feature	Number of Dimensions
CDR3α sequence	171
CDR3β sequence	191
Epitope sequence	177
HLA allele	226
α chain V gene	235
α chain J gene	86
β chain V gene	81
β chain J gene	157

Table 6: The optimised number of dimensions in the embedding space for each of the model input features (after hyperparameter optimisation with Hyperband).

4. Results

After completing hyperparameter optimisation, the resulting model (illustrated in *Figure 19*) was evaluated against the testing dataset. Against the testing dataset the model achieved an accuracy of 0.9859 and an AUC of 0.9944.

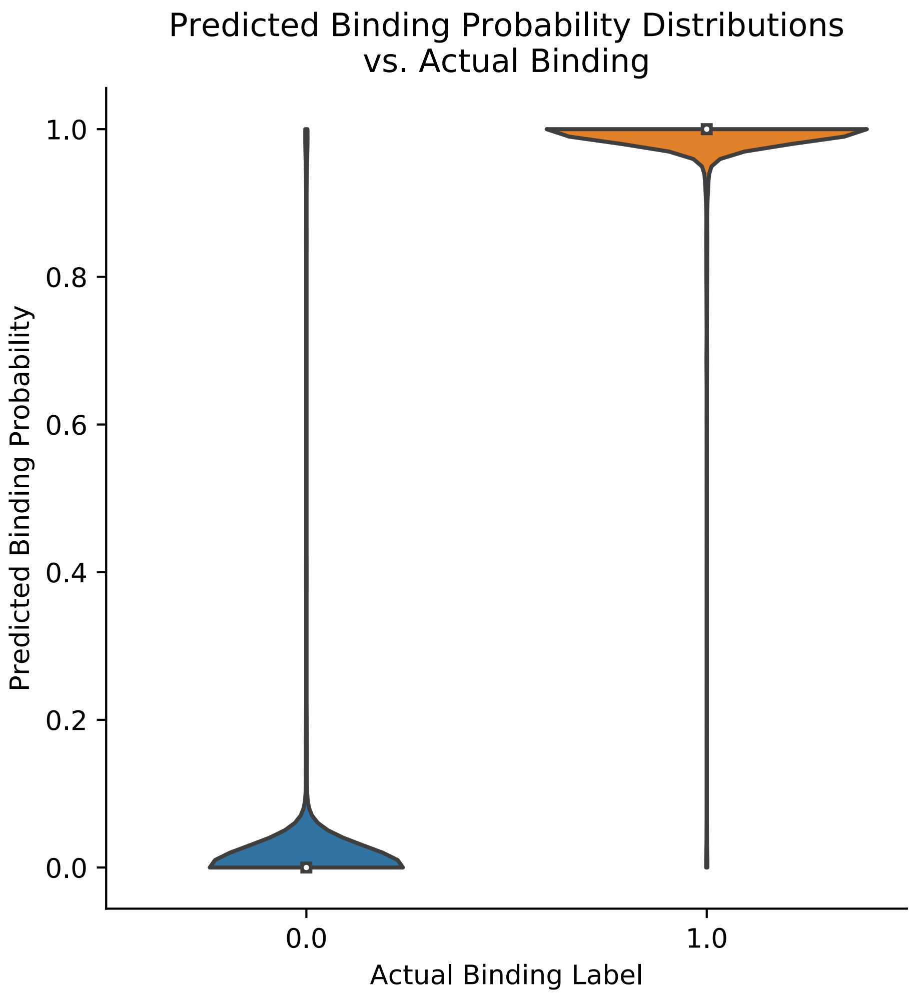


Figure 20: A kernel density plot of the distribution of predicted binding probability for non-binding (0.0) and binding (1.0) TCR-epitope pairs in the testing dataset.

As well as achieving a high accuracy, the predicted probabilities output by the model seem to correlate well with the ground truth labels. As *Figure 20* shows, the predicted p-values for most binding TCR-epitope pairs are close to 1.0, and for most non-binding TCR-epitope pairs are close to 0.0.

As can be seen in *Figure 17*, some epitopes have received more research attention than others and so have a disproportionate number of known binding TCRs in the dataset. Because of this, a model that performs well for an overrepresented epitope may perform poorly on most other epitopes and still achieve a reasonable aggregate accuracy. In order to confirm that the model generalises well across epitopes (and has not specialised in epitopes that are overrepresented), error analysis was performed. Accuracy was computed for each epitope in the testing dataset. Summarising that data, *Figure 21* shows a kernel density plot of the model accuracy by epitope.

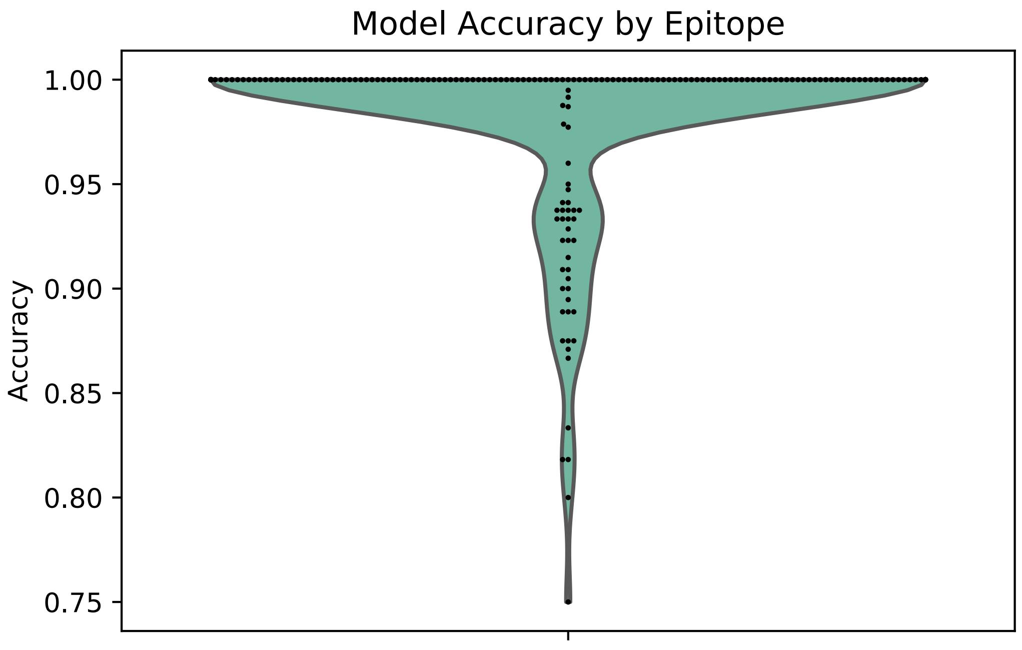


Figure 21: A kernel density plot of accuracy scores by epitope, overlaid with a swarm plot of the same. In the swarm plot, each dot represents a single epitope, with horizontal offset added to ensure that no points overlap. Note that there are 211 epitopes in the testing dataset and 167 of those epitopes lie at accuracy=1.00)

From *Figure 21* we can see that the accuracy of the model predictions for most of the epitopes in the test dataset (167/211) is 1.00. Only 4 epitopes have an accuracy below 0.85, and the minimum accuracy is 0.75. In sum, the model generalises well across epitopes and the high overall accuracy of the model is not an artefact of the imbalanced dataset with respect to epitopes.

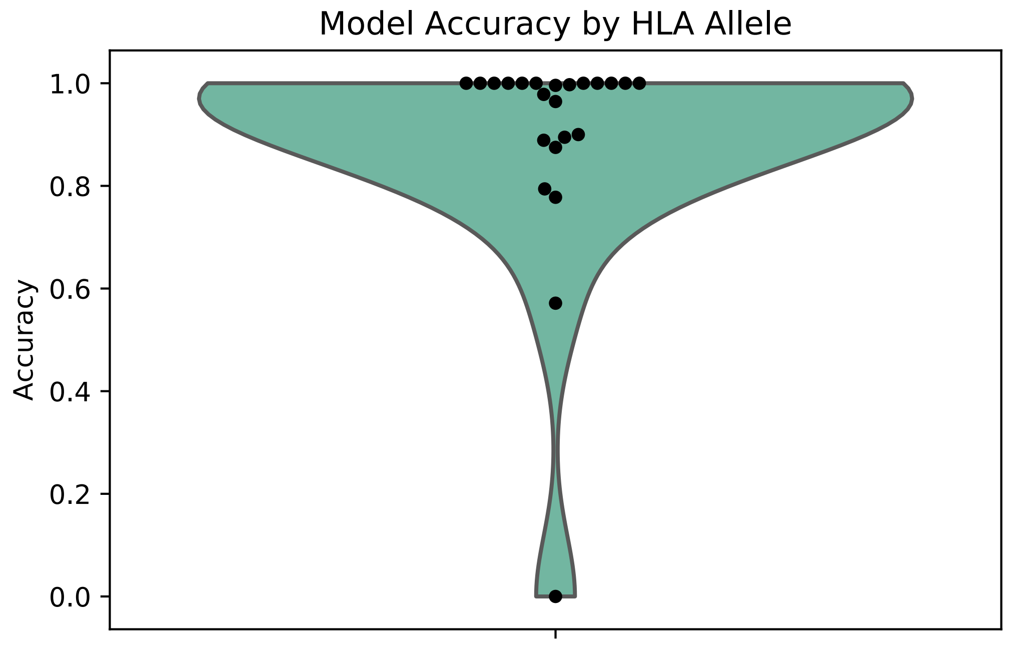


Figure 22: A kernel density plot of accuracy scores by HLA allele, overlaid with a swarm plot of the same. In the swarm plot, each dot represents an HLA allele, with horizontal offset added to ensure that no points overlap.

Similarly, in *Figure 22* we can see that the model accuracy is quite strong across almost all HLA alleles in the test set. There are two HLA alleles, however, for which the model does not perform well: HLA-B*44:05 and HLA-B*44:05:01 (which are synonymous mutations). As *Figure 23* shows, this is due to insufficient training data. In *Figure 23,* we can see that the model accuracy is strong for HLA alleles until the number of available training examples for an HLA allele falls below 15. One possible explanation for why the model performs so poorly with insufficient training data for this particular HLA allele is that its pMHC complexes have unusually high conformational plasticity, leading to atypical binding dynamics (recall the dynamic discussed in *1.4* and illustrated in *Figure 4(e)*) (Fodor *et al.*, 2018). More generally, the results suggest that binding dynamics vary considerably for some HLA alleles, much more so than by epitope or TCR.

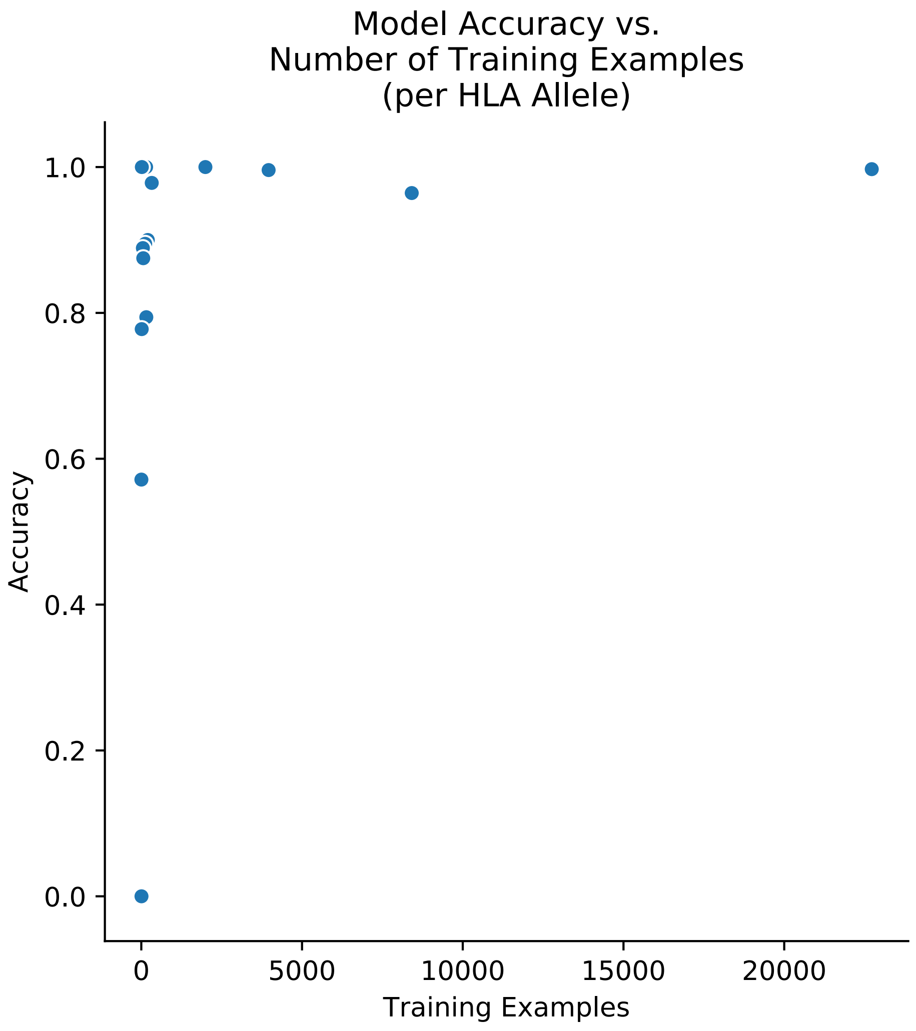


Figure 23: A scatterplot of model accuracy vs. number of training examples for each HLA allele in the testing dataset. Note that accuracy is quite high until the number of available training examples falls below 15.

5. Discussion and Conclusions

5.1. Results vs. objectives

The primary aim of the project was to lay the groundwork for a usable pipeline for in silico prediction of TCR-epitope interaction that would support pathogen diagnosis and immunotherapy development. The model that has been developed here certainly achieves that goal and has promise for clinical applications.

A secondary goal of this work was to establish through experimentation performant feature representations and model architectures for deep learning approaches to the problem of TCR-epitope binding. Through rigorous experimentation and a wide breadth of alternatives tested via hyperparameter optimisation, this has likewise been achieved.

Finally, an ambition of this work was to develop a model that is scalable and applicable to novel TCRs. The model developed here has been built on an architecture that can be parallelised to increase performance and easily scale to petabytes of training data. As seen in *Figure 20*, the model performs strongly on *de novo* TCRs. As seen in *Figure 21*, the model generalises well across epitopes.

5.2. Comparative performance

A number of projects have been developed with the aim of applying machine learning techniques to predicting TCR-epitope specificity (see *Table 2*). It was an early ambition of this work to comprehensively benchmark the comparative performance of these projects with the model developed here. Unfortunately, this proved impractical for a number of reasons.

Firstly, NetTCR only provides binding predictions for HLA allele HLA-A*02:01, which represents ~21% of the dataset, or 982 examples in the testing dataset (Jurtz *et al.*, 2018). TCRex only has models for 50 different epitopes (43 at the time of publication), which collectively represent ~21% of the dataset, or 1015 examples in the testing dataset (Gielis *et al.*, 2018). TCRGP suffers from a similar problem. Because TCRGP likewise trains a separate model per epitope, prediction is limited in scope to epitopes for which there is enough data to train a model. In the TCRGP paper, the set of epitopes were limited to those having at least 50 associated TCRs in VDJDB, of which there were 22 (Jokinen *et al.*, 2019). These 22 epitopes represent ~16% of the dataset or 761 examples in the testing dataset. Constructing a benchmarking dataset that contained only HLA allele HLA-A*02:01, the subset of epitopes supported by both TCRex and TCRGP, and examples not used in the training of these three models only left 654 examples, which was not enough to establish the performance differences between these models with any reasonable confidence.

While a robust comparison of model accuracy was not possible, there are some marked differences which are worth highlighting. Firstly, our model is able to make accurate predictions across all HLA alleles for which there is a meager amount of training data (>15 examples, see *Figure 22*). Secondly, our model is able to make accurate predictions even for epitopes for which there is little or no experimental data (see *Figure 21*). Finally, our model is able to make accurate predictions for previously unseen TCRs (see *Figure 20*). In these ways, the model presented here represents a significant advance on the previous state of the art, and suggests that we may have successfully captured some of the underlying biological dynamics.

5.3. Further work

The biological dynamics represented in this model have some limitations in scope which it might be desirable to address in future work. As noted in *3.2*, the model training excluded data for CD4+ 'helper' T cells and their associated MHC II antigen-presenting cells due to a lack of available data. There may be data sources not explored in this work which could open this class up to further investigation.

Similarly, due to the data sources used for this model, the training data only contained linear epitopes. Though an edge case for T cells, there are discontinuous epitopes which are known to bind with CD8+ T cells and may be relevant for producing highly immunogenic synthetic vaccines (Wu *et al.*, 2016). The ability to model discontinuous epitopes would also be an important precondition for adapting this type of model to B cell specificity, since an estimated 85% to 88% of B cell epitopes are discontinuous (Sivalingam and Shepherd, 2012).

As discussed in *1.8.2*, structural information was not included in this model. However, recent work has been published which attempts to model, from sequence, multiple possible structural conformations of a given TCR, using observed conformations of CDRs similar to the component CDRs of the TCR of interest (Wong *et al.*, 2020). It is worth exploring whether including information about known CDR conformations could improve models of TCR-epitope binding.

Though the model presented here makes predictions for TCR-epitope pairs, it could easily be extended to applications with repertoire sequencing (rep-seq) data. For example, all TCRs in the rep-seq data could be scored against a full panel or database of known epitopes. This would of course require a large number of combinations to be scored, but the model is performant enough to score large inputs quickly. By way of example, the testing dataset of 4,748 TCR-epitope pairs were scored in ~500ms running in a single-threaded process running on CPU. This could be further improved with

GPU processing and could be parallelised to score petabytes of rep-seq data if required. Once scored against the database of epitopes, positive predictions could be aggregated to report relative abundance by epitope or antigen.

Alternatively, the model architecture could be restructured slightly. As has been done in many transfer learning applications, the current output layer and epitope input could be removed, and the model retrained to predict antigens directly with a new output layer of a softmax classifier.

These are just two examples of how the performance and flexibility of this model provides ample opportunities for applications in diagnostics. Equally the model could be adapted to other domains, such as identifying candidate immunotherapies and understanding potential adverse cross-reactivity issues with immunotherapies.

6. References

Abadi,M. *et al.* (2016) TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467 [cs].*

Asgari,E. and Mofrad,M.R.K. (2015) Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics. *PLOS ONE*, **10**, e0141287.

Baker,B.M. *et al.* (2012) Structural and dynamic control of T-cell receptor specificity, cross-reactivity, and binding mechanism. *Immunological Reviews*, **250**, 10–31.

Bergstra,J. and Bengio,Y. (2012) Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, **13**, 281–305.

Birnbaum,M.E. *et al.* (2014) Deconstructing the peptide-MHC specificity of T cell recognition. *Cell*, **157**, 1073–1087.

Borbulevych,O.Y. *et al.* (2009) T Cell Receptor Cross-reactivity Directed by Antigen-Dependent Tuning of Peptide-MHC Molecular Flexibility. *Immunity*, **31**, 885–896.

Boyer,M. *et al.* (2009) Accelerating leukocyte tracking using CUDA: A case study in leveraging manycore coprocessors. In, *2009 IEEE International Symposium on Parallel Distributed Processing.*, pp. 1–12.

Dash,P. *et al.* (2017) Quantifiable predictive features define epitope specific T cell receptor repertoires. *Nature*, **547**, 89–93.

DeWitt,W.S. *et al.* (2018) Human T cell receptor occurrence patterns encode immune history, genetic background, and receptor specificity. *eLife*, **7**.

Fischer,D.S. *et al.* (2019) Predicting antigen-specificity of single T-cells based on TCR CDR3 regions. *bioRxiv*, 734053.

Fodor,J. *et al.* (2018) Previously Hidden Dynamics at the TCR–Peptide–MHC Interface Revealed. *The Journal of Immunology*, **200**, 4134–4145.

Gielis,S. *et al.* (2019) Detection of Enriched T Cell Epitope Specificity in Full T Cell Receptor Sequence Repertoires. *Front Immunol*, **10**.

Gielis,S. *et al.* (2018) TCRex: a webtool for the prediction of T-cell receptor sequence epitope specificity. *bioRxiv*, 373472.

Glanville,J. *et al.* (2017) Identifying specificity groups in the T cell receptor repertoire. *Nature*, **547**, 94–98.

Gonzalez-Galarza,F.F. *et al.* (2020) Allele frequency net database (AFND) 2020 update: gold-standard data classification, open access genotype data and new query tools. *Nucleic Acids Res*, **48**, D783–D788.

Gowthaman,R. and Pierce,B.G. (2018) TCRmodel: high resolution modeling of T cell receptors from sequence. *Nucleic Acids Res*, **46**, W396–W401.

Gray,J.J. *et al.* (2003) Protein–Protein Docking with Simultaneous Optimization of Rigid-body Displacement and Side-chain Conformations. *Journal of Molecular Biology*, **331**, 281–299.

Guo,Y. and Chen,X. (2019) A deep learning framework for improving protein interaction prediction using sequence properties. *bioRxiv*, 843755.

Hochreiter,S. and Schmidhuber,J. (1997) Long Short-Term Memory. *Neural Computation*, **9**, 1735–1780.

Jaderberg,M. *et al.* (2017) Population Based Training of Neural Networks. *arXiv:1711.09846 [cs].*

Jokinen,E. *et al.* (2019) Determining epitope specificity of T cell receptors with TCRGP. *bioRxiv*, 542332.

Jurtz,V. *et al.* (2017) NetMHCpan-4.0: Improved Peptide-MHC Class I Interaction Predictions Integrating Eluted Ligand and Peptide Binding Affinity Data. *J. Immunol.*, **199**, 3360–3368.

Jurtz,V.I. *et al.* (2017) An introduction to deep learning on biological sequence data: examples and solutions. *Bioinformatics*, **33**, 3685–3690.

Jurtz,V.I. *et al.* (2018) NetTCR: sequence-based prediction of TCR binding to peptide-MHC complexes using convolutional neural networks. *bioRxiv*, 433706.

Kandasamy,K. *et al.* (2016) High Dimensional Bayesian Optimisation and Bandits via Additive Models. *arXiv:1503.01673 [cs, stat].*

Kawashima,S. and Kanehisa,M. (2000) AAindex: Amino Acid index database. *Nucleic Acids Res*, **28**, 374.

Kim,Y. *et al.* (2019) A computational framework for deep learning-based epitope prediction by using structure and sequence information. In, *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM).*, pp. 1208–1210.

Kimothi,D. *et al.* (2016) Distributed Representations for Biological Sequence Analysis. *arXiv:1608.05949 [cs, q-bio].*

Klausen,M.S. *et al.* (2015) LYRA, a webserver for lymphocyte receptor structural modeling. *Nucleic Acids Res*, **43**, W349–W355.

Lanzarotti,E. *et al.* (2019) T-Cell Receptor Cognate Target Prediction Based on Paired α and β Chain Sequence and Structural CDR Loop Similarities. *Frontiers in Immunology*, **10**.

LeCun, Y.,L. *et al.* (1990) Handwritten Digit Recognition with a Back-Propagation Network. In, *Advances in Neural Information Processing Systems*. Morgan Kaufmann, pp. 396–404.

Li,L. *et al.* (2018) Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *arXiv:1603.06560 [cs, stat].*

Liu,W. *et al.* (2017) A survey of deep neural network architectures and their applications. *Neurocomputing*, **234**, 11–26.

Liu,X. (2017) Deep Recurrent Neural Network for Protein Function Prediction from Sequence. *arXiv:1701.08318 [cs, q-bio, stat]*.

Madi,A. *et al.* (2017) T cell receptor repertoires of mice and humans are clustered in similarity networks around conserved public CDR3 sequences. *eLife*, **6**.

Mann,S.E. *et al.* (2020) Multiplex T Cell Stimulation Assay Utilizing a T Cell Activation Reporter-Based Detection System. *Front. Immunol.*, **11**.

Mendes,M.F.A. *et al.* (2015) Improved structural method for T-cell cross-reactivity prediction. *Molecular Immunology*, **67**, 303–310.

Meysman,P. *et al.* (2019) On the viability of unsupervised T-cell receptor sequence clustering for epitope preference. *Bioinformatics*, **35**, 1461–1468.

Mikolov,T. *et al.* (2013) Distributed Representations of Words and Phrases and their Compositionality. In, Burges,C.J.C. *et al.* (eds), *Advances in Neural Information Processing Systems 26.* Curran Associates, Inc., pp. 3111–3119.

Mösch,A. *et al.* (2019) Machine Learning for Cancer Immunotherapies Based on Epitope Recognition by T Cell Receptors. *Front Genet*, **10**.

Pettersen,E.F. *et al.* (2004) UCSF Chimera--a visualization system for exploratory research and analysis. *J Comput Chem*, **25**, 1605–1612.

Pierce,B. and Weng,Z. (2008) A Combination of Rescoring and Refinement Significantly Improves Protein Docking Performance. *Proteins*, **72**, 270–279.

Pierce,B.G. and Weng,Z. (2013) A flexible docking approach for prediction of T cell receptor–peptide–MHC complexes. *Protein Sci*, **22**, 35–46.

Pogorelyy,M.V. *et al.* (2019) Detecting T cell receptors involved in immune responses from single repertoire snapshots. *PLOS Biology*, **17**, e3000314.

Pogorelyy,M.V. and Shugay,M. (2019) A Framework for Annotation of Antigen Specificities in High-Throughput T-Cell Repertoire Sequencing Studies. *Front. Immunol.*, **10**.

Reynisson,B. *et al.* (2020) 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 Res*, **48**, W449–W454.

Ritvo,P.-G. *et al.* (2018) High-resolution repertoire analysis reveals a major bystander activation of Tfh and Tfr cells. *PNAS*, **115**, 9604–9609.

Rossjohn,J. *et al.* (2015) T Cell Antigen Receptor Recognition of Antigen-Presenting Molecules. *Annu. Rev. Immunol.*, **33**, 169–200.

Rudolph,M.G. *et al.* (2006) How TCRs bind MHCs, peptides, and coreceptors. *Annu. Rev.*

Immunol., **24**, 419–466.

Sanchez-Trincado,J.L. *et al.* (2017) Fundamentals and Methods for T- and B-Cell Epitope Prediction. *Journal of Immunology Research.*

Sewell,A.K. (2012) Why must T cells be cross-reactive? *Nat. Rev. Immunol.*, **12**, 669–677.

Shugay,M. *et al.* (2018) VDJdb: a curated database of T-cell receptor sequences with known antigen specificity. *Nucleic Acids Res*, **46**, D419–D427.

Sivalingam,G.N. and Shepherd,A.J. (2012) An analysis of B-cell epitope discontinuity. *Molecular Immunology*, **51**, 304–309.

Snoek,J. *et al.* (2012) Practical Bayesian Optimization of Machine Learning Algorithms. *arXiv:1206.2944 [cs, stat].*

Sutskever,I. *et al.* (2011) Generating Text with Recurrent Neural Networks. In, *ICML.*

Tickotsky,N. *et al.* (2017) McPAS-TCR: a manually curated catalogue of pathology-associated T cell receptor sequences. *Bioinformatics*, **33**, 2924–2929.

Vita,R. *et al.* (2019) The Immune Epitope Database (IEDB): 2018 update. *Nucleic Acids Res.*, **47**, D339–D343.

Wong,W.K. *et al.* (2020) TCRBuilder: multi-state T-cell receptor structure prediction. *Bioinformatics*, **36**, 3580–3581.

Wooldridge,L. *et al.* (2012) A Single Autoimmune T Cell Receptor Recognizes More Than a Million Different Peptides. *J Biol Chem*, **287**, 1168–1177.

Wu,M. *et al.* (2016) DNA vaccine with discontinuous T-cell epitope insertions into HSP65 scaffold as a potential means to improve immunogenicity of multi-epitope Mycobacterium tuberculosis vaccine. *Microbiol. Immunol.*, **60**, 634–645.

Yin,Y. and Mariuzza,R.A. (2009) The Multiple Mechanisms of T Cell Receptor Cross-reactivity. *Immunity*, **31**, 849–851.

Yu,T. and Zhu,H. (2020) Hyper-Parameter Optimization: A Review of Algorithms and Applications. *arXiv:2003.05689 [cs, stat].*

Zarnitsyna,V.I. *et al.* (2013) Estimating the Diversity, Completeness, and Cross-Reactivity of the T Cell Repertoire. *Front Immunol*, **4**.

Zhang,W. *et al.* (2020) PIRD: Pan Immune Repertoire Database. *Bioinformatics*, **36**, 897–903.

Zvyagin,I.V. *et al.* (2020) An overview of immunoinformatics approaches and databases linking T cell receptor repertoires to their antigen specificity. *Immunogenetics*, **72**, 77–84.