2017

Data Representation in Machine Learning Methods with its Application to Compilation Optimization and Epitope Prediction

Yevgeniy Sher
University of Central Florida

DATA REPRESENTATION IN MACHINE LEARNING METHODS WITH ITS APPLICATION TO COMPILATION OPTIMIZATION AND EPITOPE PREDICTION

by

GENE SHER
M.S. University of Central Florida, 2014
B.S. University of California Santa Cruz, 2007

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2017

Major Professor: Shaojie Zhang

ABSTRACT

In this dissertation we explore the application of machine learning algorithms to compilation phase order optimization, and epitope prediction. The common thread running through these two disparate domains is the type of data being dealt with. In both problem domains we are dealing with categorical data, with its representation playing a significant role in the performance of classification algorithms.

We first present a neuroevolutionary approach which orders optimization phases to generate compiled programs with performance superior to those compiled using LLVM's -O3 optimization level. Performance improvements calculated as the speed of the compiled program's execution ranged from 27% for the ccbench program, to 40.8% for bzip2.

This dissertation then explores the problem of data representation of 3D biological data, such as amino acids. A new approach for distributed representation of 3D biological data through the process of embedding is proposed and explored. Analogously to word embedding, we developed a system that uses atomic and residue coordinates to generate distributed representation for residues, which we call 3D Residue BioVectors. Preliminary results are presented which demonstrate that even the low dimensional 3D Residue BioVectors can be used to predict conformational epitopes and protein-protein interactions, with promising proficiency. The generation of such 3D BioVectors, and the proposed methodology, opens the door for substantial future improvements, and application domains.

The dissertation then explores the problem domain of linear B-Cell epitope prediction. This problem domain deals with predicting epitopes based strictly on the protein sequence. We present the DRREP system, which demonstrates how an ensemble of shallow neural networks can be combined with string kernels and analytical learning algorithm to produce state of the art epitope

prediction results. DRREP was tested on the SARS subsequence, the HIV, Pellequer, AntiJen datasets, and the standard SEQ194 test dataset. AUC improvements achieved over the state of the art ranged from 3% to 8%.

Finally, we present the SEEP epitope classifier, which is a multi-resolution SMV ensemble based classifier which uses conjoint triad feature representation, and produces state of the art classification results. SEEP leverages the domain specific knowledge based protein sequence encoding developed within the protein-protein interaction research domain. Using an ensemble of multi-resolution SVMs, and a sliding window based pre and post processing pipeline, SEEP achieves an AUC of 91.2 on the standard SEQ194 test dataset, a 24% improvement over the state of the art.

ACKNOWLEDGMENTS

I would like to thank Dr. Shaojie Zhang for his support and encouragement throughout my Ph.D study. His passion for research, insuring that all work is tested and done rigorously, and his attention to detail, were essential for the improvement of my research. I also wish to thank my co-adviser Dr. Damian Dechev, whose support, encouragement, and guidance was always of great help during my studies at UCF.

I also wish to thank Dr. Gary Leavens, Dr. Degui Zhi, and Dr. Avelino Gonzalez, for taking time to serve in my dissertation committee and reviewing my dissertation.

Chapter 2 is adopted from **Gene Sher**, Kyle Martin, and Damian Dechev. *Preliminary results for neuroevolutionary optimization phase order generation for static compilation*. Proceedings of the 11th Workshop on Optimizations for DSP and Embedded Systems. ACM, 2014. The dissertation author was primary author of this paper, and was responsible for the research.

Chapter 4 is adopted from **Gene Sher**, Degui Zhi, and Shaojie Zhang. *Deep Ridge Regressed Epitope Predictor* which has been presented at a peer reviewed conference ICIBM 2016, and accepted into the BMC Genomics journal 2017, but not yet published. The dissertation author was primary author of this paper, and was responsible for the research.

Chapter 5 is adopted from material in submission, **Gene Sher**, Shaojie Zhang, Degui Zhi. *SEEP: Conjoint Triad Feature Based Epitope Predictor*. The dissertation author was primary author of this paper, and was responsible for the research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

The field of machine learning concentrates on developing algorithms that give ”computers the ability to learn without being explicitly programmed” [65, 89]. Today, people say ”machine learning” when they refer to any algorithm which has the ability to learn from data and then generalize to make predictions on new previously unseen data. Such algorithms come in vast number of types of functionality, and there are different approaches to their construction. There are machine learning algorithms that are based on and are inspired by biological systems and environments, these methods include the likes of neural networks, evolutionary algorithms, artificial immune systems, and ant colonies. Other approaches are based on statistics, such as the naive bayes classifier. There are strengths and weaknesses associated with each; a balance of speed, computational complexity, ability to generalize, differs for different approaches, and too does their performance on specific problem domains.

Some of the most commonly used approaches are artificial neural networks (ANN), shallow and deep variations, support vector machines (SVM) which some consider to be a variation of shallow neural networks but using a distinctly different learning method, and evolutionary computation. There are of course variations, combinations, and ensembles of these approaches, combined together to produce even better performance than any of the approaches on their own.

There are also different types of learning. The three learning types are: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning requires examples of what the system should learn, an expected output for every input. Unsupervised approach includes algorithms that primarily perform some type of clustering, or dimensionality reduction. Finally, reinforcement learning methods only require the knowledge of whether one solution is better than another, which is enough to provide the system with the needed positive or negative reinforcement

signal to modify its parameters.

The problem domains this dissertation covers, requires reinforcement learning for the compilation phase optimization, and supervised learning for the classification problem of epitope prediction. All the methods and problem domains presented in this dissertation share a common thread, they all have to deal with categorical data, either on the input, the output, or both ends of the machine learning algorithm. Discrete optimization phases have to be represented to the neuroevolutionary system we use to order them, while in bioinformatics we have to deal with amino acids, which too require a manner in which to encode them and represent to the machine learning methods. Both problems require us to represent categorical data in such a way that the machine learning algorithms can perform efficient calculations on it. Furthermore, throughout the entire dissertation, one of the main issues being dealt with, is the representation of data, and how to embed within it useful domain specific knowledge. The following sections provide a brief background on the two problem domains, and an overview of the chapters that follow.

1.1 Compilation Optimization Background

The LLVM project [52] is an open source framework for building compilers with support for a wide variety of operating systems and hardware architectures. LLVM utilizes a highly portable and descriptive Intermediate Representation (IR) which front-end compiler developers can target to leverage the numerous platforms supported by the IR compiler back-end. Such an approach is used by the clang C/C++ front-end included with LLVM. Uniquely, LLVM's IR includes type information to enable optimizations to be applied directly to IR code and is structured to allow modules, functions, and blocks to be represented. LLVM includes numerous module and function level optimizations, such as inter-procedural optimization, dead code elimination, constant propagation, loop unrolling, etc. Optimizations are applied using a flexible pass management API which

can resolve dependencies between analysis and transformation passes automatically. Though designed for module level optimizations (the same passes are applied to all functions) it is extensible enough to support function level optimizations.

There has previously been only one known attempt to evolve neural networks to select optimization phases based on program features [50]. This technique utilized the NEAT [93] neuroevolutionary system to evolve networks for selecting optimizations to apply during dynamic compilation in the Jikes RVM Java JIT compiler. Compared to the Genetic Algorithm based techniques used previously [19, 47], neuroevolution demonstrated significant speedups over standard compiler optimizations. Additionally, using evolved neural networks avoids the high cost of evolving optimization phase-orderings directly during compilation and allows for reuse and possible specialization of the evolved networks. These results motivated the present study in the context of static compilation of LLVM IR code.

Another interesting paper covering this problem domain is [48], which deals with exhaustive exploration of the optimization phase sequences. This is of course only possible by limiting the total number of optimization types explored, and the length of the sequences. This paper explores and provides hints at some features of the solution space within this problem domain. Many other machine learning algorithms have been previously applied to the phase-ordering problem. The methods explored have ranged from hill climbers and evolutionary computation algorithms, to the more complex approaches utilizing predictive modelling [1, 3, 38, 49, 44]. Statistical methods to find optimization flags have been tested as well, with positive results [41, 33, 9], but heuristic algorithms have also been shown to produce comparable performance gains [19, 3].

1.2 Epitope Prediction Background

Epitopes are 3D structures on a surface of an antigen which can be recognized by an antibody. There are two types of epitopes, linear, and conformational. Linear epitopes are self contained on a single continuous amino-acid subsequence, while conformational epitopes are composed of multiple subsequences which form a single continuous 3D surface structure, due to the folding of the protein to bring those subsequences spatially together.

There are many reasons why one might want a computational approach to epitope prediction. The most direct applications deal with vaccine search and design. A less direct reason for a computational approach deals with the complexity and time required to search for epitopes through experimental methods. It is much faster to search computationally through the large quantities of data currently available, and only then to follow up with the much slower experimental search within the regions marked by the computational method. By having a tool which, even if it is not 100% accurate, can guide us to the locations where an epitope might be located, greatly decreases the amount of time spent on in-vivo search, and accelerates our ability to develop vaccines. Thus, it is of great use to have an epitope prediction tool which can either predict or guide us to epitope locations.

The first linear epitope prediction methods were developed in the 1980s, and were based on propensity scales [57, 37]. These were built up experimentally, and based on the statistical correlation of a physicochemical property of a residue and it belonging to an epitope. Later systems used multiple propensity scales together, these systems include the likes of PEOPLE [2], PREDITOP [70], BEPITOPE [67], and BcePred [78]. A decade later, these propensity scales were coupled with various predictive algorithms, after it was shown that predictions based purely on propensity scales produce results only slightly better than random [12].

Starting in 2006, machine learning algorithms coupled with new types of amino acid sequence encoding methods and propensity scales, began to emerge. The first of such systems was ABCPred [79], based on a recurrent neural network with an input vector of 16 residues using a sparse binary encoding. During the same year, BepiPred [51] was released, and was based on Hidden Markov Models [10] rather than neural networks. The input to BepiPred was based on numerous physicochemical properties and protein secondary structure. In 2007 AAP [17] was released, and was the first predictor using a support vector machine based model, and proposed the use of a new type of antigenicity propensity scale. AAP's improved performance ushered a new era of epitope predictors based on SVM algorithms.

In 2008, BCPred [25], and later in 2010, FBCPred [113] were published, both using SVM. BCPred and FBCPred demonstrated that predictive improvement can be achieved by using methods developed within the text mining community. These two systems used string kernels [55, 56] and SVM to make their predictions. BCPred operates on a fixed length input sequence window, whereas FBCPred can be applied to variable length input sequences. LEP-LP [16] is an SVM predictor released in 2008, and was based on multiple numerically profiled propensity scales as input.

Due to SVM's excellent classification performance, the SVM based predictor trend continues to this day. CBTOPE [6] converts residues in the sliding window into a "Composition profile of patterns", which is a vector of amino acid ratios within the window. BEST [28] epitope predictor uses a 20-mer sliding window and an SVM classifier. COBEpro [94] is another epitope predictor which uses SVM to predict short epitope sub-sequences. All three, CBTOPE, BEST, and COBEpro, can also predict conformational epitopes using a secondary clustering algorithm.

Around the same time, in 2009, EPITOPIA [77] was released. EPITOPIA is based on a naive bayes classifier and is capable of predicting conformational epitopes. It uses structure and sequence based inputs, with the sequence input being based on a sliding window and multiple (14) propensity

scales. BaysB [105] is an epitope predictor based on the naive bayes method and an SVM model. BRORacle [102] uses an SVM predictor whose input data is based on sequence features, secondary structure, and physicochemical properties such as solvent accessibility and disorder. LEPS [101] was released in 2011 and is an extension of LEP-LP. LEPS' SVM based model is used to discard LEP-LP's less likely candidates, resulting in a more accurate classification. SVMTriP [108] is an SVM based predictor, but for input it uses "Tri-peptide similarity and Propensity scores."

One of the most recently published epitope predictors is LBtope [90]. LBtope is also based on an SVM model, which is coupled with a nearest neighbour algorithm. In the paper presenting LBtope, Singh et al. notes that until now, most predictors (ABCPred, BCPred, FBCPred, BEST) have been trained on negative datasets composed of random peptides. Furthermore, the training datasets have been small, with a size of roughly 1500 total samples. To solve this problem, Singh et al. composed a new dataset of epitopes and non-epitopes, an order of magnitude larger and using the available data from IEDB [73, 100]. In this LBtope-dataset, the non-epitope sequences were based on confirmed data.

Finally, in 2015 deep learning models began entering the bioinformatics domain. Deep learning, and in particular convolutional deep networks, are currently state of the art in classification. The deep maxout network based model called DMN-LBE [109], was the first deep learning approach which was applied to linear epitope prediction. This predictor used the new LBTope dataset for training and testing, and used 5-fold cross validation. The system's classification performance was reported to be slightly higher than that of LBTope. Unfortunately, just like LBtope, it was not applied to actual long protein sequences in the published paper.

Taking all of this information into account, in this work we develop a first of its kind, deep analytically learning network using string kernels. Our system, DRREP, due to using string kernels, can be applied to the sequence directly and without any type of pre-processing. Furthermore, DRREP

outputs a vector of residue-by-residue scores, rather than scores for a single fixed k-mer window. Thus, DRREP can be used to predict the presence of epitopes in variable length sequences, and applied to entire protein chains. It is a convolutional deep network, with the first layer being a convolutional string kernel, the second an average pooling layer, the third a linear neuron layer, the fourth an average pooling layer, and finally fifth being a single threshold neuron.

1.3 Chapter Overview

Chapter 2 discusses a neuroevolutionary solution to the optimization phase ordering problem. Neuroevolutionary system DXNN and NEAT are used to evolve optimization phase order for the compiler, with the systems tested on the benchmark programs ccbench and bzip2. When compiling the programs using a standard -O0 through -O3 flags, the optimizations and their orders are statically pre-set, and are independent of the source code they are applied to. The study explores the utility of ordering the optimization phases based on the features of the source code. The performance is based on how fast the compiled programs perform, and their relative speed compared to the same source codes compiled using LLVM's -O3 compilation flag.

Chapter 3 formulates and explores a new concept of 3D BioVectors. The chapter demonstrates a new method which performs embedding of 3D biological data. Applying machine learning directly to biological 3D information is an extremely difficult task, due to the issue of representation, and high dimensionality. This new approach aims to provide a solution, through a distributed representation of residues and residue clusters, which we call 3D BioVectors. The method is a derivative of the approach known as word embedding, utilized within the natural language processing community. The chapter concludes by presenting preliminary results of applying 3D Residue BioVectors to conformational epitope, and protein-protein interaction classification problems.

Chapter 4 presents a linear B-Cell epitope predictor called DRREP, Deep Ridge Regressed Epitope Predictor. DRREP is an analytically trained and string kernel using deep neural network, which is tailored for continuous epitope prediction. DRREP was tested on long protein sequences from the following datasets: SARS, Pellequer, HIV, AntiJen, and SEQ194. DRREP was compared to numerous state of the art epitope predictors, including the most recently published predictors called LBtope and DMNLBE. Using area under ROC curve (AUC), DRREP achieved a performance improvement over the best performing predictors on SARS (13.7%), HIV (8.9%), Pellequer (1.5%), and SEQ194 (3.1%), with its performance being matched only on the AntiJen dataset, by the LBtope predictor, where both DRREP and LBtope achieved an AUC of 0.702.

Chapter 5 demonstrates a B-Cell epitope predictor based on an ensemble of multi-resolution Support Vector Machines (SVMs) which leverage conjoint triad feature (CTF) protein encoding. The CTF encoding of primary protein sequences has been utilized within the protein-protein interaction (PPI) for a decade. But this methodology, which is based on grouping amino acids into 7 groups based on their biochemical properties, has not been utilized within the epitope prediction community. The encoding uses triads, which further captures spatially local information, and residue ordering. We call our system SEEP, SVM Ensemble Epitope Predictor. SEEP is a variable length epitope classifier, capable of achieving an AUC of 91.2 on the SEQ194 standard test dataset, an improvement of 24% over the current state of the art.

Chapter 6 concludes the dissertation by summarizing the works presented within the dissertation, and discussing potential future extensions and works.

CHAPTER 2: NEUROEVOLUTIONARY OPTIMIZATION OF PHASE ORDER GENERATION FOR STATIC COMPILATION

There exists a plethora of optimizations that can be applied to code to make it more efficient. Just a minute fraction of which are for example the following: global value numbering, code factoring, instruction scheduling, reordering computations, deforestation, dead code elimination, code-block reordering etc. or replacing certain methods/functions with expert tailored versions. The order in which any of the optimizations are applied affects the form of the code available for the optimization phases that follow. Any optimization, once it has passed through the code and has affected it, thus affects what type of code is available to other optimizations, and whether they will be able to or not, to further optimize it. Thus there exists a complex interaction between the optimization phases applied to a program, a problem known as the ”optimization phase ordering problem”. This same complex web of optimization phase interactions makes it difficult to find an optimal sequence of optimization phases.

For this reason, the optimization phases for numerous compilers, are ordered either in no particular order, or in some pre-set order that seems to produce relatively good results. But optimal ordering is based on a particular program, which cannot be predicted ahead of time by any engineer setting up a static ordering of the optimization phases. Furthermore, the optimal order is based on the hardware as well, not to mention what the goals of a particular optimization sequence are (such as for example: low memory usage, execution speed etc.).

To solve this, we want the optimization sequence to be based on the features of the software being optimized, and the hardware on which it is set out to run. One of the most powerful and flexible feature mapping approaches in machine learning is the approach through neural networks [75, 22]. A neural network is a graph composed of signal processing nodes, a universal function

approximator. But given that we do not know ahead of time neither what the optimal sequence is, nor where in the solution space this optima is located, we need a method for searching for the optimal neural network topology and its parameters that is based simply on whether the resulting optimization phase order is better or not than the one it is being compared to with regards to producing a more optimal program. This type of optimization algorithm belongs within the realm of reinforcement learning, and in our particular case: evolutionary computation.

In this research we explore the use of evolved neural networks for the purpose of ordering optimization phases when compiling a program. For the compiler, the LLVM [52] compiler infrastructure was used. We connect a neural network to LLVM, and then use the compiler to generate the program features under consideration, which are then fed into the neural network which makes the decision on what optimization to use (based on a provided list of optimizations available). Of course to generate better and better neural networks capable of making such decisions, we need a process, and that is where evolutionary computation enters.

2.1 A Neuroevolutionary Approach To LLVM Phase Optimization

A neural network can be evolved to map from features of a program to a particular optimization phase that should be applied to it. The evolutionary approach can optimize the neural network so that it chooses an optimal sequence of such optimizations, and stops when it is done as per the fitness function used to evolve these mapping neural networks. The fitness function can be based on either performance of the program (how fast it runs), how much memory it takes (when we're optimizing for it to run on limited hardware for example), or all of the above, in a multi-objective approach, and thus creating a Pareto front of optimal neural networks.

In this application domain, two different approaches are evaluated for the purpose of evolving

neural networks which select optimizations during static compilation. In the first approach, an evolved neural network selects optimizations to apply to an entire LLVM module based on the aggregate features of all the functions defined in the module. Contrast this with the second approach, where each function in a module is independently optimized based solely on its features. In both approaches optimizations are iteratively selected based on the features of the module or function extracted after the previous optimization is applied. Iteratively optimizing a module or function in this fashion allows the most appropriate optimization to be applied at each step. The function level approach is closest in form to that used in by Kulkarni et al., [50], where each Java method is individually compiled and optimized during JIT compilation, but in this case is applied during static compilation using LLVM.

Both the module and function level approaches share a number of implementation details. We first have to extract a list of features used as inputs to the NN, based on which it will be making its selections. The list of features is long and will not be listed here in full (module level uses 48 features while function level uses 44 features). Module level features include the number of functions, size of functions as a percentage of total instructions, the instruction histogram over the entire module, etc. Function level features include the number of blocks within the function, the instruction histogram for the function, etc. As noted previously, LLVM provides numerous optimizations which can be applied at the module and function levels, of which only a subset are utilized here (53 at module level, 34 at function level). Each neural network output is associated with a particular optimization to be applied. There is also a single output which indicates that iteration should be ceased and no further optimizations are to be applied. The highest valued output for a particular set of module or function feature inputs indicates the next optimization to be applied. After each optimization is applied, a new set of features are extracted for the next iteration. This process continues until the the neural network chooses to cease iteration or the maximum number of optimizations has been applied.

The general architectural pipeline of our system is shown in Fig-2.1. An LLVM fronted such as Clang is used to generate IR bytecode from a program's source code. This bytecode is then passed to a module or function level optimizer along with a neural network used to select optimizations. In each iteration of the optimizer, features are extracted and set as the inputs to the neural network. The resulting outputs determine the optimization to apply in the next iteration or whether iteration should terminate and the optimized bytecode passed to the backend to finish compiling. During evolution, benchmark programs are optimized using an evolved neural network and then executed to measure the resulting performance of the program on which the fitness of each evolved neural network is based. The speedup provided by a particular neural network is measured relative to the standard *-O3* optimization flag provided by Clang. For both the module and function level optimizers the *ccbench* [github.com/ucb-bar/ccbench] and *bzip* [bzip.org] benchmarks are used to provide feedback.

To evolve neural networks for module level optimization, DXNN is used, which has been shown to perform superbly in complex problem domains such as Artificial Life and Currency Trading, all of which are multi-dimensional problem domains, and deal with complex fitness landscapes. DXNN supports recurrent neural networks, which are used to provide memory of previously observed features and applied optimizations, potentially improving the selected optimizations. To evaluate the fitness of each evolved neural network, a custom LLVM module feature extractor and the LLVM *opt* tool are iteratively used to generate the neural network inputs and apply the selected optimizations to a module. The custom LLVM module feature extractor calculates and aggregates features of all functions in a module to determine neural network inputs for each iteration. The LLVM *opt* tool is used to apply each optimization selected by the neural network to a particular module. This applies the same optimization to every function contained in the module.

The function level approach uses NEAT to evolve neural networks in a manner similar to [50]. The overall architecture is similar to that of the module level approach, with the exception of

NEAT replacing DXNN, and the requirement of generating multiple optimization sequences for each function instead of a single sequence applied to the entire module. The MultiNEAT implementation of NEAT is used to evolve the neural networks and, as was done by Kulkarni et al., only feedforward (non-recurrent) neural networks are evolved. This approach is complicated by the fact that LLVM does not provide tools to independently optimize particular functions. For this reason, a custom optimizer, implemented using the LLVM framework, is used to extract function features and directly activate a neural network evolved using NEAT to iteratively select optimizations to apply to the function.

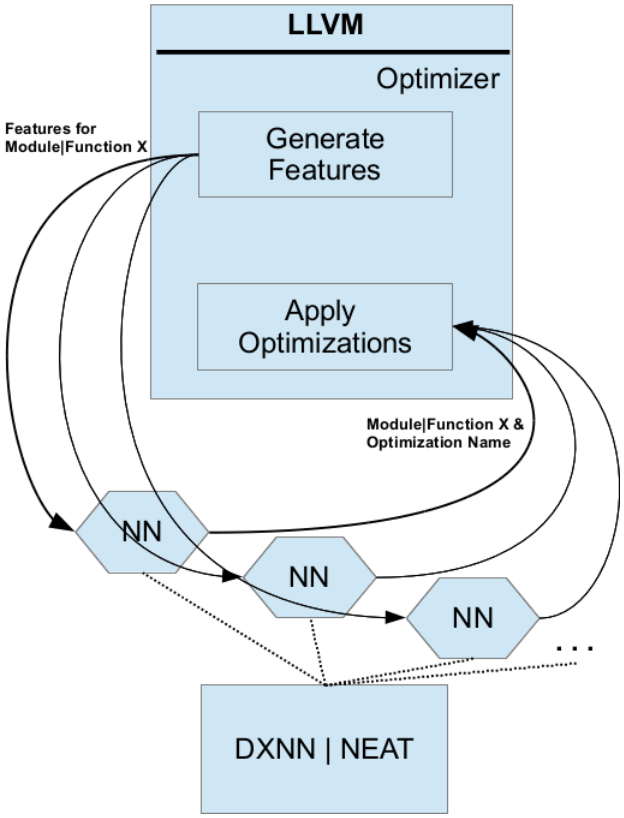

Figure 2.1: The general architectural pipeline

2.2 Comparison of Evolved Optimization Phase Ordering to LLVM Built-in Ordering

For the module level approach, DXNN was ran in different types of modes and applied to both, bzip2 and ccbench. In all DXNN based experiments, a population of 10 was used, and memetic algorithm employed. For bzip2 benchmark, DXNN ran for 5000 evaluations averaged over 10 evolutionary runs, using only program speed as the fitness function and using only tanh as the activation function. The experiment was then performed again for 5000 evaluations and averaged over 10 evolutionary runs, but where the system had access not only to a sigmoid activation function, but instead the following list of functions: [tanh, sin, abs, gaussian], referred to in the graph as "all". A multi-objective experiment was performed, with the fitness now being a vector: [ProgramSpeed, TotalOptimizationsApplied], in an attempt to see whether we could minimize the number of optimizations applied while improving speed, at the same time. The results were poor, and begs the question of why, and thus is currently being explored. Finally, another benchmark was ran for 50000 evaluations, to see whether further improvement can be achieved if more computational time is provided. The graphed validation results are shown in Fig-2.2.

Then the system was applied to the ccbench for 15000 evaluations, only with tanh as the activation function. A graph of which is shown in Fig-2.3.

For the function level approach, NEAT was ran for 1000 generations over a population of 30 (30000 fitness evaluations using ccbench) and averaged over 4 runs. The maximum, minimum, and mean of the best fitness in each generation is shown in Fig-3.19. Note that the negative of the running time is used so that higher fitness corresponds to lower running time. For comparison, the results of 100 runs of ccbench compiled with clang optimization flags $-O0$ and $-O3$ are shown in Table-2.1. A graph of the evolved function level optimization neural network based orderers is shown in Fig-2.4.

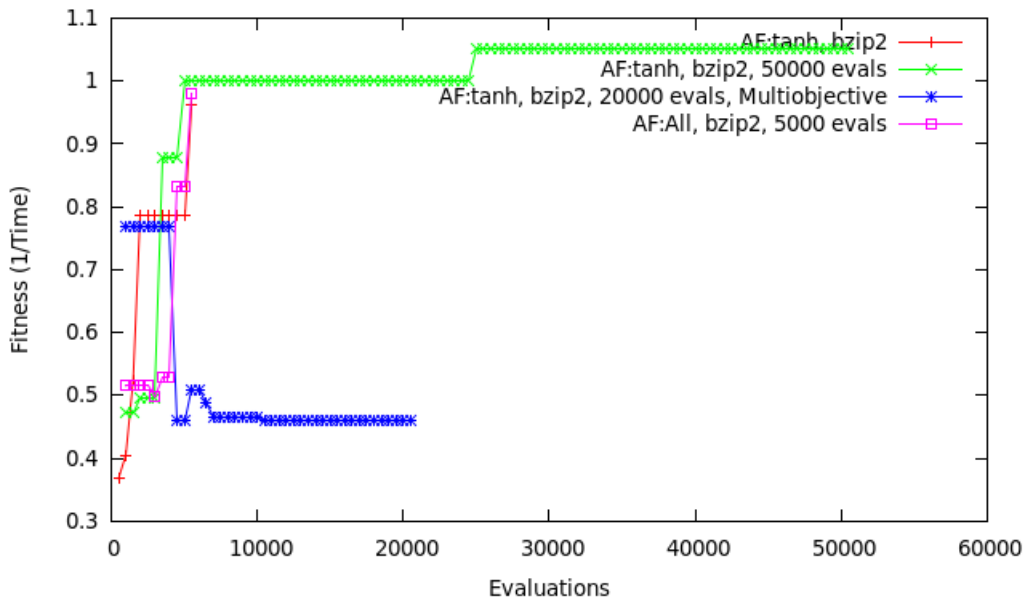

Figure 2.2: Module level based Bzip2 results

Table 2.1: Results of 100 runs of ccbench compiled with clang optimization flags '-O0' and '-O3'.

Mean	Min	Max	OptimizationFlag
2.849	2.558	3.284	-O0
2.819	2.612	3.016	-O3

2.2.1 Discussion

In Fig-2.2 we can see the DXNN's slow and steady fitness increase on the BZip2 benchmark. The 5000 evaluation experiment ended with a final most fit NN capable of ordering the optimization phases that results in the BZip2 benchmark executing in 1.1s, compared to the 1.77s optimized by -O3. Running the evolutionary run for a longer period of time, did not result in significantly better results, but did show an improvement, with the final optimized program running in 0.953s, an improvement of 40.8%. Also, for at this time unknown reasons, the multi-objective approach did not produce positive results.

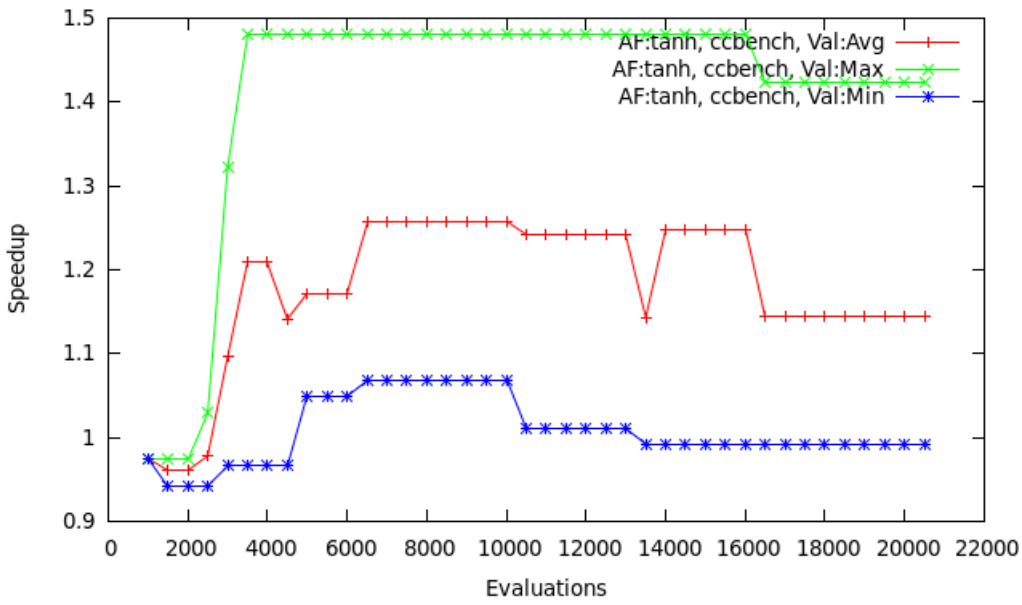

Figure 2.3: Module level based ccbench results

Fig-2.3 presents DXNN's evolutionary run that evolved the ccbench phase ordering NNs. Here too an improvement is seen, with the best validated NN capable of ordering phases such that the final result runs in 0.752s, compared to the 1.035, a 27% improvement. The results demonstrate that this approach produces significant improvements over the -O3 optimization level.

We can see clearly in Fig-2.4 that NEAT is having difficulty with this highly complex optimization space (44 inputs and 34 outputs). This is likely due to a combination of several factors: a small population makes it difficult for optimal traits to persist over many generations, evolutionary parameters were not optimized, and the optimization space is highly non-linear. While little can be done to reduce the complexity of the optimization space, increasing the population size while reducing the total number of generations and informed selection of evolutionary parameters may provide some improvement. Of particular interest is the fact that even the lowest best fitness in a generation (roughly 2.68s around generation 860) is approximately 5% faster than the average

running time using '-O3', while the best fitness found (1.860s in generation 442) is approximately 34% faster. Though these results are not directly comparable to [50], they demonstrate that function level optimization phase-ordering is also beneficial in static compilation.

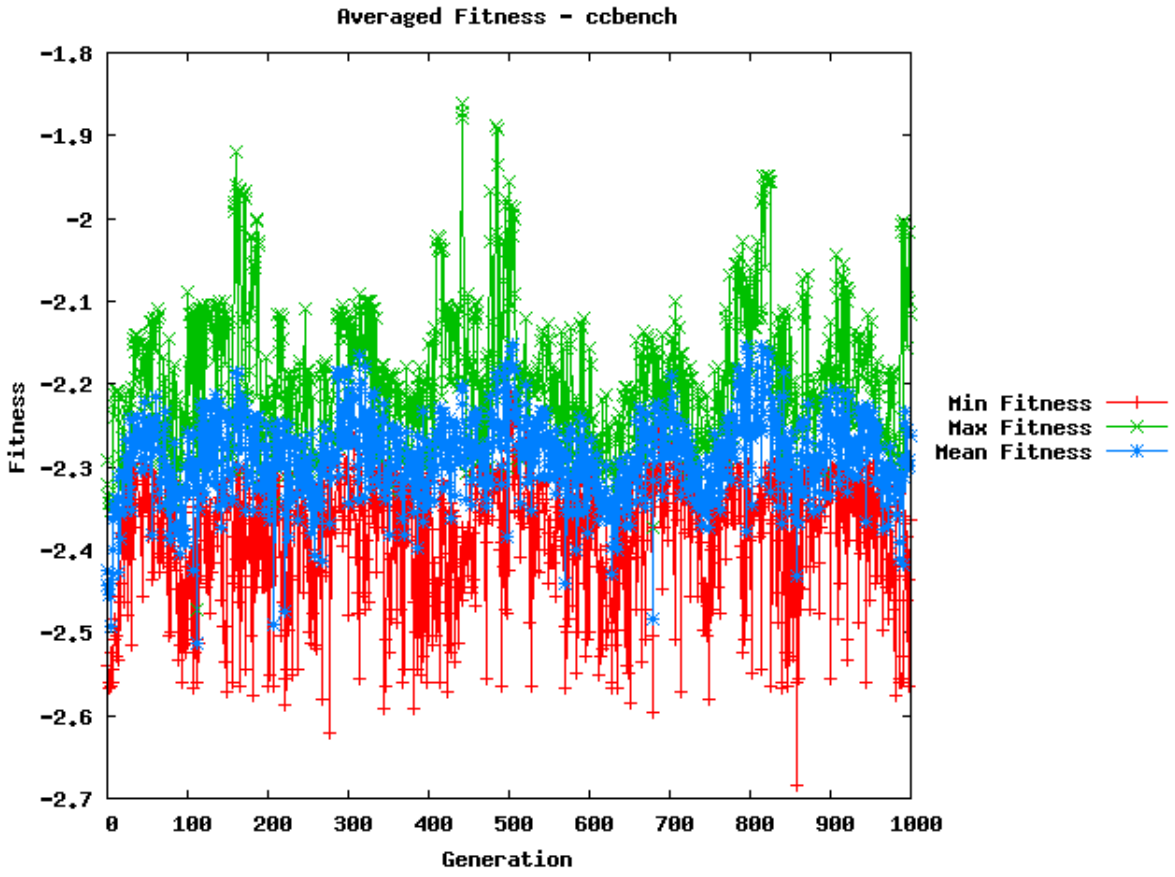

Figure 2.4: Function level based ccbench results

One significant issue is the challenge of accurately measuring the running time of the benchmarks. For these results, the output of the bash shell's built-in time command was used, summing the user and system running times to determine the overall running time. The wall-clock running time was found to vary significantly depending on system load. While the user and system times are measured at the kernel-level, they too appear to vary with load, but much less than the wall-

clock time. No readily available alternative to the time command is known. However, kernel-level performance tracing tools may provide results that are more accurate and will be tested in future work.

2.3 What We Learned

In this chapter we have developed two neuroevolutionary based pipelines for the purpose of optimization phase ordering, used with the LLVM and Clang compiler. One was module level optimization based on DXNN neuroevolutionary system, and one was function level and based on NEAT. In both cases we have demonstrated improvements, as per bzip2 and ccbench benchmarks, performance improvements ranging from 5% to 50%. Furthermore, we have explored different types of activation function sets, feedforward and recurrent neural networks, and even a multi-objective evolutionary run, though which has not yielded positive results.

One significant issue which must be considered in future work is that a number of optimizations are applied regardless of the optimizations specified when invoking opt. A customized optimizer which eliminates these optimizations will likely need to be used to better evaluate the performance of our results. It is also likely that these optimizations may provide a better starting point for both module and function level phase-ordering. It may be useful to also investigate whether selecting specific sequences of optimizations, such as those used by standard optimization sequences, may be more effective that selecting individual optimizations alone. We also still need to explore the role of plasticity and multi-objective optimization within this problem domain, for which significantly more time is required. Finally, Superoptimization [5,6] is the search for optimal loop free instruction sequences implementing a particular function template. Though seemingly unrelated, superoptimization could potentially be leveraged as another optimization phase that can be selected by the neural network during phase ordering.

CHAPTER 3: EXPLORATION OF BIOLOGICAL DATA REPRESENTATION, AND THE CONCEPT OF 3D BIOVECTORS

When it comes to biological data, the main issue that we are dealing with and are trying to overcome, is data representation. How do we encode amino acids in a useful way, in a manner that embeds within the encoding some useful biochemical and spatial information? Is there a way to directly extract useful biochemical and spatial information from the 3D structure of proteins, and embed that information into some distributed representation that can be used by machine learning algorithms? This was the premise for the study.

We explored other research domains which faced similar data representation problems. This eventually lead us to Natural Language Processing (NLP), and the issues faced by the community with regards to representing words, and text data in general. Similarly to the discrete biological data, like amino acids, NLP researchers also face the difficulty of representing words in a way that embeds their properties, and contextual information. Their solution to this, is word embedding.

Majority of machine learning algorithms are not capable of being applied to strings or text data *directly and in its raw form*. Word embedding is simply a method of representing text in numerical form. More generally, distributed representation is a method of representing categorical data through continuous numerical vectors. In this chapter we explore a concept of representing 3D biological data using a method based on an analogy of word embedding.

The methodology of word embedding is a process by which high dimensional semi-discrete or categorical data is embedded into a continuous lower dimensional vectors (distributed representation). This type of representation was shown to produce extremely positive results within the NLP field when Word2Vec was released and popularized [61]. Word2Vec was developed for text

based data, and then extended to *1D BioVectors*, produced by applying the methodology to protein and genetic sequences [7]. But these approaches only concentrate on the primary sequence information and text data, rather than 3D and spatial biological information. Nevertheless, Asgari et al. demonstrated that even these 1D BioVectors can be successfully applied to the protein-protein interaction classification problem.

In this work we have formulated a new method and developed a system which performs embedding on biological 3D data. We call these distribute representations: 3D BioVectors. In this chapter we explore and introduce the concept of 3D BioVectors, a method for generating them, and show preliminary results of low dimensionality residue based 3D BioVectors applied to *Conformational Epitope Classification* and *Protein-Protein Interaction* (PPI). We demonstrate that using only the basic information provided within the protein database (PDB) files, coordinates of residues and atoms composing the 3D structure, 3D BioVectors can be generated which embed within themselves useful domain specific information, and thus might have utility in practical applications. We believe that 3D BioVectors can encode within themselves useful biochemical and spatial information, data that is not available when building 1D BioVectors, and thus be applicable in more application domains, and produce superior results.

3.1 Distributed Representation

Word embedding is a method of creating a distributed representations of words. Traditionally NLP represented words as independent Ids, unrelated to each other semantically. This means that not only can you not get any useful information about the word's context by simply looking at its representation (some unique Id), but also that it would be difficult to process it using a machine learning algorithm. Research shows that this is not how the brain stores words, instead it uses distributed representations [64]. In distributed representations, semantically similar words occupy nearby lo-

cations within the multi-dimensional word vector space. It is this type of representation which also allows us to link together similar ideas, and perform transfer learning from one field to another, if they share similar concepts. These properties were some of the advantages that researchers within NLP field wanted to take advantage of when developing the distributed representation of words through word embedding.

Though the concept of word embedding, and the term itself was coined back in 2003 by Bengio et. al. [11], it was not until Word2Vec [62] was developed that the methodology took of. Word2Vec is currently the state of the art system which performs word embedding using text data. The method is based on the standard approach of using shallow neural networks and skip-grams [32], but modified (by incorporating optimizations such as negative sampling and sub-sampling) to work efficiently with extremely high dimensional input and output vectors (based on the size of the vocabulary), and very large training datasets. We extend the concept of embedding from 1D text, to 3D spatial biological data.

3.2 From Word Embedding to 3D Information Embedding

Word embedding can be done using numerous approaches. Some of the common methods include probabilistic approaches [29], dimensionality reduction approaches [53], co-occurrence matrices [58, 59], and most commonly, neural networks [61]. All methods are based in one way or another on the concept popularized by Firth [27] through his comment that ”a word is characterized by the company it keeps”. A basic version of the skip-gram method which generates distributed representations of words is set up as shown in Fig-3.1.

In this setup, a large body of text is broken down into tuples composed of target word at location N in the sentence, and the context words near the target word. The context words (”the company

that the word keeps") are defined by context radius; in the Fig-3.1 example the radius is 1. The context words for the target word at location N are the words at locations $N-1$ and $N+1$. The input and output dimension of the neural network is defined by the vocabulary size on which the system is trained. Finally, the words are represented using one-hot encoding.

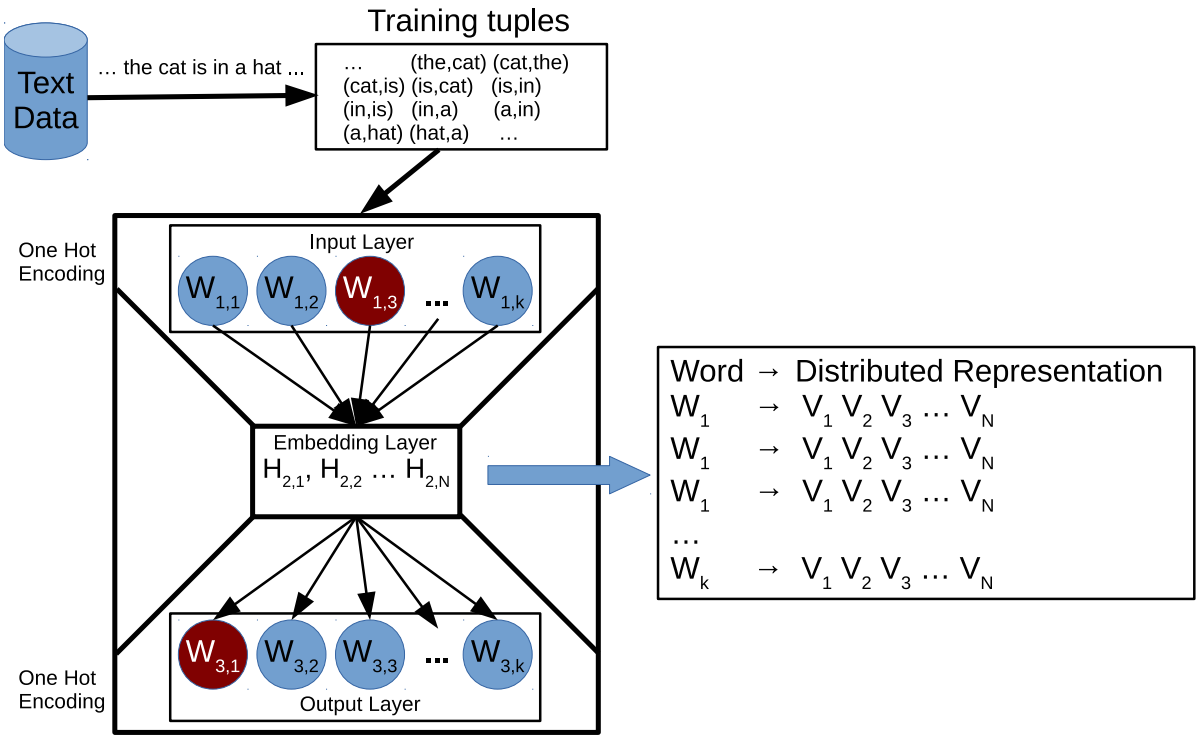

Figure 3.1: **Basic Word Embedding Framework.** Demonstrates a simple feedforward neural network approach for generating distributed representations of words with a context radius of 1.

Using these $(TargetWord, ContextWord)$ tuples, we attempt to teach the NN to predict a probability distribution of context words, given some particular target word. Because there are thousands of words, the input and output dimensions are very high. We train the neural network with a hidden layer that is dimensionally lower than the input and output layers. The output layer uses softmax activation function, while the neurons in the hidden layer do not use activation functions. What does such a neural network learn? If the entire body of text is broken down in this manner, than the

neural network predicts the statistical distribution of any word appearing next to the target word. For example, on average there will be many more training samples where the word ”green” is next to the word ”light” rather than some other word like ”orange”. After training the NN, if the word ”green” is used as input to the NN, it will produce a vector of probabilities of the words in the vocabulary appearing next to the input word. But learning these associations this is just a ”throw away” problem, learning this type of probability distribution is not the end goal.

To learn such probability distributions, the NN is forced to encode useful information within the hidden layer, it is forced to embed information that is useful to make contextual associations. It is this hidden layer that represents the distributed representation, and thus provides us with a word to distributed representation mapping. It is this hidden layer encoding that is the end goal of the training.

After the NN is trained, we throw away the output layer, and then feed to the NN each word within the vocabulary to acquire its distributed representation. These distributed representations can then be used with other machine learning methods for various applications. These representations have been shown to be extremely useful, and to have numerous useful properties. For example, words with similar semantics clustered together within the multidimensional word vector space [97]. Examples of these trained distributed representations and their clustering of contextually similar words, have been presented in a number of recent works [62, 18, 92].

Thinking analogously, we extend the idea of word embedding into three dimensional space, concentrating primarily on biological structures. Firth [27] said that ”You shall know the word by the company it keeps”, our intuition about amino acids runs along similar lines. We think that *”You shall know the amino acid by the company it keeps”*. There are numerous types of amino acid interactions within the protein and on its surface. The 3D structure of the protein is based on how the entire protein sequence folds, and the different types of residue-residue interactions. Thus, em-

bedding can be performed on buried amino acids, on all amino acids (buried and surface), or just surface amino acids. Each of these residue-residue interaction types would have their own specific environmental, biochemical, and spatial properties. The distances, and the type of residue-residue interaction that occurs between buried residues is different than that which occurs between surface amino acids. Because protein-protein interaction occurs particularly through the surfaces of the two interacting proteins, in this exploration we chose to primarily concentrate on the surface residues.

The 1D BioVectors generated from the primary protein sequences [7] were based on training tuples where the target is a residue trimer within the sequence, and the context are the residue trimers on the left and the right of the target trimer. In this work we explore a low dimensional distributed representation based on residues rather than trimers, we call these residue distributed representation vectors: *3D Residue BioVectors*. For the case of 3D embedding, as it applies to proteins, we define the target and context as the *target residue* and *context residue*. The 3D Residue BioVectors are produced by a neural network which is trained on the tuples composed of these target and context residues.

Analogously to word embedding and skip-gram approach, context radius is a distance around the target residue, and any residue within that distance belongs to the target residue's context or environment. Figure-3.2 shows an example of how training tuples are formed. First we find all surface amino acids on the protein. Then, we go through each surface amino acid, designating it as the target residue, and defining context residues as those amino acids which are within its context radius. The context radius in this example is defined to be a distance of 6A, as per the average distance between two residues in direct contact, as is specified by Ofran et al. [68]. The tuples are composed by producing (TargetResidue, ContextResidue) pairs. Because we will primarily deal with the 20 most common amino-acids, the input target residues will be defined as 20 dimensional one-hot vectors. Similarly, the context amino-acids are also

defined as 20 dimensional one-hot vectors. Thus, the resulting training pairs have the following format: $([TR_1, TR_2, ...TR_{20}], [CR_1, CR_2, ...CR_{20}])$, where TR_i specifies the target residue, and CR_i specifies the context residue.

An approach which will be able to incorporate more information within the distributed representation is one that is trained on higher dimensional input and output vectors, such as trimer residue clusters for example. Amino acid cluster target and context pairs is something we plan to explore in the future, once we have analysed and understood this much lower dimensional embedding, and have explored its performance.

Because we take our target residues from the surface of a protein, we must first define the pool of surface residues. As was noted by Miller et al. [63], surface residues have an average relative surface accessible area of $5 - 50\%$, or an average of $2.5A^2$ and above of absolute solvent accessible surface (SAS) area. Thus, we start by defining our surface residues which have at least $2.5A^2$ SAS, but leave it as a tunable parameter as we explore the performance of the derived 3D Residue BioVectors.

Though there are many factors that make up the context/environment of a target surface amino acid, we start our exploration using the most easily accessible and available information, which is present within all PDB files: the nearby amino acids. In the example we define the context radius to be $6A$. This value is roughly equivalent to context radius of 1 within word embedding, because it defines those residues which are directly in contact with the target residue. In general, any amino acid pairs whose C_a or C_b atoms are within $4A$ to $12A$ of one another [68], are considered to be in direct contact. Similar to the surface accessible area, the environmental radius is also a parameter, which we can vary to explore its affect on the generated 3D Residue BioVectors.

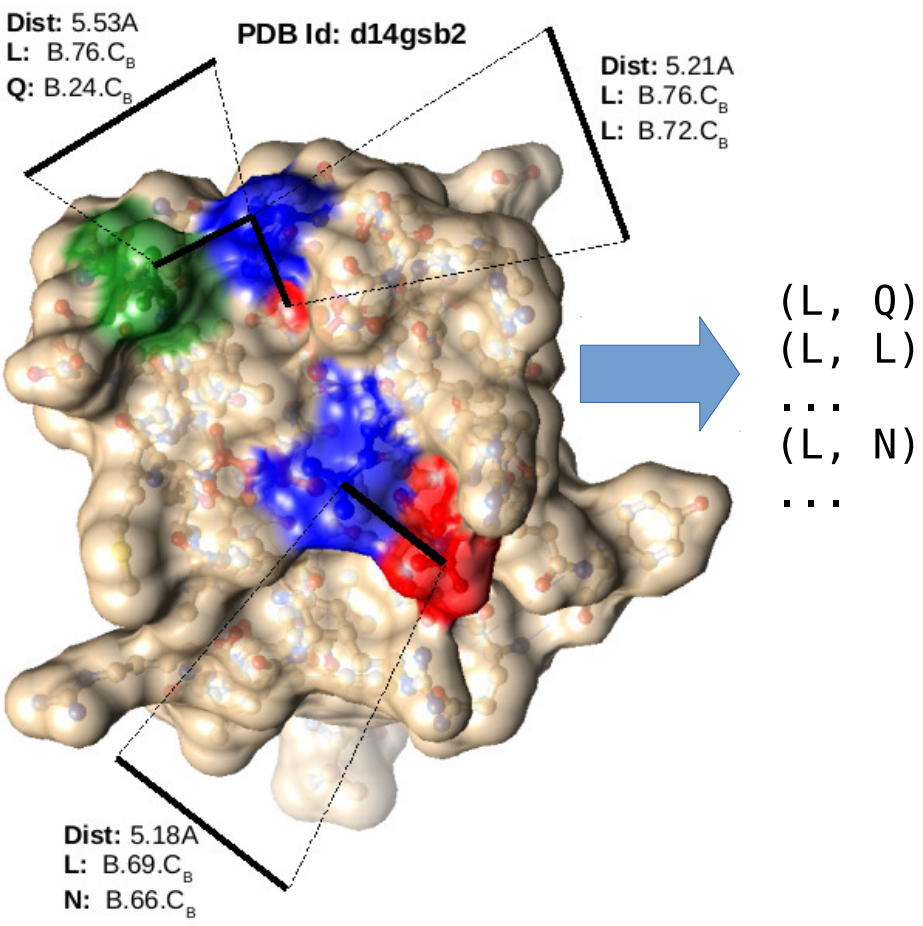

Figure 3.2: **An example of solvent accessible residues.** Solvent accessible residues are those residues which are located on the surface of a protein, defined by having on average of at least $2.5A^2$ of solvent accessible surface. The figure presents a protein (Id d14gsb2), and though there is internal structure, the residues we concentrate on are those on the surface. The figure shows 5 surface residues, with target residues coloured blue, and their nearby context residues coloured in red and green. The context residue is defined in this example as a residue that is no more than $6A$ away from the target residue.

3.3 Unsupervised 3D Residue BioVector Generator

The training dataset tuple formation and 3D BioVector generation pipeline shown in Fig-3.3, works as follows: We use the *SCOPe: Structural Classification of Proteins - extended* database, because it it is composed of non redundant set of PDB files. The next step is to calculate solvent accessible surface (SAS) area for every atom in the PDB. Following the calculation of SAS, a training-tuple extractor goes through each PDB file, and marks all residues with $SAS > A$, where A is the minimal area threshold. Then for every marked residue, we calculate its context residues by finding all residues that are within distance D of the target amino acid. After the context residues are found, we form tuples: ($TargetResidue$, $ContextResidue_k$), for all k context residues.

The next step is to train a feed forward neural network (FFNN) with N hidden neurons on the training dataset. We set aside 10% of the dataset to be used for validation, and train the neural network until its performance on the validation dataset begins to decrease. The FFNN is validated every 1000 evaluations.

Once the neural network is trained, each residue is ran through the NN. During this step, we store the output of the hidden layer, which results in a list of 20 tuples of the form: (Target Residue, Hidden Layer Output), one tuple for each residue. The tuples represent the amino acid to distributed representation mapping, and N is the dimensionality of the resulting distributed representation, the 3D Residue BioVector.

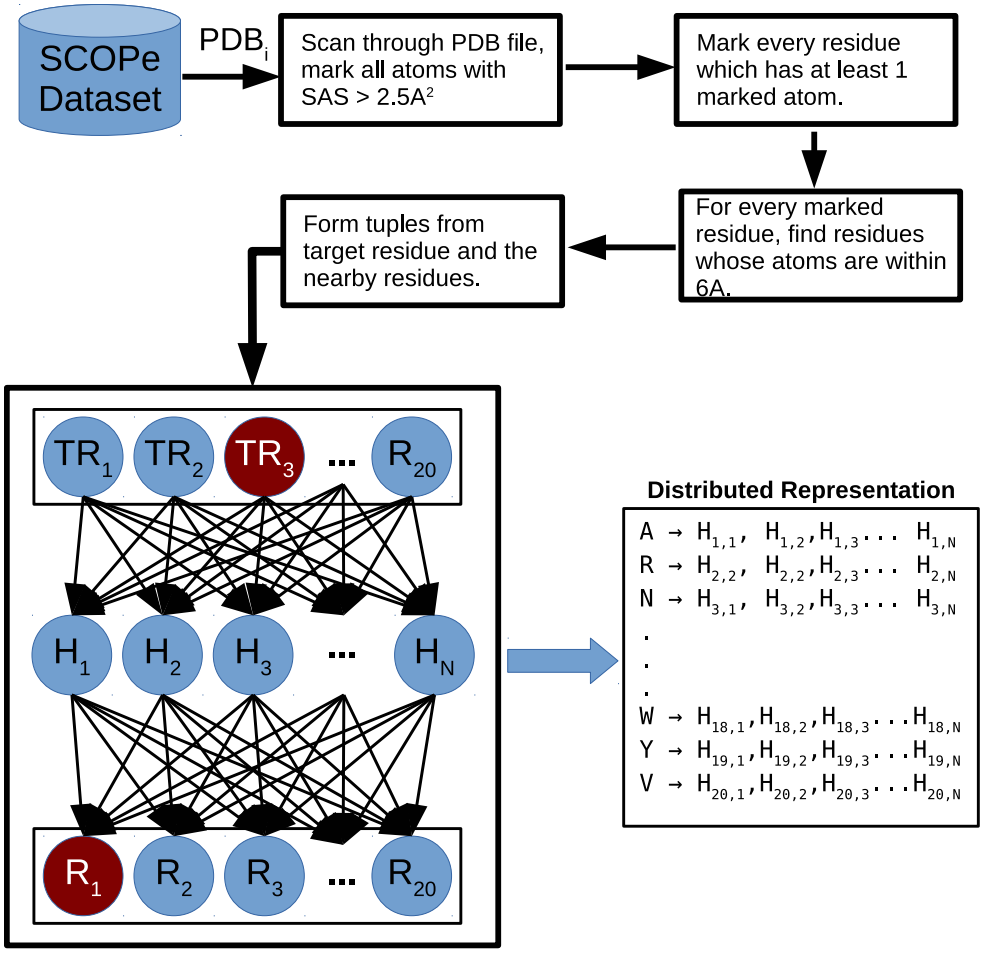

Figure 3.3: **Training Dataset Generation.** The pipeline for generating the training dataset using the SCOPe database PDB files.

3.4 Compositional Residue Encoding

The easiest way to compare protein sequences of different sizes using a machine learning algorithm, such as SVM or Neural Network, is if both of the sequences are represented through a vector of an invariant length. This way, the two vectors are appended, and their appended length is also invariant (independent of the protein sequence's length). The dimensionality of the appended

vectors is the input dimensionality we then initiate NN or SVM with, and then train it to classify the two proteins. One of such representations is the conjoint triad feature [83], which encodes a protein sequence into a 343 dimensional vector, regardless of how long the protein sequence is. We create an invariant length continuous vector representation using the 3D Residue BioVectors using *vector composition*.

Vector composition representation of a protein sequence is accomplished by composing the vectors for each residue. Rather than replacing each residue with its distributed representation, we add up the continuous vectors, and then normalize the result. This encoding approach is demonstrated in Fig-3.4, which also presents a learned example of a 5 dimensional distributed representation for all 20 residues.

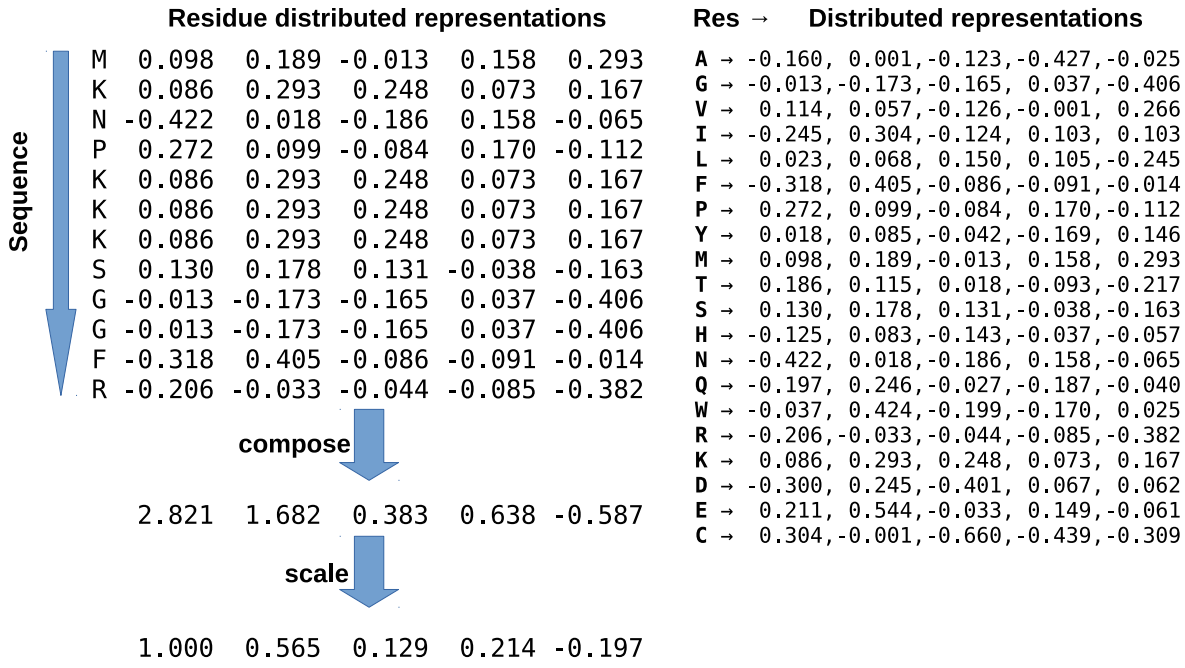

Figure 3.4: **Compositional representation.** A compositional representation of a protein sequence, through addition of 3D Residue BioVectors. The example demonstrates the compositional representation of a sequence of length 12, encoded using 3D Residue BioVectors of length 5, resulting in a final vector representation of length 5.

3.5 Results

To test if any useful information was embedded within the generated 3D Residue BioVectors, we applied this new encoding to two different problem domains: 1. Conformational B-Cell Epitope Prediction, and 2. Protein-Protein Interaction. We used the conformational epitope dataset provided by Kringelum et al. [45], which is composed of 76 sequences, containing a total of 25200 residues. To test the encoding's performance on PPI classification, we used Pans PPI dataset [69] for training, and tested the trained SVM classifier on the external test datasets based on Drosophila, Yeast, and Caenorhabditis elegans, provided by Guo et al. [31].

We generated 3D Residue BioVector of length 5, 10, and 15. We specifically chose the dimensions lower than 20, based on the hypothesis that the NN will have to more densely encode useful information by having to compress the already very limited amount of information provided by the single residues. Our aim was also to explore whether the low dimensional hidden layer could embed any useful information at all. Multiple versions were generated by varying the solvent accessible surface area SAS, and by varying the context radius threshold D. Based on research literature, our expectation is that by using SAS of $2.5A^2$, which has been experimentally demonstrated to be the average threshold value for surface residues, and by using a threshold distance of $6A$ as context radius, we will be able to produce high performing low dimensionality 3D Residue BioVectors.

During the process of training an SVM classifier on the generated 3D Residue BioVectors, we tried Linear, Polynomial, and RBF kernels, for each sliding window size. In Table-3.1 only the top 2 best performing parameter combinations for each window size are shown when applying the trained classifier to the conformational epitope test dataset, which was also the test dataset for DiscoTope 2.0 [42].

Table 3.1: **Conformational Epitope Prediction Preliminary Results** AUC results based on the application of compositional 3D Residue BioVector encoding to conformational epitope prediction.

RSV Length	Window Size	Kernel	SAS Area	Dist. Threshold	AUC
5	12	Linear	$5A^2$	10A	63.2
10	12	Poly	$5A^2$	10A	65.6
10	20	Poly	$5A^2$	10A	63.1
15	20	Poly	$5A^2$	10A	63.4
5	30	Linear	$5A^2$	10A	61.3
10	30	Poly	$5A^2$	10A	61.3
5	12	Poly	$2.5A^2$	6A	65.5
10	12	Linear	$2.5A^2$	6A	66.4
10	20	Linear	$2.5A^2$	6A	66.3
15	20	Linear	$2.5A^2$	6A	64.6
5	30	Linear	$2.5A^2$	6A	**67.6**
10	30	Linear	$2.5A^2$	6A	65.5
5	12	Poly	$2.5A^2$	10A	**68.7**
10	12	Linear	$2.5A^2$	10A	**68.5**
15	12	Poly	$2.5A^2$	10A	**68.4**
10	20	Linear	$2.5A^2$	10A	62.0
15	20	Linear	$2.5A^2$	10A	61.5
5	30	Poly	$2.5A^2$	10A	64.3
10	30	Poly	$2.5A^2$	10A	64.7
15	30	Poly	$2.5A^2$	10A	65.7
5	12	Linear	$2.5A^2$	20A	60.2
10	12	Linear	$2.5A^2$	20A	61.1
15	12	Linear	$2.5A^2$	20A	63.3
5	20	Poly	$2.5A^2$	20A	60.1
15	20	Linear	$2.5A^2$	20A	61.3
5	30	Poly	$2.5A^2$	20A	61.7
15	30	Linear	$2.5A^2$	20A	63.1

This is a very compact encoding, and undoubtedly does not encode all the needed information to make accurate predictions. We further note that using a residue vector of length 5, the SVM classifier was still able to reach an AUC of 67.6. This performance was reached when using sliding windows of size 30, and with the surface vectors generated using solvent accessible area of $2.5A^2$, and a context radius of 6A. These SAS and distance threshold values are the ones that are

associated with the average SAS to determine a surface residue [63], and the distance threshold which is noted to be the average distance between residues directly in contact [68].

A more inclusive threshold distance of 10A produced slightly better results, reaching an AUC of **68.7**. This makes sense in retrospect, because as noted by Ofran et al., the residue contact distances range from 4A to 12A, dependent on the size of the residue. Thus, a more inclusive distance would cover larger number of residue types, and potentially allow the distributed representation to encode a more accurate correlation. A SAS threshold of $5A^2$ produced 3D Residue BioVectors which performed worst of all, which fits with the intuition that such a high SAS excludes a lot of residues which are indeed on the surface of the protein, thus miss-representing contextual associations. On the other extreme end, setting the threshold distance to 20A, also produces a set of results lower than those achieved by using a threshold distance of 6A or 10A.

We believe that these preliminary results demonstrate that the distributed representations produce different performance results, based on the parameters used to generate them, and the information that is embedded within. In Table-3.2 we compare the best AUC achieved in Table-3.1 to the AUCs achieved by DiscoTope-2.0, DiscoTope [34], ElliPro [74], and PEPPITO [95], on the same dataset. Results for these classifiers are taken from Table-4 of the paper published by Kringelum et al. [45]. Though the 3D Residue BioVector produced classification results which are not state of the art, it is encouraging that the extremely low dimensional 3D Residue BioVector performed on the same level as the more complex ElliPro classifier.

Table 3.2: **Conformational Epitope Prediction Comparison** Comparing results of 3D Residue BioVector of dimensionality 5, to other modern methods.

Method	AUC
3D R. BioVector	68.7
DiscoTope-2.0	73.1
DiscoTope	70.5
ElliPro	68.6
PEPITO	73.2

Table-3.3 presents preliminary results for PPI classification. The best results for Drosophila was an AUC of 68.0. The SVM achieved an AUC of 69.0 when applied to the protein pairs from the C.elegans dataset. Interestingly, on the Yeast dataset the SVM only achieved an AUC of 53.2 when using the 3D Residue BioVector, which will require further analysis to better understand the reason. Again, though the results are not state of the art, this exploration of the data representation shows utility in 3D BioVectors, and the method proposed and explored here. Finally, though the compositional distributed representation encoding dimensionality ranged only from 5 to 15, Pan et al., dataset contains proteins which range in size from $51aa$ to $18074aa$. Thus further makes the classification results interesting and encouraging.

Table 3.3: **PPI Classification Preliminary Results** Application of residue surface vector encoding to Protein-Protein interaction.

Dataset	RSV Length	Kernel	SAS Area	Dist. Threshold	AUC%
Drosophila	5	Linear	$2.5A^2$	6A	67.8
Drosophila	10	Linear	$2.5A^2$	6A	67.4
Drosophila	15	Linear	$2.5A^2$	6A	68.0
Yeast	5	RBF	$2.5A^2$	6A	51.5
Yeast	10	RBF	$2.5A^2$	6A	53.2
Yeast	15	RBF	$2.5A^2$	6A	53.0
C.elegans	5	RBF	$2.5A^2$	6A	68.2
C.elegans	10	Linear	$2.5A^2$	6A	69.0
C.elegans	15	Linear	$2.5A^2$	6A	68.4

3.6 The Future of Residue Surface Vectors

The work presented here is exploratory, with the aim being the exploration of 3D biological data representation, and generation of distributed representations from the 3D protein data itself. When using strictly residues as target elements, the input vector is only of length 20, and thus we can not expect for too much information to be embedded. This is unlike word embedding, with an extremely large vocabulary dataset which allows for input/output dimensions ranging in the thousands. This allows for a lot more learning to occur, due to a lot more input/output correlations that exist in the much larger dimension of the input and output vectors, and the larger hidden layer.

Asgari et. al. used word embedding on the primary protein sequence, but the target and context elements forming the training dataset were based on trimers. This allowed for 8000 (20*20*20) dimensional input/output vector lengths. From this representation, 100 dimensional 1D BioVectors were generated. The resulting embedded representation produced state of the art predictive results when applied to PPI classification. In our 3D distributed representation, the next step is to use residue trimer clusters, extending the dimensionality of the input and output to 6840 (20*19*18) dimensional vectors (residue order is not applicable in 3D amino acid clusters). This will further extend the complexity of the information that the system will attempt to learn, and allow us to use larger hidden layers and produce 3D BioVectors which will embed more information. Furthermore, at this time only the residues from the same protein have been used to form the training tuples. It is possible to explore the tuples based on the known PPI contact maps, which better represent the correlations of residue interaction between two independent proteins. Finally, tuning the distance and SAS threshold values that define the target and the context around it, will further allow the system to embed more accurate spatial and biochemical features, as is suggested by our preliminary benchmark results in conformational epitope prediction.

CHAPTER 4: DEEP RIDGE REGRESSED EPITOPE PREDICTOR

In this work we present a sequence based continuous epitope predictor called DRREP (Deep Ridge Regressed Epitope Predictor). Our *linear B-Cell epitope* predictor is based on a deep neural network (DNN) [54], which utilizes a string mismatch function based first hidden layer, a second normalization pooling layer, an analytically computed third hidden layer, followed by another non-linear pooling layer, and a final fifth layer composed of a single threshold neuron. Our intuition is that because there is structure within protein sequences, and because we are dealing with sequences composed of characters, these structures and patterns are based on the k-mers within the sequences. Thus, a way to find and extract them, is through the use of string based activation functions, similar to methods applied in text mining. Because we do not know ahead of time the actual structures, lengths, and patterns of these k-mers, one way to solve the problem of exposing them is to generate a large number of our own random k-mer patterns, tiling the sequence with them, and counting how many, and which of our generated k-mers match the k-mers within the sequence being analysed. This k-mer tiling method extrapolates the protein sequence into a large feature space, which we can then cluster, separate, and classify through regression. In DRREP, we perform this regression step using the Moore-Penrose generalized inverse [72].

DRREP was tested on long protein sequences from the following datasets: SARS, Pellequer, HIV, AntiJen, and SEQ194. DRREP was compared to numerous state of the art epitope predictors, including the most recently published predictors called LBtope and DMNLBE. Using area under ROC curve (AUC), DRREP achieved a performance improvement over the best performing predictors on SARS (13.7%), HIV (8.9%), Pellequer (1.5%), and SEQ194 (3.1%), with its performance being matched only on the AntiJen dataset, by the LBtope predictor, where both DRREP and LBtope achieved an AUC of 0.702.

DRREP is an analytically trained deep neural network, thus capable of learning in a single step through regression. By combining the features of deep learning, string kernels, and convolutional networks, the system is able to perform residue-by-residue prediction of continues epitopes with higher accuracy than the current state of the art predictors.

4.1 A Text Mining Approach Through Random KMer Generation

Majority of the published epitope classifiers are based on Support Vector Machines [106], Neural Networks trained through backpropogation [14, 66], Naive Bayes Classifiers [25], and Propensity Scales [30]. A very limited number of the more obscure methods have also been explored, such as Ant Colony Optimization [23], for example. In the last decade a number of new and innovative classification and regression algorithms have been demonstrated and published, the most promising of which falls into the category of Deep Learning [35]. These types of systems are only now starting to be explored in the bioinformatics, and more concretely, the epitope prediction domain.

In this paper we develop a new epitope classification pipeline called Deep Ridge Regressed Epitope Predictor (DRREP). DRREP is a deep neural network which uses a string mismatch activation function, and is trained using an analytical method based on ridge regression. Because DRREP learns using an analytical method in a single step (going through the data only once), the system learns faster than SVM and other traditional iterative learning methods (exp. those based on error back-propagation).

4.1.1 Benchmark Datasets And Their Importance

There is a need to standardize epitope prediction benchmarking. Different papers discussing their predictors tend to use different test datasets. For example, BCPRED/FBCPRED used a short 193

residue SARS-COV sequence, BepiPred used an HIV dataset composed of 2706 residues, and BEST predictor used a large SEQ194 dataset, composed of 128180 residues. But the Accuracy and AUC achieved by a system on one dataset, can not be compared to the accuracy and AUC achieved by another system on a different dataset. This makes the task of comparing the different epitope predictors a bit difficult, requiring the re-application of the published predictors on some common datasets. Thus, we found the most current test datasets used by other state of the art predictors, and applied our system to those datasets, and when possible (when an epitope prediction server was available), applied the competing predictor to the test datasets it has not been applied to in its original paper. This allowed us to compare the AUC of different predictors on multiple datasets, each with very different epitope densities.

We have chosen to use the following 5 datasets: 1. SARS [24], which is a relatively short (193 residues) sequence, with a high epitope density. 2. HIV [46] dataset on which a number of other predictors have been tested and reported their AUC on, composed of 2706 residues. 3. SEQ194, which is a large dataset derived from BciPep, composed of 194 protein sequences, with a total of 128180 residues, and used as a test dataset by numerous predictors [28]. 4. AntiJen [96] used by BepiPred as a validation dataset. and 5. Pellequer [71], which was used as BepiPred's training dataset [51].

The SEQ194, HIV, Pellequer, and AntiJen sequences were all calculated by measuring the cross-reactivity between the intact protein and the peptide fragments. AntiJen and SEQ194 have extremely low epitope densities (1.4% and 6.6%, respectively); HIV and Pellequer have an order of magnitude higher epitope densities (37.1% and 37.6%, respectively); and SARS has the highest epitope density of the five datasets (63.7%). Thus, together these 5 datasets represent a realistic test of the classifier that is to be used to search for new epitopes within new protein sequences, covering a wide spectrum of possible epitope densities.

We also wanted a relatively common training dataset that has been used by other predictors, and which did not share any of its epitopes with the test datasets we have selected. We have searched through the literature and found that the BciPep [80] dataset has been utilized as a training dataset by a variety of predictors. The BCPred/FBCPred further pointed out some of the weaknesses within that dataset, producing a variation of it without protein sequence duplicities. A training dataset based on it was also used by the BEST predictor. Thus, given its common use, it represents a good training and validation dataset, and was chosen by us to train and validate DRREP on.

4.1.2 Training and Validation Dataset

DRREP was trained on the BCPred's 80% homology reduced dataset [8], which is itself a refined, homology reduced BciPep dataset [80]. The BCPred group based their dataset on the BciPep's shared 1230 unique linear B-Cell epitope dataset, by only keeping the 80% homology reduced subset. Afterwards, any epitope present in the subset that had a length less than 20 amino-acids, was extended/buffered on both sides by random anti-gen sequences from SwissProt [8]. This resulted in a new dataset composed of 1230 linear B-Cell epitopes, each of length *20* or greater. This dataset was further filtered to remove sequences that due to the buffering became too similar. The final dataset was composed of 701, 80% homology reduced sequences, each composed of 20 residues. For this 701 epitope sequence based dataset, non-epitope peptides were then generated by randomly extracting non-eptipe sequences from the SwissProt database, with the final dataset composed of 701 epitopes and 701 non-epitopes.

Finally, from this base dataset, the BCPred/FBCPred group generated 10 final datasets, composed of sequence sizes: 12, 14, 16, 18, 20, 22, 24, 26, 28, 30. To create the 22, 24, 26, 28, and 30 residue sized epitopes, the 20 residue sized epitopes and non-epitopes were extended on both ends. To create the 12, 14, 16, and 18 residue sized datasets, the 20 residue sized epitopes were truncated

on both ends. By creating these 10 different sized sequence length based dataset variations, the BCPred/FBCPred group was hoping to see how classification accuracy of a system changes when one changes the sliding window length. BCPred/FBCPred group made the original non homology reduced dataset, and the 10 derived datasets, available online at [110]. Because our system is also based on a sliding window method, and thus requires finding an optimal sliding window, we chose to train it using these 10 datasets.

4.1.3 Benchmark Measures

Our system can be applied to residue chains of any length by utilizing a sliding window approach that moves forward one residue at a time along the chain. Once it reaches the end of the entire protein sequence, it provides a score for each residue. Thus, benchmark measurements, accuracy, and AUC, are more fine grained and are based on the correctly predicted epitope residues, rather than correctly predicted epitopes. Those predicted residues which all fall into a single continuous sequence, are considered by DRREP to form a single continuous epitope. This classification approach allows DRREP to provide smoother decision boundaries and classify variable length eptiope and non-epitope sub-sequences within some large sequence to which it is applied, as opposed to providing scores for fixed length blocks of residues. Accuracy, Sensitivity, and Specificity, are calculated as follows:

Sensitivity $= TP/(TP + FN)$
Specificity $= TN/(TN + FP)$
Accuracy $= (TP + TN)/(TP + FP + TN + FN)$

where TP (True Positive), FP (False Positive), TN (True Negative), FN (False Negative), are residue based. The Receiver Operating Characteristic (ROC) plot is True Positive Rate (Sensi-

tivity) vs. False Positive Rate (1-Specificity), with AUC calculated as the area under the ROC curve. AUC has been demonstrated to be highly correlated with the general performance of a classifier, with a higher AUC being correlated with a classifier capable of high sensitivity, specificity, and accuracy.

4.2 Deep Ridge Regressed Epitope Predictor

The Deep Ridge Regressed Epitope Predictor (DRREP) is a deep neural network composed of 5 hidden layers, but only a single learning layer. The first layer is a randomly generated array of k-mers, used to perform feature extraction using basic string mismatch functions, with the mismatch number set to 0. Because the activation function just counts and outputs how many times a particular k-mer occurs in the input string, it can also be considered to be using a bagging method introduced by Leo Breiman [13]. But because each k-mer is slid across the entire input sequence, with the second neural layer performing a pooling computation, the first layer can also be considered as performing a convolutionary computation. The second layer is composed neurons which form a normalization pooling layer. The third is a layer of linear neurons, whose weights are set analytically using a simplified ridge regression algorithm [36]. The hidden weights of the linear neural layer are analytically computed using a matrix inverse, in our case, the Moore-Penrose generalized inverse, a method also used in a number of other machine learning algorithms [40, 81, 112, 4]. This is followed by a fourth scaled average pooling layer, and then a final fifth thresholding layer. This final fifth layer is composed of a single threshold neuron whose synaptic weights are deduced by DRREP using the validation scores of the sub-networks it is composed of, acquired during the training process. These validation scores are used to weigh the sub-network contributions to the final classification score. In essence, making the DRREP function as a type of ensemble of sub-networks.

In the following subsections we discuss the Moore-Penrose generalized inverse calculated synaptic weights, followed by a pseudo-code and a detailed discussion of the entire DRREP pipeline.

4.2.1 Calculating Synaptic Weights Analytically

DRREP can be applied to a continues sequence of any length, producing a score for each residue. The way DRREP does this internally is by using its sliding window to cut the long sequence into sub-sequences, score each subsequence, and then recompose the sub-sequences, averaging out the prediction for each subsequence such that the resulting longer sequence has a score for each residue. Thus, DRREP has a long sequence as input, and then it internally cuts it down to create a dataset of Y columns and X rows, where Y is the length of the sliding window used (chosen by the researcher during the training phase), $X = Tot\_Residues - SlidingWindowLength + 1$, and $Tot\_Residues$ is the total number of residues in the original long input sequence.

Each of these sliding window sized sub-sequences is passed through the first string function based layer and the second norm-pooling layer. The second layer outputs a matrix: **H**, which then acts as an input matrix to the third linear neural layer containing the synaptic weight matrix: β. During the training phase, the input data is labelled, and is usually composed of a dataset of sliding window sized sub-sequences, each of which is either an epitope or a non-epitope. Thus, we expect for the hidden linear neural layer to produce the expected training output (labels/classes) matrix: **E**, based on the available labelled input and β. We can calculate β by solving:

$$\mathbf{H}\beta = \mathbf{E}$$

where matrix **E** is composed of target labels, or in our case, epitope and non-epitope classes, and matrix **H** is composed of the output vectors of the 2nd pooling neural layer which is based on

the output of the first string function neural layer that processed the labelled input vectors. The optimal weight matrix of the linear hidden neural layer, β, is then computed through the use of Moore-Penrose generalized inverse, all in a single step, as follows:

$$\beta = \mathbf{H}^{\dagger}\mathbf{E}$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$.

Because the string function based first hidden neural layer which performs the extrapolation of the input data into the feature space, is randomly generated, and because regression is performed using the Moore-Penrose generalized inverse, the algorithm is fast, and is used here akin to the way it is used in [39]. Because there is no pre-training, or long phases of iterative training as is done in the more standard approaches based on gradient descent, it opens doors to potentially training the system on *big data*.

4.3 Training, Validation, and DRREP Construction

DRREP is a 5 layer deep neural network based on a parallel stack of independently trained 3 layer based sub-networks, each with a single learning layer, a randomly generated string (k-mer) based activation function first hidden layer, and a pooling transformational layer, as shown in Fig-4.1. Training is done in multiple phases. First, N number of 3 layer neural networks, called Sub_DRREPs, are generated and trained independently (each such Sub_DRREP network is composed of the DRREP's first 3 layers). The N networks are then stacked in parallel, with each network's output aggregated, and then normalized by the norm-pooling 4th layer. The normalized signals are then passed on to the threshold neuron in the 5th layer.

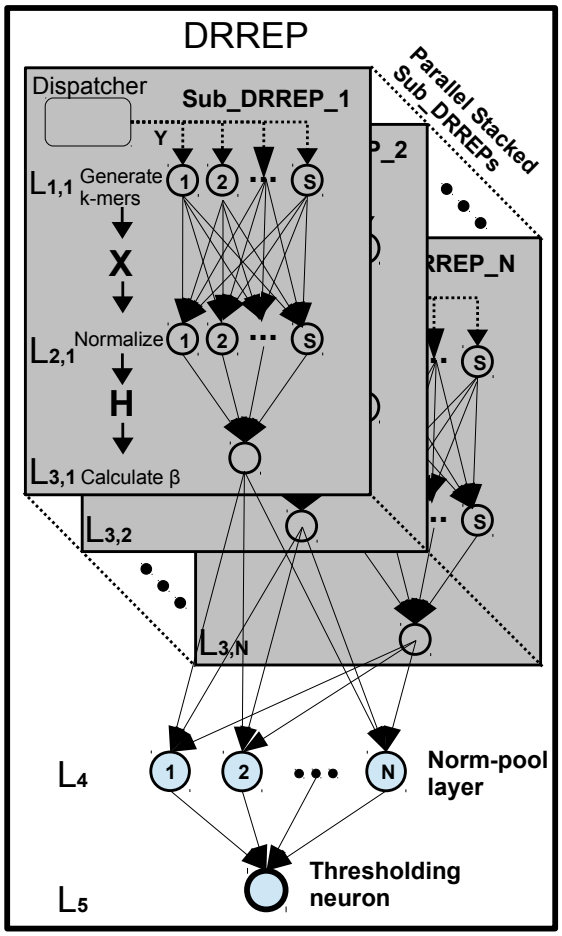

Figure 4.1: **DRREP Architecture** DRREP is composed of N Sub_DRREPs. Each Sub_DRREP's first layer, L1, is composed of S neurons, each of which is made out of a randomly generated k-mer based mismatch function. The second layer, L2, is composed of S nodes, with the entire layer performing normalization of the L1's signals. Layer L3, is the learning linear neural layer, whose synaptic weights are calculated using the Moore-Penrose generalized inverse. All N Sub_DRREPs are stacked in parallel. The L4 is a norm-pooling layer, composed of N nodes, which normalizes the signals from each Sub_DRREP. The next layer, L5, is composed of a single thresholding neuron, which weighs each contribution from the Sub_DRREPs based on that Sub_DRREP's relative validation score, and passes this value through the threshold to output the final score for the input sliding window.

The way this is performed, is by putting these sub-networks in parallel, to form a single, wider, deep network. Then the fourth layer is added, which normalizes and pools the outputs from these sub-networks. The fifth layer is composed of a single thresholding neuron. The scaling factor for

each sub-network is based on its relative validation AUC score, which act as weights for the single thresholding neuron in the final 5th layer, which decides whether the input vector belongs to an epitope. The DRREP pipeline is shown in Fig-4.2.

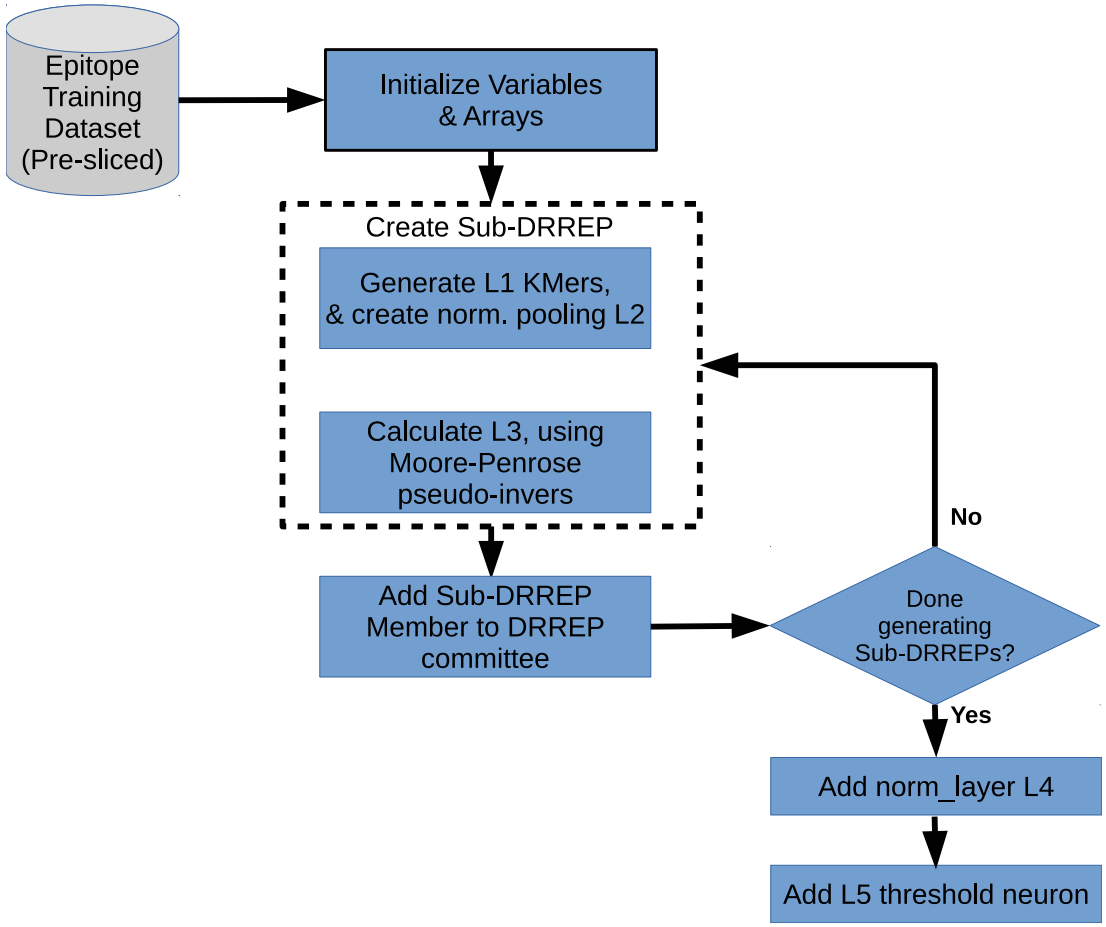

Figure 4.2: **DRREP Training Pipeline** DRREP composition and training pipeline.

The Sub_DRREP networks can be trained on input sliding windows of different sizes Y. We have explored sliding window sizes: $12, 14, 16, 18, 20, 22, 24, 26, 28$, and 30. Though DRREP can be composed of Sub_DRREPs of different sized sliding windows, in this paper we have explored composing DRREP where all Sub_DRREP networks use the same sized sliding windows. We have explored different values for Y, and different values for the parameter N (total number of

Sub-DRREPs), and settled on $Y = 24$ and $N = 20$, which resulted in the best validation score, and a DRREP that was fast to train.

DRREP makes its predictions purely based on the amino acid sequence. The first hidden layer in each Sub_DRREP is composed of a random number S of basic mismatch activation functions, each of which uses a randomly generated k-mer whose size ranges between 1 and the size of the sliding window Y. Based on our experiments, a string mismatch activation function which allows for 0 mismatches, produces the best results. Thus, each neuron using the basic mismatch activation function in the first layer counts the number of times its k-mer occurred in the sequence of the sliding window.

This allows the second normalization layer to calculate the proportions of various types of k-mers occurring within the window. Our intuition is that there are numerous small k-mers which are particularly antigenic, but we do not know which ones, or in which order and ratios they should be to trigger an immune response. Our system generates a large number of random k-mers, and through regression the system finds the correlation between the ratio and combination of the presence of these k-mers, and antigenicity.

Through meta-parameter optimization, DRREP was found to perform best (highest Validation dataset AUC) when for each Sub_DRREP, S was randomly chosen between 2 and 4000 (done in 2 bouts with a randomly generated value between 1 and 2000 for each). DRREP's sliding window moves through the long input sequence, and for each sliding window, DRREP's basic mismatch functions in the first hidden layer output the number of times their k-mer appeared in the window. The second pooling hidden layer in DRREP normalizes these scalar values, producing a k-mer ratio vector, and then passes this vector onwards to the 3rd layer. The third layer is the learning layer, whose synaptic weights are computed in a single step after all the training input vectors (windows) have been processed by the first 2 layers to produce the matrix $\mathbf{H}$. The synaptic weights

are computed using the Moore-Penrose generalized inverse, using the provided labels for the training dataset. This is done for each Sub_DRREP independently. Their (Sub_DRREPs) outputs are then passed onwards to layer 4, where they are pooled and normalized (this time, between the Sub_DRREPs, rather than the neurons within each single Sub_DRREP as was done in the 2nd hidden layer). Finally, the 5th layer is composed of a single threshold neuron with N weights, one for each Sub_DRREP. After the training and validation phases for the entire DRREP are completed, the synaptic weights for this neuron are set to the validation AUC scores of each Sub_DRREP, so that the voting contribution of each Sub_DRREP is based on its performance on the validation dataset. The neuron calculates its output in the standard linear neuron fashion, through the application of the dot product, resulting in the final output score. This output score can then further be passed through a threshold, so that the output is a classification rather than a score. By default, the threshold of the neuron is set to the mean score of the entire score sequence that DRREP produces (made possible by DRREP first calculating all the scores, and then calculating the threshold based on the mean).

4.3.1 The Pipeline

First a training dataset is $90/10$ split into subsets, with 90% of the total dataset used for training, and 10% set aside to be used for validation. The training dataset is designated by the input dataset and its expected labels/classes as: (Trn_I, Trn_Exp) and validation dataset with its labels/classes as: (Val_I, Val_Exp). I and Exp postfixes designate **I**nput and **Exp**ected (label) matrices of the dataset. The 3rd hidden layer in each Sub_DRREP is composed and trained using the method shown in Fig-4.3.

Algorithm 1 Training DRREP using the Training dataset, and validating it using the Validation dataset, both using epitope and non-epitope vectors of size 24.

1 Initialize $N, Y, J, (\textbf{Trn\_I}, \textbf{Trn\_Exp}), (\textbf{Val\_I}, \textbf{Val\_Exp})$, where: N = 20, Y = 24, J = 2, **Trn_I** = Training data input, **Trn_Exp** = Training labels, **Val_I** = Validation data input, **Val_Exp** = Validation labels

2 **L1, L2, L3, L4, L5** $\leftarrow []$

3 $Old\_AUC \leftarrow 0$

4 For every **Sub_DRREP_i** in N, perform:

(a) Loop J number of times:

1. Generate $rand(2000)$ of mismatch activation functions, each with a randomly generated k-mer of size $rand(Y)$, and append this vector to **L1**

2. Generate a list of $length(\textbf{L1})$ nodes, where each node $Node\_k$ performs a normalization function of the signals coming from **L1**, and outputs a normalized signal of neuron $L1\_k$, for k ranging from 1 to $length(\textbf{L1})$

3. Calculate the hidden layer output vector for **Trn_I** by first passing this training input matrix through **L1**, and then normalizing the **L1** output by passing it through **L2**, producing matrix **H**

4. Calculate synaptic weights β for the **Sub_DRREP_i**'s neuron in **L3**, given the labelled data **Trn_Exp**, and the hidden layer output matrix **H**, using Moore-Penrose generalized inverse, s.t. $\beta \leftarrow \textbf{H}^{\dagger}\textbf{Trn\_Exp}$

5. Use the now formed **L1, L2, L3** to calculate the vector **Output** of epitope prediction scores for **Val_I**

6. Compare **Output** for **Val_I** to **Val_Exp**, and calculate AUC

7. If AUC is higher than Old_AUC, keep current **L1, L2, L3**, set Old_AUC to AUC, and loop. Otherwise remove from **L1** the newly added mismatch activation functions, reset **L2, L3** to empty lists, and loop

(b) **SubDRREP_i** $\leftarrow$ **[L1,L2,L3]**

5 Form **DRREP**'s first 3 layers by parallel stacking **Sub_DRREP**s

6 Create **DRREP**'s **L4** normalization pooling layer composed of N nodes

7 Create **DRREP**'s **L5** threshold neuron composed of N weights

Figure 4.3: **Training DRREP** The figure presents the algorithm used to train DRREP.

Once DRREP is trained and validated using the provided dataset, it can then be used for epitope discovery and classification by applying it to protein sequence datasets, as shown in Fig-4.4.

DRREP can be updated with new Sub_DRREP networks over time, as new training data becomes available. This is done by simply stacking the new sub-networks in parallel with the existing sub-networks within the DRREP pipeline. In a similar fashion, sub-networks can also be removed if needed (exp. a Sub_DRREP is found to contribute negatively to the final prediction).

Algorithm 2 Loading & applying DRREP to an unknown sequence.

1. Open the file where DRREP parameters are stored.
2. Open the sequence file.
3. **Sequence** $\leftarrow load(Sequance\_FileName)$
4. Initialize the score matrix **Output_M** to an empty matrix
5. $SeqLen \leftarrow length($**Sequence**$)$
6. For every **Sub_DRREP** in **DRREP** do:
   (a) Load **Sub_DRREP**
   (b) Calculate the **Output** vector of prediction scores for the sequence using the **Sub_DRREP**
   (c) Add the **Output** score vector to the **Output_M** score matrix.
7. Pass the **Output_M** matrix through **DRREP**'s 4th norm-pooling layer to produce a normalized **Output_M** score matrix
8. Pass the normalized **Output_M** score matrix through **DRREP**'s scaling and thresholding 5th layer neuron to produce a rescaled and summed version (using a standard dot product, where each weight is based on the correlated **Sub_DRREP**'s validation score), and then pass it through the neuron's thresholding function to produce the final **Score_Vector**
9. **Return Score_Vector**

Figure 4.4: **Loading and Applying DRREP** The figure presents the algorithm used to load and apply DRREP.

DRREP was first optimized with regards to its meta-parameters. We explored multiple sliding window sizes, and multiple first hidden layer sizes, and optimal number of Sub_DRREPs to form the DRREP. We found that sliding window of size 24, with 20 total Sub_DRREPs, each composed of around 4000 randomly generated string mismatch functions in its first layer, produced the highest validation AUC. Once the meta-parameters were optimized based on the best validation AUC score, the system was then tested by being applied to long continuous protein sequences. DRREP was implemented using JuliaLang, a high performance technical computing programming language. But because DRREP is composed of nearly 80000 first hidden layer neurons, and stored in human readable XML format, there is roughly a 40 second overhead in loading the system into memory, which is only done once.

4.4 Experiment Results

The DRREP pipeline was applied to 5 datasets (SARS, HIV, Pellequer, AntiJen, and SEQ194) composed of long continuous protein sequences, with the AUC and accuracy at 75% specificity shown in Table 4.1. In the same table, we also list the AUC scores reported by other epitope predictors, such as the self reported AUC values of BCPred on the SARS dataset, BepiPred on the HIV dataset, and multiple systems on the SEQ194 dataset. Where possible, we ran the server-available predictors on the SARS and HIV datasets, these included the CBTOPE, Epitopia, ABCPred, LBtope, and DMN-LBE predictors.

CBTOPE and Epitopia servers produced score based outputs, whereas ABCPred produced a list of index start locations of the predicted 16-sized window based epitopes. This required a conversion to a single residue score based format, and was performed by using the highest epitope score for each residue's location. We have also used the CBTOPE and ABCPred servers to calculate scores for the HIV dataset. We had difficulty running Epitopia on the longer HIV dataset, as the server produced run-errors. Also, unfortunately, DMN-LBE server predicts one sequence at a time. Thus, due to the large number of sequences AntiJen and SEQ194 datasets are composed of, we were only able to run the smaller SARS, HIV, and Pellequer datasets on the server (doing so manually, one sequence at a time). For the missing predictors, or where the AUC scores are not listed in the table, we did not get a response from the authors as to the proper conversion from the output format produced by their predictor, to the score based format we needed to calculate AUC and accuracies, or the server for that predictor was not available. Nevertheless, we applied DRREP to every dataset (without retraining), so that it could be compared to every system which was originally tested on it.

Table 4.1: Accuracy and AUC results of applying DRREP to long protein sequence datasets, and the AUC results of other epitope predictors.

DataSet	Tot Residues	Epitope%	System	75spec	AUC
SARS	193	63.3	**DRREP**	**86.0**	**0.862**
			BCPred	80.3	_
			ABCPred	67.9	0.648
			Epitopia	67.2	0.644
			CBTOPE	75.6	0.602
			LBtope	65.8	0.758
			DMN-LBE	59.1	0.561
HIV	2706	37.1	**DRREP**	61.4	**0.683**
			BepiPred	_	0.60
			ABCPred	61.2	0.55
			CBTOPE	60.4	0.506
			LBtope	61.2	0.627
			DMN-LBE	**63.6**	0.63
Pellequer	2541	37.6	**DRREP**	62.7	**0.629**
			LBtope	60.9	0.62
			DMN-LBE	**62.8**	0.61
AntiJen	66319	1.4	**DRREP**	73.0	**0.702**
			LBtope	**74.2**	**0.702**
			DMN-LBE	_	_
SEQ194	128180	6.6	**DRREP**	**75.9**	**0.732**
			Epitopia	_	0.59
			BEST10	_	0.57
			BEST16	_	0.57
			ABCPred	_	0.55
			CBTOPE	_	0.52
			COBEpro	_	0.55
			LBtope	73	0.57
			DMN-LBE	_	_

Fig-4.5 demonstrates the type of output DRREP provides when using the classification threshold, rather than simply outputting a list of residue scores. The figure shows the SARS sequence, with the line (True) showing the actual epitope locations, line (DRREP) shows DRREP's epitope predictions when using a 75% specificity threshold, line (BCPred) designates BCPred's predicted

epitopes, line (ABCPred) designates ABCPred's predicted epitopes, and similarly for EPitopia, CBTOPE, and finally LBtope. The true epitope locations are coloured green, and for each predictor, the incorrect predictions are coloured in red, and the correct are coloured blue.

4.4.1 Discussion

When applied to the SARS sequence, DRREP achieved an AUC of 0.862, and an accuracy of 86.0% at specificity set to 0.75. BCPred reported an accuracy of 80.3% at the same specificity. We also used the ABCPred, Epitopia, CBTOPE, LBtope, and DMN-LBE servers to generate predictions for the SARS sequence. Their resulting AUCs were 0.648, 0.644, 0.602, 0.758, and 0.561 respectively. Their accuracies calculated at a specificity set to 0.75 are also listed in the table. DRREP achieved a higher accuracy and AUC than all competing predictors on this sequence. This is an improvement in accuracy of 5.7% over BCPred, and an AUC improvement over the best performing predictor on that dataset (LBtope here, because BCPred did not report it's AUC for this sequence) of over 13.7%.

Furthermore, from Fig-4.4 we can see that DRREP predicted correctly a larger number of residues than other predictors. But, DRREP classified the four sub-sequences as all belonging to a single epitope. This could potentially be alleviated by adding a post-processing filter which calculates not just a score, but changes in a score as well. We base this hypothesis on the fact that we observed a number of cases where the score transitioned significantly between continuous sequences of epitope and non-epitope sequences, yet still held above the epitope threshold for both cases. Based on this observation, perhaps the system could be further improved by taking into account radical score transitions. This methodology is planned to be explored in our future research. From all the test datasets on which LBtope was tested, it performed the worst on SARS. For the tests performed, the version of LBtope used was based on the one trained on a fixed 20 residue win-

dow based dataset, which was used in the original LBtope publication. When using this version, the server does not predict the last residue, hence it was designated with an asterisk. An LBtope trained on flexible window based original dataset was also tested on SARS, because that version does predict the last residue, but it performed much worse than the version shown, and thus was not included. The Epitopia server also did not provide the classification for the 193rd residue in the SARS sequence. Another interesting anomaly in residue prediction results was produced by ABCPred. ABCPred server gives a score for 16 residue slices, with a default epitope score threshold set to 0.5. Based on this, ABCPred classified numerous such 16 residue slices as epitopes, and when combined together, this included all but the first two residues in the SARS sequence.

When applied to the HIV dataset, DRREP produced an AUC of 0.683. We can compare it to LBtope and DMN-LBE, which were also tested on the HIV dataset, their AUCs were 0.627 and 0.63 respectively. We ran ABCPred, BepiPred, and CBTOPE servers on the same dataset, and their resulting AUCs were 0.60, 0.55 and 0.506, respectively. Thus, DRREP achieves an AUC higher by 0.053 than the best predictor in the list (DMN-LBE), an improvement of 8.4%. Interestingly, at 75% specificity DMN-LBE had a higher accuracy.

BepiPred was trained on the Pellequer dataset, and was thus disqualified from being compared on it. We had a difficult time running this dataset on multiple predictor servers, and neither could we find their performance on this particular dataset within published literature. The only server we were able to run on the dataset was LBtope, which achieved an AUC of 0.62, which is 3.2% lower than DRREP's AUC of 0.629. On Pellequer DMN-LBe, though having a lower AUC score, at 75% specificy achieved an accuracy of 62.8%, which was .1% higher than DRREP.

AntiJen is a dataset much larger than HIV and Pellequer, and thus we could not get it to run on some of the listed predictor servers, nor find their published performance on this dataset. The only server that allowed us to run such a large dataset was LBtope. DRREP and LBtope tied on this

dataset with an AUC of 0.702. LBtope did achieve a 1.2% higher accuracy at 75% specificity.

DRREP achieved an AUC of 0.732 on the SEQ194 dataset, which we compared to Epitopia, ABCPred, CBTOPE, and COBEpro, whose AUCs on this dataset were acquired from [77], and BEST10/16 system whose AUC is listed in [28]. This dataset was also ran on the LBtope server, which achieved an AUC of 0.71 when calculating per window, and 0.57 when calculating per residue by averaging the window scores. DRREP achieved an AUC performance improvement of 5.6% over the best performing predictor (LBtope). It should be noted that the dataset SEQ194 is the most recently published of all datasets, with the largest number of long, FASTA encoded sequences. Furthermore, LBtope was ran using the *LBtope_Fixed_non_redundant (non redundant dataset)* version, which was the one reported in their most recent paper, and we considered to be the best performer of the versions available.

Thus, DRREP achieved a higher AUC performance on 4 of 5 datasets than all other predictors, and particularly the state of the art LBtope and DMN-LBE predictors. DRREP tied with the LBtope on the remaining fifth dataset, the AntiJen dataset. And though DMN-LBE achieved a higher accuracy at 75% specificity on the HIV dataset, and parity on Pellequer, it will not be possible to know the 75% specificity threshold when the systems are applied to new and unknown sequences, thus AUC is still the best indicator of system's general performance. These results demonstrate DRREP's markedly higher general performance.

4.5 Conclusion

In this chapter we have presented a novel deep network based classifier using a string activation function based first layer, multiple non-linear transformational pooling layers, and a single learning layer. The learning layer synaptic weights are calculated analytically using the Moore-Penrose

generalized inverse, making the training phase faster than that of SVM, and standard gradient descent based models. When DRREP was applied to the SARS sequence, the achieved classification accuracy at 75% specificity was 86.0%, which is 5.7% higher than the BCPred/FBCPred, it's AUC was higher by over 13.7% than that of LBtope on the same sequence. When applied to the HIV, Pellequer, and SEQ194 datasets, DRREP achieved an AUC performance improvement of 8.4%, 3.2%, and 5.6% respectively, over the best performing predictors in the list, which were the most recently published DMN-LBE and LBtope predictors. The only dataset on which DRREP did not achieve a performance improvement was the AntiJen dataset, on which both DRREP and LBtope achieved the same AUC score. We believe that these results represent a substantial and highly regular and stable improvement over the current state of the art.

DRREP is a promising new method. Its generalization capabilities are stable across all tested datasets, with different levels of epitope densities. We plan to further improve DRREP's performance by incorporating new advancements within the deep learning domain, by further exploring convolutional layering, local receptive field layering, and other types of topologies and pooling paradigms. We also plan to further explore the effect of training dataset refinement on the system's performance. The DRREP system [84] and its datasets [85], are freely available on the GitHub server, and can be downloaded from the referenced URLs.

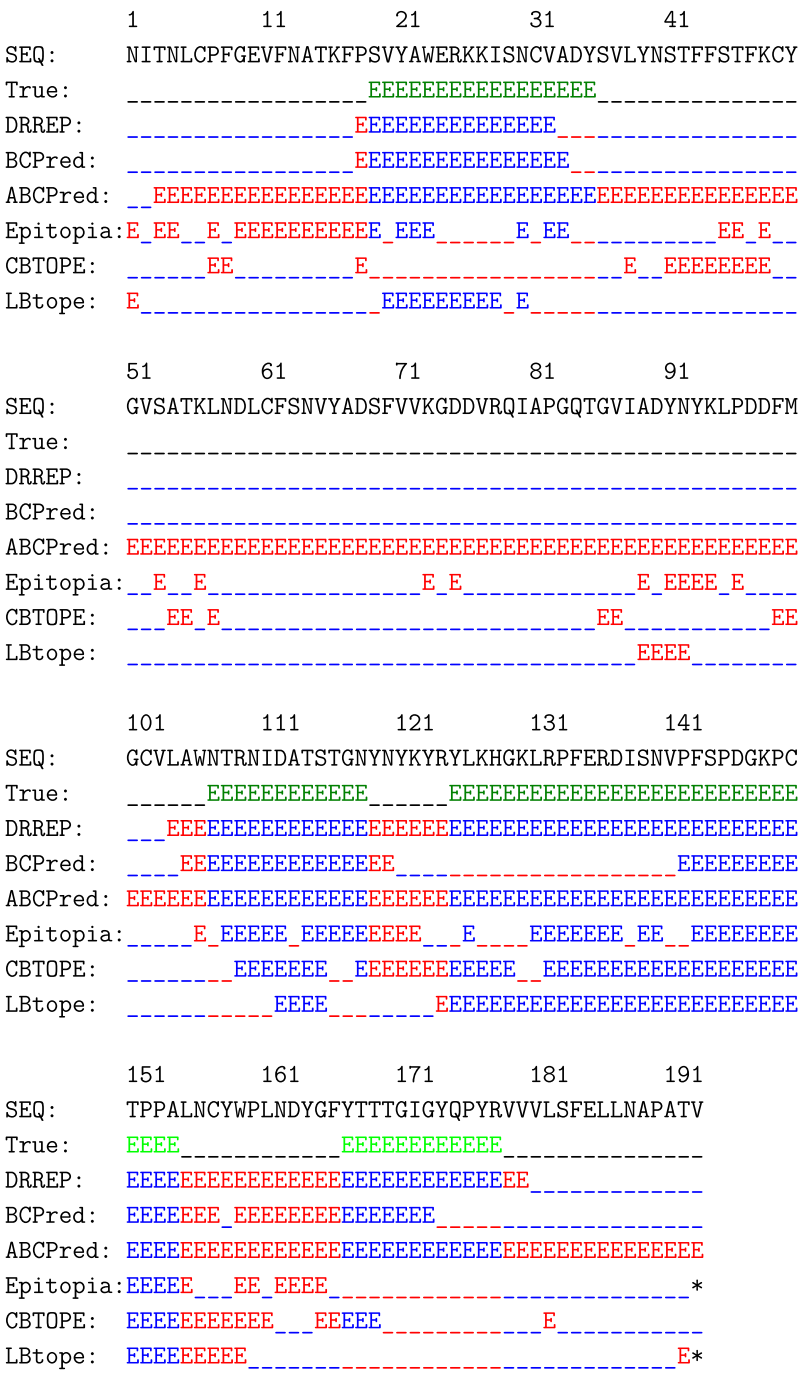

Figure 4.5: **Predicting Epitopes in SARS** SARS sequence (SEQ), true epitope locations (True), DRREP's predicted epitopes (DRREP), BCPred's predicted epitopes (BCPred), ABCPred's predicted epitopes (ABCPred), EPitopia's predicted epitiopes (Epitopia), CBTOPE's predicted epitopes (CBTOPE), and LBtope's predicted epitopes (LBTope). The incorrect predictions are colored in red, the correct are colored in blue, and the true epitope locations are colored green.

CHAPTER 5: SEEP: CONJOINT TRIAD FEATURE BASED EPITOPE PREDICTOR

Epitopes are part of an antigen relevant for recognition by immune molecules. The domain of epitope prediction is intertwined with that of protein-protein interaction (PPI). Most of the tools built for epitope prediction are not taking advantage of the vast amounts of domain specific knowledge discovered within the PPI sector. In this paper we present a system which leverages the domain specific knowledge acquired within the PPI research domain, and apply it to the prediction of B-Cell linear epitopes. We demonstrate how the Conjoint Triad Feature introduced in 2007 within the PPI sector, is just as useful within the epitope prediction domain. We developed an **S**VM **E**nsemble **E**pitope **P**redictor (SEEP), based on the Conjoint Triad Feature (CTF) encoding, and demonstrate its substantial performance improvement over the current state of the art. SEEP accepts a protein sequence, recodes it into CTF ratio vectors, and then produces a score for each residue in the sequence through a consensus of SVM predictors. SEEP achieved an AUC of **0.912** on the standard SEQ194 test dataset, an improvement of 24% over the current state of the art. Furthermore, we applied the system to the Zika virus, showing that the system is capable of recognizing the currently known linear B-Cell epitopes, and pointing us towards new epitope locations which should be further investigated experimentally.

5.1 Improved Data Represetation Through Domain Knowledge

Though 90% of all epitopes are conformational [98], the tools which perform conformational epitope prediction are usually two-stage systems, which first predict linear epitope sequences, and then using a second clustering stage deduce which of the amino acid subsequences belong together to form a single conformational epitope. Thus, linear epitope prediction is at the core of both, lin-

ear and conformational epitope prediction tools. For this reason, we developed SEEP to be a linear epitope predictor, with potential extensions of using it in future conformational epitope prediction pipelines.

It is well accepted that Sequence specifies structure [5], thus it should theoretically be possible to predict protein interaction, and by extension epitope prediction, from the protein sequence alone. And though having more information about the 3D structure might provide some advantages, as of this writing, there is still substantially more accurate sequence based data than there is 3D. Thus, because we expect that most of the structure is indeed specified by the sequence, and because there is so much more of curated protein sequence information than 3D, an accurate and fast sequence based epitope predictor is of great use.

5.1.1 Domain Specific Knowledge In Data Representation

As of this writing, majority of the existing epitope predictors use only the protein sequence as input. Numerous machine learning approaches and protein sequence representations have been explored over the past decade. Some of the first such systems were primarily reliant on propensity scales [57, 37], but the predictive results were only slightly better than random [12]. Starting in 2006, papers which leveraged machine learning began to surface, starting with ABCpred [79] which utilized a backpropagation feed-forward neural network (FFNN). Other approaches were being explored shortly thereafter, ranging from Hidden Markov Models (HMM) [10] and Naive-Bayes methods [105], to Support Vector Machines (SVM) [17, 105, 25, 113, 16], and now even deep learning [109].

Currently, most of the methods within the epitope prediction domain are based on support vector machines. Some methods apply SVMs to protein sequences represented by propensity scale based vectors, such as the LEP-LP [16] predictor for example. Other methods have also began to utilize

research ideas from the text mining community [55, 56]. In 2010 BCPred [25] was published, which uses string kernels. A year later FBCPred [113] was released, which predicts variable length epitopes. The reason such a large number of modern epitope prediction systems use SVMs is not a coincidence. Rather, it is because of this method's excellent ability to generalize well when learning on datasets smaller (keeping all other things equal) than those needed by neural network or genetic algorithm based approaches. At the same time, SVMs have shown to also generalize well when dealing with high dimensional input, through an easily tunable regularization parameter. All of these issues are being faced within the epitope prediction problem, due to the type of data being analysed. One of the main issues we are facing, as was noted by Singh et al. [90], is that the amount of curated and highly accurate data is small (even though there is a lot of published data), while the methods we use to encode protein sequences, in many cases result in a high dimensional representation.

During the same time frame, researchers working on Protein-Protein Interaction (PPI) mapping, have also been concentrating on finding useful ways to represent protein sequences. Similarly to the epitope prediction community, the PPI research community also came to the conclusion that there is substantially more sequence rather than 3D based data, and an amino acid based PPI predictor would be of great utility [83, 15, 107]. Thus, both research communities have been battling very similar problems, and working on very similar data types, data representation, and prediction problems.

Though the binding of paratopes to epitopes does partially fall under the category of Protein-Protein Interaction (PPI), until now none of the noted epitope prediction systems have taken advantage of the vast amounts of knowledge that has been acquired within the PPI research with regards to sequence representation. Majority of the PPI predictors are mainly based on protein sequences. Almost all current PPI predictors [83, 15, 107, 31, 103, 111] make use of the Conjoint-Triad Feature (CTF) at some point in their prediction pipeline. CTF was developed and published

back in 2007 by Shen et al, and has been used by almost all PPI predictors ever since.

CTF groups the amino acids based on their biochemical properties. CTF forms seven amino acid clusters based on the dipoles and side chain volumes of the residues. The encoding further has the property of suiting synonymous mutation, and abstracting the features of protein pairs. The 7 groups made for the 20 amino acids are shown in Table 5.1.

Table 5.1: **Conjoint Triad Feature grouping.** Each group contains amino acids which share similar dipoles and side chain volume properties.

Group	Amino Acid Group
1	Ala, Gly, Val
2	Ile, Leu, Phe, Pro
3	Tyr, Met, Thr, Ser
4	His, Asn, Gln, Trp
5	Arg, Lys
5	Asp, Glu
7	Cys

There are a total of 343 (7x7x7) trimer combinations when using these 7 groups. The way SEEP encodes the protein sequence, discussed in greater detail within the methods section, is by converting it into a ratio vector of these trimers. This means that each sequence presented to the machine learning system has a dimensionality of 343. Furthermore, by using triads, the encoding also incorporates some local information, and amino acid ordering.

In our previous work [88], the DRREP pipeline utilized a string kernel, which virtually reduced each neuron in the first layer of the neural network to a single input rectifier. This worked well with the learning algorithm used, but unfortunately, the same learning algorithm tends to require a large dataset to train on, as the dimensionality of the input vector increases, otherwise it overtrains. Comparatively, SVM approaches have demonstrated higher generalization capabilities [20, 104, 60] when the amount of data is sparse yet dimensionality of the input is relatively high. Thus,

due to the type of data and encoding we utilize, we chose to use SVM as part of our prediction pipeline.

We leverage the domain knowledge acquired within the PPI field, the now well tested and highly utilized CTF encoding, and the data Pre and Post processing method we developed in our previous work [88], further modified by our min-pooling consensus algorithm here, to create an SVM Ensemble based Epitope Predictor (SEEP). This new sequence based epitope predictor has achieved an AUC of 0.912 on the large SEQ194 test dataset [28], on which all other most recently published state of the art epitope predictors, such as LBTOPE [90], BEST [28], and our previously published system called DRREP, achieve a maximum AUC of 0.732.

Furthermore, we have applied SEEP to the Zika virus, and have demonstrated that our system does see the overlapping epitope sequences published on IEDB, and show where else on the polyprotein sequence our system predicts a high probability of containing epitopes. Finally, we have made SEEP freely available [86], which accepts FASTA encoded files, providing both: a per residue score, and a list of continuous linear epitopes.

5.2 The Committee Based Multi-Resolution CTF Encoding Approach

SEEP leverages the domain specific knowledge derived CTF encoding, a pre and post processing method we introduced in our previous works to calculate per-residue prediction, a consensus algorithm to improve the system's committee performance, a curated and tested training dataset, and a committee of multi-resolution predictor systems based on SVM, which was chosen due to its generalization ability when used with a small training dataset relative to the dimensionality of the input.

5.2.1 Datasets

SEEP was trained on the BCPred training dataset, which itself was based on the Bcipep [80] database. It contains 1230 unique linear B-Cell epitopes, from which an 80% homology reduced set of 701 positive and 701 negative peptides were extracted. Furthermore, the epitopes longer than X residues were truncated equally from both sides, and those shorter than X residues were extended based on their corresponding full antigen sequences retrieved from SwissProt [8]. X was set to 12, 14, 16, 18, 20, 22, 24, 26, 28, and 30, to produce a total of 10 datasets. This was done to explore which slice size produces the best results. In our case, to create an ensemble of multi-resolution trained predictors, a predictor was trained on each slice size based dataset, and then a committee was formed using the top 60% of the predictors (the percentage was derived during the optimization phase). Ensemble systems [21] have been demonstrated to produce superior performance as compared to singular predictors in numerous applications. We too have found that for this application, an ensemble performs better than any individual member forming it.

We used the HIV [46] dataset for validation. This dataset is composed of a total of 2706 residues, with an epitope density of 37.1%. The sequence was generated by measuring the cross-reactivity between the intact protein and the peptide fragments.

SEEP was tested on the SEQ194 dataset. SEQ194 is a low epitope density dataset, containing only 6.6% of epitopes. Given the low epitope density in standard protein sequences, this test dataset is a good indicator of how the system will perform in general. SEQ194 contains a total of 128245 residues, and has been used as the test dataset for all recently published state of the art B-Cell epitope predictors (BEST, LBTope, MN-LBE, and DRREP). The SEEP system was compared to these and other predictors on this dataset to demonstrate its relative performance.

Finally, we also ran SEEP on the ZIKA virus polyprotein, *Acc. ALU33341.1*. IEDB lists 7 contin-

uous B-Cell epitopes, each with a single reference and based on 2 or more assays. All 7 epitopes were overlapping, together spanning the sequence: IAPAYSIRCIGVSNRDFV.

5.2.2 Benchmark Measurements

The benchmark measurements, AUC, Accuracy, Sensitivity and Specificity, are all per-residue based. SEEP calculates an epitope score for each residue rather than for a window of some size N. This is done by the post-processing method that is part of the SEEP pipeline, and allows for the system to mark variable length epitopes rather than provide guesses whether there is or not an epitope within some fixed sized window. The measurements are calculated as follows:

$Sensitivity = TP/(TP + FN)$

$Specificity = TN/(TN + FP)$

$Accuracy = (TP + TN)/(TP + FP + TN + FN)$

Where, TP is True Positive, FP is False Positive, TN is True Negative, FN is False Negative, and are calculated per residue. The Receiver Operating Characteristic (ROC) curve is calculated by plotting True Positive Rate (Sensitivity) vs. False Positive Rate (1-Specificity). AUC is calculated as the area under ROC curve. We use AUC as the main benchmark of the system's performance because it has been shown to be highly correlated with the performance of a classification system, and is the standard benchmark measurement used by all other predictors.

5.2.3 SEEP Pipeline

SEEP accepts as input a protein sequence, and outputs a sequence of scores, a vector with a score for each residue within the input sequence. The scores are between 1 (most likely belongs to an epitope) and -1 (least likely to belong to an epitope). Internally each member of the committee within SEEP slices the sequence into subsequences of length N, where N is specific to each par-

ticular SVM member, and depends on the resolution it was trained on. Afterwards, each residue within the slice is re-encoded into the 7 groups presented in Table 5.1. Finally, based on these 7-group based vectors, the system calculates the CTF ratio for each slice, and these CTF ratio vectors are then used as input to the committee members. The process is separated into the following 7 steps, and summarized in Figure 5.1:

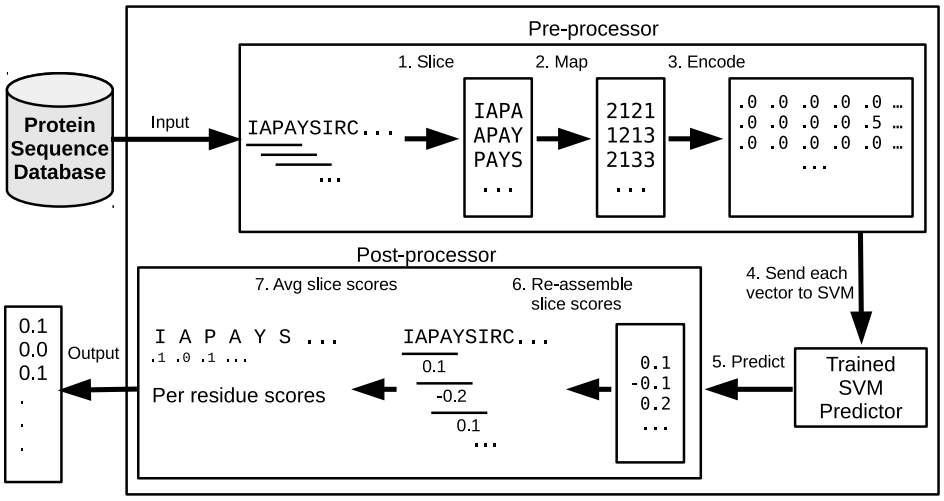

Figure 5.1: **SEEP Member** The architecture of the SEEP committee member, with its pre and post processing.

Step-1 Cut the amino acid sequence of length N into an array of $N - K + 1$ slices, each of length K, by scanning the sliding window along the protein sequence.

Step-2 For each slice, map the residues into their respective classes. As per the CTF descriptor, the 20 amino acid types are clustered into the groups: $G_1 = \{A, G, V\}, G_2 = \{I, L, F, P\}, G_3 = \{Y, M, T, S\}, G_4 = \{H, N, Q, W\}, G_5 = \{R, K\}, G_6 = \{D, E\}$, and $G7 = \{C\}$. Every residue P_i is mapped to its respective group G.

Step-3 The conjoint triad is composed of 3 continuous groups. There are a total of 343 $(7x7x7)$ trimer permutations. We count the occurrence number of each trimer within the slice, and then

divide each occurrence value by the length of the slice to produce a trimer ratio vector with a dimension of 343.

Step-4 Each trimer ratio vector is then sent to SVM for classification.

Step-5 The SVM predicts a score for each trimer ratio vector.

Step-6 Because each score vector spans a slice, spanning score vectors are re-arranged under their respective sequence positions.

Step-7 Each residue is given a score which is the average of the scores whose spans cover it.

Each member of the SVM committee calculates a score for each window, which are then reassembled into a single sequence, averaging the scores between the slices, as shown in Figure 5.2, to produce a single continuous score sequence. The system then calculates a consensus for each residue between the produced score sequences of the committee members. This is done by Min-Pooling the votes between all committee member predictions. This final consensus sequence of scores is then output by the system. To produce a strictly binary output, further threshold can be used to mark each residue as belonging to an epitope or not. This threshold can be based on the specificity one wishes the system to operate with.

Max, *Mean*, and *Min* pooling have been standard operators within deep learning. A pooling can be considered as a type of consensus calculation. In deep learning, over time, mean-pooling has been replaced by max-pooling [82] and min-pooling [26] which have been shown to work better in practice, for certain problem domains. We have tested different consensus algorithms, and found that min-pooling outperforms mean-pooling and max-pooling in this application. We argue that similarly to findings made by Eldar H. et al., in our application too, this particular pooling results in establishing a ”minimum reliable scale” for the detection of sparsely present signals (epitopes).

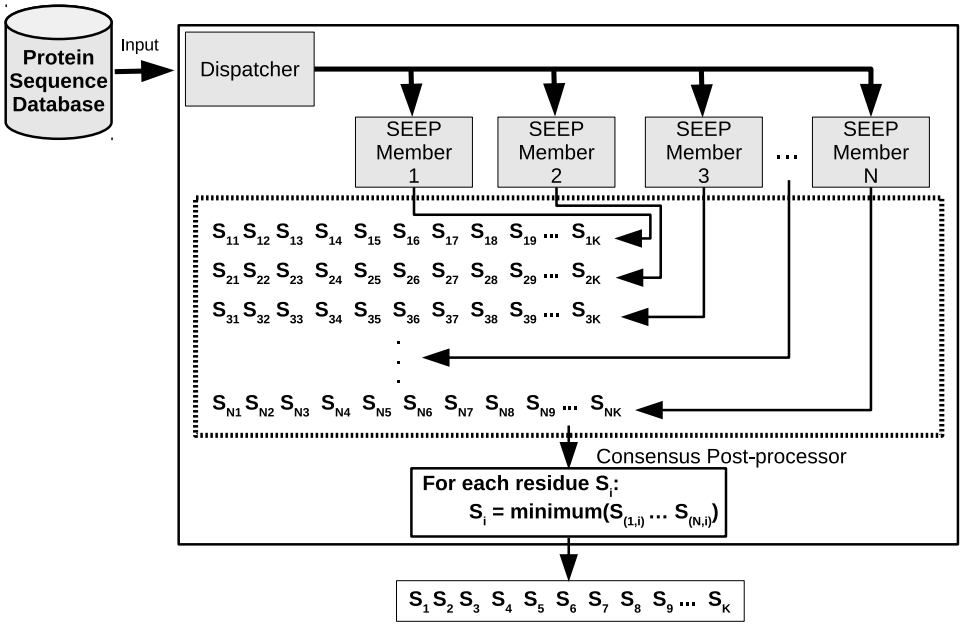

Figure 5.2: **SEEP Data-Flow Architecture** The data-flow architecture of the entire SEEP system.

5.2.4 Training and Validation

We use 10 different versions of the training dataset, composed of the slice sizes: 12, 14, 16, 18, 20, 22, 24, 26, 28, and 30. The sequences in each version of the dataset are converted into CTF ratio vectors, and then a different SVM is trained on each dataset separately. We used the SVMLight [43] package for the SVM predictors composing the members within the SEEP committee machine. HIV was used as the validation dataset, with the metaparameters for each SVM tuned using grid search, which found that the best kernel for this particular encoding was Polynomial, with all parameters set to default, and C set to 32.

We leverage the 10 different ”resolution” versions of the dataset for training, such that the final committee is based on members trained on different sliding window sizes. Our intuition is based on these different resolutions will pick up and better suited for different length epitopes. Using a pooling method, we can then optimize the final classification, based on the cumulative prediction

of the individual multi-resolution members.

Once all the SVM systems were trained, we further chose only the top 60% of the members, from which a committee was formed. This was done due to a large drop in validation performance for predictors which used slice sizes of length lower than 20. This resulted in a final committee composed of 6 SVMs, with window sizes 20, 22, 24, 26, 28 and 30. The training pipeline is shown in Figure 5.3.

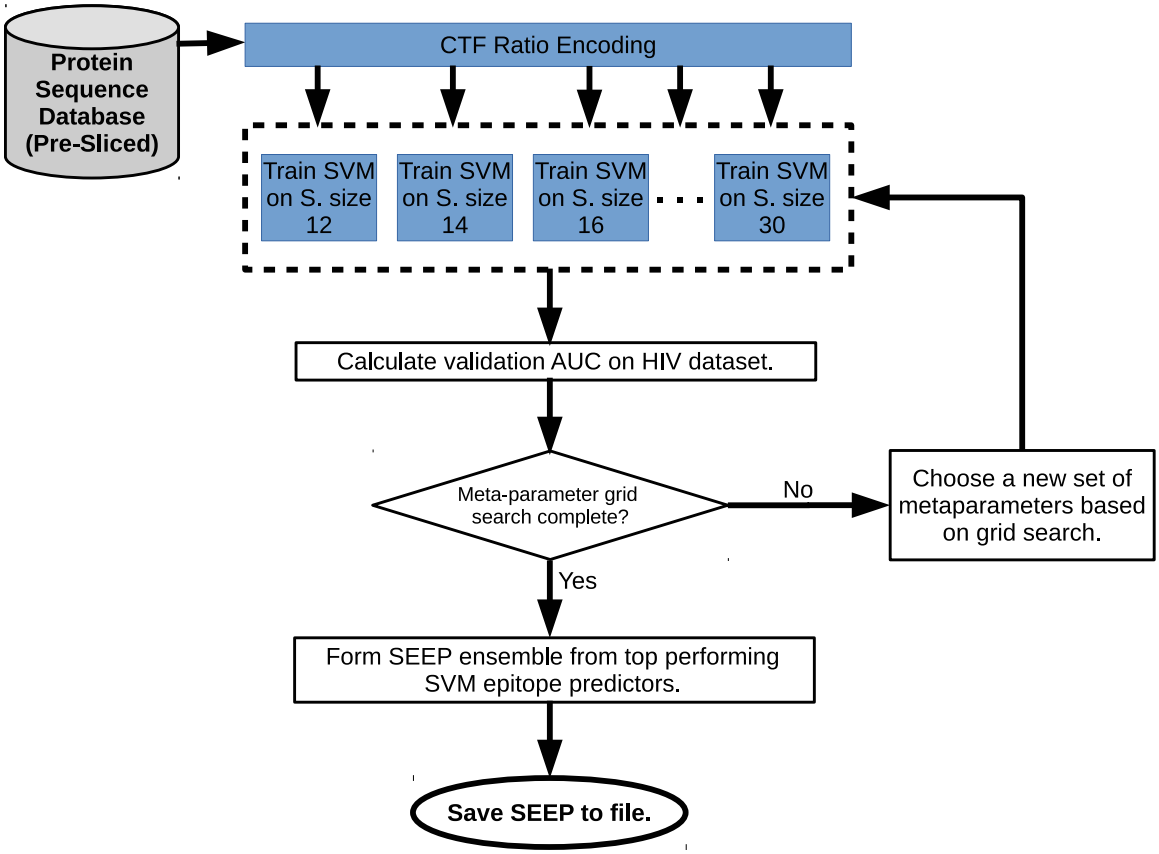

Figure 5.3: **SEEP training pipeline.** Demonstrates the training procedure of the SVM Ensemble Epitope Predictor system which uses the BciPep training dataset, and the HIV validation dataset.

5.3 Greatly Improved Prediction Results

SEEP was tested on the SEQ194 dataset, which has been used as the test dataset by all latest epitope predictors such as BEST, LBTope, and DRREP. The AUC results are shown in Table 5.2, the table also shows accuracy calculated for best specificity and sensitivity pairs. The ROC curves for the top 3 performing systems in the table: LBTope, DRREP, and SEEP, are shown in Figure 5.4.

Table 5.2: **Accuracy and AUC** Results of applying SEEP to the SEQ194 test dataset, and the AUC results of other epitope predictors. SEQ194 contains a total of 194 sequences, 128245 residues, and has an epitope density of 6.6.

Method	**Max(Spec*Sens)**	**Accuracy @ Max**	**AUC**
$DRREP$	$(70.8, 64.5)$	70.4	0.73
$Epitopia$	_	_	0.59
$BEST10$	_	_	0.57
$BEST16$	_	_	0.57
$ABCPred$	_	_	0.55
$CBTOPE$	_	_	0.52
$COBEpro$	_	_	0.55
$LBTope$	$(50.3, 59.0)$	58.6	0.57
SEEP	(**82.6**,**89.3**)	**82.9**	**0.912**

The maxmum specificity*sensitivity pairs, and accuracy of the method at this specificity and sensitivity pair, are only shown for the top 3 systems. The structure based predictor Epitopia results were obtained from [77]. DMN-LBE results are not presented due to the server no longer being available. BEST10 and BEST16 were reported in [28]. LBtope results were acquired by running the SEQ194 dataset on the LBTope server [91] specified as ”trained on the original dataset”.

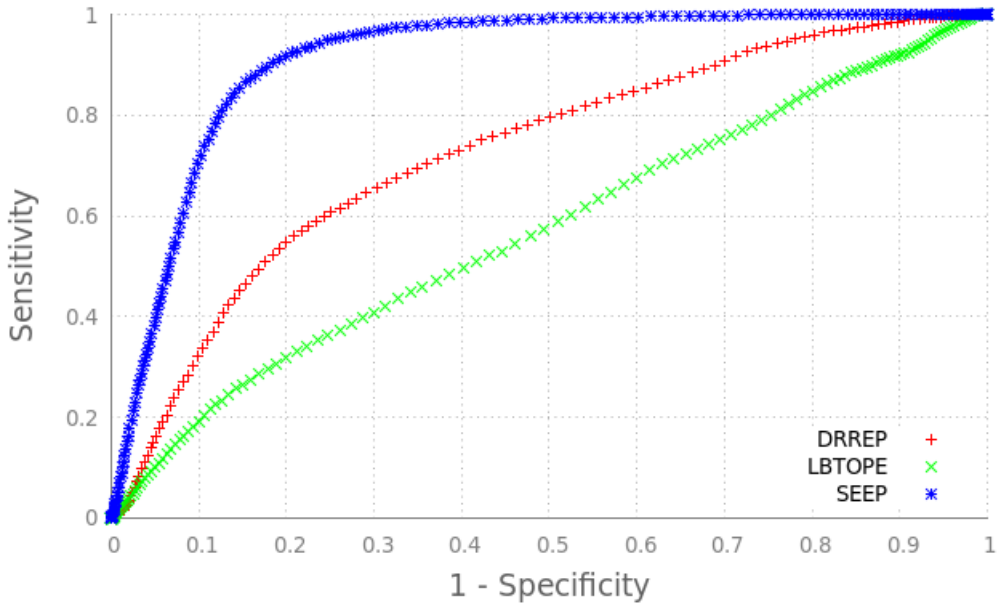

Figure 5.4: **SEQ194 ROC curve results.** SEQ194 ROC curves for SEEP and two competing top predictors (LBtope and DRREP). For SEEP, the maximum(Sensitivity*Specificity) occurs at threshold set to 0.083, with sensitivity of 89.3%, specificity of 82.6%, and resulting in an accuracy of 82.9%.

To demonstrate the higher performance of the consensus prediction compared to any single member, we plot the AUC of each individual member, and that of the committee. The ROC curves of each individual and the committee as a whole are shown in Figure 5.5. The graph shows a clear difference in performance between the committee members, and the committee as a whole.

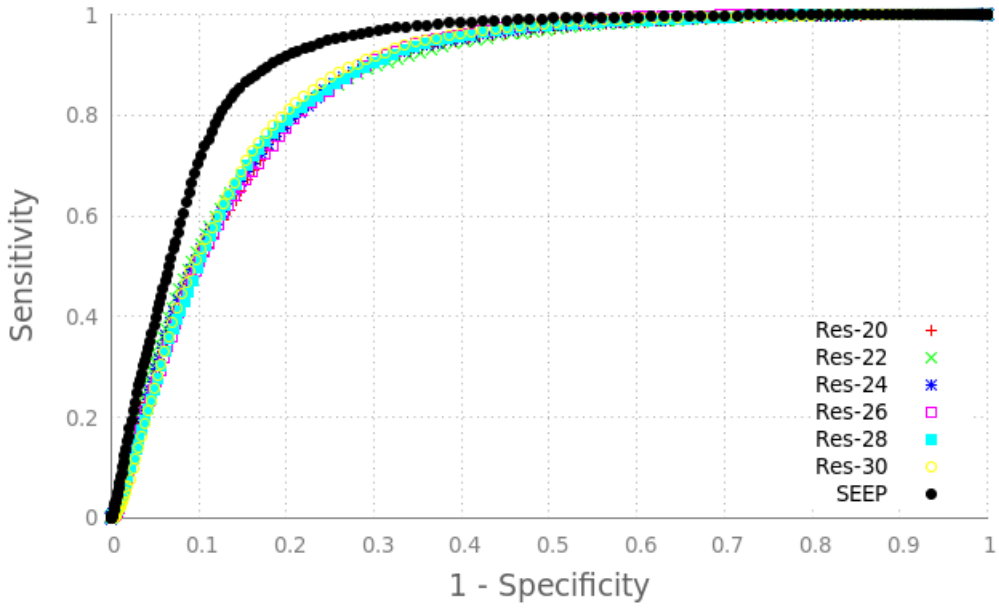

Figure 5.5: **Performance improvement through min-pooling consensus.** ROC curves for each member of the consensus committee composing SEEP, and the improved ROC curve for the SEEP as a whole, produced by using a consensus of the committee members to make its prediction.

Finally, we applied SEEP to the Zika Virus polyprotein, *Acc. ALU33341.1*. As of this writing, there are a total of 7 linear epitopes specified in IEDB, all of which overlap to produce a single sequence: IAPAYSIRCIGVSNRDFV. We applied SEEP to the entire Zika Virus polyprotein sequence, and it predicted numerous epitopes (scores greater than the mean-score threshold), amongst which was a sequence that covered most of the overlapping sub-sequences, as follows:

```
IAPAYSIRCIGVSNRDFVEGMSGGTWVDVVLEHGGCVTVMAQ Sequence
.......EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE SEEP
EEEEEEEEEEEEEEEEE.........................  IEDB
```

Results of SEEP applied to Zika virus polyprotein *Acc. ALU33341.1* are available from our github repository [87]. The score vector, and the position of the various tracks corresponding to the protein sequence, is presented in Figure 5.6.

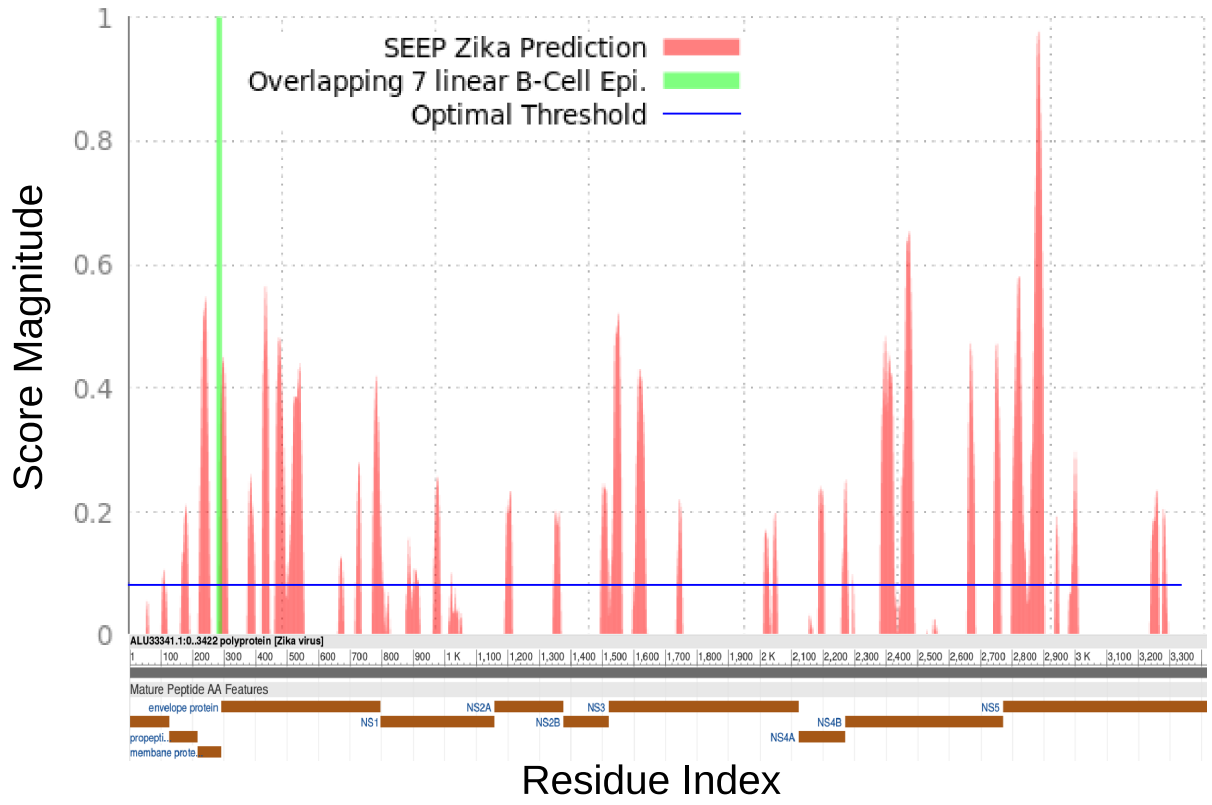

Figure 5.6: **Zika Virus polyprotein and SEEP's predicted epitopes.** Predicted epitopes for Zika Virus Polyprotein Acc. ALU33341.1. Presented in the graph are positive scores, the NCBI listed tracks, and the horizontal line representing our suggested optimal threshold of 0.083. Only positive scores are shown.

5.3.1 Discussion

Table 5.2 demonstrates that SEEP drastically outperforms other systems on the SEQ194 dataset. At the same time, due to it's reliance on the highly optimized SVMLight package, the system is fast, and it linearly scales with the size of the input sequence. SEEP achieves an AUC of 0.912 on the test dataset, which is .34 higher than LBTope server produced results and .14 higher than DRREP. This is a 24% improvement over the current state of the art. SEEP achieves an accuracy of 82.9% at 82.6% specificity and 89.3% sensitivity, which is the system's optimal specificity/sensitivity pair,

achieved at a cutoff prediction threshold of 0.083.

In Figure 5.4 we show the ROC curves for LBTope, DRREP, and SEEP. We want to note that the presented LBTope ROC curve was created based on the per residue classification, using the score results from the LBTope server [91], when feeding it the SEQ194 FASTA file and averaging out the window scores to produce per residue rather than per window scores. When averaging the 20 residue sized window scores from the LBTope prediction server, the achieved AUC was 0.57. SEEP greatly outperforms both, DRREP and LBTope on the SEQ194 dataset.

Figure 5.5 demonstrates the advantage of the multi-dimensional ensemble architecture of the SEEP system. Here we present the ROC curves for separate members of the ensemble, and then the performance of the consensus based ensemble system as a whole. We can see that as a committee, the individual predictors compensate each other's errors, resulting in a higher performance when working together. At the same time, it should be noted that the SVM member AUCs range from 0.86 to 0.88, which demonstrates that it is the CTF trimer ratio based encoding that is the main factor in improving the predictive performance of SEEP over other systems.

The scoring patterns plotted in Figure 5.6 shows 2 sequences which contain a single epitope each, and 2 sequences which contain two epitopes each. The epitope locations predicted by SEEP, clearly align with the expected location of the epitopes. It is also interesting to note that in graph *d)*, we can see the local minima located at the place where the separation between the two epitopes is present. Thus, when choosing high specificity/threshold, the system has the ability to differentiate even between two epitopes which are highly proximal.

Finally, we applied SEEP to a Zika Virus polyprotein. There are only a few confirmed linear B-Cell epitopes that have been discovered and entered into the IEDB database [99]. As of this writing, there are 7 in total, with Epitope Ids: 569587, 572137, 591614, 591675, 591676, 591677, 591800. All 7 epitopes overlap each other, spanning the sequence: `IAPAYSIRCIGVS-`

NRDFV. SEEP predicts an epitope within the same area, but spanning the sequence: **RCIGVS-NRDFV**EGMSGGTWVDVVLEHGGCVTVMAQ, using the residue score mean as threshold. The overlapping section is shown in boldface. It is difficult to deduce whether the system overestimates the span of the overlapping epitopes, or whether other epitopes have simply not yet been discovered. In either case, we see that the system is very useful in marking regions of interest. Figure 5.7 shows a score graph produced by SEEP for the entire Zika polyprotein. Presented are positive scores, with the amplitude being proportional to the certainty of the prediction. The horizontal threshold line is our suggested optimal threshold of 0.083.

Due to the high accuracy of the system, we think that SEEP represents an extremely useful tool. We have demonstrated on the SEQ194 test dataset the excellent performance of the system. This is achieved by taking advantage of the domain specific knowledge offered by the CTF based representation, a pre and post-processor that converts sliding window prediction into a per-residue prediction allowing us to make variable length epitope predictions, and a min-pooling consensus algorithm which allows us to use predictors of different dimensions, and combine their results to yield an even more accurate classification.

5.4 Concluding Remarks & Potential Future SEEP Extensions Based on PPI Domain Knowledge

In this chapter we have presented the **S**VM **E**nsemble **E**pitope **P**redictor called **SEEP**, which leverages the Conjoint-Triad Feature encoding developed within the PPI domain. We have demonstrated that when trained on a well curated dataset and validated on the HIV dataset, its performance on the SEQ194 test dataset achieves an AUC of 0.912, which is 24% higher than the current best state of the art predictors. The next two best and most recently published sequence based predictors, achieve an AUC of 0.73 (DRREP) and 0.57 (LBTope). We then applied it to the Zika virus

polyprotein, and demonstrated that the regions the system marks as having a high probability of containing an epitope, match well with the linear B-Cell epitope locations that have been published to IEDB. The system further predicts other areas, which have not yet been experimentally verified, and which we believe to be of great interest, and should be looked into.

We plan to further expand the system, through larger and improved training datasets, and by further incorporating domain specific knowledge. We believe that by utilizing a more varied ensemble with regards to dimensionality and training datasets used, by adding local descriptors [111], and incorporating auto-covariance [31] information, the system can be improved even further. Finally, we plan on extending the system to the conformational epitope prediction problem, by training a secondary clustering phase, that will be able to predict which of the subsequences belong together to form a single conformational epitope.

CHAPTER 6: CONCLUSION & FUTURE WORK

This dissertation has presented 3 systems which dealt with categorical data, and an approach which allows us to generate distributed representations of amino acids. Chapter 2 presented a neuroevolutionary system which was applied to the problem of compilation phase ordering. Unlike the standard pre-set, order-fixed, source code independent optimizations which are applied by the compiler to the program when using the $-O3$ flag, we created a method that evolves neural networks capable of selecting optimizations based on the source code's features. The evolved NNs were able to order the optimization phases to produce substantially higher performance. The performance was based on the speed of the compiled benchmark programs, as compared to the same programs compiled by LLVM's $-O3$ option. Performance gains ranged from 20% to 40%.

Chapter 3 explored the problem of representing biological data like amino acids, in such a way that the representation encodes useful biochemical and spatial information. The chapter proposed a system which functions analogously to word embedding and skip-gram, but uses 3D biological data. We presented some preliminary results, and demonstrated that this approach to distributed representation has future potential. Furthermore, distributed representations, though being low dimensional, still had utility. This hints that such 3D BioVectors could also potentially be informationally high density.

Chapter 4 presented an epitope predictor called DRREP, which is a committee of ridge regressed classifiers, which with an addition of preprocessing and postprocessing neural layers, formed an analytically trained deep network. DRREP generates input neurons composed of randomly generated kmers, and optimizes the total number of input neurons and thus kmers, based on the training dataset it is applied to. This approach is analogous to using a bag of words, but with the bag of words being generated randomly, and optimized for particular data and problem domain. The

system had performance results on the test datasets that were slightly higher than the state of the art.

Chapter 5 presented a different approach to the problem of encoding and representing protein sequences, and epitope prediction. In this chapter SEEP was presented. SEEP is an ensemble of multi-resolution SVMs, a set of pre and post processing methods which allow it to predict variable length epitopes, and protein encoding which leverages domain specific information. Using a ratio of trimers based on the Conjoint Triad Features, SEEP produced state of the art performance on the standard SEQ194 test dataset, 24% higher than the state of the art.

SEEP leveraged the encoding method originally introduced within the PPI research domain. It demonstrated that when a concise encoding method is used, particularly when it incorporates domain specific knowledge and useful biochemical properties, spatial and order based information provided by using the conjoint traids, high performance can be achieved. This demonstrates the importance of nominal multi-dimensional representation which incorporates within it, useful domain specific information.

For this reason, future work is expected to concentrate on further exploration of the concept of our 3D BioVectors. The preliminary results of 3D Residue BioVectors have demonstrated some potential, but the performance was limited. This is because the compositional representation and the use of residues rather than clusters, resulted in very low dimensional distributed representations. Thus, embedding of information was very limited. We plan to continue exploring this approach, concentrating on trimers, which have shown great performance when generated based on the primary sequence, leading us to expect that 3D BioVectors will be even more useful, and embed higher quality biochemical and spatial information. We plan on exploring which physical properties, and which type of target and environment pairs generate better performing distributed representations. Furthermore, we believe that the same approach that produces 3D BioVectors can be applied to

produce clusters and grouping of residues, akin to the CTF grouping, but which will be formed based on the particular dataset and problem domain. We expect that this might produce groupings specific to problem domains, and thus lead to superior performance. 3D BioVectors open for us an exciting and interesting new research path.

LIST OF REFERENCES

[1] Agakov, F., Bonilla, E., Cavazos, J., Franke, B., Fursin, G., O'Boyle, M. F., Thomson, J., Toussaint, M., and Williams, C. K. (2006). Using machine learning to focus iterative optimization. In *Proceedings of the International Symposium on Code Generation and Optimization*, pages 295–305. IEEE Computer Society.

[2] Alix, A. (1999). Predictive estimation of protein linear epitopes by using the program people. *Vaccine*, **18**(3), 311–314.

[3] Almagor, L., Cooper, K. D., Grosul, A., Harvey, T. J., Reeves, S. W., Subramanian, D., Torczon, L., and Waterman, T. (2004). Finding effective compilation sequences. *ACM SIGPLAN Notices*, **39**(7), 231–239.

[4] An, S., Liu, W., and Venkatesh, S. (2007). Face recognition using kernel ridge regression. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–7. IEEE.

[5] Anfinsen, C. B. (1972). Studies on the principles that govern the folding of protein chains.

[6] Ansari, H. and Raghava, G. (2010). Identification of conformational b-cell epitopes in an antigen from its primary sequence. *Immunome research*, **6**(1), 1.

[7] Asgari, E. and Mofrad, M. R. (2015). Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS one*, **10**(11), e0141287.

[8] Bairoch, A. and Apweiler, R. (2000). The swiss-prot protein sequence database and its supplement trembl in 2000. *Nucleic acids research*, **28**(1), 45–48.

[9] Bansal, S. and Aiken, A. (2006). Automatic generation of peephole superoptimizers. In *ACM Sigplan Notices*, volume 41, pages 394–403. ACM.

[10] Baum, L. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, **37**(6), 1554–1563.

[11] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, **3**(Feb), 1137–1155.

[12] Blythe, M. and Flower, D. (2005). Benchmarking b cell epitope prediction: underperformance of existing methods. *Protein Science*, **14**(1), 246–248.

[13] Breiman, L. (1996). Bagging predictors. *Machine learning*, **24**(2), 123–140.

[14] Buus, S., Lauemøller, S., Worning, P., Kesmir, C., Frimurer, T., Corbet, S., Fomsgaard, A., Hilden, J., Holm, A., and Brunak, S. (2003). Sensitive quantitative predictions of peptide-mhc binding by a query by committeeartificial neural network approach. *Tissue antigens*, **62**(5), 378–384.

[15] Chang, D. T.-H., Syu, Y.-T., and Lin, P.-C. (2010). Predicting the protein-protein interactions using primary structures with predicted protein surface. *BMC bioinformatics*, **11**(1), S3.

[16] Chang, H., Liu, C., and Pai, T. (2008). Estimation and extraction of b-cell linear epitopes predicted by mathematical morphology approaches. *Journal of Molecular Recognition*, **21**(6), 431–441.

[17] Chen, J., Liu, H., Yang, J., and Chou, K. (2007). Prediction of linear b-cell epitopes using amino acid pair antigenicity scale. *Amino acids*, **33**(3), 423–428.

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

[19] Cooper, K. D., Schielke, P. J., and Subramanian, D. (1999). Optimizing for reduced code space using genetic algorithms. In *ACM SIGPLAN Notices*, volume 34, pages 1–9. ACM.

[20] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, **20**(3), 273–297.

[21] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer.

[22] Dominic, S., Das, R., Whitley, D., and Anderson, C. (1991). Genetic reinforcement learning for neural networks. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pages 71–76. IEEE.

[23] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, **26**(1), 29–41.

[24] Drosten, C., Günther, S., Preiser, W., Van Der Werf, S., Brodt, H., Becker, S., Rabenau, H., Panning, M., Kolesnikova, L., Fouchier, R., *et al.* (2003). Identification of a novel coronavirus in patients with severe acute respiratory syndrome. *New England Journal of Medicine*, **348**(20), 1967–1976.

[25] EL-Manzalawy, Y., Dobbs, D., and Honavar, V. (2008). Predicting linear b-cell epitopes using string kernels. *Journal of molecular recognition*, **21**(4), 243–255.

[26] Elder, J. H. and Zucker, S. W. (1998). Local scale control for edge detection and blur estimation. *IEEE Transactions on Pattern Analysis and machine intelligence*, **20**(7), 699–716.

[27] Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955.

[28] Gao, J., Faraggi, E., Zhou, Y., Ruan, J., and Kurgan, L. (2012). Best: improved prediction of b-cell epitopes from antigen sequences. *PloS one*, **7**(6), e40104.

[29] Globerson, A., Chechik, G., Pereira, F., and Tishby, N. (2007). Euclidean embedding of co-occurrence data. *Journal of Machine Learning Research*, **8**(Oct), 2265–2295.

[30] Greenbaum, J. A., Andersen, P. H., Blythe, M., Bui, H., Cachau, R., Crowe, J., Davies, M., Kolaskar, A., Lund, O., Morrison, S., *et al.* (2007). Towards a consensus on datasets and evaluation metrics for developing b-cell epitope prediction tools. *Journal of Molecular Recognition*, **20**(2), 75.

[31] Guo, Y., Yu, L., Wen, Z., and Li, M. (2008). Using support vector machine combined with auto covariance to predict protein–protein interactions from protein sequences. *Nucleic acids research*, **36**(9), 3025–3030.

[32] Guthrie, D., Allison, B., Liu, W., Guthrie, L., and Wilks, Y. (2006). A closer look at skip-gram modelling. In *Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006)*, pages 1–4.

[33] Haneda, M., Knijnenburg, P. M., and Wijshoff, H. A. (2005). Automatic selection of compiler options using non-parametric inferential statistics. In *Parallel Architectures and Compilation Techniques, 2005. PACT 2005. 14th International Conference on*, pages 123–132. IEEE.

[34] Haste Andersen, P., Nielsen, M., and Lund, O. (2006). Prediction of residues in discontinuous b-cell epitopes using protein 3d structures. *Protein Science*, **15**(11), 2558–2567.

[35] Hinton, G., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, **18**(7), 1527–1554.

[36] Hoerl, A. and Kennard, R. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, **12**(1), 55–67.

[37] Hopp, T. and Woods, K. (1981). Prediction of protein antigenic determinants from amino acid sequences. *Proceedings of the National Academy of Sciences*, **78**(6), 3824–3828.

[38] Hoste, K. and Eeckhout, L. (2008). Cole: compiler optimization level exploration. In *Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization*, pages 165–174. ACM.

[39] Huang, G., Zhu, Q., and Siew, C. (2004). Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 985–990. IEEE.

[40] Huang, G., Zhu, Q., and Siew, C. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, **70**(1), 489–501.

[41] Hunter, W. G., Hunter, J. S., and George, E. (1978). *Statistics for experimenters: an introduction to design, data analysis, and model building*. Wiley New York.

[42] Jens Vindahl Kringelum, Claus Lundegaard, O. L. M. N. (2017). Discotop-2 dataset: http://www.cbs.dtu.dk/suppl/immunology/DiscoTope-2.0/

[43] Joachims, T. (2008). Svm light: http://svmlight.joachims.org/

[44] Knijnenburg, P. M., Kisuki, T., and O'Boyle, M. F. (2003). Combined selection of tile sizes and unroll factors using iterative compilation. *The Journal of Supercomputing*, **24**(1), 43–67.

[45] Kringelum, J. V., Lundegaard, C., Lund, O., and Nielsen, M. (2012). Reliable b cell epitope predictions: impacts of method development and improved benchmarking. *PLoS computational biology*, **8**(12), e1002829.

[46] Kuiken, C., Korber, B., and Shafer, R. (2003). Hiv sequence databases. *AIDS reviews*, **5**(1), 52.

[47] Kulkarni, P., Zhao, W., Moon, H., Cho, K., Whalley, D., Davidson, J., Bailey, M., Paek, Y., and Gallivan, K. (2003). Finding effective optimization phase sequences. In *ACM SIGPLAN Notices*, volume 38, pages 12–23. ACM.

[48] Kulkarni, P. A., Whalley, D. B., Tyson, G. S., and Davidson, J. W. (2006). Exhaustive optimization phase order space exploration. In *Proceedings of the International Symposium on Code Generation and Optimization*, pages 306–318. IEEE Computer Society.

[49] Kulkarni, P. A., Whalley, D., and Tyson, G. (2007). Evaluating heuristic optimization phase order search algorithms. In *Code Generation and Optimization, 2007. CGO'07. International Symposium on*, pages 157–169. IEEE.

[50] Kulkarni, S. and Cavazos, J. (2012). Mitigating the compiler optimization phase-ordering problem using machine learning. *ACM SIGPLAN Notices*, **47**(10), 147–162.

[51] Larsen, J., Lund, O., and Nielsen, M. (2006). Improved method for predicting linear b-cell epitopes. *Immunome Res*, **2**(2), 1–7.

[52] Lattner, C. and Adve, V. (2004). Llvm: A compilation framework for lifelong program analysis & transformation. In *Code Generation and Optimization, 2004. CGO 2004. International Symposium on*, pages 75–86. IEEE.

[53] Lebret, R. and Collobert, R. (2013). Word emdeddings through hellinger pca. *arXiv preprint arXiv:1312.5542*.

[54] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, **521**(7553), 436–444.

[55] Leslie, C., Eskin, E., and Noble, W. (2002). The spectrum kernel: A string kernel for svm protein classification. In *Pacific symposium on biocomputing*, volume 7, pages 566–575. World Scientific.

[56] Leslie, C., Eskin, E., Cohen, A., Weston, J., and Noble, W. (2004). Mismatch string kernels for discriminative protein classification. *Bioinformatics*, **20**(4), 467–476.

[57] Levitt, M. (1976). A simplified representation of protein conformations for rapid simulation of protein folding. *Journal of molecular biology*, **104**(1), 59–107.

[58] Levy, O. and Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185.

[59] Li, Y., Xu, L., Tian, F., Jiang, L., Zhong, X., and Chen, E. (2015). Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *IJCAI*, pages 3650–3656.

[60] Matykiewicz, P. and Pestian, J. (2012). Effect of small sample size on text categorization with support vector machines. In *Proceedings of the 2012 workshop on biomedical natural language processing*, pages 193–201. Association for Computational Linguistics.

[61] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[62] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013b). Effic ient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[63] Miller, S., Janin, J., Lesk, A. M., and Chothia, C. (1987). Interior and surface of monomeric proteins. *Journal of molecular biology*, **196**(3), 641–656.

[64] Mitchell, T. M., Shinkareva, S. V., Carlson, A., Chang, K.-M., Malave, V. L., Mason, R. A., and Just, M. A. (2008). Predicting human brain activity associated with the meanings of nouns. *science*, **320**(5880), 1191–1195.

[65] Munoz, A. (2014). Machine learning and optimization. *URL: https://www. cims. nyu. edu/˜ munoz/files/ml_optimization. pdf [accessed 2016-03-02][WebCite Cache ID 6fiLfZvnG]*.

[66] Nielsen, M., Lundegaard, C., Worning, P., Lauemøller, S., Lamberth, K., Buus, S., Brunak, S., and Lund, O. (2003). Reliable prediction of t-cell epitopes using neural networks with novel sequence representations. *Protein Science*, **12**(5), 1007–1017.

[67] Odorico, M. and Pellequer, J. (2003). Bepitope: predicting the location of continuous epitopes and patterns in proteins. *Journal of Molecular Recognition*, **16**(1), 20–22.

[68] Ofran, Y. and Rost, B. (2003). Analysing six types of protein–protein interfaces. *Journal of molecular biology*, **325**(2), 377–387.

[69] Pan, X.-Y., Zhang, Y.-N., and Shen, H.-B. (2010). Large-scale prediction of human protein-protein interactions from amino acid sequence based on latent topic features. *Journal of Proteome Research*, **9**(10), 4992–5001.

[70] Pellequer, J. and Westhof, E. (1993). Preditop: a program for antigenicity prediction. *Journal of molecular graphics*, **11**(3), 204–210.

[71] Pellequer, J., Westhof, E., and Van Regenmortel, M. (1990). Predicting location of continuous epitopes in proteins from their primary structures. *Methods in enzymology*, **203**, 176–201.

[72] Penrose, R. (1955). A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge Univ Press.

[73] Peters, B., Sidney, J., Bourne, P., Bui, H., Buus, S., Doh, G., Fleri, W., Kronenberg, M., Kubo, R., Lund, O., *et al.* (2005). The immune epitope database and analysis resource: from vision to blueprint. *PLoS biol*, **3**(3), e91.

[74] Ponomarenko, J., Bui, H.-H., Li, W., Fusseder, N., Bourne, P. E., Sette, A., and Peters, B. (2008). Ellipro: a new structure-based tool for the prediction of antibody epitopes. *BMC bioinformatics*, **9**(1), 514.

[75] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, **65**(6), 386.

[76] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *science*, **290**(5500), 2323–2326.

[77] Rubinstein, N., Mayrose, I., Martz, E., and Pupko, T. (2009). Epitopia: a web-server for predicting b-cell epitopes. *BMC bioinformatics*, **10**(1), 287.

[78] Saha, S. and Raghava, G. (2004). *Artificial immune systems*. Springer, New York.

[79] Saha, S. and Raghava, G. (2006). Prediction of continuous b-cell epitopes in an antigen using recurrent neural network. *Proteins: Structure, Function, and Bioinformatics*, **65**(1), 40–48.

[80] Saha, S., Bhasin, M., and Raghava, G. P. (2005). Bcipep: a database of b-cell epitopes. *BMC genomics*, **6**(1), 79.

[81] Saunders, C., Gammerman, A., and Vovk, V. (1998). Ridge regression learning algorithm in dual variables. In *(ICML-1998) Proceedings of the 15th International Conference on Machine Learning*, pages 515–521. Morgan Kaufmann.

[82] Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. *Artificial Neural Networks–ICANN 2010*, pages 92–101.

[83] Shen, J., Zhang, J., Luo, X., Zhu, W., Yu, K., Chen, K., Li, Y., and Jiang, H. (2007). Predicting protein–protein interactions based only on sequences information. *Proceedings of the National Academy of Sciences*, **104**(11), 4337–4341.

[84] Sher, G. (2016a). DRREP classifier repository: https://github.com/gsher1/DRREP

[85] Sher, G. (2016b). DRREP datasets: https://github.com/gsher1/DRREP_Datasets

[86] Sher, G. (2017a). SEEP classifier repository: https://github.com/gsher1/SEEP

[87] Sher, G. (2017b). SEEP datasets: https://github.com/gsher1/SEEP_Datasets

[88] Sher, Z. and Zhang (2017). DRREP: Deep ridge regressed epitope predictor. *BMC genomics*, page 17.

[89] Simon, P. (2013). *Too big to ignore: the business case for big data*, volume 72. John Wiley & Sons.

[90] Singh, H., Ansari, H., and Raghava, G. (2013). Improved method for linear b-cell epitope prediction using antigens primary sequence. *PloS one*, **8**(5), e62216.

[91] Singh H, Ansari HR, R. G. (2013). Lbtope server: http://www.imtech.res.in/raghava/lbtope/

[92] Socher, R., Ganjoo, M., Manning, C. D., and Ng, A. (2013). Zero-shot learning through cross-modal transfer. In *Advances in neural information processing systems*, pages 935–943.

[93] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, **10**(2), 99–127.

[94] Sweredoski, M. and Baldi, P. (2009). Cobepro: a novel system for predicting continuous b-cell epitopes. *Protein Engineering Design and Selection*, **22**(3), 113–120.

[95] Sweredoski, M. J. and Baldi, P. (2008). Pepito: improved discontinuous b-cell epitope prediction using multiple distance thresholds and half sphere exposure. *Bioinformatics*, **24**(12), 1459–1460.

[96] Toseland, C., Clayton, D., McSparron, H., Hemsley, S., Blythe, M., Paine, K., Doytchinova, I., Guan, P., Hattotuwagama, C., and Flower, D. (2005). Antijen: a quantitative immunology database integrating functional, thermodynamic, kinetic, biophysical, and cellular data. *Immunome Res*, **1**(4), 82–93.

[97] Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

[98] Van Regenmortel, M. H. (1996). Mapping epitope structure and activity: from one-dimensional prediction to four-dimensional description of antigenic specificity. *Methods*, **9**(3), 465–472.

[99] Vita, R., Overton, J. A., Greenbaum, J. A., Ponomarenko, J., Clark, J. D., Cantrell, J. R., Wheeler, D. K., Gabbard, J. L., Hix, D., Sette, A., *et al.* (2014). The immune epitope database (iedb) 3.0. *Nucleic acids research*, **43**(D1), D405–D412.

[100] Vita, R., Overton, J., Greenbaum, J., Ponomarenko, J., Clark, J., Cantrell, J., Wheeler, D., Gabbard, J., Hix, D., Sette, A., *et al.* (2015). The immune epitope database (iedb) 3.0. *Nucleic acids research*, **43**(D1), D405–D412.

[101] Wang, H., Lin, Y., Pai, T., and Chang, H. (2011a). Prediction of b-cell linear epitopes with a combination of support vector machine classification and amino acid propensity identification. *BioMed Research International*, **2011**.

[102] Wang, Y., Wu, W., Negre, N., White, K., Li, C., and Shah, P. (2011b). Determinants of antigenicity and specificity in immune response for protein sequences. *BMC bioinformatics*, **12**(1), 1.

[103] Wang, Y.-C., Wang, X.-B., Yang, Z.-X., and Deng, N.-Y. (2010). Prediction of enzyme subfamily class via pseudo amino acid composition by incorporating the conjoint triad feature. *Protein and Peptide Letters*, **17**(11), 1441–1449.

[104] Way, T. W., Sahiner, B., Hadjiiski, L. M., and Chan, H.-P. (2010). Effect of finite sample size on feature selection and classification: a simulation study. *Medical physics*, **37**(2), 907–920.

[105] Wee, J., Simarmata, D., Kam, Y., Ng, F., and Tong, J. (2010). Svm-based prediction of linear b-cell epitopes using bayes feature extraction. *BMC genomics*, **11**(4), 1.

[106] Woodrow, G. (1997). An overview of biotechnology as applied to vaccine development. *New Generation Vaccines*, **25**.

[107] Yang, L., Xia, J.-F., and Gui, J. (2010). Prediction of protein-protein interactions from protein sequence using local descriptors. *Protein and Peptide Letters*, **17**(9), 1085–1090.

[108] Yao, B., Zhang, L., Liang, S., and Zhang, C. (2012). Svmtrip: a method to predict antigenic epitopes using support vector machine to integrate tri-peptide similarity and propensity. *PloS one*, **7**(9), e45152.

[109] Yao, L., Huang, Z., Meng, G., and Pan, X. (2015). An improved method for predicting linear b-cell epitope using deep maxout networks. *Biomedical and Environmental Sciences*, **28**(6), 460–463.

[110] Yasser EL Manzalawy, D. D. and Honavar, V. (2008). BCPred and fbcpred datasets: http://ailab.ist.psu.edu/bcpred/data.html

[111] You, Z.-H., Lei, Y.-K., Zhu, L., Xia, J., and Wang, B. (2013). Prediction of protein-protein interactions from amino acid sequences with ensemble extreme learning machines and principal component analysis. *BMC bioinformatics*, **14**(8), S10.

[112] Zhang, L. and Suganthan, P. (2015). A comprehensive evaluation of random vector functional link networks. *Information Sciences*.

[113] Zhang, W. and Niu, Y. (2010). Predicting flexible length linear b-cell epitopes using pairwise sequence similarity. In *Biomedical Engineering and Informatics (BMEI), 2010 3rd International Conference on*, volume 6, pages 2338–2342. IEEE.