

**IDENTIFYING AND GENERATING CANDIDATE ANTIBACTERIAL PEPTIDES
WITH LONG SHORT-TERM MEMORY RECURRENT NEURAL NETWORKS**

A Dissertation
Presented to
The Academic Faculty

By

Michael Thomas Youmans

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Department of Biomedical Engineering

Emory University &
Georgia Institute of Technology

May 2019

**IDENTIFYING AND GENERATING CANDIDATE ANTIBACTERIAL PEPTIDES
WITH LONG SHORT-TERM MEMORY RECURRENT NEURAL NETWORKS**

Approved by:

Dr. Peng Qiu, Advisor
Department of Biomedical Engineering
Georgia Institute of Technology

Dr. Eberhard Voit
Department of Biomedical Engineering
Georgia Institute of Technology

Dr. Jeffrey Skolnick
School of Biological Sciences
Georgia Institute of Technology

Dr. Zsolt Kira
Georgia Tech Research Institute
Georgia Institute of Technology

Dr. Wendy Kelly
School of Chemistry and Biochemistry
Georgia Institute of Technology

Dr. Eva Dyer
Department of Biomedical Engineering
Georgia Institute of Technology

Date Approved: January 11, 2019

To my parents who helped make this opportunity possible.

ACKNOWLEDGEMENTS

Earning a Ph.D. can be a grueling task often requiring a village of people to mint a new scientist. First and foremost I'd like to thank my adviser, Dr. Peng Qiu, for giving me the opportunity to return to Georgia Tech and complete the minting process. His kindness in letting me explore the new area of deep learning as part of my thesis project was wonderful. Next, I'd like to thank Dr. Christian Spainhour. His helpful conversations and advice on navigating the rigors of a Ph.D. program were very welcome. I want to also thank Jenny Jeong and Xingyu Yang for being good lab mates who were remarkably tolerant of my somewhat chatty nature while in lab. Also, the new members of the lab, Hong Seo Lim, Qi Zhang, and Qinwei Zhuang for the interesting discussions during lab meetings. I want to thank the committee for being present for advice and intellectual support as needed. A special nod goes to Dr. Eberhard Voit for his recommended edits to the thesis and his general mentoring nature during my grad school career. I thank Dr. Zsolt Kira for his help with all things deep learning and his ability to be fascinated the biological side of the project despite arguably insufficient help from me. Dr. Jeffrey Skolnick earns special mention due to his helpful skepticism that led us to ask the right question more than once. Also I thank Dr. Robert Lee for his service on my committee and helpful advice during our individual meetings. I thank Dr. Wendy Kelly for her words of encouragement during our meetings. Finally, I want to thank Dr. Eva Dyer for being an extremely helpful committee member following Robert Lee's retirement. Her willingness to serve on my committee at the last second was extremely gracious. I also want to thank my family for their constant support throughout this endeavor.

TABLE OF CONTENTS

Acknowledgments	iv
List of Tables	ix
List of Figures	x
Nomenclature	xi
Summary	xiii
Chapter 1: Introduction	1
1.1 The Growing Problem of Antibiotic Resistance	1
1.2 Interest in Antibacterial Peptides	2
1.3 Antibacterial Peptides Databases	3
1.4 Machine Learning Methodologies Applied in this Area	4
1.4.1 Classifiers	5
1.4.2 Subsequence Classification	7
1.5 Neural Networks	7
1.5.1 Feedforward Neural Networks	7
1.5.2 Recurrent Neural Networks	10
1.5.3 Long Short-Term Memory Architecture	15

1.5.4	Attention Mechanisms	17
1.5.5	Vector Embeddings	17
1.5.6	Generative Adversarial Networks	18
Chapter 2: Training Long Short-Term Memory Networks To Identify Antibacterial Peptides		20
2.1	Motivation, Background and Overview	20
2.2	Materials and Methods	21
2.2.1	Creating the Dataset	21
2.2.2	Representation of the Peptides	22
2.2.3	Algorithm for Creating Reduced Sequence Similarity Dataset	24
2.2.4	Classification Algorithms and Implementations	25
2.2.5	The LSTM and Shuffled Versions of Positive Instances	27
2.3	Results	28
2.3.1	Results of LSTM	29
2.3.2	Results of Random Forests	31
2.3.3	Results of kNN	32
2.3.4	Overall Results	33
2.3.5	Results of Shuffling Positive Instances on the LSTM Predictions	34
2.4	Discussion	34
2.4.1	The Implications of the Shuffled Positive Instances	36
Chapter 3: Identifying Critical Regions of Antibacterial Peptides		40
3.1	Motivation, Background and Overview	40

3.1.1	Attention Mechanisms	40
3.2	Peptide Point Mutations	41
3.2.1	Chosen Peptides for Point Mutations	43
3.2.2	Effect of Point Mutations on the Class Probability	43
3.3	Attention Mechanism with Bidirectional LSTM	46
3.3.1	Precise Soft Attention LSTM Topology	46
3.3.2	Toy Data Set	46
3.3.3	Results on Toy Data Set	47
3.4	Discussion	48
Chapter 4: Identifying and Generating Potential Antibacterial Peptides		50
4.1	Motivation, Background, and Overview	50
4.1.1	Background and Overview of GANs	50
4.1.2	SeqGAN and Similar Techniques	55
4.2	Random Peptides Generation and the Bidirectional LSTM Classifier	59
4.2.1	Simple Technique to Generate a Large Number of Random Peptides	59
4.2.2	Identifying Potentially Promising Peptides	60
4.2.3	Discussion of the Random Peptides Technique	64
4.3	Generative Adversarial Networks Progress	65
Chapter 5: Thesis Contributions and Future Work		68
5.1	Thesis Contributions	68
5.2	Alternative Attention Mechanisms	69

5.3	Making the Classifier Less Overconfident when Classifying Antibacterial Peptides	70
5.4	Finishing the SeqGAN	70
References	72

LIST OF TABLES

2.1	Algorithm Performance on Original Data	31
2.2	Algorithm Performance on Reduced Sequence Similarity Data	31
3.1	Peptides for Point Mutations	43
4.1	Random Peptides Top 31 Part 1	62
4.2	Random Peptides Top 31 Part 2	63

LIST OF FIGURES

1.1	Multilayer Perceptron	9
1.2	Recurrent Neural Network	10
1.3	Unfolded RNN	11
1.4	Unfolding of an RNN	13
1.5	Unfolded Bidirectional RNN	14
1.6	LSTM versus RNN	16
2.1	Features Used in LSTM	23
2.2	LSTM Topology	30
2.3	Shuffled Peptides Probabilities	39
3.1	LSTM with Attention	42
3.2	Point Mutations on DBAASP ID 759	44
3.3	Point Mutations on DBAASP ID 7869	45
4.1	Random Peptides Histogram	61

NOMENCLATURE

DBAASP Abbreviation for Database of Antimicrobial Activity and Structure of Peptides.

end-to-end differentiable Simply refers to the idea that a particular neural network's loss function is differentiable with respect to its weights. In the case of networks with attention means that even with the additional attention neural network, the overall configuration is differentiable with respect to both the original network's weights and the auxiliary attention network's weights.

GAN Abbreviation for Generative Adversarial Network.

Generative Adversarial Network A method to use two competing neural networks to build a generative model. Samples drawn from the generative model post training should be indistinguishable from those drawn from the real distribution that generated the training data.

K-Lipschitz a real-valued function, f , is K-Lipschitz if there exists a real constant K such that for all real x_1, x_2 both in the domain of f , $|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$. 1-Lipschitz is when K is equal to 1.

Long Short-Term Memory A special class of recurrent neural networks that replaces more typical simple recurrent units with nonlinearities such as hyperbolic tangent with a more sophisticated LSTM block that performs the same function but is augmented with a memory cell and gates that allow it to better control the flow of information along the sequence.

LSTM Abbreviation for Long Short-Term Memory, refers to a special type of recurrent neural network.

MCC Abbreviation for Matthews Correlation Coefficient.

Matthews Correlation Coefficient A popular performance metric for binary classifiers that takes into account the number of false positives and false negatives in addition to the true positives and true negatives. Considered a balanced measure even when the positives and negative classes are unbalanced in size. +1 indicates perfect classification performance and -1 indicates that all instances were misclassified.

MIC Abbreviation for Minimum Inhibitory Concentration.

Minimum Inhibitory Concentration lowest concentration of a chemical that prevents visible growth of bacterium.

MNIST A popular dataset consisting of images of digits which must be classified.

one-hot A special type of vector with a single entry set to one and all other entries set to zero. Essentially a canonical unit vector.

QSAR Abbreviation for Quantitative Structure Activity Relationships.

WGAN Abbreviation for Wasserstein Generative Adversarial Network.

SUMMARY

There is a growing need to deal with increasing rates of resistance to antibiotics among pathogenic bacteria. It is becoming widely recognized that the development of resistance in bacteria to current antibiotics poses a global health hazard. One potential angle of attack on this problem is antibacterial peptides which form an active area of current research that may aid in the development of new methods to deal with pathogenic bacteria. Identifying such antibacterial peptides experimentally, however, is a time consuming task. It is hoped that machine learning strategies will provide a more efficient strategy to identify potential antibacterial peptide candidates that can accelerate what can be done through experimental testing alone. Many of the current machine learning methodologies employed to identify antibacterial peptides rely on constructing a finite length feature vector that is based on amino acid level features. Since peptides may contain different numbers of amino acids, it is not obvious how best to take amino acid features and turn them into a feature vector representing the entire peptide. Many methods for constructing such features search for periodic patterns in the amino acid level features and then use a scalar representing the strength of the periodic pattern to create feature for the whole peptide. This approach can be hit or miss as it is difficult to know which periodic patterns are relevant to the classification or regression task in advance. Deep learning has emerged as a popular new machine learning paradigm in recent years. In this work we examine the ability of deep learning architectures like recurrent neural networks to perform classification tasks on antibacterial peptides. Recurrent neural networks that can take in variable length sequences of amino acid features and automatically extract a feature representation that is appropriate for the given machine learning task are used to develop classifiers in Chapter 2. The trained recurrent neural network classifier learns to distinguish between antibacterial peptides and non antibacterial peptides while using surprisingly simple features at least as well as a random forest classifier and a k-nearest neighbor algorithm developed using more complicated

features.

In Chapter 3 we pursue methods to identify important regions of antibacterial peptides. We are essentially searching for small groups of amino acids that seem to have an outsize probability on the peptide being antibacterial. First we take the approach of using the recurrent neural network classifier from Chapter 2 and observe changes in its probabilistic predictions as we apply point mutations to two known positive instances. We see that at least for one of the peptides we can identify likely important amino acids using this method. Next, we examine the possibility of using a neural network attention mechanism to aid in identification of important regions in the peptide. An attention mechanism in this case is an auxiliary neural network that aids the recurrent neural network that was developed in Chapter 2. It aids the original recurrent network by placing weights on each of the amino acids composing a peptide. These weights indicate the amount of attention that the original or primary network should pay to each amino acid of the peptide when it attempts to determine the overall probability of being antibacterial. Before applying our attention augmented network to the peptides, a toy data set was generated to test the topology. Due to disappointing results on the toy data, we decided to not apply it to our primary dataset for now. Instead, we propose exploring a more sophisticated version of attention mechanism in future work.

In Chapter 4, we investigate methods to identify or generate potentially interesting new peptides. We begin with the simple method of uniformly generating peptides with the appropriate lengths and amino acids. This is done by first uniformly selecting a length from among a certain range and then uniformly sampling over the twenty amino acids we consider for each position. We generated one million peptides using this method and applied the recurrent neural network from Chapter 2 to identify some candidate antibacterial peptides. We also describe the results of this experiment both its current drawbacks and advantages as well as some noticeable issues with the recurrent neural network from Chapter 2 that we would like to correct in future work. Next, we took a different tack for deter-

mining candidate antibacterial peptides by attempting to use a popular new deep learning framework known as generative adversarial networks to build a generative model of the peptides. A generative adversarial network is a pair of neural networks locked in a competition. One is trained to generate samples similar to the instances drawn from the real world while the second attempts to distinguish between these generated samples and the real world instances. By competing, the second neural network forces the first to become so good at generating reasonable samples that they are indistinguishable from the real instances. In our case, we looked into generative adversarial networks that were appropriate for our sequential data, the peptides composed of their respective amino acids. Although we have not finished implementing the sequential generative adversarial network for our dataset, we hope to explore it more in future work.

Finally in Chapter 5, future directions that may improve the results are described. One important technique is to better calibrate the classifiers in Chapter 2 so that they do not assign overconfident probabilities to random peptides. New techniques for performing such calibrations are discussed. These techniques may allow for a more informative classifier when trying to identify the most likely candidates from among the one million random peptides that were considered in Chapter 4. We also hope to continue to make progress on implementing the sequential generative adversarial network for our data as well as improving on the attention mechanisms.

CHAPTER 1

INTRODUCTION

This thesis explores the use of recurrent neural networks and related architectures for the identification of antibacterial peptides.

1.1 The Growing Problem of Antibiotic Resistance

There is a growing need to deal with increasing rates of resistance to antibiotics among pathogenic bacteria. The development of resistance in bacteria to current antibiotics poses a global health hazard, and it is necessary to introduce new antibiotic compounds to keep up with the pace of increasing resistance [1]. It is also clear that antibiotic resistance is a global problem with the ability to affect many different aspects of health-care. This includes not only deaths due to resistant infections, but also the increasing costs of treating resistant infections. It also impacts medical procedures that require effective antibiotics to prevent infection [1]. An estimated 25,000 people die in Europe each year due to antibiotic-resistant bacteria. It is estimated that 2 million illnesses and 23,000 deaths occur each year in the United States due to antibiotic resistance. It is suggested that the situation may even be worse in low-income and middle income countries [1]. Furthermore, the clinical incidence of drug-resistant microbes has increased in recent decades [2]. Until approximately the last decade, the pharmaceutical industry developed new antibiotics or upgraded existing ones in an attempt to keep up with antibiotic resistance [3]. In recent years, the number of new antibiotics licensed for human use in different parts of the world has been lower than before. The pharmaceutical industry has largely abandoned their anti-infective research programs due to financial reasons, and currently the government and large academic institutions are not investing the necessary resources to keep pace with the growth of antibiotic resistance [4].

1.2 Interest in Antibacterial Peptides

Antimicrobial peptides (AMPs), part of the innate immune system, are the first line of defense against microbial pathogens [3, 5]. AMPs are gene-coded, short (<100 amino acids), amphipathic molecules with broad-spectrum antimicrobial activity. They have the potential to form a new class of antibiotics [3].

Due to the large number of possible amino acid combinations that can be used to create peptides ranging between 5 and 100 amino acids in length, it is nearly impossible to use experimental testing alone to identify likely candidate AMPs. Scientists are turning to computational methodologies for predicting peptide activity that can later be evaluated clinically [6].

Originally, AMPs were believed to act primarily through disruption of the bacterial membranes through several different proposed models. More recently, however, it is clear that AMPs not only act through these forms of disruption. Instead, many AMPs are now known to translocate across the bacterial cell membrane and interfere with internal targets such as DNA or RNA synthesis, protein synthesis, cell wall synthesis, cell division, and protein folding [7]. However, regardless of their precise mode of action, antibacterial peptides are dependent on interaction with the bacterial cell membrane. This step involves the electrostatic attraction between the typically cationic peptide and negatively charged components present on the outer bacterial envelope [7]. This method of interaction also lends notion to the idea that AMPs are selective for bacterial cell membranes over plant and animal cell membranes. In bacterial cells, for example, the outermost leaflet of the bilayer is heavily populated by lipids with negatively charged head-groups which can attract the typically cationic AMPs. This contrasts with the outer leaflet of plants and animals which is typically composed of lipids with no net charge [8].

A popular method for aiding in the design of potential AMPs is known as quantitative structure-activity relationship (QSAR) modeling. These methods allow one to predict the

activity of an AMP from various physicochemical and structural descriptors. Classically, these methods use linear regression [9] or methods such as partial least squares regression to allow for easy interpretation of how descriptors or features affect the measured property [10]. Once the QSAR model is determined, it is possible to perturb properties of the AMP sequence and observe the change in AMP activity to some degree. Many of the classic methodologies employed in QSAR, however, are being replaced by other machine learning techniques such as the use of neural networks [11]. The machine learning methods may be harder to interpret, but they can learn highly non-linear relationships between descriptors or features and the activity level of the AMP.

Over the past decade or so, a number of mechanisms have emerged that allow bacteria to develop various forms of resistance to AMPs. One example of this is the ability of some bacteria to modify the anionic lipids or wall components to reduce the net negative charge of their membranes. This reduces the ability of the cationic AMPs to interact with the membranes of potentially pathogenic bacteria and confers some resistance to the bacteria [12].

1.3 Antibacterial Peptides Databases

Over the past several years a large number of databases have been developed with the intention of aiding researchers studying various antimicrobial peptides. One such database is the Database of Antimicrobial Activity and Structure of Peptides (DBAASP). It contains a large number of monomeric AMPs and has many searchable properties such as the ability to provide minimum inhibitory concentrations against particular organisms. This database served as the primary source for all datasets used in this project [13, 14].

The Collection of Antimicrobial Peptides database is another large database of antimicrobial peptides. This database attempts to use Hidden Markov Models to extract patterns from the AMPs that can be used to group similar AMPs together. It also provides various classifiers to identify likely AMP sequences from non-AMPs [15, 16, 17].

The data repository of antimicrobial peptides (DRAMP) database contains a large number of general AMPs as well as a large number of patented peptide sequences associated with AMPs [18].

1.4 Machine Learning Methodologies Applied in this Area

Many scientists have turned their hand to developing machine learning methodologies for classification of AMPs and predicting their properties [6]. In 2004 quantitative structure-activity relationship (QSAR) methods were employed to predict the antimicrobial potency of peptides. Specifically, a feedforward neural network was applied to whole peptide descriptors to predict the potency in terms of averaged minimum inhibitory concentration (MIC) measurements for the peptides [19].

In 2013, Lira et al. used a decision tree model to aid in AMP synthesis. They used a parent peptide as a scaffold and proceeded to model and then generate synthetic peptides. Two of these synthetic peptides, subsequently known as colossomin C and colossomin D, were shown to be antibacterial towards both *Staphylococcus aureus* and *Escherichia coli* [20].

In Antimicrobial Peptides Design by Evolutionary Multiobjective Optimization, MacCari et al. designed alpha-helical AMPs with both proteinogenic and non-proteinogenic amino acids [21]. Their features were developed by encoding QSAR descriptors into auto- and cross covariance values (ACC). ACC values describe the average interactions between residues distributed a certain distance apart in the peptide. The ACC values result in fixed length vectors describing the variable length peptide sequences. The approach demonstrated the ability to turn a non-antimicrobial peptide into a highly active AMP by including non-natural amino acids in the sequence.

1.4.1 Classifiers

In 2007, AntiBP analyzed 486 antibacterial peptides obtained from the APD database [22, 23]. Artificial Neural Networks (ANN or feedforward neural networks), Quantitative Matrices (QM), and Support Vector Machines (SVM) to classify peptides as antibacterial or non-antibacterial. Interestingly, the dataset used peptides extracted from non-secretory proteins as negative examples [24].

Also in 2007, Fjell et al. developed AMPer, a combination of both a clustering algorithm to generate important groups of peptides as well as a collection of hidden Markov models to model the clusters developed [25].

In 2010, AntiBP2 [26] used an SVM to classify antibacterial peptides using an updated version of the APD2 database [23]. Another group of scientists classified protein sequences with cysteine knot motifs as having or lacking antimicrobial properties using an SVM [27].

The next year, Torrent et al. [28] used an artificial neural network to map physico-chemical features of potential antimicrobial peptides to a classification as antimicrobial or non-antimicrobial. They also used an artificial neural network on the CAMEL dataset originally published in the 2004 paper by Cherkasov et al. to predict the potency of antimicrobial peptides. One interesting aspect of this work was the inclusion of a parameter representing the aggregation propensity of the peptide which aided in identification of the peptides activity or lack thereof. The negative examples of antimicrobial peptides for the classifier built in this work were pulled from the Uniprot [29] database. Specifically, 991 peptides not reported as toxic or antimicrobial were used.

Also in 2011, Wang et al. [30] used both a sequence alignment technique combined with a feature selection method to classify peptides as antimicrobial or not antimicrobial. The training portion of the dataset used in this study had positive instances derived from the CAMP database. The negative instances were initially derived from the Uniprot database by merely requiring them to be non-secretory and lacking the antimicrobial annotation. The final negative instances were then sliced out of this pool of peptides to make them the

proper length by randomly cutting a peptide from the original negative sequences that was 10-80 amino acids in length.

Fernandes et al. [6] used an adaptive neuro-fuzzy inference system (ANFIS) to classify antibacterial peptides. In addition to the accuracy of 96.7% and Matthews correlation coefficient (MCC) of 0.9356 on the dataset these scientists constructed for this work, the paper also demonstrated additional effort to obtain a proper negative dataset by running all potential negative examples through Phobius [31], a combined signal and transmembrane peptide predictor, as a database filter to ensure intracellular localization of the peptide and presumed non-antimicrobial activity. Interestingly, the paper employed only seven or eight whole peptide descriptors or features in the ANN and ANFIS methods they employed.

Joseph et al. [32] compared the ability of random forests and support vector machines to classify peptides as antibacterial, antifungal, or antiviral. In the same year, Porto et al. [33] used an SVM based method with physicochemical features to predict antimicrobial activity of protein sequences with cysteine knot motifs.

Ng et al. developed an algorithm for identifying AMPs based on both sequence alignment and a support vector machine [34]. They used a dataset consisting of 2,752 positive examples of AMPs and 10,014 negative examples. They also created a dataset with 0.7 sequence similarity by eliminating homologous sequences in the dataset. This dataset contained 870 positive instances and 8,861 negative instances. Furthermore they used two different test sets one from the work of Wang et al. with 1136 peptides and another based off of the CAMP database consisting of 2420 peptides. Their results indicated superior performance to the online methods associated with the CAMP database.

In 2015, Camacho et al. [35] used a class of unsupervised neural networks known as stacked autoencoders to extract new features from existing feature sets. They then used a support vector regression task to predict the MIC values of 101 peptides, each 15 amino acids in length. They improved on the result that Cherkasov et al. demonstrated on this dataset in the original use of it in 2004 [19].

In Machine Learning Assisted Design of Highly Active Peptides for Drug Discovery, Gigure et al. [36] developed a method to design highly active peptides for various binding tasks that employs the generic string kernel and a graph based algorithm to decide the optimal peptide for a particular binding task of an initially chosen length. They demonstrated the applicability of this approach on a dataset of 101 synthetic peptides. Using their approach they were able to generate two peptides that had activities equal to the best peptide of the 101 synthetic peptides.

1.4.2 Subsequence Classification

In 2015, Chang et al. attempted to predict the critical regions of AMPs that are responsible for the AMPs activity. A dataset consisting of critical regions was obtained by beginning with 2,497 peptides from a database, creating a nested AMP family where each AMP sequence in the dataset was associated with its full-length source protein, and finally finding overlapping regions of amino acids within each family to denote the critical regions. They employed a discriminative model known as Conditional Random Fields on this dataset to learn to identify the critical regions of these peptides by mapping each residue of the protein sequence to a corresponding label (critical or non-critical region). Finally the authors also analyzed the critical regions of AMPs to see which features were most important for the classifiers success. [37]

1.5 Neural Networks

1.5.1 Feedforward Neural Networks

A simple feedforward neural network has an input layer containing a number of nodes equivalent to the number of features and one additional node called bias, a second hidden layer containing some additional number of nodes with another bias node, and an output layer whose structure depends on the type of output desired without a bias node. Every node in the input layer is connected through a weighted edge to every node in the hidden

layer except for the hidden layers bias node. Similarly, every node in the hidden layer is connected through a weighted edge to every node in the output layer. In addition to the weighted edges, every node in the hidden layer represents an activation function such as a hyperbolic tangent function. Nodes in the hidden layer experience an activation equivalent to the sum of products between weights on input to hidden layer edges and the real value each input node sends to each hidden layer node. This activation is fed into the activation function resulting in a value that is fed along the edges connecting hidden layer nodes to output nodes. The output layer, depending on its structure, essentially compares the value it receives from the hidden nodes to a label associated with the features that were fed originally to the input nodes. Based on the difference between output and label, a loss function calculates the loss for a particular instance fed into the network. Due to the use of essentially differentiable non-linearities, the gradient of the loss can be computed with respect to the weights and also the inputs. The loss is backpropagated through the network by using the gradient of the loss with respect to each weight to update each weight in the network [38].

The notion of depth in feedforward neural networks is designed to refer not merely to the number of hidden layers the network contains or the number of nodes in each hidden layer, but more importantly to the notion that when additional layers or neurons are added to a network it gains the ability to combine simpler representations in the initial hidden layers to form more complex representations of the inputs in later hidden layers. This abstract notion is important to the idea that deep learning is about learning representations that are expressed in terms of other simpler representations [38]. With the advent of new graphical processing units, clever choices of activation functions, and unsupervised pre-training [39], the ability to train deeper and wider neural networks has become reality. Deep Neural Networks (DNNs) have even achieved the first instance of superhuman visual pattern recognition results. This occurred during a controlled competition, the IJCNN 2011 traffic sign recognition contest [40].

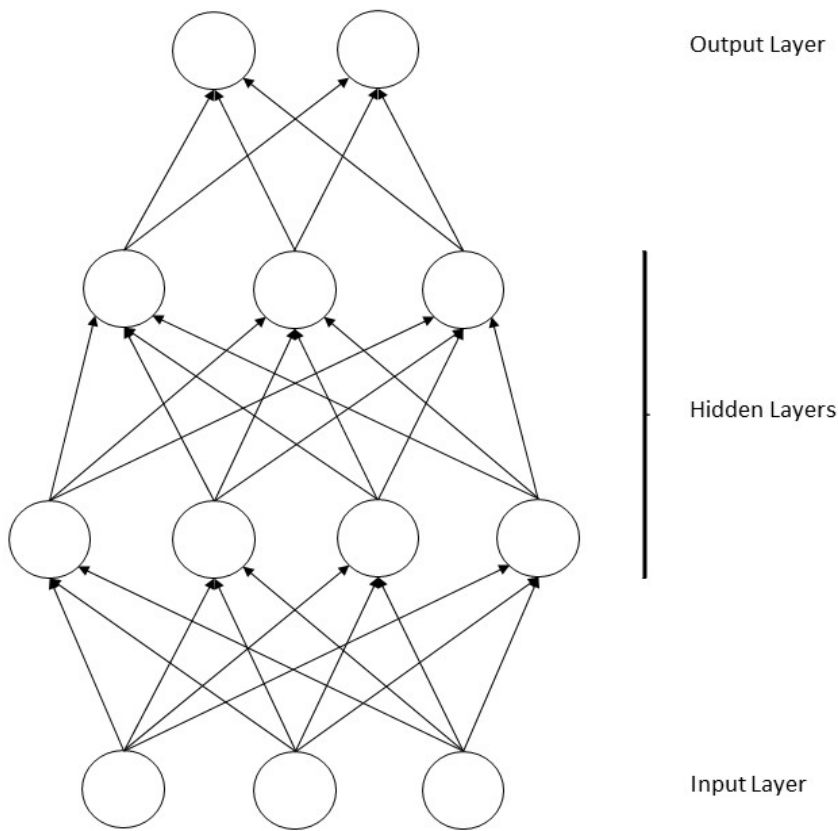


Figure 1.1: A multilayer perceptron with two hidden layers and no bias units shown.

1.5.2 Recurrent Neural Networks

A recurrent neural network (RNN) is a special form of neural network that allows for sequences of inputs to be mapped to a sequence of outputs. This is done by allowing each element in a sequence to influence the hidden layer activations of future values in the sequence. For example, if each element of the sequence represents the state of some system at different points in time, then outputs of the hidden layer of an earlier time state are summed into the activation in the hidden layer of the next immediate time step. This process is repeated for each element of the sequence. In essence the recurrent connections demonstrated in figure 1.2 allow for the network to remember the influence of earlier portions of the sequence [41].

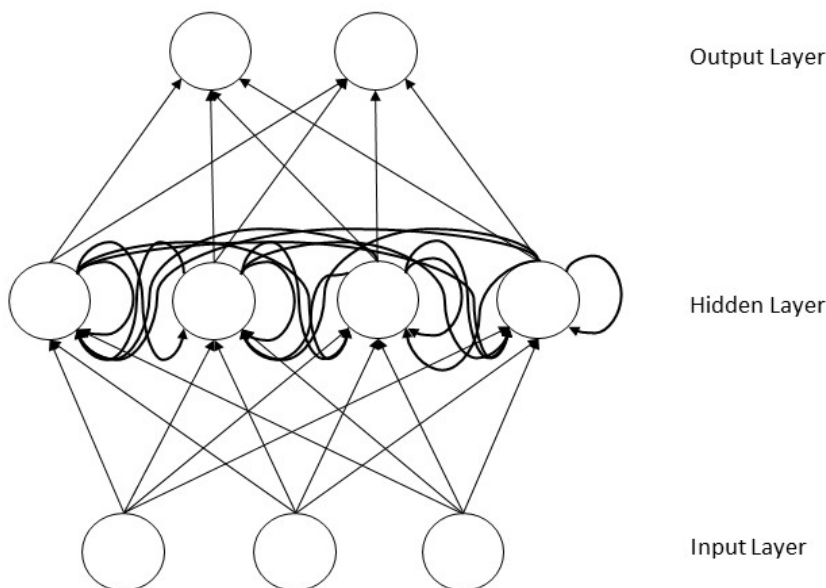


Figure 1.2: A recurrent neural network.

One useful way to visualize a recurrent neural network is to unfold the network along the input sequence. An example of an unfolded recurrent neural network is shown in figure 1.3. Note that biases are omitted and each layer of the network is turned on its side.

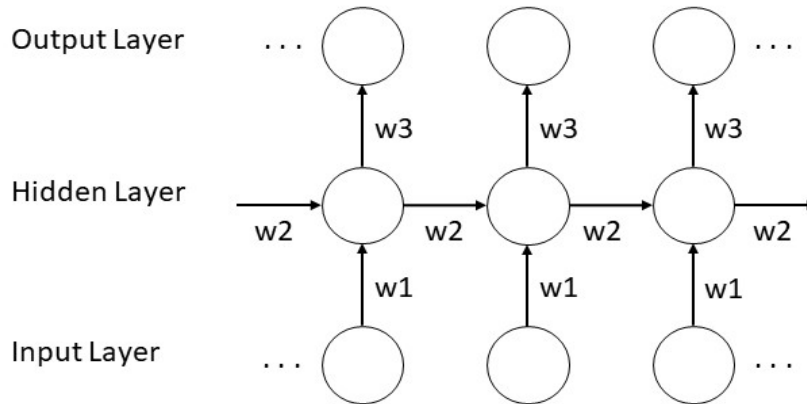


Figure 1.3: Unfolding of an RNN. Adapted from Fig. 3.4 of Graves [41].

Essentially each hidden layer circle or node represents all the hidden layer units in the network with a similar interpretation intended for the output and input layer. Another important characteristic to note of the RNNs is that weights are shared along the sequence. This is visible in figure 1.3 in that each instance of w_1 in the figure represents the same weight matrix mapping inputs to the activations of the hidden layer. The w_2 s in the figure represent the weight matrix of feedback from hidden layer outputs to the next hidden layer as one moves along the sequence. Finally, w_3 represents the shared weights between the hidden layer outputs and the output layers activations.

Forward propagation in an RNN begins with the specification of the initial hidden state, $\mathbf{h}^{(0)}$, which is often set to all zeros. For each time step the following update equations are

applied as shown in figure 1.4:

$$\mathbf{a}^{(s)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(s-1)} + \mathbf{U}\mathbf{x}^{(s)} \quad (1.1)$$

$$\mathbf{h}^{(s)} = \tanh(\mathbf{a}^{(s)}) \quad (1.2)$$

$$\mathbf{o}^{(s)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(s)} \quad (1.3)$$

$$\hat{\mathbf{y}}^{(s)} = \text{softmax}(\mathbf{o}^{(s)}) \quad (1.4)$$

where the parameters are given by the bias vectors \mathbf{b} and \mathbf{c} , and the weight matrices \mathbf{U} , \mathbf{V} , and \mathbf{W} which represent input-to-hidden, hidden-to-output, and hidden-to-hidden connections respectively. This is an example of a recurrent network that maps each input in the sequence to an output. The total loss for a sequences of inputs, \mathbf{x} values, paired with a sequence outputs, \mathbf{y} values, is the sum of the losses, denoted by $\mathbf{L}^{(s)}$, at each time step. The gradient computation involves forward pass from left to right the the unrolled graph shown in figure 1.4, followed by a backward propagation moving from right to left through the graph [38].

Since ordinary RNNs process sequences in the order of the elements in the sequence, they are unable to take into account future context. One way to resolve this problem is to train an RNN on both the forward and backward directions of the sequence and to connect their recurrent hidden layers to the same output layer. This elegant solution provides the complete past and future context for every point in the input sequence [41], [42, 43, 44]. Figure 1.5 displays an instance of an unfolded bidirectional RNN (BRNN). Note that the unfolded graph is acyclic which allows for backpropagation to be applied to train the networks weights. Also note the weight sharing that is similar to the case of the ordinary RNN.

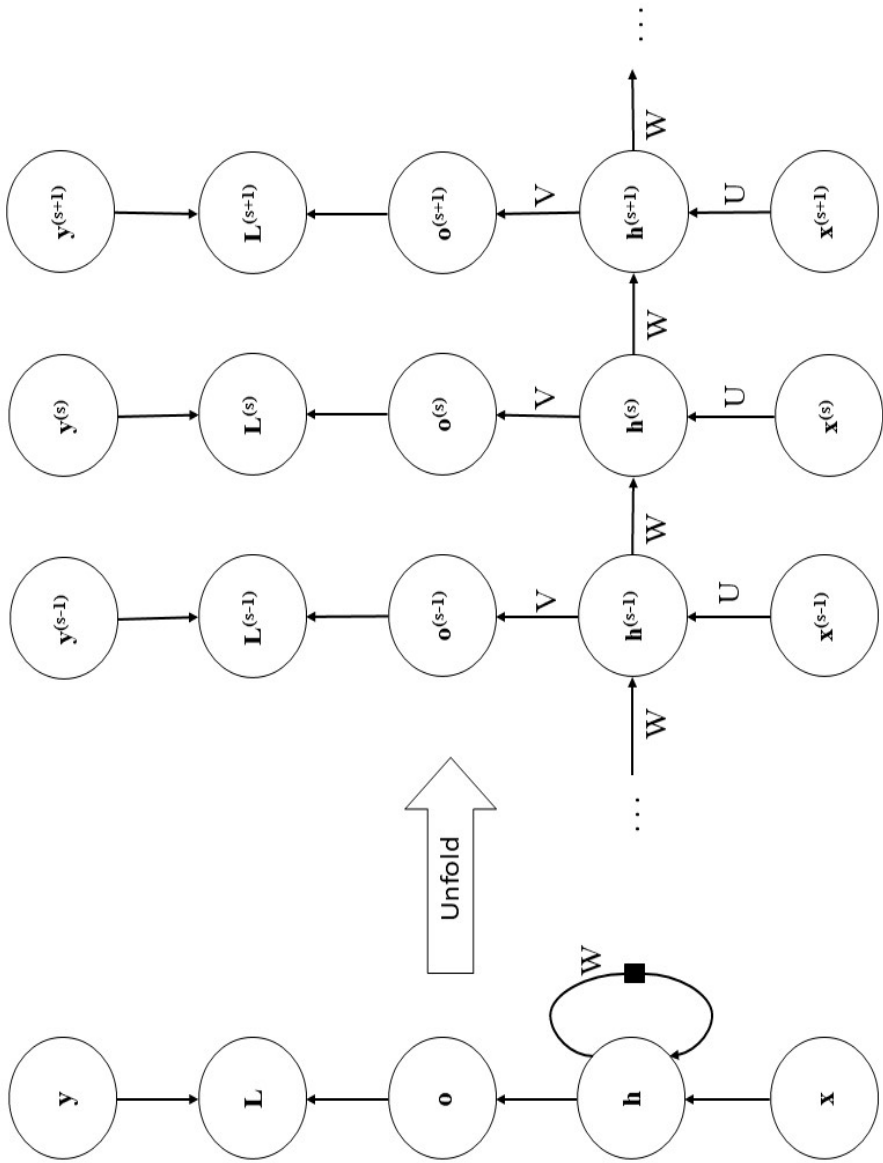


Figure 1.4: A standard unfolded RNN. Adapted from Fig. 10.3 of Goodfellow, Bengio, and Courville [38].

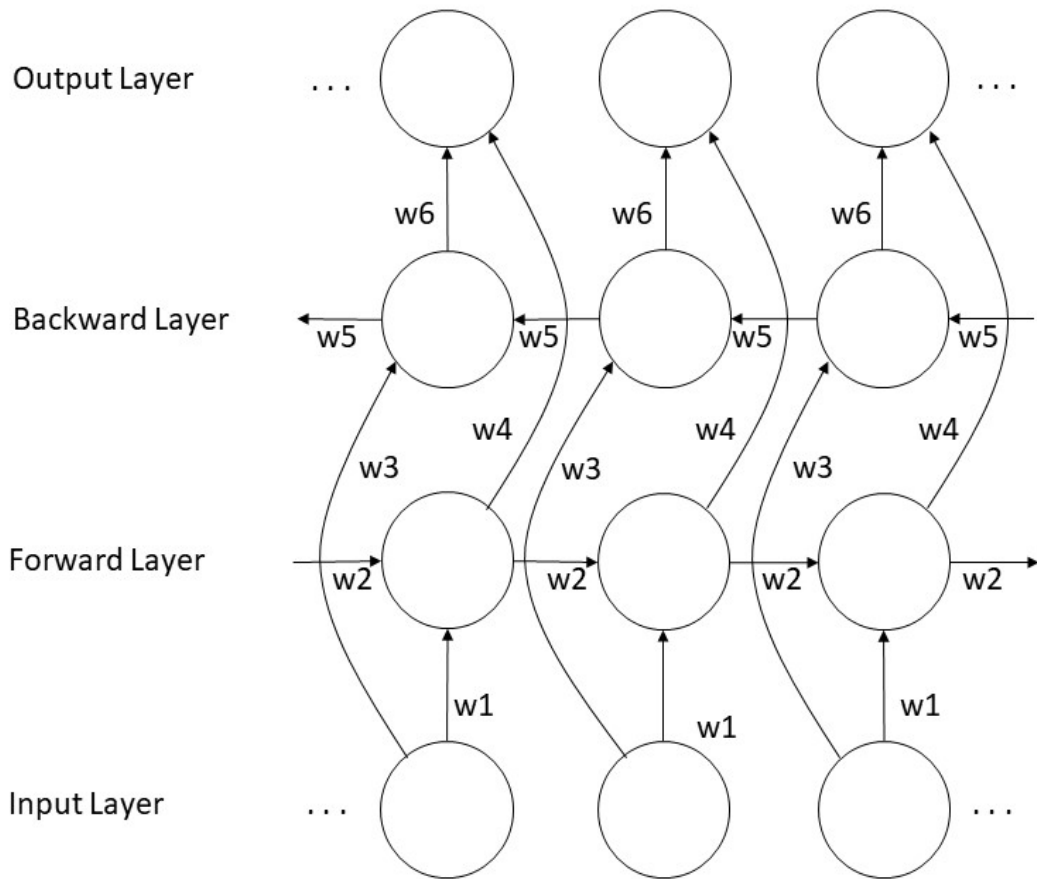


Figure 1.5: An unfolded bidirectional RNN. Adapted from Fig. 3.5 of Graves [41].

1.5.3 Long Short-Term Memory Architecture

An important recurrent neural network architecture that helps to solve the vanishing gradient problem is known as the Long Short-Term Memory architecture [45]. Long Short-Term Memory neural networks can essentially be thought of as modified conventional recurrent neural networks with the simple hidden units in the recurrent networks replaced by the more sophisticated LSTM unit as shown in 1.6.

The standard equations for a Vanilla LSTM as described in [46] and associated with 1.6 are shown below. The functions h and g are usually taken to be hyperbolic tangent (\tanh), and σ is typically the logistic sigmoid function.

$$\mathbf{z}^s = g(\mathbf{W}_z \mathbf{x}^s + \mathbf{R}_z \mathbf{y}^{s-1} + \mathbf{b}_z) \quad (1.5)$$

$$\mathbf{i}^s = \sigma(\mathbf{W}_i \mathbf{x}^s + \mathbf{R}_i \mathbf{y}^{s-1} + \mathbf{p}_i \odot \mathbf{c}^{s-1} + \mathbf{b}_i) \quad (1.6)$$

$$\mathbf{f}^s = \sigma(\mathbf{W}_f \mathbf{x}^s + \mathbf{R}_f \mathbf{y}^{s-1} + \mathbf{p}_f \odot \mathbf{c}^{s-1} + \mathbf{b}_f) \quad (1.7)$$

$$\mathbf{c}^s = \mathbf{i}^s \odot \mathbf{z}^s + \mathbf{f}^s \odot \mathbf{c}^{s-1} \quad (1.8)$$

$$\mathbf{o}^s = \sigma(\mathbf{W}_o \mathbf{x}^s + \mathbf{R}_o \mathbf{y}^{s-1} + \mathbf{p}_o \odot \mathbf{c}^s + \mathbf{b}_o) \quad (1.9)$$

$$\mathbf{y}^s = \mathbf{o}^s \odot h(\mathbf{c}^s) \quad (1.10)$$

LSTMs offer an elegant solution to the problem of vanishing gradient and allow for information to easily pass along the length of the sequence. They are also typically considered to be easier to train than their simpler RNN counterparts, as RNNs often require special optimization techniques in order to achieve similar or superior performance [47].

Like their simpler RNN counterparts LSTMs also allow for bidirectional application of the architecture to the sequence. The extension to bidirectional LSTM is similar to that from RNNs to bidirectional RNNs except that the forward and backward hidden units in the BRNN are replaced by LSTM units. Note that one may also stack multiple layers of

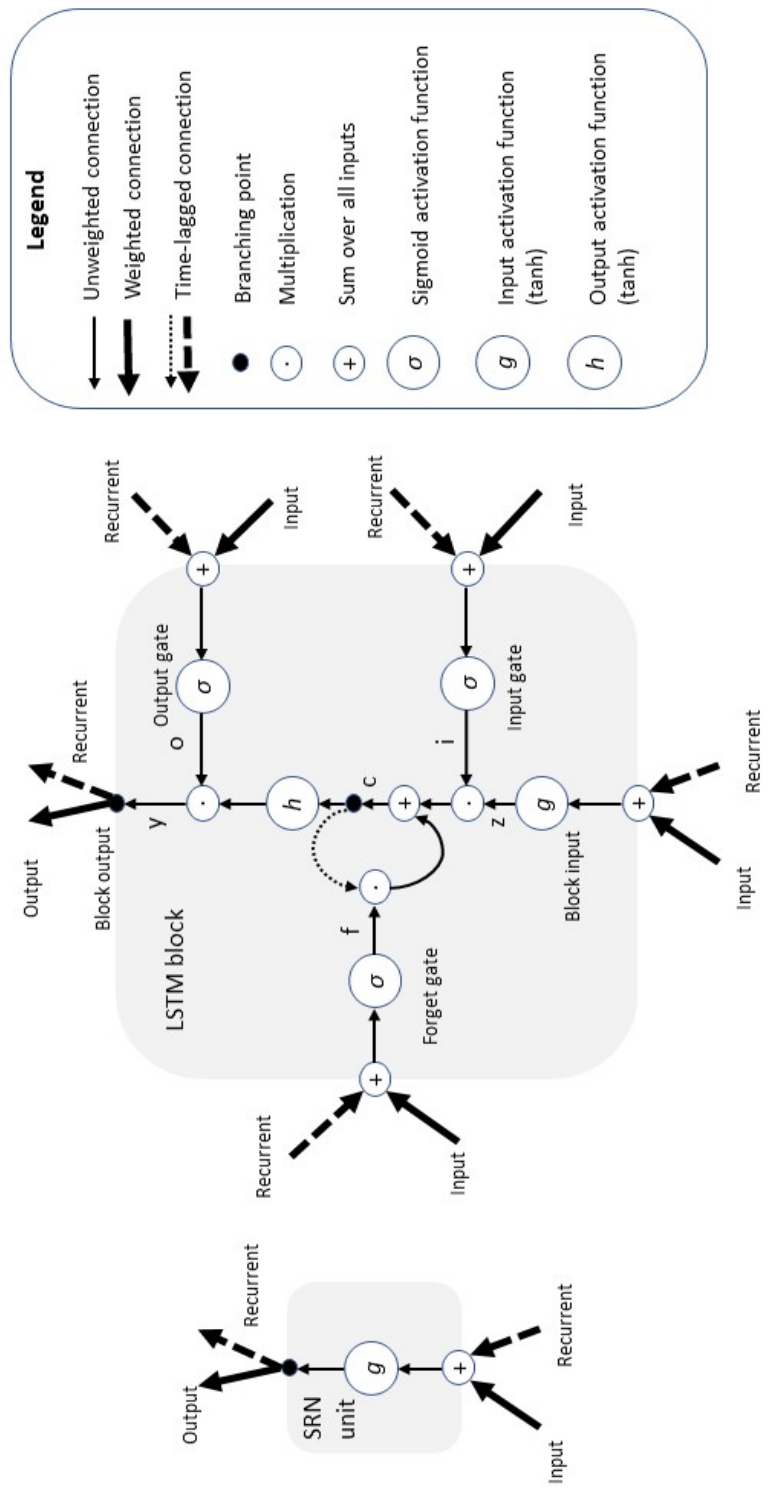


Figure 1.6: An LSTM block versus a standard RNN unit. Adapted from Fig. 1 of Greff, Srivastava, Koutnk, Steunebrink, and Schmidhuber [46].

hidden units or LSTM units before the output layer in order to create a deep bidirectional RNN or LSTM architecture.

1.5.4 Attention Mechanisms

Developed based on ideas taken from the field of psychology, attention mechanisms in the context of neural networks depend on two main aspects: decide which parts of the input needs to be focused on, and allocate the limited processing resources to the important part. In a survey on the attention based RNN model, Wang et al. describe four major types of attention mechanisms: item-wise soft attention, item-wise hard attention, location-wise soft attention, and location-wise hard attention [48]. Item-wise attention requires that the input sequence have clearly delineated items (such as amino acids in the case of antimicrobial peptides). By contrast, location-wise is more likely to be useful when dealing with inputs such as images where the location of objects in the image needs to be recognized. The image itself is one large feature map without clearly defined boundaries between objects. The difference between hard versus soft attention mechanism involves how the items or locations in an image are selected. In the case of hard attention, a sub-region of the image or an item or set of items in the image is picked discretely, while in soft attention a weighted linear combination of the items or sets of items or locations is created where the weights indicate the amount of attention given. Also, for soft attention the attention module is typically differentiable with respect to the inputs and the entire model including the attention mechanism can be trained end to end with backpropagation. In contrast, hard attention mechanisms make discreet decisions and are not end to end differentiable. They must typically be trained using special mechanisms [48].

1.5.5 Vector Embeddings

In natural language processing tasks distributed representations of words in a vector space help learning algorithms achieve better performance [49]. The notion of distributed rep-

representations of words in a vector space developed as an alternative to simple one-of-K encodings that given a total number of words in a dictionary (say K words in the dictionary), create a vector of length K with a single one in a unique position for each word. All the other positions are assigned the value zero. Such vectors are also referred to as one-hot vectors due to the single one entry. In contrast, the notion of a vector-space word embedding develops a vector for each word in the vocabulary that depends on the words context, the other words surrounding the immediate word of interest in large bodies of text. It has been noted that the vector representations of words computed using neural networks such as the Skip-gram model [50] , can encode many linguistic patterns and regularities. For instance, in Mikolov and Chen et al., the result of a vector calculation $\text{vec}(\text{Madrid}) - \text{vec}(\text{Spain}) + \text{vec}(\text{France})$ is closer to $\text{vec}(\text{Paris})$ than any other word vector of a city developed which illustrates the idea that Paris is the capital of France just as Madrid is the capital of Spain [50]. One can imagine doing something similar for the amino acids in proteins and peptides. The amino acid would serve as the word and the amino acids most commonly surrounding the one of current interest in a large collection of Proteins and Peptides would be the context that this amino acid is mostly found within. One can hope that when these amino acid-to-vector features are fed to an LSTM instead of one-hot vectors, they would aid in learning the classification result.

1.5.6 Generative Adversarial Networks

In 2014, Goodfellow et al. [51] introduced the concept of a generative adversarial network (GAN). A GAN essentially consists of two neural networks locked in competition where one network attempts to generate samples that fool the other network, and the second network tries not to get fooled. The first is termed the generator represented by G , and the second is termed the discriminator represented by D . G attempts to learn to model the distribution underlying the data while D attempts to distinguish between instances drawn from real data which is the training set and samples invented by G . D assigns a probability

of being real, meaning it was drawn from the training set, to each example it sees. The term example is used here to refer to either instances from the real data or samples from G . The two networks force each other to improve at their respective tasks. G attempts to develop samples so similar to the real data that D cannot distinguish between them and real training set instances while D attempts to get better at distinguishing between the two types of examples. Once training is complete, it is hoped that the samples invented by G are an accurate representation of the probability distribution that is responsible for generating the real data while D should output probability of approximately 0.5 for both the real data instances and the samples from G meaning that it can no longer distinguish between the real data and the samples generated by G .

Recently, work has been developed to extend the GAN concept to sequential data through the use of techniques such as SeqGAN [52]. These methods will be described in more depth in Chapter 4 of the thesis.

CHAPTER 2

TRAINING LONG SHORT-TERM MEMORY NETWORKS TO IDENTIFY ANTIBACTERIAL PEPTIDES

Much of the work in this chapter is in preparation for publication as Michael Youmans, John C. G. Spainhour, Peng Qiu " *Classification of Antibacterial Peptides using Long Short-Term Memory Recurrent Neural Networks*, IEEE/ACM Transaction on Computational Biology and Bioinformatics, under review, 2018.

2.1 Motivation, Background and Overview

In this chapter we introduce the core tool for the entire thesis, the Long Short-Term Memory Recurrent Neural Network, and apply it to a dataset composed of antibacterial and nonantibacterial peptides in order to understand its performance. We directly compare the LSTM approach to two other algorithms, a random forest classifier and a k-nearest neighbor classifier. Due to the LSTM architecture's flexibility and ease of generating appropriate features for amino acids composing the peptides, it was hoped that they would provide good performance on this task. In contrast, the random forest classifier and k-nearest neighbors approach require generating complicated features to describe each peptide before any learning can occur. In contrast the LSTM approach extracts a good feature vector during training using purely low level representations of the amino acids. In addition, we later attempt to apply the LSTM approach to tasks that are not very feasible with the random forest classifier or k-nearest neighbors approach. These tasks includes identifying important subregions of an antibacterial peptide and generating peptides similar to those in the training set or identifying important peptides from a random generation scheme. These last two tasks, however, will have to wait for Chapters 3 and 4.

2.2 Materials and Methods

Our approach employs a Long Short-Term Memory (LSTM) recurrent neural network, a Random Forests (RF) implementation, and a k-Nearest Neighbors (kNN) algorithm to classify peptides as either antibacterial or non-antibacterial. These methods are compared and contrasted on an identical dataset consisting of known antibacterial peptides and presumably non-antibacterial peptides.

2.2.1 Creating the Dataset

The positive antibacterial instances for the dataset were derived from the Database of Antimicrobial Activity and Structure of Peptides (DBAASP) [13]. This database contains peptides known to be active against various bacterial species from approximately 100 bacterial genera. We used peptides in this database that were active against at least one bacterial species, were less than 55 amino acids in length, and did not contain any non-proteinogenic or D-amino acids. The maximum lengths of peptides in the DBAASP database was approximately 100, but the relative number of positive instances with lengths greater than 55 is small. The negative instances were gathered from the manually annotated portion of the Universal Protein Resource (UniProt), which is also known as UniProtKB or Swiss-Prot [53]. This database was queried for peptides between 2 and 55 amino acids in length, with known function, no mention of being antibacterial, and not a secreted peptide according to the subcellular location parameter. After obtaining an initial set of positive and negative instances, we reduced the final collection of instances further by considering only the twenty proteinogenic amino acids composing the standard genetic code and removing any sequence containing an amino acid other than these. Any sequence with 3 or fewer amino acids was ultimately removed from the dataset as well as any sequence containing more than 55 amino acids. Reasons for the removal of sequences with 3 or fewer amino acids concern the use of a protein-protein BLAST+ for one of the algorithms compared [54].

The length cap of 55 was chosen due to the relatively small number of positive instance peptides with lengths larger than this in the DBAASP database. This helped balance the relative numbers of positive and negative instances in the dataset. Biopython was used to finalize a file containing all the peptide sequences of interest [55].

A second dataset was also constructed by taking the positive and negative instances described above and reducing the sequence similarity as determined by protein-protein BLAST+ among the collection of peptides. This is done by removing peptides until all peptide pairs remaining in the dataset have a sequence similarity less than a given threshold.

2.2.2 Representation of the Peptides

Two different feature representations of the peptides were used in this work. The first representation consists of an equivalent number of features for each peptide and was generated using the ProtDCal software [56]. ProtDCal provided 45,494 features. These features are developed based on features from individual residues such as hydrophobicity and electronic charge. These are augmented based on their local neighboring amino acids as well as grouped based on distantly related but similar amino acids in the sequence. After the groups are formed, operations such as mean value or the sum across all the features in the groups is obtained and serves as a single feature for the entire peptide. In some cases, if a peptide does not contain a single value from a group, the software will return a null value for that peptide. After removal of features that returned null values across all peptides in the dataset, the total number of features reduced to 45,378. Any remaining null values were replaced with 0 prior to scaling and normalization operations on the dataset.

The second representation takes in the amino acid sequence composing a peptide and returns a sequence of finite equivalent length vectors representing each amino acid individually. As the peptides composing the dataset vary in length, the number of finite length vectors varies for different peptides. The mapping from an amino acid to a finite length feature vector relies upon a one-hot vector (20), three different substitution matrices (20

from each), and a set of 6 physicochemical features known as NNAAIndex factors derived from Liang et al. [10]. Overall, each amino acid is represented by a vector of length 86. Each substitution matrix supplies twenty features for each of the twenty amino acids appearing in the dataset. These features are essentially the columns of the substitution matrix for each amino acid minus any irrelevant elements in the columns. The three substitution matrices used were PAM30, PAM70, and BLOSUM80 [57, 58]. They were chosen due to the lengths of typical peptides in the dataset, and were all downloaded from an NCBI ftp server (<ftp://ftp.ncbi.nih.gov/blast/matrices/>). A figure illustrating this arrangement of features can be seen in Fig. 2.1.

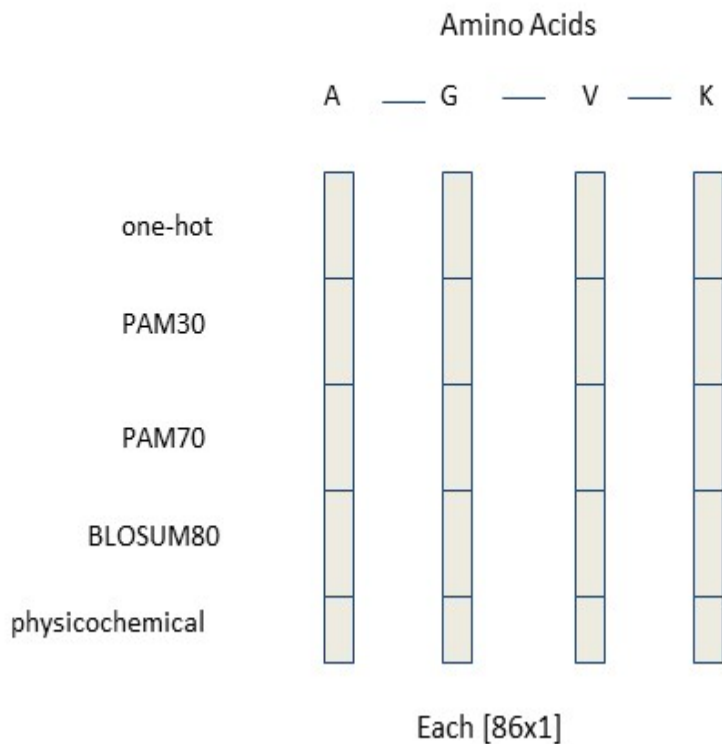


Figure 2.1: Illustration of the features used in the LSTM.

A third feature representation is similar to the second representation mentioned, but

adds predicted secondary structure features to the vector representation of each amino acid. This addition is eight features in length where each added features represents the probability that a particular amino acid in the peptide takes on a particular secondary structure. The secondary structure features were derived by computing the predicted secondary structure of all peptides according to an algorithm developed Wang et al. [59]. The eight types of secondary structure used are a more fine-grained version of the alpha-helix, beta-strand and coil region classes.

The final method used in the work, kNN relies upon NCBI protein-protein BLAST+ to create a distance matrix between all peptides in the dataset [54].

2.2.3 Algorithm for Creating Reduced Sequence Similarity Dataset

The algorithm for creating the second dataset containing only peptide pairs whose similarity lies below a threshold can be summarized as follows. First assign all peptides in the original dataset to a set A. Step one is to pick an element from set A at random and assign it to a set of peptides we would like to keep for our reduced dataset, say set K. Also remove this individual peptide from set A. Compare the similarity of all peptides still in set A to the most recently added peptide in set K. If any peptide currently in set A is too similar to the peptide in set K remove it from set A and place it in a set we intend to exclude from the second dataset, set E. After checking all the peptides currently in A against the most recently added element to set K, return to step one and repeat the process above until all peptides from set A are now located in either set K or set E. Note that each time we add a peptide to set K we already know it cannot be too similar to other peptides already located in set K. The random selection step makes sure we never bias the algorithm toward positive or negative instances in case they were not shuffled in advance. The steps of this approach are illustrated in Algorithm 1.

Algorithm 1 Algorithm for Creating Reduced Sequence Similarity Dataset

Input: A (collection of peptides), T (a similarity threshold)

Output: K (peptides to keep), E (peptides to exclude)

```
1: while  $A$  is non-empty do
2:   Randomly select a peptide from  $A$  and assign to  $K$ 
3:   Remove this peptide from  $A$ 
4:   for each remaining peptide in  $A$  do
5:     if this peptide from  $A$  and the most recently added peptide in  $K$  violate  $T$  then
6:       Place this peptide in  $E$ 
7:     end if
8:   end for
9: end while
10: return  $K, E$ 
```

2.2.4 Classification Algorithms and Implementations

Recurrent neural networks are a special class of neural networks that take in sequences of inputs and produce an output for each element of the input based on a hidden layer representation. These hidden layer representations in a recurrent neural network can be thought of as feeding back into the network for the next position in the sequence, and the weights composing the recurrent network are shared across the length of the sequence. In the case of recurrent neural networks for classification, the last internal feature representation produced by the hidden layer is often used as a representation of the whole sequence and is then fed into a feedforward neural network consisting of a series of linear and element-wise nonlinear transformations. This feedforward network culminates in a softmax layer that returns the probability of the peptide belonging to a given class. These probabilities are used in the calculation of the loss for the network. Recurrent neural networks can also proceed down both a forward and reverse direction along the sequence with the hidden layer representations for the two directions being concatenated into a single vector representing the output of the bidirectional recurrent neural network for each element of the sequence [42]. The first and last outputs can again be concatenated into an even larger vector which is then fed into a feedforward network. The reason for using bidirectional recurrent neural networks for these peptides is to more accurately obtain a representation that allows infor-

mation to flow in both directions along the backbone of the sequence and does not bias the network towards inputs nearer the classification layer.

Long Short-Term Memory recurrent neural networks are a special class of recurrent neural networks that use a special type of recurrent hidden layer unit [45]. Instead of the recurrent weight matrix and the input weight matrix in a simple recurrent neural network, LSTM blocks use several weight matrices [46]. Six such matrices control gates that regulate information flow into and out of an internal memory cell within each block. These added weights and the internal memory serve to help gradients flow across the length of the sequence and help control how information is passed along a sequence [41].

The vector formulas for a standard forward pass of a unidirectional LSTM are given in (1.5)-(1.10) [46]. In these equations \mathbf{x}^s is the input vector at position s in the input sequence, \mathbf{W} are rectangular input matrices, the \mathbf{R} are square recurrent weight matrices, the \mathbf{p} are peephole weight vectors, and \mathbf{b} are bias vectors. The σ , g and h represent point-wise non-linear activation functions where σ usually represents the logistic sigmoid function while g and h are typically the hyperbolic tangent function. The symbol \odot represents point-wise multiplication of two vectors. These equations are identical to those in Chapter 1 but are reiterated here for convenience.

$$\mathbf{z}^s = g(\mathbf{W}_z \mathbf{x}^s + \mathbf{R}_z \mathbf{y}^{s-1} + \mathbf{b}_z) \quad (2.1)$$

$$\mathbf{i}^s = \sigma(\mathbf{W}_i \mathbf{x}^s + \mathbf{R}_i \mathbf{y}^{s-1} + \mathbf{p}_i \odot \mathbf{c}^{s-1} + \mathbf{b}_i) \quad (2.2)$$

$$\mathbf{f}^s = \sigma(\mathbf{W}_f \mathbf{x}^s + \mathbf{R}_f \mathbf{y}^{s-1} + \mathbf{p}_f \odot \mathbf{c}^{s-1} + \mathbf{b}_f) \quad (2.3)$$

$$\mathbf{c}^s = \mathbf{i}^s \odot \mathbf{z}^s + \mathbf{f}^s \odot \mathbf{c}^{s-1} \quad (2.4)$$

$$\mathbf{o}^s = \sigma(\mathbf{W}_o \mathbf{x}^s + \mathbf{R}_o \mathbf{y}^{s-1} + \mathbf{p}_o \odot \mathbf{c}^s + \mathbf{b}_o) \quad (2.5)$$

$$\mathbf{y}^s = \mathbf{o}^s \odot h(\mathbf{c}^s) \quad (2.6)$$

TensorFlow was used in constructing the LSTM used here [60]. This software provides an opensource numerical tool for constructing and training over many common neural net-

work topologies.

The Random Forests algorithm was applied to the ProtDCal feature representation. The implementation of RF in scikit-learn [61] was applied. Grid search and 5-fold cross validation on the training set were used to tune several hyperparameters, including the maximum allowed depth for decision trees generated, the minimum number of samples required to perform an additional split at an internal node, the minimum number of samples required to appear at a leaf node, the criterion used to determine the split such as Gini impurity or Entropy, as well as the maximum number of features to consider when looking for the best split. The 45,378 features from ProtDCal were used to train the RF classifier. This algorithm was chosen as a potentially good algorithm for this dataset due to its inherent feature selection properties.

The kNN algorithm used is also a part of the scikit-learn toolkit for machine learning [61]. The distance matrix provided to the kNN algorithm was constructed using bit-scores returned by NCBI protein-protein BLAST+ [54]. The bit-score is essentially based on the summed cost of matches and mismatches of residues in an alignment, but modified so that bit-score is independent of the size of the database as well as query sequence length [62].

2.2.5 The LSTM and Shuffled Versions of Positive Instances

In order to further evaluate the performance of the LSTM methodology on the peptides, we took 199 positive instances and created 200 permuted amino acid sequences for each of the 199 original positive instances. An LSTM was once again trained on the same dataset but with a potentially different weight initialization as the LSTM already described. This was due to difficulty loading the original LSTM into the latest version of TensorFlow. The newly trained LSTM used identical hyperparameters as the original. The performance of the newly trained LSTM essentially matched the original on test set data.

We also trained an additional LSTM with the number of epochs reduced to 30 from 55. It was hoped that this may help better regularize the network. Once again this was essen-

tially identical to the original LSTM with only the number of epochs hyperparameter being changed. The more severe early stopping may have slightly improved the performance of the LSTM as it did show an improvement in test set accuracy and MCC. The 30 epoch network was originally trained in the expectation that improved regularization would assist the network in better dealing with shuffled peptides.

We ran the permuted and original sequences through the newly trained LSTMs in order to see the probability of the peptides being classified as antibacterial changing both before and after shuffling.

2.3 Results

In this work, we trained three different machine learning algorithms on two identical datasets described in the Materials and Methods section. The first dataset contains 2609 antibacterial and 3170 likely non-antibacterial peptides with 5779 total peptides, and we denote it as the original dataset. The second dataset was developed by taking the original dataset and reducing the sequence similarity between all pairs of peptides below a certain threshold, a bit-score of 17. The second dataset contains a total of 2475 peptides with 565 positive instances and 1910 negative instances. Due to the slightly unbalanced nature of the first dataset, stratified splits were used to create a held out test set consisting of 20% of the data as well as to split the remaining training instances to determine hyperparameters during 5-fold cross-validation. Stratified splits were also used on the second dataset of peptides with reduced sequence similarity. On all algorithms compared we measured both accuracy and Matthews correlation coefficient (MCC). It is important to note that MCC is arguably the preferred measure to the unbalanced nature of the both the original data and the reduced sequence similarity dataset.

2.3.1 Results of LSTM

The LSTM recurrent neural network applied to the amino acid sequence feature representation of the peptides was bidirectional. This means the neural network recurred over each sequence of amino acids from both the N-terminus to C-terminus direction and the C-terminus to N-terminus direction. The idea of applying a bidirectional LSTM to protein sequences, a relatively similar task, has been done before [63]. For each amino acid composing the sequence an input of 86 features was provided. The hidden layer consisted of 512 LSTM hidden units for each direction so the bidirectional hidden layer representation was a 1024 length vector after concatenation. The first and last outputs from the bidirectional hidden layer representation were then concatenated to generate a 2048 length vector, and passed through a feedforward dense layer into a 2 element softmax layer for classification. A figure illustrating the topology can be seen in Fig. 2.2. An identical topology was used for all the LSTMs trained both on the original dataset and the reduced similarity dataset as well as for those with and without secondary structure features.

Hyperparameters for the LSTM network were tuned manually using the same 5-fold cross validation as the other methods compared. The Adam, adaptive momentum, optimizer was used with default parameters including a learning rate of $1.0e-4$. A batch size of 128 was chosen, and the final network was trained for 40 epochs. The network also did not employ peepholes, the default setting in TensorFlow. This means the terms containing the symbol \mathbf{p} in (1.6), (1.7) and (1.9) are essentially zeroed out and can be ignored.

As mentioned above, the 86 amino acid features served as the input sequence to the LSTM. The features were centered and scaled so that the mean input for each feature across all sequence elements in the training set has mean zero and unit-variance except for the NNAAIndex factors which were only mean centered and the one-hot vectors which were neither centered nor scaled. For the LSTMs incorporating predicted secondary structure an additional 8 unscaled features were incorporated representing a discrete probability distribution over the different possible types of secondary structure. This brought the total

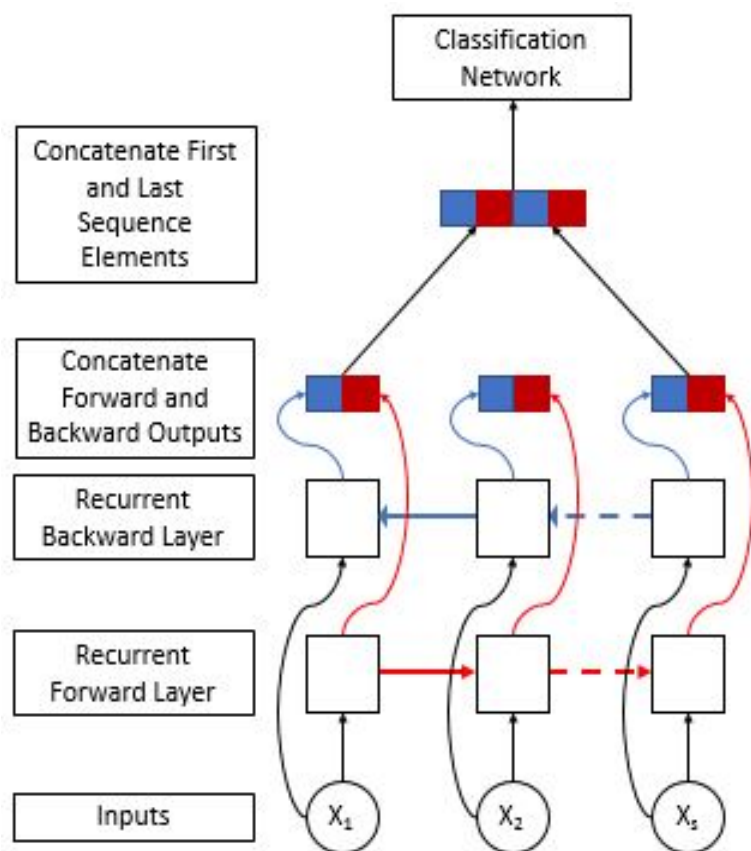


Figure 2.2: The LSTM topology used for all LSTM classifiers in Chapter 2.

feature count for the secondary structure LSTMs to 94 amino acid features. Also the final secondary structure LSTM was trained for 50 epochs.

The accuracy of 94.64% and Matthews correlation coefficient of .8920 shown in Table 2.1 was achieved by the LSTM on the original data without predicted secondary structure. When the predicted secondary structure was added to the feature representation of each amino acid in the peptide, the performance remained almost perfectly unaltered with an extremely minor change in MCC, a slight increase from .8920 to .8988. These results suggest the peptide representation derived by LSTM at the concatenation layer efficiently summarized the sequence of amino acid feature vectors in a way relevant to the classification task.

In the context of the reduced sequence similarity dataset, we see that the LSTM loses some performance with MCC dropping from .8920 and .8988 to .8240 and .8270 respec-

tively for the LSTM with and without secondary structure. This illustrates reduction of the sequence similarity of the peptides has a definite negative impact on the LSTM performance and shows that the reduced similarity dataset is tougher; a trend that can be seen in all the algorithms tested. Also note a drop in accuracy is also present across both the LSTM with secondary structure and the one without in the case of the reduced sequence similarity dataset. These results can be seen by comparing Table 2.1 with Table 2.2.

Table 2.1: Algorithm Performance on Original Data

Algorithm	Accuracy	MCC
kNN	91.78%	.8340
RF	94.29%	.8855
LSTM	94.64%	.8920
LSTM with Secondary Structure	94.98%	.8988

Table 2.2: Algorithm Performance on Reduced Sequence Similarity Data

Algorithm	Accuracy	MCC
kNN	86.87%	.5970
RF	92.32%	.7796
LSTM	93.73%	.8240
LSTM with Secondary Structure	93.94%	.8270

2.3.2 Results of Random Forests

The RF algorithm performed nearly as well as the LSTM on the original dataset. As can be seen in Table 2.1, the RF algorithm achieved 94.29% accuracy and an MCC of .8855. On the reduced sequence similarity dataset, the performance drop was greater than that for the LSTM algorithms especially in terms of MCC. The algorithm reported a new MCC score of .7796 for the reduced similarity dataset.

The optimal performing RF during cross validation on the original dataset used: information gain for measuring the quality of splits in the decision trees, and the square root of

the total number of features during each split. Also, each tree required at least 3 samples at each leaf node. The forest was composed of 128 decision trees, and no max depth for each tree was set. On the reduced sequence similarity dataset, the optimal performing RF according to cross validation used: the information gain criterion for measuring the quality of splits, and the square root of the total number of features during each split. The forest was composed of 128 decision trees, which could also required at least 3 samples to be present at each leaf node.

2.3.3 Results of kNN

In order to determine how much sequence similarity contributes to the performance of machine learning algorithms on this dataset, we applied protein-protein BLAST+ to the data to generate a pairwise similarity matrix, and used kNN to classify the test set based on nearest neighbors in the training data. This allows us to compare the classification performance based purely on sequence similarity to the above algorithms that take in physicochemical descriptors and other information in addition to sequence representations.

The kNN algorithm was tuned using 5-fold cross validation. The potential number of nearest neighbors employed ranged from 1 to 20. NCBI protein-protein BLAST+ requires an E-value threshold to return metrics for peptide similarity or distance. If a peptide pair has an E-value greater than this threshold the alignment information is not returned [54]. The E-value or Expectation value indicates the probability that an alignment in a database search occurred by chance [62]. In order to obtain as much information regarding the distance between peptides as possible, the E-value threshold was set to an extremely large value, $10e30$.

The best performing kNN algorithm on validation data used 3 neighbors and the distance between the query point and its neighbors calculated as $1/(1+\text{bit-score})$, where bit-score is the similarity between pairs of peptides. As mentioned earlier, the bit-score is essentially based on the raw score or the summed cost of matches and mismatches of residues

in the alignment, but bit-score is independent of the size of the database as well as query sequence length. For this reason we preferred bit-score as the means of determining similarity via calls to protein-protein BLAST+ [54, 62]. On the original data, The kNN test set classification accuracy was 91.78% and its MCC was .8340, which indicate it underperformed both random forests and the LSTM. On the reduced similarity dataset, the best performing number of neighbors was 2 according to validation data and the algorithm used distance between neighbors again. An accuracy of 86.87% and an MCC of .5970 was reported on the test set.

2.3.4 Overall Results

The final test set scores on the original data are shown for the three algorithms in Table 2.1. They show the LSTM outperforms the RF and kNN methods in terms of accuracy but the difference in performance between the LSTM and RF is likely insignificant with a difference of approximately 1% on an identical test set. A potential reason for the underperformance of the kNN algorithm is its total reliance on sequence similarity alone. The LSTM and RF likely performed better because of their ability to incorporate some other features such as physicochemical properties that are important for identifying antibacterial peptides. Note also that the addition of predicted secondary structure features for the LSTM in Table 2.1 showed virtually no improvement to the performance of the algorithm.

After reducing the sequence similarity among the peptides in the dataset as discussed in Algorithm 1 and checking performance of the same algorithms, it became clear that the LSTM both with and without secondary structure features better maintained its original performance. In essence, all classifiers do worse on the new harder dataset, but the gap between the performance of the two LSTM algorithms and the RF and kNN widens. Once again, the addition of the predicted secondary structure features had little impact on the performance of the LSTM with and without them.

2.3.5 Results of Shuffling Positive Instances on the LSTM Predictions

It was hoped that by shuffling the positive instances we would observe a decrease in the classifier's confidence that the shuffled peptides should be given the same label as the original unshuffled peptide. This was not typically observed for the training set instances. Specifically, when a positive training set instance is shuffled and then run through either the 55 epoch retrained network or the 30 epoch retrained network, the classifier continues to classify the shuffled peptide in the same manner as the original peptide in the training set and does so with high confidence. The same is not necessarily true of the test set instances, as the network has never observed them. If one shuffles a test set instance, one may very well see a reduction in probability of the positive test set instance being classified as antibacterial. If one examines 2.3, one can see that the vast majority of the shuffled peptides are present in the upper right quadrant of the figure. This means that if a peptide from the positive instances is predicted to be positive by the classifier, then on average its 200 shuffled versions are also likely to be classified the same way. Most of the exceptions to this obvious trend are likely due to test set peptides being chosen which the classifier hasn't seen. These likely account for the majority of the circles that are not in the top right quadrant of the image. We discuss the full implications of these results in the next section.

2.4 Discussion

Although RF performance is comparable to the LSTM performance on the original dataset, the LSTM features are much simpler and more easily produced than those produced for the RF classifier because the LSTM features consist of amino acid level features only. There is also nothing preventing the inclusion of some of the features used in the RF classifier being translated into features usable by the LSTM. By repeating the RF feature for each amino acid feature vector, one could deliver a constant signal to the LSTM over the length of a peptide. This feature would change from peptide to peptide, but would be constant for

each individual peptide.

When one shifts attention to the reduced similarity dataset, it becomes clear that the LSTM algorithms suffer less from the effect of the new dataset. It is clear that the new dataset should be harder for the kNN due to this algorithms total dependence on sequence similarity for successful classification results. It is especially interesting that the gap between the LSTM algorithms and RF widens as one passes from the original dataset to the reduced similarity data. Perhaps this also indicates that the LSTM is less dependent on sequence similarity than the RF features that were generated.

As briefly mentioned in the introduction, another important concern is the quality of the negative instances used in the dataset. Most datasets constructed to date for the purpose of identifying antibacterial peptides do not actually contain guaranteed negative instances based on experimental measurements such as a Minimum Inhibitory Concentration (MIC). In order to build such a negative instances set for the notion of antibacterial versus nonantibacterial, one would need to obtain poor MIC measurements against some reasonable number of bacterial organisms in order to have confidence that this peptide is indeed a negative instance. The dataset used in this work is no exception to this problem as the negative instances for it come entirely from peptides within Swiss-Prot with known functions that are not considered antibacterial and are not secreted [53]. One possible solution to this issue that will be explored in future work is to consider designing organism specific classifiers on positive and negatives instances with as many known MIC values as possible. The DBAASP database used in this work for generating the positive instances does contain fairly large numbers of peptides active against potential target organisms such as *E. coli* with some negative instances also available [13]. The caveat is that as the quality of the negative instances is improved due to these examples, the number of positive instances will shrink and the number of genuine negative instances with MIC measurements is still likely to be relatively small.

2.4.1 The Implications of the Shuffled Positive Instances

The fact that the LSTM does not significantly alter the probabilities of the shuffled positive training set instances essentially indicates that the classifier is basing a large aspect of its predictive capacity on the composition of the peptides rather than the peptides' full primary structure. This does not necessarily a fault of the classifier, however, as it is possibly due to the nature of the positive instances and the fact that there are very few, perhaps zero, negative instances that can contradict the "sufficiency of composition" in the training set. Also as will be discussed, it is not clear how important the full primary structure is to the activity levels of the peptides.

The reason for the lack of negative instances capable of illustrating to the classifier the likely insufficiency of composition is simply due to the enormous size of the instance space, on the order of 20^{55} possible peptides with only approximately 4500 negative instances. This means the odds of having good negative instances on hand to properly contradict the "sufficiency of composition" are extremely low. The basic problem here is that for the dataset we are currently able to provide to the LSTM, the LSTM can obtain very good training and test set performance while not needing to fully leverage the full primary structure of the peptide. The natural question is whether or not the LSTM should be needing to leverage the full primary structure. According to some researchers it is not clear how much the full primary structure of the peptide, secondary structure, or tertiary structure should characterize the activity of these peptides. Despite decades of intense search for sequence-structure-function relationships for antimicrobial peptides, such relationships are rarely found and evidence for well defined transmembrane pores is rarely seen [64].

Specifically, there have been experimental studies where researchers have permuted the order of amino acids in a known antibacterial peptide and then tested the permuted versions. In the work of Hilpert et al., 49 peptide variants of a known antibacterial peptide were generated through permuting the amino acids in the original peptide. This means each variant had an identical amino acid composition to the original as well as the same overall physic-

ochemical properties. The authors state that this is a situation ideally suited for examining the importance of primary amino acid sequence for the activity. Very interestingly they found that approximately half of the peptides tested were at least as active as the original. Exactly 6 were classified as being substantially more active than the original peptide and 4 were classified as being slightly more active than the original. 14 of the variants showed similar activity to the original peptide. The other half of the peptides were slightly less active or less active than the original with only 2 of the peptides showing weak killing activity even at the highest concentrations and were considered inactive. The authors go on to state that in addition to composition there are likely a few additional features that determined the activity in this scenario. They postulate the existence of these additional features due to the fact that if amino acid composition alone was sufficient then all peptides would show very similar activity to the original [65].

In summary, composition seems to be an important predictor of antibacterial activity and may serve as a critical feature in a successful classifier. Composition on its own however should not be seen as a completely sufficient predictor of antibacterial activity, and as Hilpert et al. [65] indicate should be complemented by at least some aspects of physicochemical properties and their relative arrangements in the primary sequence while not necessarily demanding the full primary sequence exist in a certain form. While the LSTM classifier we have demonstrated here can certainly leverage the composition successfully, it may still be overconfident with respect to composition for training set instances due to the sparseness of the dataset. A potentially obvious fix for this problem is to make sure the training set is enriched for permuted variants of the original peptides which are also correctly labeled according to their activity. This however would require testing these variants and adding them to the training set similar to what Hilpert et al. performed [65]. Another method to hedge the LSTM classifier's tendency to bet highly on the composition is the string kernel idea, but it is likely that adding the permuted variants to the training set would be a better approach. In fact, the LSTM should truly excel at the composition, physico-

chemical, and regions of primary structure that are critical to predict the true label, but the needed testing for the variants to tease out the latter two aspects that are most critical may be expensive.

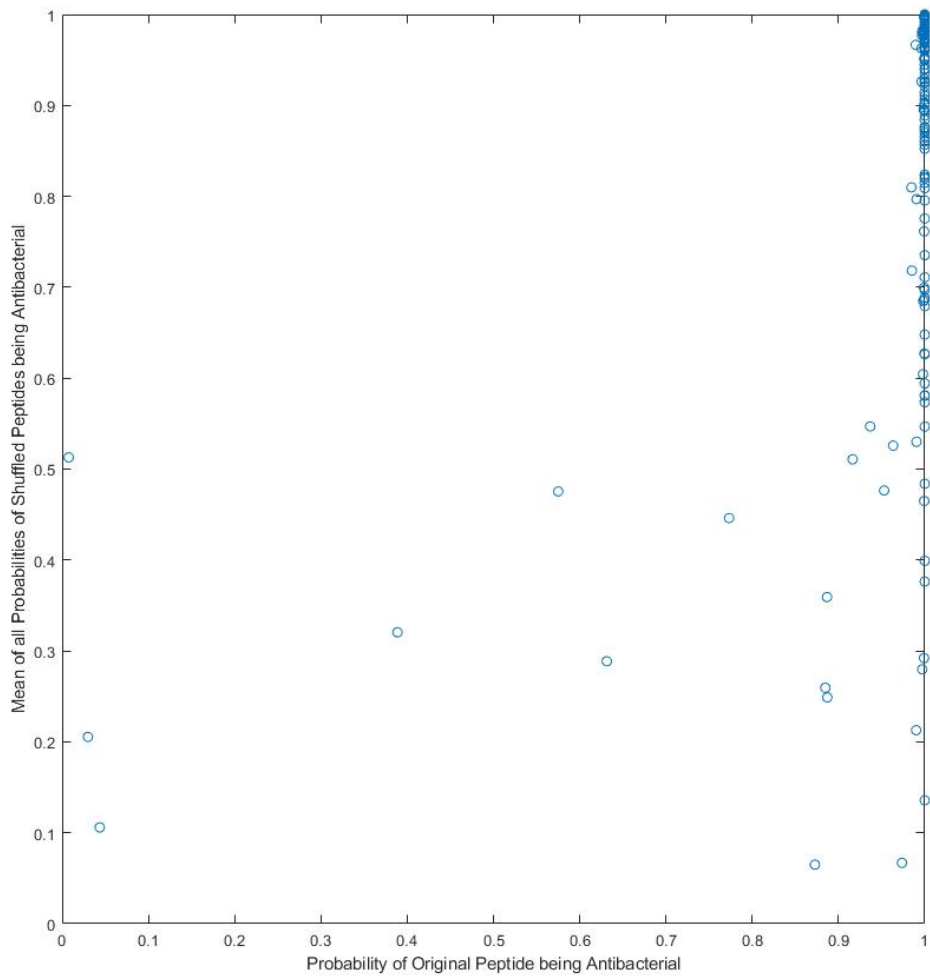


Figure 2.3: The effect of shuffling positive instance peptides on their probability of being antibacterial according to the classifier.

CHAPTER 3

IDENTIFYING CRITICAL REGIONS OF ANTIBACTERIAL PEPTIDES

3.1 Motivation, Background and Overview

In this chapter we develop the second major goal of the thesis, to provide some technique for denoting the importance of certain amino acids in a peptide to the overall probability of the peptide being antibacterial. We take two major approaches. The first rather simplistic approach is to take the classifier from Chapter 2 as well as peptides from the positive instances to note how point mutations affect the probabilities provided by the classifier. The second approach attempts to use an attention mechanism as briefly described in Chapter 1 to identify important regions as well. Before describing the specifics of each approach, we will provide some background on the second major approach taken, attention mechanisms in recurrent neural networks.

3.1.1 Attention Mechanisms

Attention mechanisms were first introduced by Bahdanau et al. [66] in an encoder-decoder framework for machine translation. Here a soft attention mechanism was introduced that allowed a machine translation model to focus attention on intermediate representations of words in a source or input sentence so that the model could better translate the source sentence into another language. This involves using a set of weights in a linear combination of a representation of the words to distribute attention. The soft attention mechanism also allows one to train both the attention neural network and the original model jointly [48].

For this work with a bidirectional LSTM, it was decided to design the attention mechanism in a similar manner to two papers that dealt with the same task in different areas. The first of the two papers is due to Yang et al. [67] who used the idea of a bidirectional

LSTM with attention to perform document classification. The topology used in their work is similar to the types of bidirectional LSTMs trained in Chapter 2 as shown in figure 2.2. In figure 3.1, we can see the changes made to what was done in Chapter 2. In fact, from the position of the inputs to the outputs of the bidirectional LSTM the networks are essentially identical. After the outputs from the LSTM are generated, however, the network using attention incorporates an attention neural network that takes in each output from the LSTM that represents an individual amino acid and generates a weight which we will denote by α_i where $i \in \{1, s\}$ and s is the length of the peptide. The α_i form a discrete distribution over the amino acids composing the peptide and when combined with the output vectors in a linear combination a new vector is obtained that is then fed into the classification network to determine the class label. The precise mathematical description is given below in the subsection 3.3.1. In [68] Sønderby et al. also use a very similar architecture to augment a bidirectional LSTM with attention, but their work is geared toward the subcellular localization of proteins. This work is followed up with the work of Armenteros et al. that also employs an attention mechanism which they argue not only improves the performance of their bidirectional LSTM network at the subcellular localization of proteins, but also serves as a way to identify important regions of proteins for subcellular localization [69]. The work of Armenteros et al. [69] makes the attention mechanism more sophisticated and relies upon a decoder recurrent network similar to what was done in the original work of Bahdanau et al. [66]. The decoder network allows the attention weights to be updated a greater number of times in comparison to the simpler attention mechanism that was used in Yang et al. [67].

3.2 Peptide Point Mutations

In this section the idea of using the classifier trained in Chapter 2 to elucidate important regions of antibacterial peptides is developed. This is done by taking peptides that are known to be active and are predicted to be active by the LSTM then examining the changes

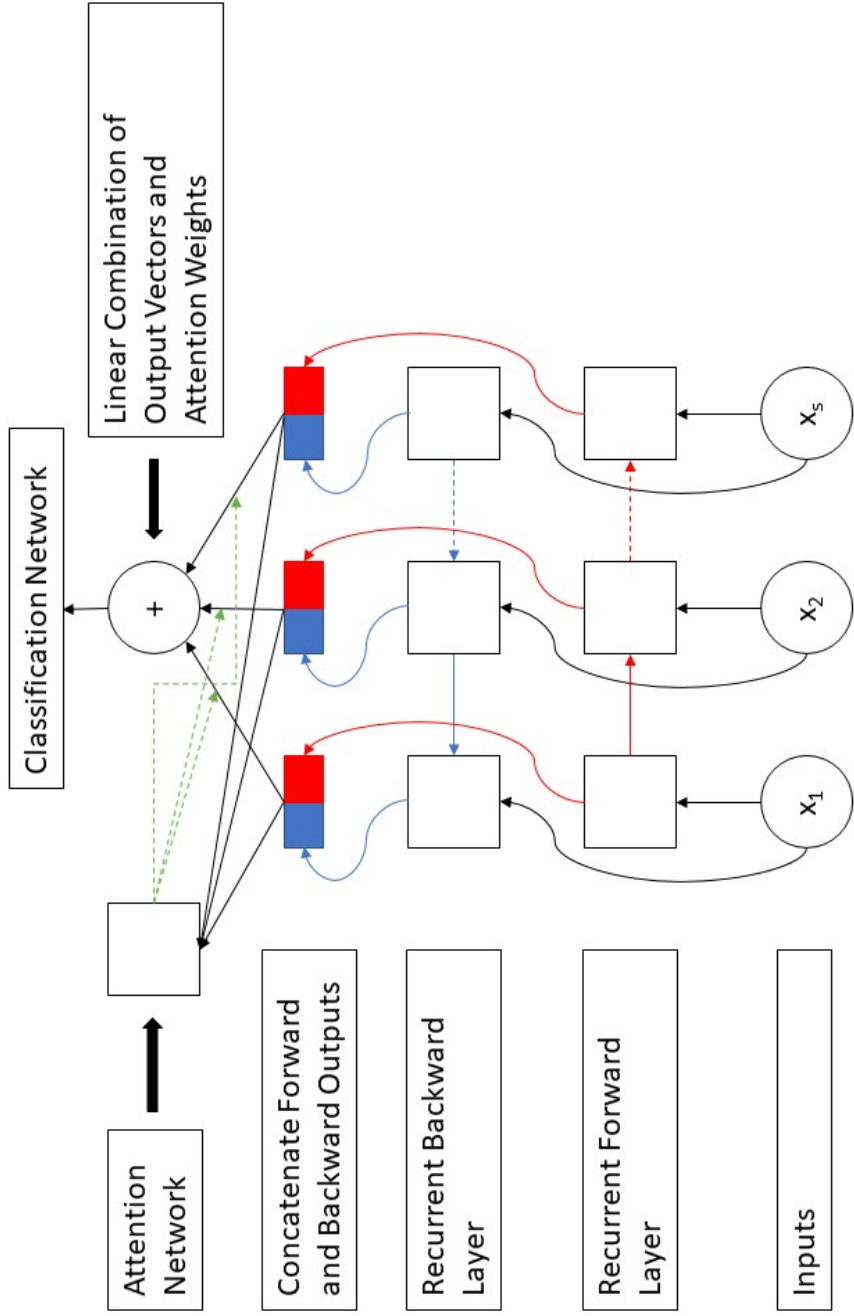


Figure 3.1: An illustration of an LSTM with an attention mechanism on the outputs of the hidden layer.

in probabilities provided by the LSTM after mutating individual amino acids in the original peptides.

3.2.1 Chosen Peptides for Point Mutations

Two peptides were chosen for providing a proof of concept for the point mutations strategy for identifying critical amino acids in peptides solely through the use of the same type of bidirectional LSTM used in Chapter 2. The two peptides chosen were both positive instances that were predicted to be positive with high probability by the neural network. The peptides' sequences are provided in Table 3.1.

Table 3.1: Peptides for Point Mutations

DBAASP ID	<i>Peptide Sequence</i>
759	ACYCRIPACIAGERRYGTTCIYQGRLWAFCC
7869	RWCVYAYRRVRGVLVRYRRCW

3.2.2 Effect of Point Mutations on the Class Probability

The effect of the point mutations on the class probability is illustrated in figure 3.2 and figure 3.3.

In figure 3.2, it is clear that the fifth residue in the original amino acid sequence, an arginine residue, may play an important role in the class label assigned as the mean probability across the point mutations yields a value of approximately 0.8. Although it occurs at only on one residue, it does seem to indicate that this particular arginine plays an outsize importance in the class label assigned. Also note that the other arginine residues in the same peptide do not have the exact same effect across class probabilities. this indicates in reference to Chapter 2 that amino acid composition is not the only feature that these recurrent networks are basing their classification upon.

In figure 3.3, one can see that only one point mutation in the first position is able to flip

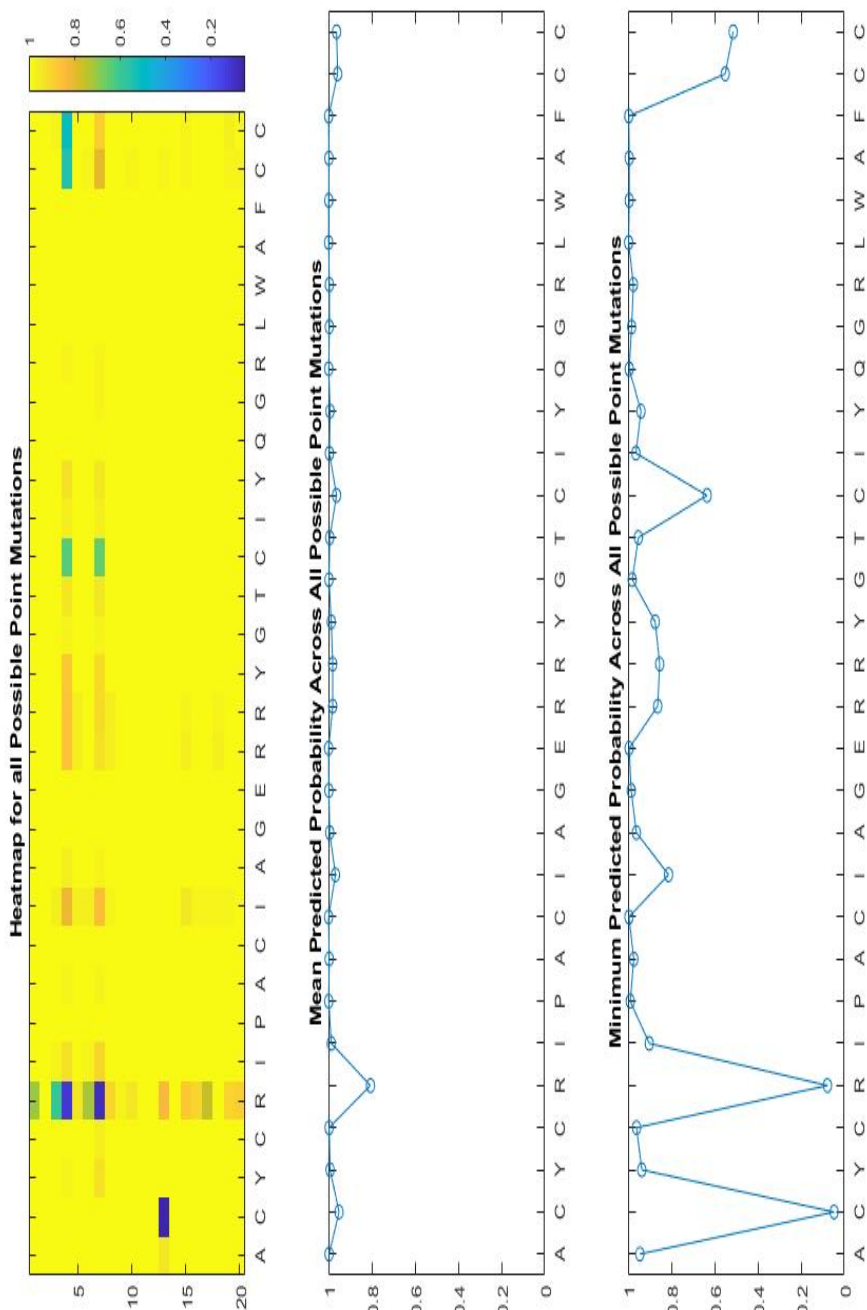


Figure 3.2: A collection of plots illustrating the effect of point mutations on the peptide with DBAASP ID 759.

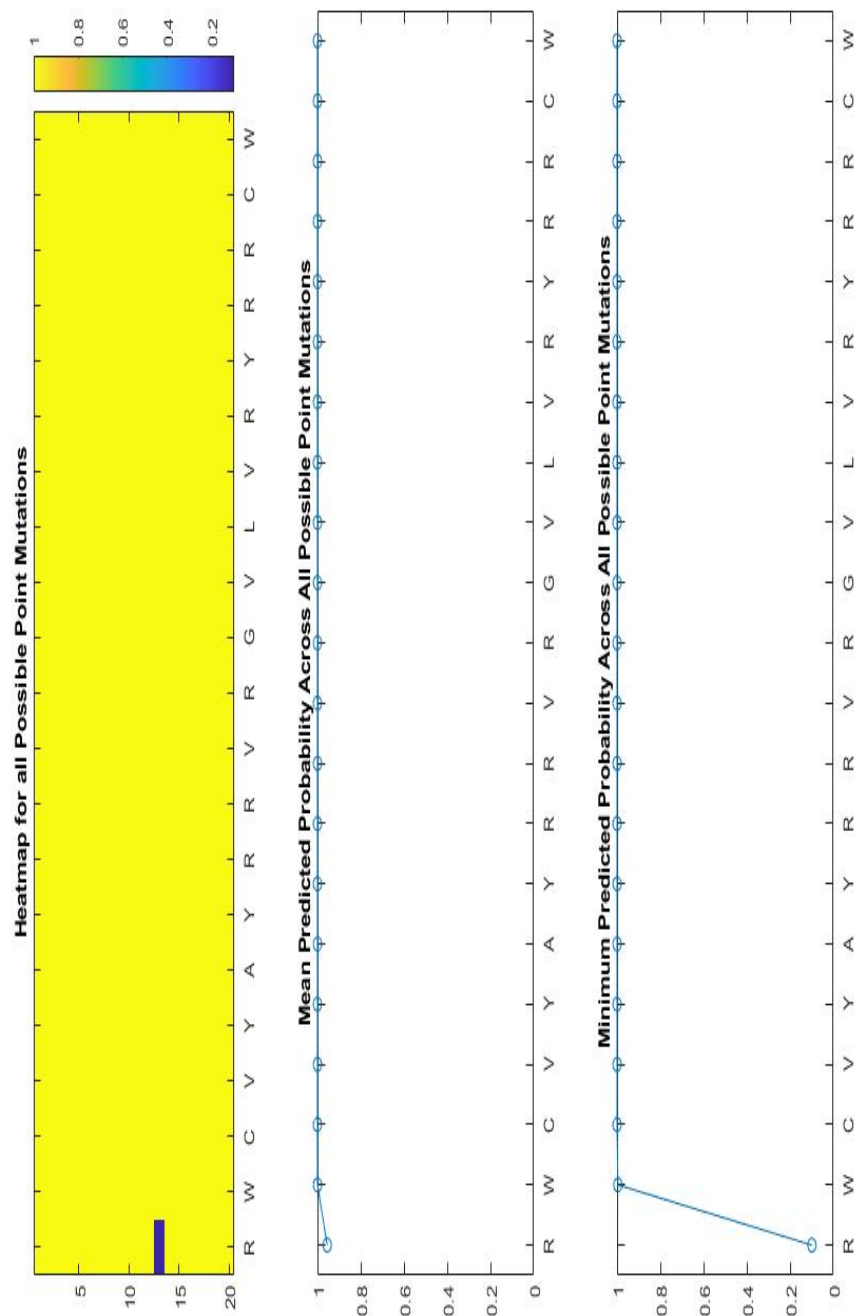


Figure 3.3: A collection of plots illustrating the effect of point mutations on the peptide with DBAASP ID 7869.

the class label from antibacterial to nonantibacterial. Overall there is little evidence that any single position is able to have large impact on the class probability assigned.

3.3 Attention Mechanism with Bidirectional LSTM

In this section, the development of a Bidirectional LSTM with attention mechanism will be described and the results of its application to toy data sets will be discussed.

3.3.1 Precise Soft Attention LSTM Topology

A Bidirectional LSTM Topology with attention mechanism on the outputs from the hidden layer was developed in TensorFlow [60]. The idea is that the hidden layer directly associated with each amino acid input is representative of that amino acid's contribution to the feature representation that is used to classify the peptide as a whole. The attention network takes in the hidden layer representation and then provides a weight indicating the importance of this particular amino acid feature representation to the overall class label assigned. This is done by using the weight, a number between zero and one with the sum of all weights across all amino acid in the peptide equal to one, to scale the relative contribution of that hidden layer representation to a summed feature representation of the peptide as a whole. The summed representation is simply the sum of all hidden layer amino acid representations after being multiplied by their respective weight. This summed representation can then either be fed through a feedforward network and ultimately a softmax for classification.

3.3.2 Toy Data Set

In order to test the effectiveness of the LSTM with attention mechanism at identifying important portions of a sequence, an extremely simple toy dataset was developed. This toy dataset consisted of a sequence of twenty -0.5 or 0.5 and an associated label indicating whether this particular sequence belong to class one or zero. The dataset was designed so

that only the fifth entry in the sequence was relevant to the label, and in fact received the label 0 if the value was -0.5 or 1 if the value of the fifth entry was 0.5. The other entries were all random with respect to the labels. There were 100000 training set and test set sequences randomly developed in accordance with the previously mentioned criteria.

3.3.3 Results on Toy Data Set

A bidirectional LSTM with attention was trained on the training set of 100000 instances with 16 units in the hidden layer. It was trained using adaptive momentum (ADAM) with a learning rate of $1.0e-4$. The topology consisted of a single LSTM layer with 16 units in the hidden layer and an attention layer with 4 units. The basic idea is the same as what was illustrated in figure 3.1. The inputs as described previously were either a single -0.5 or 0.5 for each element of the sequence and the sequences were 20 elements in length. The network was trained for 20 epochs, and at the conclusion of training the network was misclassifying only one test set instance out of 100000 and was classifying the training set instances perfectly.

Next, the attention weights were examined. It was expected that the majority of the attention would be focused on the fifth entry in the sequence as the value of this element was the only appropriate predictor of the label. Unfortunately, this is not what was observed. Just a brief glance at the weight values indicated that no special attention was being paid to the fifth entry, the overall attention paid to each entry seemed fairly uniform and did not yield any truly meaningful interpretation. Regardless of the explanation, this technique did not seem clearly helpful in identifying the single element of the sequence responsible for the label assigned.

One potential problem with the result obtained is that perhaps the toy data set is too simple and does not sufficiently challenge the neural network; to force it to attend in order to obtain good performance. Another potential issue is the location of the attention mechanism. We decided to place the attention mechanism immediately after the LSTM layer as

others in the literature have used this placement as well [67, 68]. One possibility that may be investigated in future work is to try to place the attention mechanism prior to the LSTM layer, though this would require careful thought to ensure the weights have the intended meaning. By attempting to place the attention here it would prevent forward and backward information flowing through the LSTM hidden layer from interfering with the attention determination. The topology following the LSTM hidden layer could be converted back to something like Chapter 1 as opposed to using the summing technique. It is likely, however, that placing the attention here would mean the weights would not behave as desired. For example if two identical inputs were given the attention weights would have no forward or backward information to distinguish which deserves more attention. Another possibility is to use a more sophisticated attention mechanism such as the decoder recurrent network similar to what was done in [69].

3.4 Discussion

A proof of principle using an LSTM to identify important amino acids has been shown. The method simply revolves around mutating each position in the amino acid sequence to each of the other potential amino acids to determine and to determine these mutations effect on the class label. If a particular amino acid in the original sequence plays an outsize role in the efficacy of the antibacterial peptide, then one will see a reduction in the average predicted class probability across the other amino acids. The next step may be to extend the mutation regions from point mutations to small contiguous regions of the peptide, say two or three consecutive amino acids. After checking the change in class probability due to either all or some subset of the possible mutations associated with multiple contiguous amino acids, one may obtain a better idea of the relative importance of regions of the antibacterial peptide that are responsible for its activity.

The second major approach applying the attention mechanism to the bidirectional LSTM as illustrated in figure 3.1 did not work as well as was hoped. There is the potential, how-

ever, to use a recurrent decoder network to give the attention weights more of an opportunity to zero in on this simple toy data set's only relevant feature for the label. Whether or not this helps improve the efficacy of the attention mechanism will be explored in future work. Also a more challenging toy data set may be sufficient to force the network to attend to the important regions with or without the decoder network change to the current topology. This concept provides another avenue forward and may also be explored in future work.

CHAPTER 4

IDENTIFYING AND GENERATING POTENTIAL ANTIBACTERIAL PEPTIDES

4.1 Motivation, Background, and Overview

In this chapter we describe the third major goal of the thesis: to generate or identify potentially new and interesting antibacterial peptides. Part of the chapter will describe efforts to generate random peptides from the instance space associated with the LSTM classifier discussed and developed in Chapter 2 and to identify potentially promising peptides using the classifier. The latter portion of the chapter will describe the attempt to build a generative model of the peptides using a fairly new neural network technique known as Generative Adversarial Networks (GANs). First, we begin with a section introducing the notion of a Generative Adversarial Network.

4.1.1 Background and Overview of GANs

The concept of a Generative Adversarial Network was developed in 2014 by Goodfellow et al. [51]. The essential premise is to train two models, a generative model which we will denote by G that is trained to capture the probability distribution associated with the data, and a discriminative model, D , that estimates the probability that a sample came from the training data rather than G .

In the work due to Goodfellow et al. [51] the idea of a two-player minimax game is used to describe the overall concept of a generative adversarial network. Although the work is described using the concept of two multilayer perceptrons to represent the generator G and discriminator D with their associated collections of parameters, most of the theoretical results presented would more accurately apply to function spaces for both the generator and the discriminator. The paper refers to this as the non-parametric limit of the generator

and discriminator given "enough capacity". D outputs the probability that a sample \mathbf{x} came from the true data rather than being a generated sample. G takes in a sample of input noise, represented by \mathbf{z} , and sampled from $p_{\mathbf{z}}(\mathbf{z})$ and maps it to the data space as the output of $G(\mathbf{z})$. As described, G and D play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))] \quad (4.1)$$

Goodfellow et al. [51] go on to prove some theoretical results involving GANs. They begin by stating that the generator G implicitly defines a probability distribution p_g of the samples $G(\mathbf{z})$ when $\mathbf{z} \sim p_{\mathbf{z}}$. The desire is to have p_g converge to p_{data} assuming the multi-layer perceptrons are given enough capacity. They then go on to state that the results of the theory section are done in a nonparametric setting where it is assumed that the model has infinite capacity and that the convergence is done in the space of probability density functions. This ignores the parametric character of the multilayer perceptron representations of G and D . This parametric character likely prevents D from ever obtaining optimality over the space of probability density functions. The authors readily state this, but hope multi-layer perceptrons are sufficiently rich so that this is still an excellent empirical technique despite the lack of theoretical guarantees.

The authors also bring up an important issue regarding the case D is not regularly re-trained as the training of G progresses. G could in theory learn to map different values of $\mathbf{z} \sim p_{\mathbf{z}}$ to the same value in the data space, x a training set element perhaps. If G collapses too many values of \mathbf{z} to too few values in the data space, the probability distribution associated with G , p_g , then fails to capture the true nature of p_{data} . This situation is often referred to as mode collapse.

Of particular interest to the work here, Goodfellow et al. [51] also mention at the end of the original paper introducing GANs that a conditional generative model $p(\mathbf{x}|\mathbf{c})$ can be

made by adding a conditional label or some other entity \mathbf{c} as input to both G and D . We move onto to discussing both these conditional GANs and improvements to the original GAN framework in what follows. Before that we provide the algorithm Goodfellow et al. gave in the paper introducing GANs.

Algorithm 2 Algorithm for Generative Adversarial Networks

Input: k the number of steps to apply to the discriminator

- 1: **for** number of training iterations **do**
- 2: **for** k steps **do**
- 3: Sample minibatch of m noise samples $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}$ from noise prior $p_g(\mathbf{z})$.
- 4: Sample minibatch of m samples $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ from data generating distribution $p_{data}(\mathbf{x})$.
- 5: Update the discriminator by ascending its stochastic gradient where ϕ are the parameters associated with the discriminator:

$$\nabla_{\phi} \frac{1}{m} \sum_{i=1}^m [\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)})))] \quad (4.2)$$

- 6: **end for**
- 7: Sample minibatch of m noise samples $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}$ from noise prior $p_g(\mathbf{z})$.
- 8: Update the generator by descending its stochastic gradient where θ are the generator parameters:

$$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(\mathbf{z}^{(i)}))) \quad (4.3)$$

- 9: **end for**
-

Conditional GANs

Generative adversarial networks can be extended to a conditional model if both the generator and the discriminator are conditioned on some additional information denoted by \mathbf{c} which can be a class label or data from other sources [70]. The conditioning is typically performed by feeding \mathbf{c} into both the discriminator and generator as an additional input. Mirza and Osindero illustrate the technique on the MNIST dataset where the image label was provided to both discriminator and generator as an encoded one-hot vector [70].

Wasserstein Metrics and Other GAN Techniques

Arjovsky et al. [71] introduce the concept of a Wasserstein GAN by applying the Earth-Mover distance which is also known as Wasserstein-1 distance in the task of training a GAN. First they compare this distance metric to other popular probability distances and divergences that are used in the context of learning distributions. Next, they develop the idea of a Wasserstein GAN that minimizes a reasonable and efficient approximation of the Earth-Mover distance. Finally, they show empirical evidence the Wasserstein GANs (WGANs) solve some of the difficult training aspects of the GAN framework. In particular, training WGANs does not require the careful balance of updating discriminator and generator. The authors also make the claim that the mode dropping phenomenon in GANs is reduced. They also state that a practical benefit of WGANs is the ability to "continuously estimate" the EM distance by training the discriminator to optimality.

First lets begin by describing the Earth-Mover (EM) distance which is also known as the Wasserstein-1 metric. Let \mathcal{X} be a compact metric set and let Σ denote the set of all Borel subsets of \mathcal{X} . Let $\text{Prob}(\mathcal{X})$ be the space of probability measures defined on \mathcal{X} . Let $\mathbb{P}_r, \mathbb{P}_g \in \text{Prob}(\mathcal{X})$. The equation provided in the work of Arjovsky et al. [71] is below:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma} [||x - y||], \quad (4.4)$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the collection of all joint distributions $\gamma(x, y)$ whose marginals are \mathbb{P}_r and \mathbb{P}_g respectively. The authors state the intuitive description of this distance is given by the how much "mass" must be transported from x to y in order to transform \mathbb{P}_r into \mathbb{P}_g , and that EM distance can be thought of as the cost of this transportation scheme.

Arjovsky et al. [71] compare the EM distance to three other metrics or divergences. The two we will focus on are the *Kullback-Leibler* (KL) divergence and the *Jensen-Shannon* (JS) divergence.

The KL divergence is defined as:

$$KL(\mathbb{P}_r, \mathbb{P}_g) = \int \log \left(\frac{p_r(x)}{p_g(x)} \right) p_r(x) d\mu(x), \quad (4.5)$$

where both \mathbb{P}_r and \mathbb{P}_g are assumed to be absolutely continuous with respect to μ and therefore admit the densities $p_r(x)$ and $p_g(x)$. The JS divergence is defined as:

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r || \mathbb{P}_m) + KL(\mathbb{P}_g || \mathbb{P}_m), \quad (4.6)$$

where \mathbb{P}_m is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$.

The authors make the case that simple sequences of probability distributions are capable of converging under the EM distance but will not converge under either KL or JS divergence. They state that KL and JS divergences are not good cost functions when learning distributions supported by low dimensional manifolds, but that the EM distance remains reasonable in such a situation.

Arjovsky et al. [71] then go on to design a GAN that would focus on optimizing the EM distance (4.4) rather than say the JS divergence. Unfortunately, the infimum in (4.4) is intractable. The authors then claim that due to Kantorovich-Rubinstein duality they can instead focus on optimizing

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)] \quad (4.7)$$

where the supremum is over all 1-Lipschitz functions $f : \mathcal{X} \rightarrow \mathbb{R}$. If a family of parameterized functions $f_{w \in \mathcal{W}}$ that are all K -Lipschitz for some K then solving

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z \sim p(z)}[f_w(g_\theta(z))] \quad (4.8)$$

and assuming the supremum in (4.7) is attained for a $w \in \mathcal{W}$, then we would obtain a

calculation of $W(\mathbb{P}_r, \mathbb{P}_\theta)$ up to a multiplicative constant. Also, differentiating $W(\mathbb{P}_r, \mathbb{P}_\theta)$ via estimating $\mathbb{E}_{z \sim p(z)}[\nabla_\theta f_w(g_\theta(z))]$ and backpropagating through (4.7) is possible.

The authors, Arjovsky et al. [71], claim that since the EM distance is continuous and differentiable a.e. means that they can and should train the critic or discriminator til optimality. The idea basically suggests that one does not have to be as careful to alternate between discriminator and generator updates as in the traditional GAN framework where one can run into poor gradients due to saturation of the second term in Equation 4.1. They also argue that due to training the critic til optimality mode collapse is far less likely.

In another paper, Gulrajani et al. [72] introduce a *gradient penalty* technique for training Wasserstein GANs that does not suffer the same problems as those associated with the weight-clipping technique that [71] proposes. Gulrajani et al. that the other paper has the right idea in that a discriminator in WGANs should lie within the space of 1-Lipschitz functions, but in essence disagree about how best to enforce this constraint. The authors modify a version of equation (4.7) into a full WGAN value function with notation changes as follows:

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r} [D(\mathbf{x})] - \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g} [D(\tilde{\mathbf{x}})] \quad (4.9)$$

where \mathcal{D} is the set of 1-Lipschitz function, \mathbb{P}_r is the data distribution, and \mathbb{P}_g is the model distribution implicitly defined by $\tilde{\mathbf{x}} = G(\mathbf{z}), \mathbf{z} \sim p(\mathbf{z})$. Under an optimal discriminator, minimizing the value function with respect to the generator parameters minimizes $W(\mathbb{P}_r, \mathbb{P}_g)$.

4.1.2 SeqGAN and Similar Techniques

SeqGAN Description

There is interest in moving beyond the fixed dimension vector as a generative target for GANs, and generalizing the GAN concept to other data structures such as sequences. An early technique for doing this was made known by Yu et al. [52]. In their work they draw

upon techniques from reinforcement learning, a technique they refer to as policy gradient, to allow a sequential GAN or SeqGAN as they termed it to be trained to generate sequences of discrete elements.

Specifically, the sequence generation problem that Yu et al. [52] discuss consists of a θ -parameterized generative model G_θ to produce a sequence $Y_{1:T} = (y_1, \dots, y_t, \dots, y_T)$, $y_t \in \mathcal{Y}$, where \mathcal{Y} is the vocabulary of candidate tokens. In addition, a ϕ -parameterized discriminative model D_ϕ is trained to assist the generator G_θ in improving its performance. $D_\phi(Y_{1:T})$ is a probability of how likely a sequence is from real sequence data or not. The discriminative model is trained by providing it with sequences from the generator which are treated as negative examples as well as real training data which are treated as positive examples. The generative model G_θ is trained by employing what the authors refer to as policy gradient and a Monte Carlo search on the basis of the expected end reward received from the discriminative model D_ϕ . The reward is based on the likelihood that a sequence from the generator would fool the discriminative model D_ϕ .

The crucial ingredient to the SeqGAN approach authors Yu et al. [52] developed is the Policy Gradient technique which deals with the case where there is no intermediate reward for a partially generated sequence. The objective for the generator model $G_\theta(y_t|Y_{1:t-1})$ is to generate a sequence from the start state s_0 to maximize the expected end reward:

$$J(\theta) = \mathbb{E}[R_T|s_0, \theta] = \sum_{y_1 \in \mathcal{Y}} G_\theta(y_1|s_0) \cdot Q_{D_\phi}^{G_\theta}(s_0, y_1),$$

where R_T is the expected end reward for a completed sequence.

In the case of the GAN architecture one can use the probability of the generated sequence being real according to the discriminator as the reward:

$$Q_{D_\phi}^{G_\theta}(a = y_T, s = Y_{1:T-1}) = D_\phi(Y_{1:T}). \quad (4.10)$$

The problem with this, however, is that the discriminator only provides a reward for a

completed sequence. A Monte Carlo approach was adopted that samples future tokens to add to the sequence to generate a completed sequence. This is denoted by a roll-out policy G_β which is used to sample the future tokens which is typically chosen by the authors to be the same as the generator. The notation for an N-time Monte Carlo search is

$$Y_{1:T}^1, \dots, Y_{1:T}^N = MC^{G_\beta}(Y_{1:t}; N),$$

and $Y_{1:t}^n = (y_1, \dots, y_t)$ and $Y_{t+1:T}^n$ are sampled based on the roll-out policy G_β which can typically be taken to be the generator.

To reduce the variance associated with the Monte Carlo sampling approach, the roll-out policy is applied N-times as already suggested to obtain a batch of potential completed sequences. The action-value function of a sequence, $Q_{D_\phi}^{G_\theta}(s, a)$, is ultimately denoted by:

$$Q_{D_\phi}^{G_\theta}(s = Y_{1:t-1}, a = y_t) = \begin{cases} \frac{1}{N} \sum_{n=1}^N D_\phi(Y_{1:T}^n), Y_{1:T}^n \in MC^{G_\beta}(Y_{1:t}; N) \text{ for } t < T \\ D_\phi(Y_{1:t}) \text{ for } t = T. \end{cases}$$

As sets of increasingly realistic generated sequences are developed by the generator, the discriminator can be retrained according to the following model:

$$\min_{\phi} -\mathbb{E}_{Y \sim P_{data}}[\log D_\phi(Y)] - \mathbb{E}_{Y \sim G_\theta}[\log(1 - D_\phi(Y))], \quad (4.11)$$

where P_{data} denotes an element drawn from the real training set sequences.

Following an update to the discriminator, it becomes time to update the generator [52]. The gradient of the objective function, $J(\theta)$, w.r.t the generator's parameters θ is derived as

$$\nabla_{\theta} J(\theta) = \sum_{t=1}^T \mathbb{E}_{Y_{1:t-1} \sim G_\theta} \left[\sum_{y_t \in \mathcal{Y}} \nabla_{\theta} G_\theta(y_t | Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t) \right]. \quad (4.12)$$

The authors [52] provide a sampling scheme for approximating equation (4.12). The sampling scheme when augmented with gradient ascent based on methods such as Adam

[73] provides the means to train the generator.

The algorithm for training sequence generative adversarial nets is provided below and is equivalent to that described in Yu et al. [52].

Algorithm 3 Algorithm for Sequence Generative Adversarial Nets

Input: generator policy G_θ ; roll-out policy G_β ; discriminator D_ϕ , a sequence dataset $\mathcal{S} = X_{1:T}$

- 1: Initialize G_θ, D_ϕ with random weights θ, ϕ
- 2: Pre-train G_θ using MLE on \mathcal{S}
- 3: $\beta \leftarrow \theta$
- 4: Generate negative samples using G_θ for training D_ϕ
- 5: Pre-train D_ϕ via minimizing the cross entropy
- 6: **repeat**
- 7: **for** g-steps **do**
- 8: Generate a sequence $Y_{1:T} = (y_1, \dots, y_T) \sim G_\theta$
- 9: **for** t in $1 : T$ **do**
- 10: Compute $Q(a = y_t; s = Y_{1:t-1})$ by (4.10)
- 11: **end for**
- 12: Update generator parameters via policy gradient
- 13: **end for**
- 14: **for** d-steps **do**
- 15: Use current G_θ to generate negative examples and combine with given positive examples \mathcal{S}
- 16: Train discriminator D_ϕ for k-epochs by (4.11)
- 17: **end for**
- 18: $\beta \leftarrow \theta$
- 19: **until** SeqGAN converges

Wasserstein Approach for Adversarial Generation of Natural Language

In addition to the approach due to the SeqGAN researchers, Rajeswar et al. [74] expand the concept of a Wasserstein GAN to be used for generating sequences of data. This method leverages use of the Wasserstein metric to allow the use of one-hot vectors to represent tokens from the real data, but uses probabilistic vectors output from the generator. Due to some special properties of the Wasserstein metric, this enables one to train the generator using backpropagation instead of the policy gradient technique that Yu et al. [52] developed.

4.2 Random Peptides Generation and the Bidirectional LSTM Classifier

Next, we shift from background information on GANs to describe a simpler approach for potentially identifying interesting peptides. This attempt simply uses randomly generated peptides and the classifier from Chapter 2 in order to identify potentially promising candidate peptides. It may be that the GAN approach will yield marginal returns over the randomly generated peptides approach and due to its simplicity we begin with the random peptides approach.

4.2.1 Simple Technique to Generate a Large Number of Random Peptides

A short and simple Biopython [55] script was developed to generate 1,000,000 randomly generated peptides over the twenty amino acids for which the features were described in chapter 2, and the peptides were required to be between 4-55 amino acids in length. This is done by sampling uniformly over the integers 4 through 55 and for the length returned, sampling at each position once again uniformly over the integers 1 through 20 which represent one of the amino acids under consideration. The 1,000,000 random peptides were stored in a large fasta file. Initially, we attempted to load all 1,000,000 peptides with their respective features into main memory but received an out of memory error. The machine contains only 32GB of memory, so the file was ultimately divided up into 50 files each containing 20,000 of the sequences. Although not an elegant approach as it is not likely as fast as could be done, this was the simplest way to have all peptides run through the the neural network described in Chapter 2. After obtaining the probabilities of peptides being antibacterial for each peptide in the 1,000,000 random collection of peptides, we stored these results in 20 different csv files. Each of these contains 2 columns, one with the sequence of interest and a second with the probability of the sequence being antibacterial.

Perhaps there are ways to improve the random generation scheme to generate more potential hits. Note that in the current scheme, we generate uniformly with respect to

sequence length. Perhaps we could design a sampling scheme that would favor lengths more in line with the distribution of lengths associated to the positive instances. This would be a simple scheme to increase the relative ratio of interesting positive hits to uninteresting cases.

4.2.2 Identifying Potentially Promising Peptides

The output probabilities of the peptides being antibacterial were analyzed. Out of the total 1,000,000 peptides 267,583 had probability greater than 0.5 assigned to them being antibacterial and were classified as such. This large number indicates that some probabilistic threshold is necessary to consider picking out the most likely candidates. Of the 1,000,000 peptides run through the LSTM network, 31 had probabilities that were so close to 1.0 as to be indistinguishable under the floating point precision used. In addition approximately 1,000 peptides, had a probability of being antibacterial greater than .999999 according to the classifier. A histogram indicating the distribution of peptides over their respective probability of being antibacterial is shown in figure 4.1.

Overconfidence in Neural Network Output Distributions

Note that most modern neural networks are typically known to return overconfident probabilities for classification tasks [75], and the histogram shows that this particular network is no exception. A large number of peptides are concentrated in the ranges $[0.0, .01]$ and $[.99, 1.0]$ showing the tendency for the classifier to push most of the classified random peptides towards a high confidence value even though such high confidence may be unwarranted. We will discuss a possible remedy to this problem in the next chapter.

The Classifier's Most Promising Peptides

In Table 4.1 and Table 4.2, the most likely 31 peptides according to the classifier are given.

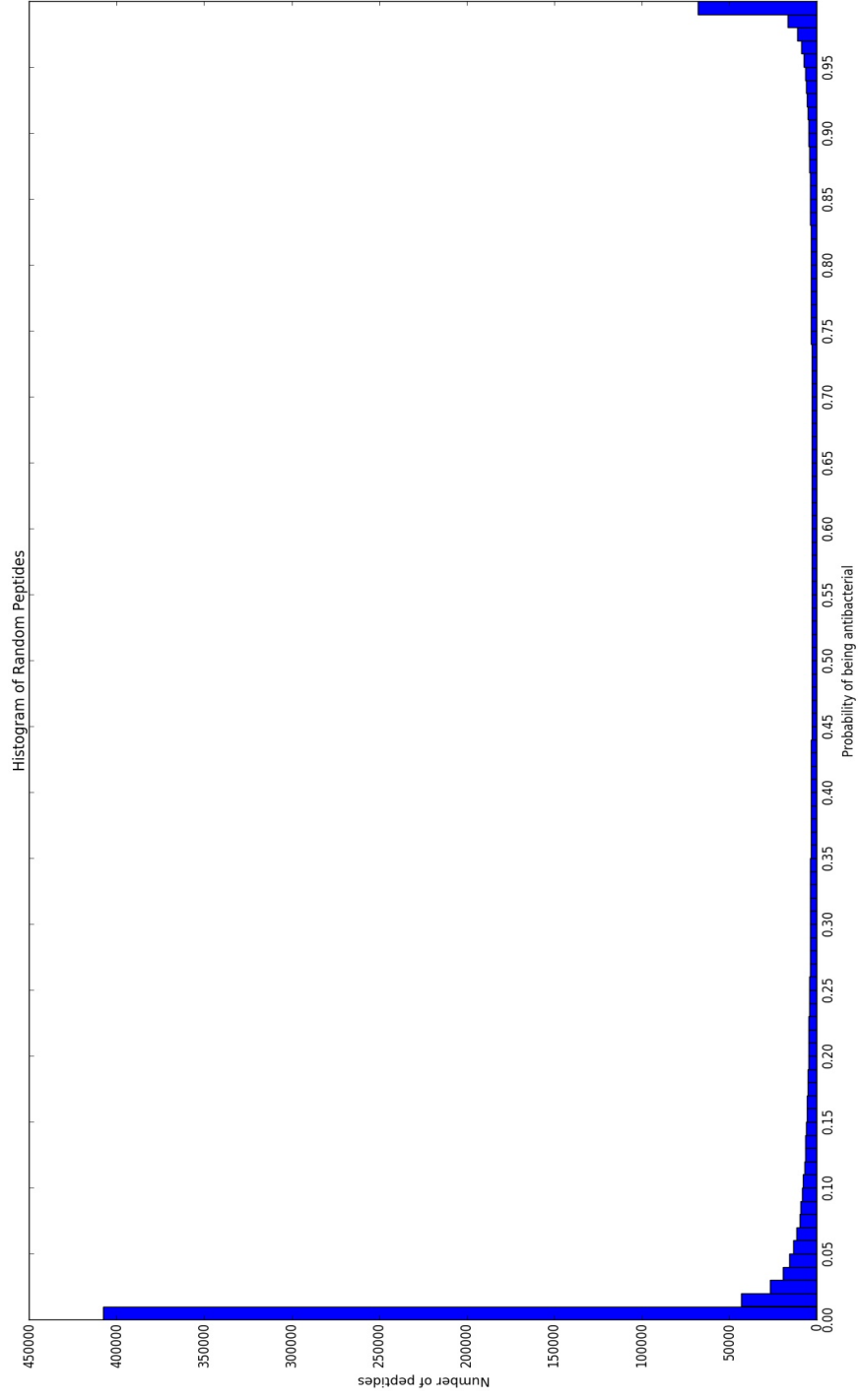


Figure 4.1: Histogram of probabilities of random peptides being antibacterial.

Table 4.1: Random Peptides Top 31 Part 1

Random Peptide Number	Peptide Sequence
6689	CIHFWGMNQILKKFKRPHHFM
31956	WWDWRRFFQMQRQMHSHFMTTYRKEWLQFW
48513	DHFSILTLLKSKRMLLEGWSLYGFQKK
59681	WVFFVSIKQVWVDKMFIRNAVAFCKSCF
74182	NWCALTWQLRNAHIKNCHYKKTCLVKF
77085	HYKWSVKLTDYKMCCKTKLRQGWPW
89951	WILGHRSHIRLKMVACWK
178131	RWYRHHGRLGPVKMIKIALVCKFH
218152	ETFLGGWLINFKILRWKTLQCSTIWRHWKNGGIGTKEVGTWTPSNISTRYWCQW
220219	WYGHMRKIMPKLNVGHFQHDFFCAF
234223	WNLRLRTQRMQMCKICVRVP
247408	WVGVLRRFTAVTHRKEFLAGCLK
285676	NWKHHWEWIMSHGYLKMELGKKRKQ
366452	GWLVTKFNTRLVVKIKPGK
387961	WCVWDLEETMKKKVIRGHCMF
396395	CFFAVWKIMHALKEHKMRPCKLILLMWCY
422081	NSFGFIWIRVEQFRWHKWGWGIVC
454526	DWLFQLTEEEKQLKRLQVIGLAFRYPVQKQTQK
489009	GWLCFCSMIQHV RDIVMRKKVKILAI
535223	FWLMWIIKWEKLLKHKMIQVRERYSK
551581	GRYTWIKQKHVKKMIVAQQQNKWTFTH

Table 4.2: Random Peptides Top 31 Part 2

Random Peptide Number	<i>Peptide Sequence</i>
608364	AWRRWIEIKAMICMNIKKHNM
614476	NDQYWWWHFLKFHKMKMNMGKFWRNTPFNKH
677933	WYPIIWATNFMMHKTKRMCFFLC
693100	DFWWMFFRWNKRRHHKYETLARLKKCLNSVNVLHLYYMW
736501	HFWYCKQKKKWWWWQQYAKIIRHPILCFATLL
778799	TLPDFWFKDLKVVFFKKKVGPIINKQRDW
815064	WLTFIVMNNFLTACKMRMTCCCKW
815282	HWWVLLGNNKMHKKINRWQ
846261	HWFFNVLRQVVNKSVMRISTQLQTYKHCT
994607	NLWKNLWKYRKVYGARMK

Water Solubility

One concern that has arisen in light of switching from the SeqGAN concept to the random peptides method is the need to address water solubility of our peptides. Due to the fact that both positive and negative instances in the dataset are more than likely to be water soluble [64] due to the sources of these instances, there is no reason for the classifier to learn to distinguish that positive instances must be water soluble while negative instances can be either.

4.2.3 Discussion of the Random Peptides Technique

Although the random peptides approach is quick to implement and easily identifies some interesting peptides according to the classifier, the water solubility issue discussed previously shows that one must take additional care with either the dataset or the classification method applied. This particular issue likely would not be very challenging to resolve as one could either use an additional classifier to assess the water solubility of the peptides, or one could add a number of water insoluble peptides to the negative instances and let the classifier learn to distinguish this characteristic as well.

Although much tougher to implement, the SeqGAN approach may allow one to build a generative model which can use physicochemical characteristics of the active peptides to generate similar properties in the peptides it generates after being trained. Although the SeqGAN would also benefit from adding water insoluble peptides to the negative instances, it may not be as critical to this generative model as it is to the classifier being evaluated on random peptides. This idea shows the relative advantages and disadvantages of the random peptide generation technique. While easy to implement one must be more careful with dataset design or add additional classifier characteristics to deal with such cases.

Multitask LSTM Networks

Another way forward for the random peptides technique, would be to train a multitask network on a set of peptides with several different characteristics of interest and labels or values for such characteristics. A multitask classifier could provide a probability for each characteristic one may be interested in evaluating for a given peptide. In fact a multitask LSTM network that takes into consideration both activity of the peptide as well as water solubility or any other characteristic one may want to consider such as eukaryotic cell toxicity or pH sensitivity would be a useful tool for finding good peptides for further analysis [64]. If one had sufficient data, the multitask network could be used to provide activity against multiple bacterial species and may help in identifying peptides with broad-spectrum activity. A network that evaluates the peptides in regards to several different criteria could be used to identify random peptides with good values across the different criteria.

4.3 Generative Adversarial Networks Progress

In our work to generate antibacterial peptides using GANs we decided to focus on the SeqGAN approach. This is partly due to its being the first available technique as well as being potentially better suited to a conditional GAN approach. This idea is supported by the work of Dai et al. [76] as they provide a conditional SeqGAN that is capable of generating natural language descriptions conditioned on images. In our work, we condition a generative LSTM network on a class label indicating whether an antibacterial or nonantibacterial peptide is to be generated. This allows us to expand the scope of the dataset and provide a more precise distribution for describing each group of peptides than we would obtain if we tried to describe either group alone. For the discriminator we also employ an LSTM approach similar to what was done in Chapter 2. Specifically, a bidirectional LSTM is trained to attempt to distinguish between the instances generated by the generator and those in the training set.

In this work, the attempt to use the original SeqGAN architecture to generate examples of antibacterial peptides is complicated by the need to train and generate in batches and the variable lengths of the peptides being used for training or those being generated. In order to make this feasible, special discrete tokens, stop tokens, were appended to the training set elements as well as added to the pool of potential amino acids during generation. In the generative setting when a stop token is first added to a growing sequences, all tokens created afterwards are automatically cast to stop tokens as well. This allows the idea of termination of the sequence and also allows one to deal with the variable sequence lengths within a generated batch. Although simple in principle, implementing the idea in TensorFlow [60] is complicated by the need to use the low level TensorFlow API elements such as a special while loop with control flow logic within it to control the addition of stop tokens as well as the behaviour of the batch as it grows when generating. The original SeqGAN architecture did not have to account for this due to fact that all sequences both trained and generated were of equivalent lengths. In fact nearly all the major issues encountered when trying to develop a SeqGAN like implementation for the antibacterial peptides revolve around the need to deal with the variable length of peptide sequences within each batch. In fact, the most important change that needs to be made to the current attempt at the variable length SeqGAN is to fix the reward calculation that is used to reward the generator for generating a sequence ultimately similar to one from the training set. The reward calculation must be altered to deal with sequences of varying lengths within the collection of Monte-Carlo roll-out sequences.

Generative Adversarial Network and the Shuffling Problem

The SeqGAN approach offers the promise of generating a wide variety of peptides that are already likely to be antibacterial. As discussed in Chapter 2, however, the fact that a shuffled variant of a peptide is still very likely to be antibacterial according to both the classifier and its MIC value, suggests that a SeqGAN trained on the current main dataset

may simply learn to shuffle the peptides in the training set. It may also generate more sophisticated peptides of interest as well, but this behaviour could likely be emulated adding some randomly generated peptides into a set of peptides to evaluate so long as the classifier is trained to evaluate on all characteristics of interest. As will be discussed in more detail in the discussion involving the random peptides, there are both advantages and disadvantages to using the SeqGAN over the random peptides approach.

CHAPTER 5

THESIS CONTRIBUTIONS AND FUTURE WORK

5.1 Thesis Contributions

This thesis set out to apply deep learning based architectures to the identification and generation of potential antibacterial peptides. As deep learning is a relatively recent development, some of the techniques had not yet been applied to the task of identifying antibacterial peptides. Most of the techniques experimented with here will likely be taken up in the future as methods for machine learning researchers to consider when working with these peptides.

In Chapter 2, we began by exploring the use of Long Short-Term Memory Recurrent Network architectures to classify peptides that were known to be antibacterial versus a group of peptides that did not indicate any such activity. We saw that despite the simple features used by the LSTM it was able to extract a feature vector which yielded a slightly stronger test set result versus a random forest classifier that used approximately 44,000 features to obtain a nearly equivalent result.

In Chapter 3, we worked to identify important subregions of antibacterial peptides. First we showed examples of two peptides that when computing all possible point mutations of their original amino acids allowed one to see the suggestion of an amino acid with heightened importance. This technique could easily be expanded to include longer subregions, maybe 2 or 3 amino acids, which would probably lead to clearer evidence of differences in subregions' influence over the peptides overall activity.

Second, we developed and attempted to demonstrate the effectiveness of an LSTM network with an attention mechanism on a toy dataset to serve as an indicator that the attention mechanism did in fact serve our purposes. Despite the fact that the attention

network did not attend well on this simple toy data set, there is potential that if the dataset was increased in difficulty we would observe a clearer attention distribution. This is largely due to the fact that on such a trivial dataset, the network does not really need to attend to attain near perfect performance. Further, there is also the potential to build a more sophisticated version of attention onto the LSTM architecture that may do a better job attending on both the toy data set and a more sophisticated dataset.

In chapter 4, we first focused on random peptide generation schemes and running these peptides through a neural network such as the one trained in Chapter 2. Although somewhat overconfident and perhaps not dealing with some issues such as water solubility well due to the original dataset also not addressing this, we were able to identify peptides that could be of further interest. Second, we attempted to build a sophisticated relatively new neural network architecture known as a SeqGAN that likely still holds promise in this area. The code for the SeqGAN architecture for our variable length sequences within each batch is currently close but still not quite complete.

5.2 Alternative Attention Mechanisms

As just discussed, in Chapter 3 we worked to develop a method for identifying interesting subregions of antibacterial peptides, but deferred it for later when we realized without ground truth there was no way to verify the attention mechanism was working properly. This is especially true in light of the failure of the attention mechanism on a perhaps a too simple toy dataset. Going forwards, however, one could certainly first attempt to make a more sophisticated toy dataset that may force the network to have to attend in order to obtain reasonable performance, but this still may not guarantee the attention mechanism is working properly on our peptides. Another perhaps more interesting avenue moving forwards is to develop the alternate version of attention briefly mentioned in Chapter 3.

5.3 Making the Classifier Less Overconfident when Classifying Antibacterial Peptides

It may be possible to correct the confidence of the classifier by implementing confidence calibration techniques as described in Guo et al. [75]. These authors introduce the concept of calibration of neural networks in the standard multiclass supervised learning problem. Given input $X \in \mathcal{X}$ and label $Y \in \mathcal{Y} = \{1, \dots, K\}$ as random variables having ground truth joint distribution $\pi(X, Y) = \pi(Y|X)\pi(X)$, let $h(X) = (\hat{Y}, \hat{P})$ where \hat{Y} is a class prediction and \hat{P} the associated confidence or probability of correctness. The goal is to have \hat{P} be well calibrated so that it represents a true probability. In this setting perfect calibration is defined as follows:

$$\mathbb{P}(\hat{Y} = Y | \hat{P} = p) = p, \forall p \in [0, 1] \quad (5.1)$$

where the probability is over the joint distribution.

Next the authors develop empirical approximations to (5.1) with the primary focus being on Expected Calibration Error (ECE). Sparing the details which can be found in the paper [75], the authors explore various regularization techniques, Platt scaling, and temperature scaling as possible calibration remedies. Temperature scaling seems to be the authors' favorite method and is something that we would like to explore in the future to improve the quality of the LSTM predictions.

5.4 Finishing the SeqGAN

As discussed in Chapter 4, the SeqGAN may offer some elegant advantages over the method of generating random peptides. The biggest issue with the current attempt at implementing the SeqGAN is the reward function for the rolled-out sequences. Due to the variable lengths within each roll-out batch it will be necessary to modify this reward function to take into account the presence of differing numbers of stop-tokens on different ele-

ments composing the batch. The differing number of stop-tokens is due to the potential for different sequences to be shorter than others. The stop-tokens serve as a form of padding during batch-wise generation. Once the reward function is finished, one may more evaluate the performance of the SeqGAN and see if it is working.

REFERENCES

- [1] R. Laxminarayan, A. Duse, C. Wattal, A. K. M. Zaidi, H. F. L. Wertheim, N. Sumpradit, E. Vlieghe, G. L. Hara, I. M. Gould, H. Goossens, C. Greko, A. D. So, M. Bigdeli, G. Tomson, W. Woodhouse, E. Ombaka, A. Q. Peralta, F. N. Qamar, F. Mir, S. Kariuki, Z. A. Bhutta, A. Coates, R. Bergstrom, G. D. Wright, E. D. Brown, and O. Cars, “Antibiotic resistance: the need for global solutions,” *The Lancet Infectious Diseases*, vol. 13, no. 12, pp. 1057–1098, 2013.
- [2] W. C. Wimley and K. Hristova, “Antimicrobial peptides: Successes, challenges and unanswered questions,” *The Journal of Membrane Biology*, vol. 239, no. 1, pp. 27–34, Jan. 2011.
- [3] M. Pasupuleti, A. Schmidtchen, and M. Malmsten, “Antimicrobial peptides: Key components of the innate immune system,” *Critical Reviews in Biotechnology*, vol. 32, no. 2, pp. 143–171, 2012.
- [4] A. J. Alanis, “Resistance to antibiotics: Are we in the post-antibiotic era?” *Archives of Medical Research*, vol. 36, no. 6, pp. 697–705, 2005, Infectious Diseases: Revisiting Past Problems and Addressing Future Challenges.
- [5] B. Deslouches, J. D. Steckbeck, J. K. Craig, Y. Doi, J. L. Burns, and R. C. Montelaro, “Engineered cationic antimicrobial peptides to overcome multidrug resistance by escape pathogens,” *Antimicrobial Agents and Chemotherapy*, vol. 59, no. 2, pp. 1329–1333, 2015.
- [6] F. C. Fernandes, D. J. Rigden, and O. L. Franco, “Prediction of antimicrobial peptides based on the adaptive neuro-fuzzy inference system application,” *Peptide Science*, vol. 98, no. 4, pp. 280–287, 2012.
- [7] A. T. Y. Yeung, S. L. Gellatly, and R. E. W. Hancock, “Multifunctional cationic host defence peptides and their clinical applications,” *Cellular and Molecular Life Sciences*, vol. 68, no. 13, p. 2161, May 2011.
- [8] M. Zasloff, “Antimicrobial peptides of multicellular organisms,” *Nature*, vol. 415, p. 389, 2002.
- [9] A. M. Czyzewski, H. Jenssen, C. D. Fjell, M. Waldbrook, N. P. Chongsiriwatana, E. Yuen, R. E. W. Hancock, and A. E. Barron, “In vivo, in vitro, and in silico characterization of peptoids as antimicrobial agents,” *PLOS ONE*, vol. 11, no. 2, pp. 1–17, Feb. 2016.

- [10] G. Liang, Y. Liu, B. Shi, J. Zhao, and J. Zheng, “An index for characterization of natural and non-natural amino acids for peptidomimetics,” *PLOS ONE*, vol. 8, no. 7, pp. 1–16, Jul. 2013.
- [11] C. D. Fjell, J. A. Hiss, R. E. W. Hancock, and G. Schneider, “Designing antimicrobial peptides: Form follows function,” *Nature Reviews Drug Discovery*, vol. 11, p. 37, 2011.
- [12] G. Marti, A. Kereszt, va Kondorosi, and P. Mergaert, “Natural roles of antimicrobial peptides in microbes, plants and animals,” *Research in Microbiology*, vol. 162, no. 4, pp. 363–374, 2011.
- [13] M. Pirtskhalava, A. Gabrielian, P. Cruz, H. L. Griggs, R. B. Squires, D. E. Hurt, M. Grigolava, M. Chubinidze, G. Gogoladze, B. Vishnepolsky, V. Alekseev, A. Rosenthal, and M. Tartakovsky, “Dbaasp v.2: An enhanced database of structure and antimicrobial/cytotoxic activity of natural and synthetic peptides,” *Nucleic Acids Research*, vol. 44, no. D1, pp. D1104–D1112, 2016.
- [14] G. Gogoladze, M. Grigolava, B. Vishnepolsky, M. Chubinidze, P. Duroux, M.-P. Lefranc, and M. Pirtskhalava, “Dbaasp: Database of antimicrobial activity and structure of peptides,” *FEMS Microbiology Letters*, vol. 357, no. 1, pp. 63–68, 2014.
- [15] S. Thomas, S. Karnik, R. S. Barai, V. K. Jayaraman, and S. Idicula-Thomas, “Camp: A useful resource for research on antimicrobial peptides,” *Nucleic Acids Research*, vol. 38, no. suppl₁, pp. D774–D780, 2010.
- [16] F. H. Wagh, R. S. Barai, P. Gurung, and S. Idicula-Thomas, “Camp3: A database on sequences, structures and signatures of antimicrobial peptides,” *Nucleic Acids Research*, vol. 44, no. D1, pp. D1094–D1097, 2016.
- [17] F. H. Wagh, L. Gopi, R. S. Barai, P. Ramteke, B. Nizami, and S. Idicula-Thomas, “Camp: Collection of sequences and structures of antimicrobial peptides,” *Nucleic Acids Research*, vol. 42, no. D1, pp. D1154–D1158, 2014.
- [18] L. Fan, J. Sun, M. Zhou, J. Zhou, X. Lao, H. Zheng, and H. Xu, “Dramp: A comprehensive data repository of antimicrobial peptides,” *Scientific Reports*, vol. 6, p. 24482, 2016.
- [19] A. Cherkasov and B. Jankovic, “Application of inductive qsar descriptors for quantification of antibacterial activity of cationic polypeptides,” *Molecules*, vol. 9, no. 12, pp. 1034–1052, 2004.
- [20] F. Lira, P. S. Perez, J. A. Baranauskas, and S. R. Nozawa, “Prediction of antimicrobial activity of synthetic peptides by a decision tree model,” *Applied and Environmental Microbiology*, vol. 79, no. 10, pp. 3156–3159, 2013.

- [21] G. Maccari, M. Di Luca, R. Nifos, F. Cardarelli, G. Signore, C. Boccardi, and A. Bifone, “Antimicrobial peptides design by evolutionary multiobjective optimization,” *PLOS Computational Biology*, vol. 9, no. 9, pp. 1–12, Sep. 2013.
- [22] Z. Wang and G. Wang, “Apd: The antimicrobial peptide database,” *Nucleic Acids Research*, vol. 32, no. suppl₁, pp. D590–D592, 2004.
- [23] G. Wang, X. Li, and Z. Wang, “Apd2: The updated antimicrobial peptide database and its application in peptide design,” *Nucleic Acids Research*, vol. 37, no. suppl₁, pp. D933–D937, 2009.
- [24] S. Lata, B. Sharma, and G. Raghava, “Analysis and prediction of antibacterial peptides,” *BMC Bioinformatics*, vol. 8, no. 1, p. 263, Jul. 2007.
- [25] C. D. Fjell, R. E. Hancock, and A. Cherkasov, “Amper: A database and an automated discovery tool for antimicrobial peptides,” *Bioinformatics*, vol. 23, no. 9, pp. 1148–1155, 2007.
- [26] S. Lata, N. K. Mishra, and G. P. Raghava, “Antibp2: Improved version of antibacterial peptide prediction,” *BMC Bioinformatics*, vol. 11, no. 1, S19, Jan. 2010.
- [27] W. F. Porto, F. C. Fernandes, and O. L. Franco, “An svm model based on physicochemical properties to predict antimicrobial activity from protein sequences with cysteine knot motifs,” in *Advances in Bioinformatics and Computational Biology*, C. E. Ferreira, S. Miyano, and P. F. Stadler, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 59–62, ISBN: 978-3-642-15060-9.
- [28] M. Torrent, D. Andreu, V. M. Nogus, and E. Boix, “Connecting peptide physicochemical and antimicrobial properties by a rational prediction model,” *PLOS ONE*, vol. 6, no. 2, pp. 1–8, Feb. 2011.
- [29] T. U. Consortium, “Uniprot: A hub for protein information,” *Nucleic Acids Research*, vol. 43, no. D1, pp. D204–D212, 2015.
- [30] P. Wang, L. Hu, G. Liu, N. Jiang, X. Chen, J. Xu, W. Zheng, L. Li, M. Tan, Z. Chen, H. Song, Y.-D. Cai, and K.-C. Chou, “Prediction of antimicrobial peptides based on sequence alignment and feature selection methods,” *PLOS ONE*, vol. 6, no. 4, pp. 1–9, Apr. 2011.
- [31] L. Kll, A. Krogh, and E. L. Sonnhammer, “A combined transmembrane topology and signal peptide prediction method,” *Journal of Molecular Biology*, vol. 338, no. 5, pp. 1027–1036, 2004.
- [32] S. Joseph, S. Karnik, P. Nilawe, V. K. Jayaraman, and S. Idicula-Thomas, “Clas-samp: A prediction tool for classification of antimicrobial peptides,” *IEEE/ACM*

Transactions on Computational Biology and Bioinformatics, vol. 9, no. 5, pp. 1535–1538, Sep. 2012.

- [33] W. F. Porto, S. Pires, and O. L. Franco, “Cs-ampred: An updated svm model for antimicrobial activity prediction in cysteine-stabilized peptides,” *PLOS ONE*, vol. 7, no. 12, pp. 1–7, Dec. 2012.
- [34] X. Y. Ng, B. A. Rosdi, and S. Shahrudin, “Prediction of antimicrobial peptides based on sequence alignment and support vector machine-pairwise algorithm utilizing lz-complexity,” *BioMed Research International*, vol. 2015, p. 13, 2015.
- [35] F. Camacho, R. Torres, and R. Ramos-Pollán, “Feature learning using stacked autoencoders to predict the activity of antimicrobial peptides,” in *Computational Methods in Systems Biology*, O. Roux and J. Bourdon, Eds., Cham: Springer International Publishing, 2015, pp. 121–132, ISBN: 978-3-319-23401-4.
- [36] S. Gigure, F. Laviolette, M. Marchand, D. Tremblay, S. Moineau, X. Liang, Biron, and J. Corbeil, “Machine learning assisted design of highly active peptides for drug discovery,” *PLOS Computational Biology*, vol. 11, no. 4, pp. 1–21, Apr. 2015.
- [37] K. Y. Chang, T.-p. Lin, L.-Y. Shih, and C.-K. Wang, “Analysis and prediction of the critical regions of antimicrobial peptides based on conditional random fields,” *PLOS ONE*, vol. 10, no. 3, pp. 1–16, Mar. 2015.
- [38] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.
- [39] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006, PMID: 16764513.
- [40] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015.
- [41] A. Graves *et al.*, *Supervised sequence labelling with recurrent neural networks*. Springer, 2012, vol. 385.
- [42] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.
- [43] M. Schuster, “On supervised learning from sequential data with applications for speech recognition,” *Daktaro disertacija, Nara Institute of Science and Technology*, vol. 45, 1999.
- [44] P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri, “Exploiting the past and the future in protein secondary structure prediction,” *Bioinformatics*, vol. 15, no. 11, pp. 937–946, 1999.

- [45] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [46] K. Greff, R. K. Srivastava, J. Koutnk, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–11, 2017.
- [47] J. Martens and I. Sutskever, “Learning recurrent neural networks with hessian-free optimization,” in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, Citeseer, 2011, pp. 1033–1040.
- [48] F. Wang and D. M. J. Tax, “Survey on the attention based RNN model and its applications in computer vision,” *CoRR*, vol. abs/1601.06823, 2016.
- [49] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2013, pp. 3111–3119.
- [50] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *CoRR*, vol. abs/1301.3781, 2013. arXiv: 1301.3781.
- [51] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.
- [52] L. Yu, W. Zhang, J. Wang, and Y. Yu, “Seqgan: Sequence generative adversarial nets with policy gradient,” in *AAAI*, 2017, pp. 2852–2858.
- [53] E. Boutet, D. Lieberherr, M. Tognolli, M. Schneider, P. Bansal, A. J. Bridge, S. Poux, L. Bougueleret, and I. Xenarios, “UniProtKB/Swiss-Prot, the Manually Annotated Section of the UniProt KnowledgeBase: How to Use the Entry View,” *Methods Mol. Biol.*, vol. 1374, pp. 23–54, 2016.
- [54] C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T. L. Madden, “BLAST+: architecture and applications,” *BMC Bioinformatics*, vol. 10, p. 421, Dec. 2009.
- [55] P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and M. J. L. de Hoon, “Biopython: Freely available python tools for computational molecular biology and bioinformatics,” *Bioinformatics*, vol. 25, no. 11, pp. 1422–1423, 2009.

- [56] Y. B. Ruiz-Blanco, W. Paz, J. Green, and Y. Marrero-Ponce, “ProtDcal: A program to compute general-purpose-numerical descriptors for sequences and 3d-structures of proteins,” *BMC Bioinformatics*, vol. 16, no. 1, p. 162, May 2015.
- [57] M. Dayhoff, R. Schwartz, and B. Orcutt, “22 a model of evolutionary change in proteins,” in *Atlas of protein sequence and structure*, vol. 5, National Biomedical Research Foundation Silver Spring, MD, 1978, pp. 345–352.
- [58] S. Henikoff and J. G. Henikoff, “Amino acid substitution matrices from protein blocks,” *Proc. Natl. Acad. Sci. U.S.A.*, vol. 89, no. 22, pp. 10 915–10 919, Nov. 1992.
- [59] S. Wang, J. Peng, J. Ma, and J. Xu, “Protein secondary structure prediction using deep convolutional neural fields,” *Scientific Reports*, vol. 6, 18962 EP –, Jan. 2016, Article.
- [60] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *CoRR*, vol. abs/1603.04467, 2016.
- [61] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [62] J. Xiong, *Essential Bioinformatics*. Cambridge University Press, 2006.
- [63] T. Thireou and M. Reczko, “Bidirectional long short-term memory networks for predicting the subcellular localization of eukaryotic proteins,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, no. 3, pp. 441–446, Jul. 2007.
- [64] C. G. Starr and W. C. Wimley, “Discovery of novel antimicrobial peptides using combinatorial chemistry and high-throughput screening,” in *Antimicrobial Peptides Discovery, Design, and Novel Therapeutic Strategies*, G. Wang, Ed., CAB International, 2017, pp. 86–100.
- [65] K. Hilpert, M. R. Elliott, R. Volkmer-Engert, P. Henklein, O. Donini, Q. Zhou, D. F. Winkler, and R. E. Hancock, “Sequence requirements and an optimization strategy for short antimicrobial peptides,” *Chem Biol*, vol. 13, no. 10, pp. 1101–7, 2006.

- [66] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *CoRR*, vol. abs/1409.0473, 2014. arXiv: 1409.0473.
- [67] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, “Hierarchical attention networks for document classification,” in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 1480–1489.
- [68] S. K. Sønderby, C. K. Sønderby, H. Nielsen, and O. Winther, “Convolutional lstm networks for subcellular localization of proteins,” in *Algorithms for Computational Biology*, A.-H. Dediu, F. Hernández-Quiroz, C. Martín-Vide, and D. A. Rosenblueth, Eds., Cham: Springer International Publishing, 2015, pp. 68–80, ISBN: 978-3-319-21233-3.
- [69] J. J. Almagro Armenteros, C. K. Snderby, S. K. Snderby, H. Nielsen, and O. Winther, “Deeploc: Prediction of protein subcellular localization using deep learning,” *Bioinformatics*, vol. 33, no. 21, pp. 3387–3395, 2017.
- [70] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.
- [71] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, International Convention Centre, Sydney, Australia: PMLR, Aug. 2017, pp. 214–223.
- [72] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of wasserstein gans,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 5767–5777.
- [73] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014. arXiv: 1412.6980.
- [74] S. Rajeswar, S. Subramanian, F. Dutil, C. Pal, and A. Courville, “Adversarial generation of natural language,” *arXiv preprint arXiv:1705.10929*, 2017.
- [75] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” *CoRR*, vol. abs/1706.04599, 2017. arXiv: 1706.04599.
- [76] B. Dai, S. Fidler, R. Urtasun, and D. Lin, “Towards diverse and natural image descriptions via a conditional gan,” *arXiv preprint arXiv:1703.06029*, 2017.