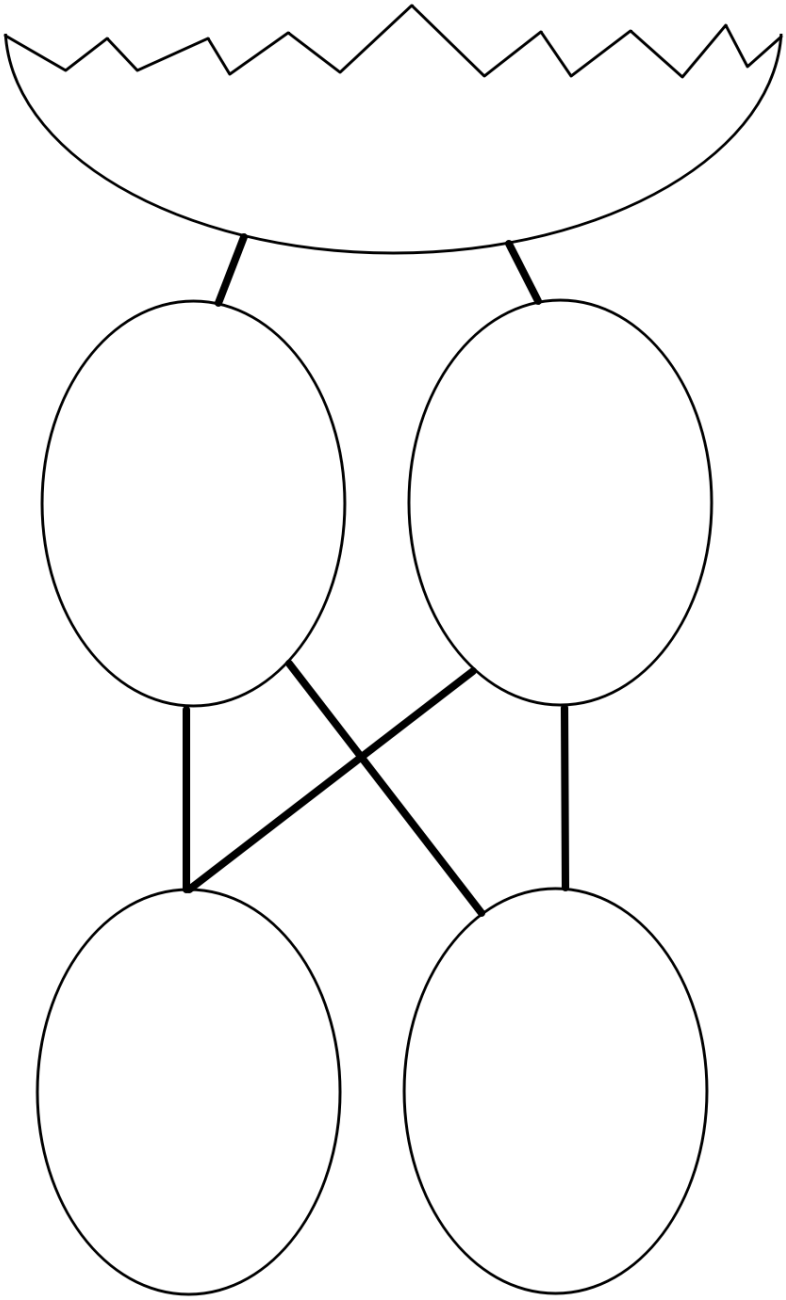
# Natural Language Processing Techniques for Identifying Bacteriocins

Lasse Buur Rasmussen

Supervised by Asker Daniel Brejnrod and Mani Arumugam

August 31, 2019



# Abstracts

*English*
Antibiotic-resistant bacteria is an increasing issue and broad spectrum antibiotics can cause long-lasting alterations of the gut microbiome. New, more narrow-spectrum antibiotics are needed as they might be a potential solution to counteract these issues. Bacteriocins, a potential solution, are small antimicrobial peptides ribosomally synthesized by bacteria. Recent results published by Hamid et al. showed neural models using Word2Vec embedding with high enough performance to challenge traditional bioinformatic classification methods. ELMo is a newer word embedding method which has some advantages over Word2Vec. The main problem of Word2Vec is the underlying assumption that every word of an alphabet only has a single context. ELMo deals with this by having different representations of each amino acid depending on its surrounding amino acids. We trained a classifier on data from bacteriocin-containing databases BAGEL, CAMP and Bactibase. We saw a significant increase in accuracy when using ELMo embedding. The freely available model by Hamid et al. obtained a test accuracy of 86.0%. The classifier model we present here has a test set accuracy of 94.8%. We applied our model on the Sberro et al.'s recently published small protein families and found 40 putative bacteriocins. We have showed that updating the embedding method to ELMo can increase the performance of these types of neural nets. It is our hope to see state of the art word embedding models trained on the vast amounts of sequence data available. This will help to further push the performance of sequence classifiers as well as the bioinformatics field.

*Danish*
Antibiotikaresistente bakterier er et stigende problem, og bredspektret antibiotika kan forårsage langvarige ændringer af tarmmikrobiomet. Nye mere smalspektrede antibiotika er nødvendige, da de kan være en potentiel løsning til at modvirke disse problemer. Bakteriociner, en potentiel løsning, er små antimikrobielle peptider syntetiseret ribosomalt af bakterier. Nye resultater offentliggjort af Hamid et al. viste neurale modeller som ved hjælp af Word2Vec "embedding" har god nok præstation til at udfordre traditionelle bioinformatiske klassificeringsmetoder. ELMo er en nyere metode til "embedding" af ord, der har nogle fordele i forhold til Word2Vec. Det største problem med Word2Vec er den underliggende antagelse om, at hvert ord i et alfabet kun har et enkelt kontekst. ELMo håndterer dette ved at have forskellige repræsentationer af hver aminosyre afhængigt af dens omgivende aminosyrer. Vi implementerede en klassifikator på data fra bakteriocin-indeholdende databaser BAGEL, CAMP og Bactibase. Vi så en markant stigning i nøjagtighed, når vi brugte ELMo-"embedding". Den frit tilgængelige model af Hamid et al. opnåede en testnøjagtighed på 86,0 %. Klassificeringsmodellen, vi præsenterer her, har en testsætsnøjagtighed på 94,8 %. Vi anvendte vores model på Sberro et al.s for nyligt offentliggjorte små proteinfamilier og fandt 40 formodede bakteriociner. Vi har vist, at opdatering af "embeddingmetoden" til ELMo kan øge ydelsen af disse typer neurale net. Det er vores håb at se state of the art "wordembedding"-modeller, der er trænet på de store mængder af tilgængelige sekvensdata. Dette vil hjælpe med til yderligere at forbedre sekvensklassifikatorers ydelse såvel som bioinformatikfeltet.

# Introduction

*Bacteriocins*

Antibiotic-resistant bacteria is an increasing issue and new antibiotics are needed[1]. What is more, the recent development of the human microbiomics field has shown that broad spectrum antibiotics can cause long-lasting alterations in the microbiome[2]. More narrow-spectrum, targeted antibiotics might be a potential solution to counteract some of the microbiome side-effects of treatment.

Bacteriocins have properties that make them very attractive candidates as new antibiotics. These are small antimicrobial peptides ribosomally synthesized by bacteria. Their precursor genes are very widespread in the genomes of the bacterial domain of life and comes in an array of different types. Typically they are surrounded by other genes important for post-translational processing and secretion. Among these are also immunity genes that confer resistance to the bacteriocin, preventing the bacteriocin-producer from harming itself. The classification of bacteriocins is continuously updated but as of this writing bacteriocins of gram positive and gram negative bacteria are divided into 12 groups and 2 categories, respectively[3].

Bacteriocins are synthesized for the purpose of fighting off other bacteria in the competition for nutrients and territory. As similar bacteria often fight for the same resources, many bacteriocins are antimicrobial towards bacteria of the same species as the producer. However, bacteriocins that act as antimicrobials towards bacteria of different genera also exist[3].

*Purpose*

Traditionally, classic bioinformatics approaches such as BLAST have been taken to identify novel bacteriocins. However, bacteriocins have been shown to have very low sequence similarity, making it hard to rely on specific sequence signatures or motifs for their identification[3].

This century's deep learning revolution has demonstrated the great potential of neural networks in other fields such as natural language processing (NLP). Similarly deep learning has started entering the bioinformatics field in recent years. The topics in bioinformatics touched by deep learning include sequence analysis, biomedical image classification and structure prediction[4]. The advantage of these approaches seem to come from the "artificial intuition" these algorithms can obtain without much need for an explanation of the problem at hand other than labels discriminating "right" from "wrong". In addition they have the benefits of rapid execution and implementation due to today's accessibility of cloud computing and software packages such as Google Colaboratory and Keras, respectively.

In this paper I describe how we approached the topic of bacteriocin discovery using some of these deep learning approaches. Our purpose was to find good amino acid representations and use them to discover bacteriocins. I show how adapting some of the approaches from NLP and image recognition such as transfer learning can help extract increased information from amino acid sequences. This strengthens bioinformatics sequence analysis while only relying on the biological sequence itself.

| Encoding | Equation | Dimensions |
|---|---|---|
| Atchley clust. | $100C$ | 100D |
| Atchley | $97KM \cdot 15AF$ | 1455D |
| One-hot | $99AA \cdot 22LE$ | 2178D |
| Reduced alphabet | $99AA \cdot 11LE$ | 1089D |
| Word2Vec clust. | $100C$ | 100D |
| Word2Vec | $97KM \cdot 200W2V$ | 19400D |
| ELMo summed | $99AA \cdot 1024ELS$ | 101376D |
| ELMo | $99AA \cdot 3072EL$ | 304128D |

Table 1: Dimensionality of the different encodings with a sequence set of maximum length 99. Legend: $C$ = cluster, $D$ = dimensions, $KM$ = k-mer, $AF$ = Atchley factors, $AA$ = Amino acids, $LE$ = Letters, $W2V$ = Word2Vec dimensions, $ELS$ = Summed ELMo dimensions, $EL$ = ELMo dimensions.

# Part 1

## Searching for good encodings

Most machine learning software requires numeric input. So to begin this analysis the first step was to identify a good way of presenting the amino acid letters of the sequences to the algorithm, referred to as sequence encoding (Table 1).

A pitfall of performing matrix operations (which is the bread and butter of machine learning) on encoded biological sequences is their varying lengths. Matrices are rectangular, meaning they have an equal amount of rows for each column and an equal amount of columns for each row. The most frequent way of dealing with this issue is to "pad" the sequence vectors with empty values so they all have the same length (Figure 1). This quickly becomes a problem when there is a big variance in sequence lengths, as some sequence vectors will become very information sparse. Fortunately, dealing with bacteriocins generally means dealing with shorter sequences, making this issue less catastrophic. Another potential solution is to use recurrent neural nets, which will be discussed later on.

A significant amount of the sequences dealt with in these experiments, contain Xs. These indicate when the amino acid in the given position is unknown. This is an issue for some of the encoding methods described below. To cope with this where necessary, the most frequent amino acid among the 7 nearest neighbors on each side replaced the X. This might be a suboptimal solution and another potential strategy is addressed when discussing *Word2Vec*. Note however, that many of the encoding methods have their own ways of dealing with missing amino acids. For instance, the word embedding methods, if trained on a peptide corpus containing Xs, will treat it as an actual amino acid with its own unique "meaning".

It is common to divide the sequences into overlapping $k$-mers

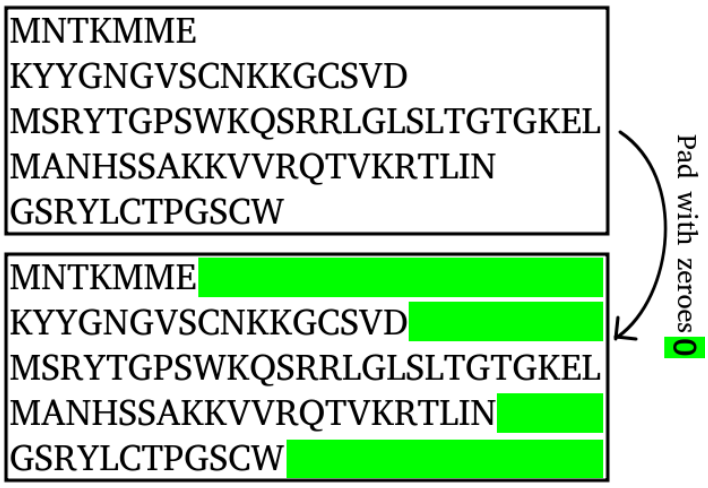



Figure 1: Sequence padding is done to obtain representations of each sequence with the same amount of rows. Only sequences with lengths equal to the maximum length observed are not padded.

$$A = \begin{array}{c} \\ G \\ H \\ \vdots \\ V \\ M \end{array} \begin{array}{c} \begin{array}{cccccccccccccccccccc} A & R & N & D & C & E & Q & G & H & I & L & K & M & F & P & S & T & W & Y & V \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 2: Example of one-hot encoding an amino acid sequence. The first row consists of the entire amino acid alphabet. As G is the 8th letter of the alphabet, the 8th digit of the vector encoding G is a one. Similarly H is the 9th, V is the 20th and M is the 13th. Source: [5].

as a way of increasing the size of the amino acid alphabet. In this way the same amino acid can have different "meanings" depending on its $k+1$ neighbors, instead of always encoding the same "meaning". This division was performed in some but not all of the encodings strategies, as described below ($KM$ in Table 1 indicates k-mer division and $AA$ indicates amino acid).

### Basic sequence encoding types

*One-hot* encoding (Figure 2) is the most basic encoding and is very frequently used for categorical variables. This is especially true when no quantitative encoding is available. To encode amino acid residues into one-hot vectors, one must first make an "alphabet" of the unique amino acids with a specific order. The first letter of the alphabet is now represented as a vector of the same length as the alphabet. It consists of zeroes with the exception of the first digit, which is a one. In the same way the second letter of the alphabet is represented with all zeroes except the second digit of the vector, which is again a one. In this way an $n$ long amino acid residue sequence with an $\alpha$ long alphabet is represented as an $n \times \alpha$ sparse matrix. In the experiments performed here, one-hot encoding was performed on single amino acid letters rather than overlapping $k$-mers.

*Reduced alphabet* encoding groups individual amino acids based on their similarity. For the Murphy et al. reduced alphabet this similarity is quantified using correlations from BLOSUM50[6]. This results in highly correlated amino acids being collapsed into a single letter. Here, we decided to go with the 10 letter reduced amino acid alphabet by Murphy et al. followed by one-hot encoding of single amino acids.

*Athcley factors* are 5 quantities specified for each amino acid relating to its polarity, secondary structure, molecular volume, codon diversity and electrostatic charge[7]. The intention with using these values rather than one-hot vectors is to feed the machine with preexisting knowledge of the properties of each amino acid. By first extracting all overlapping $(n-(k-1))$ $k$-mers with $k=3$ from an $n$ long sequence and then replacing each 3-mer with a vector of the 3 concatenated atchley factor sets of 5 each, we obtained vectors of length $d=(n-(3-1))\cdot 5\cdot 3$ (Table 1).
Inspired by Thomas et al. we also tried clustering these Athcley sequence vectors into a hundred different cluster-representing vectors[7]. To this end we used the unsupervised $k$-means algorithm. This algorithm takes a specified hyperparameter $k_m$ (not to be confused with the $k$ of "$k$"-mer), which in our case was 100. It then assigns each vector to the nearest of the 100 clusters in the $d$-dimensional space. In this way we could represent each sequence as a uniform-length vector by specifying how big a fraction of each cluster was contained in each sequence. This decreased the dimensionality to the number of clusters, which in our case was 100.

### Transfer learning

Many problems in machine learning have initial tasks that resemble one another. As an example the classification of images might start with subdividing the image into different aspects. These might be curves and edges etc., more or less irrespective of what the intended learning outcome is. Transfer learning utilizes this thinking to save enormous amounts of time, while limiting the required training data by learning on top of pre-trained models. Today this method is dominating many topics within the field of NLP[8]. As we shall see, it is also quite useful in biological sequence analysis.

### Word embedding

Much like previously discussed for biological sequences, words in text classification also need to be vectorized in a way a machine understands. This is referred to as word embedding and can be as simple as one-hot encoding. But any method that takes a word and gives it a set of coordinates in a single- or multidimensional space can be termed word embedding. As with images, text classification is likely to benefit from existing knowledge of the relationship between words. This is the idea behind the transfer learning word embeddings *Word2Vec* and *embeddings from language models (ELMo)*. But before we

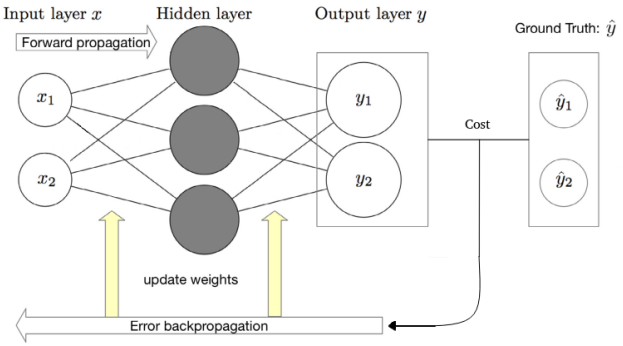



Figure 3: An overview of a deep neural network. The input values of the input vector $x$ is multiplied by their individual weights and put through an activation function. Hence, the information is propagated forwards until the output vector $y$ is obtained. The output is compared to the ground truth and the weight gradients of the cost function are computed. The gradients are used to update the weights. Source: Adapted from [4].

can dive into what exactly those are, we need a basic understanding of neural networks.

### Neural network introduction

Neural networks are among today's most popular machine learning algorithms. They come in many different flavors and here we will discuss a few of the popular ones.

*Shallow neural networks*, often exemplified by single layer perceptrons are the most simple forms of neural networks. They have a single weight $w$ for each element of the input vector as well as a bias $b$ which is added after the weights have been applied. Additionally, they have a threshold or activation function which is often simply $sign(\cdot)$. This activation function is +1 if the input is positive and -1 if the input is negative.

*Deep neural networks* (DNNs) are like shallow neural networks but with additional layers in between the input and output layer, termed "hidden" layers (Figure 3). Additional layers mean additional sets of weights as well as activations. These networks typically also have different activation functions. One such function is the frequently used rectified linear unit (ReLU) which is simply 0 if $x < 0$ and $x$ if $x >= 0$.

*Training a neural network* is carried out with a method called backpropagation. It is done by predicting the labels of the training data as well as an error or cost with the current $w$s and $b$s. By finding the gradients of the cost function with respect to the weights one can update the weights to minimize the cost function. The equations to obtain the gradients are:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$$
$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

Where $C$ is the cost function and $w_{jk}^l$ is the weight going from node $k$ of layer $l-1$ to node $j$ of layer $l$. $a_k^{l-1}$ is the activated output of node $k$ of layer $l-1$. $b_j^l$ is the bias value of node $j$ in layer $l$. $\delta_j^l$ is the error term of node $j$ in layer $l$, indicating the change in the cost function as the output of this node is changed.
Propagating backwards through the network and taking the average of the gradients over all the training samples produces a vector in the weights space. This vector points in the direction of greatest positive change of the cost function. Commonly algorithms related to stochastic gradient descent (SGD) such as "Adam" are used in order to train a neural network. SGD calculates the gradient for a random batch of samples and takes a step in the inverse direction of the gradient multiplied by a learning rate $\lambda$. It keeps doing this until a maximum step number is reached or the steps themselves become sufficiently small.

### Transfer learning sequence encodings

*Word2Vec* is a word embedding algorithm developed by Google researchers[9]. The input of the algorithm is a (typically large) corpus of sentences. The output is a vector for each unique word in the corpus of a chosen dimension $d$. These output vectors are arranged in the $d$ dimensional space so that those of similar context appear closer to each other. This means that the word "dog" is likely closer to "whale" than "chair" in the embedded vector space. This is because the former two words appear in more similar contexts relative to the latter. Following this logic "fish" is thus likely to be even closer to "whale" than "dog" is. Note however, that these relations are highly dependent on the training corpus. For example, a biology textbook might use "dog" and "whale" in more similar contexts as these animals are both mammals whereas "fish" is not.
It turns out that other interesting sentiment patterns emerge from these embeddings. A common example of this is to take the vector of the word "king" and subtract the vector of the word "man". Now add this difference to the vector of the word "woman" and you will obtain the vector of the word "queen". Other interesting observations include the relation between countries and their capital cities (Figure 4).
It is not just for the spoken language that this algorithm is applicable. Any set of diverse objects that appear together in some sort of context can be described in this way. This makes it possible to use this technology on biological sequences as well, which is what was done here.
Word2Vec comes in two different flavors, namely skip-gram and bag-of-words. Here we will focus exclusively on the skip-gram flavor, as this generally seems to be the one used in bioinformatics. To obtain the word embeddings of

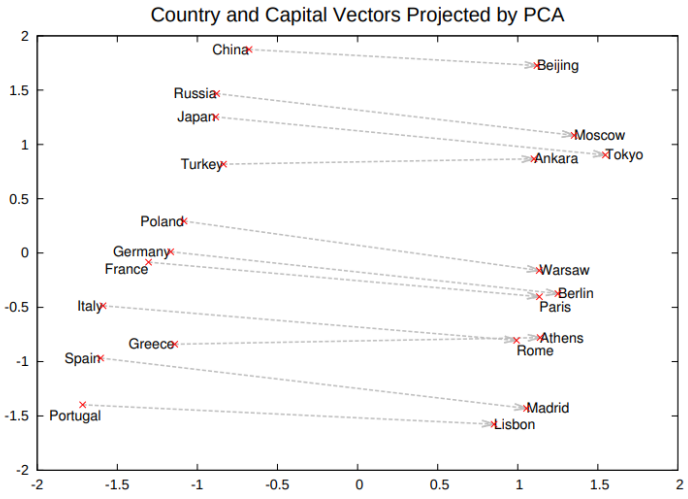



Figure 4: Capitals are related to their countries in a similar manner across multiple capital-country pairs in the Word2Vec vector space. Source: [10].

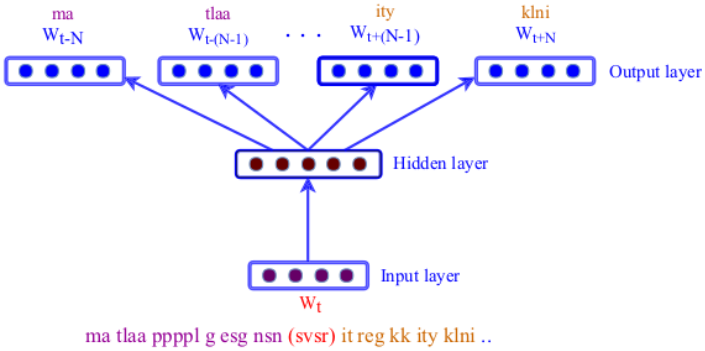



Figure 5: Skip-gram Word2Vec architecture. The lower case letter peptide sequence is split into individual k-mers (here of varying length). The neural network tries to guess the neighboring words within the neighborhood of the input word. Source: [11].

Word2Vec, a neural network with a single hidden layer is constructed. The first step of generating a Word2Vec model for amino acids is to convert the corpus of sentences to input and label vectors. Each 3-mers of the sequence is fed to the network as input once for every neighbor it has in a window of size $w$. Along with each of these input vectors, a label vector containing a neighbor of the 3-mer in question is also fed to the network. This means that the network will try to predict the 3-mers around the input 3-mer. In other words, it predicts the context (Figure 5). Note that the vectors for each 3-mer here are inputted as one-hot encoded. Therefore the input vectors will consists each of a single "one" and a large amount of zeroes. By extracting the weights from the input layer to the hidden layer we obtain the Word2Vec encodings. Since the input is one-hot encoded all the weights from the first input node will be the embeddings of the first word in the alphabet. Likewise, weights from the second node will be the embeddings of the second word etc.

For these experiments we used a pre-trained Word2Vec model released by Hamid et al.[12] trained on the UniProt TrEMBL database[13]. A principal component analysis (PCA) plot of the $d = 200$ dimensional space of all embedded 3-mers is seen in Figure 6 and 7. It is clear to see that 3-mers containing selenocysteines (U) and unknown amino acids (X) have significantly different contexts compared to the rest. The context variations of the ordinary amino acids become clearer when the 3-mers with these two special amino acids are removed (Figure 7). According to these plots the most consistently unique-context 3-mers are those that contain tryptophan (W) and cysteine (C). This makes sense, as cysteines are known to form disulfide bridges and tryptophan is a very large amino acid that does not fit everywhere. It is interesting to see how Word2Vec can pick up on these relations without being told anything prior to observing the TrEMBL database. In addition to feeding the plain Word2Vec vectors we also tried grouping them into 100 clusters as described for the Athcley factors
Another useful property of a trained Word2Vec model is its ability to infer the true identity of X amino acids given the amino acids they are surrounded by. This is known as imputing.

*ELMo* is a newer word embedding method[14] which has some advantages over Word2Vec. The main problem of Word2Vec is the underlying assumption that every word of an alphabet only has a single context. This might be a less than optimal representation of words such as "bank" and "fire", that can mean different things in different contexts. It is likely that this idea can be translated to the world of proteins, where a single amino acid might have different roles depending on what other amino acids it is surrounded by. ELMo has a way of embedding words that can take these different contexts into account by having different representations of each amino acid depending on its surroundings. However, this also means that it takes longer to encode amino acids as the embedding does not just consists of a single static vector for each word of the alphabet.
Having several representations for each word means that it is not necessary to look at overlapping k-mers. Instead we can give the ELMo model single amino acids as input and it should in theory figure out the different meanings they all carry. This also means that the size of our alphabet is significantly decreased.
ELMo is a language model. Language models are trained to predict the next word of a sentence given the preceding words. This is similar to how Word2Vec is trained to predict neighboring words. However, instead of just extracting the weights as in Word2Vec, the entire ELMo model is used to compute the embeddings of the sequences to be classified, before they are fed into the task specific classifier. The ELMo architecture uses bidirectional long short-term memory (LSTM) recurrent neural networks (RNN), which are briefly described below

*An RNN* is a type of neural network that takes the


Figure 6: 2-dimensional PCA plots of all the Word2Vec embedded 3-mers. Each plot shows whether the 3-mer contains the given amino acid residue.

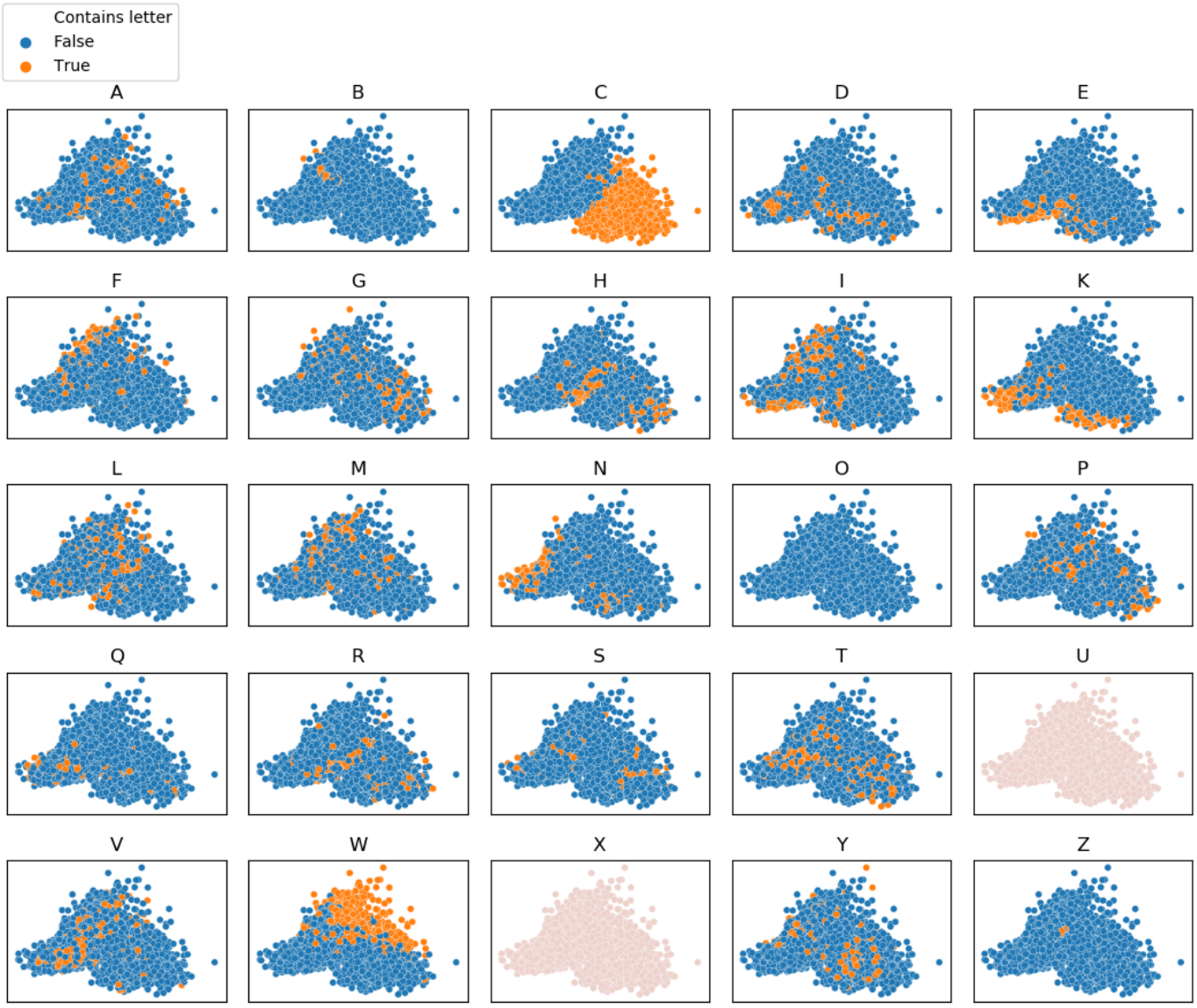



Figure 7: 2-dimensional PCA plots of all the Word2Vec embedded 3-mers. Here 3-mers containing "U" or "X" have been removed before PCA dimensionality reduction. Each plot shows whether the 3-mer contains the given amino acid residue.

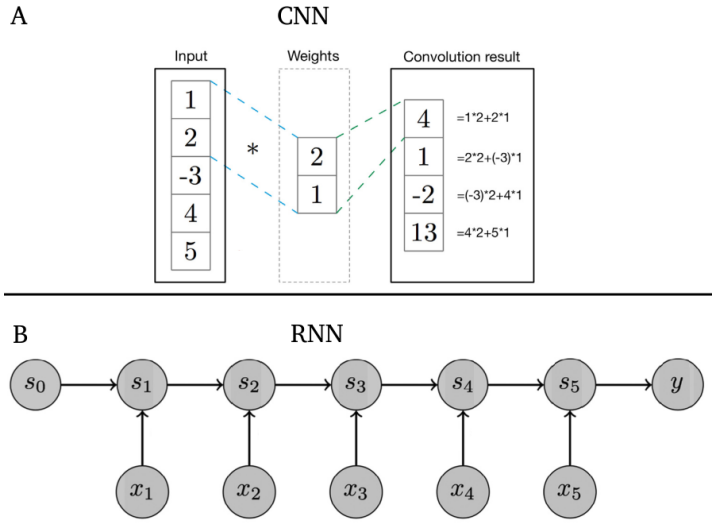



Figure 8: **A** Convolutional neural network filter illustration. The inner product is computed between the layer weights and each overlapping segment of length two of the input vector to obtain the convolution result. **B** Recurrent neural net. Each cell in the network receives information from the input sequence as well as the preceding cell. Here, the network outputs a single $y$ value after the whole sequence of length 5 has been processed. Source: adapted from [4].

hidden layer output of the previous entity in a sequence and feeds it to the hidden layer of the next entity in the sequence (Figure 8B). This is very useful when the order of the input carries meaning. This could be for the task of classifying the tone of as sentence as positive or negative. The problem with RNNs is that when computing the gradients during backpropagation, they tend to vanish for the earlier time step weights. This is what happens when many scalars $< 1$ are multiplied, and results in the initial layers not being trained properly. This is known as the problem of the vanishing gradient.

*An LSTM* solves the vanishing gradient problem by using more complicated recurrent cells. The input information flows through them via two separate paths. Namely the cell state and the hidden state. The cell state contains the information from all the previous states and is continuously selectively build upon by filtering and combining with the hidden state. The hidden state also contains information from the previous cells. Filtering of the hidden state is done by "forgetting" content that is not important for classification and blending the rest with the cell state. An RNN can be bidirectional by feeding the input vectors into both a network of recurrent cells with information flowing from right to left and from left to right.

*The ELMo architecture* itself consists of two bidirectional LSTM layers. Both of these receive the input sequences, where the first bidirectional LSTM layer also feeds its output to the second (Figure 9). Before the input is fed to the LSTM layers they are embedded using varying methods. The architecture used in our experiments uses a convolutional neural network (CNN) which is a network type defined in Part 2. A pre-trained ELMo model is used to embed a sequence by running the sequence through the model. Thereafter, the output of the convolutional and the two bidirectional LSTM layers is extracted. These three different outputs each consist of a 1024 long vector. These vectors can then be fed into a classifier to improve performance. However, in our example we also tried simply summing the three vectors to decrease dimensionality before inputting them into our classifier (Table 1). As it turned out, this worked well in practice as we shall see later on.

## Selecting best encodings

To find out what encodings would best suit our task of classifying peptides, we first compared the encodings on a different data set. We also used a classifier relatively quick to train and implement. Scikit-learn is a Python package that offers implementations of many classical machine learning algorithms and is very efficient and easy to use. By using a quicker algorithm to identify encodings outperforming the others, we could save time down the line. This enabled us to not having to train a big, computationally expensive neural network for each of the encodings.
We chose to use a support vector machine (SVM) with a linear kernel. Specifically how SVMs work is outside the scope of this paper. But said briefly: the goal of an SVM is to find the best hyperplane in a vector space to separate the training points based on their labels. The best hyperplane is the one that has the largest distance to differently labeled data points on both sides of the classification barrier.

### Data acquisition

As the amount of known bacteriocins is fairly limited we chose to pick the best encodings on different data. To this end we went to UniProt to find data resembling the kind of data of our actual classification goal. We searched the bacterial domain of life for peptide sequences up to 100 amino acid residues long and explored different gene ontology (GO) terms. We chose "glycolytic process" and "lipid A biosynthetic process" as these were sufficiently difficult to classify, giving the better encodings an opportunity to stand out.

*Sequence clustering* was also done in order to avoid data leakage. Data leakage in this case could happen if there were duplicates or very closely resembling peptides that appeared in both the train and test set. This would artificially make it seem as our accuracy was higher than in reality, as the model would simply be memorizing rather than learning. This would in turn mean poor generalization to samples outside of the test set. To take care of this potential issue we removed all but one peptide per cluster. We used Many-against-Many sequence searching 2 (MMseqs2) which

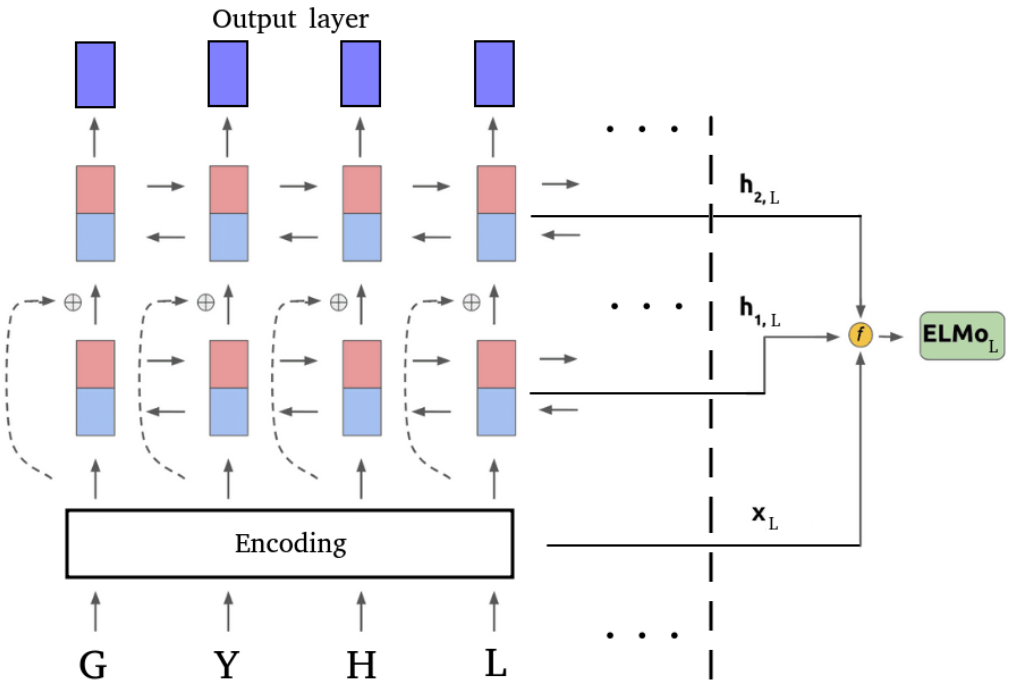



Figure 9: ELMo encoding. Each unique amino acid residue of the sequence is first encoded using a method of choice. The encoded residues are then passed to both of the connected bidirectional LSTM layers. The target value is the next residue in the sequence. To get the embeddings the output from the two bidirectional LSTM layers as well as the encodings and are extracted. These can then be processed by a function of choice. We chose summing. Source: adapted from [15]

is a fast algorithm developed by Steinegger et al.[16]. The default parameter values were used with the expection of "minimum sequence ID" which we set to 0.95.

### Cross validation strategy

A problem when training any type of classifier is the trade-off between the size of the testing data set versus the size of the training data set. A large training set results in little "waste" of data and supplying the model with lots of examples. This makes it more likely that the model has seen enough data points to learn enough about the classification problem to be useful outside of the training sample. On the other hand this scenario leaves little data for actually verifying that the trained model can generalize to data outside of the training set. Another issue is to find out what hyperparameter values are best for generalizing to unseen data. This calls for yet another data set called the validation set and is typically taken as a sample of the training data. To avoid using precious amounts of data on validation that could have been used for training the model, cross validation is a very popular approach.
Cross validation means to split up the training data in $k$ equally sized splits and train a model on all but one of the splits. The last split can now be used to validate the performance of the model. This can be done $k$ times, each time with a new split as validation set. In the end the average performance is calculated over all $k$ validation splits. This whole approach can be performed once for each unique set of choices of hyperparameters. When the optimal hyperparameters are chosen the whole training set is used to train the final model, which can now be evaluated on the unseen test set. We chose to follow this approach with $k = 10$.

### Part 1 results - Support vector machine classification

We trained, validated and tested all the encodings with the linear SVM classifier to distinguish between peptides with GO-terms "glycolytic process" and "lipid A biosynthetic process". As the results demonstrate in Figure 10 and 11, the NLP embedding methods outperformed the more basic encodings both in validation and test. For Word2Vec it turned out that clustering decreased the performance. This is likely because the order of the amino acid residues in the peptide sequence is important for classification, and that the amino acid composition is insufficient. For ELMo encoding we observed that summing and not summing the three layers had very similar performance in both validation and test. It is likely that feeding the un-summed layers into a neural network would find the optimal combination and thereby increase performance. However, as computation time is significantly shorter with summing, we continued with this embedding only, but kept in mind that there might be room for performance improvement. After evaluating the results we moved on to the next stage of the experiment with the encodings summed ELMo (from here on just simply referred to as ELMo) and Word2Vec.

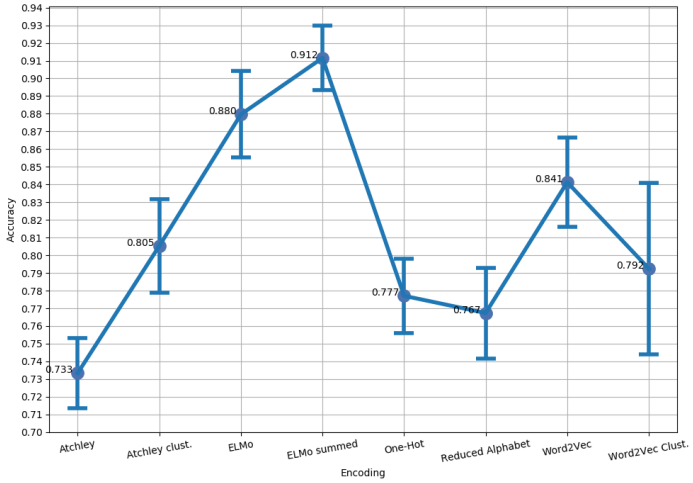



Figure 10: Cross validation accuracy scores of SVM classifiers trained on peptide data with the GO-terms "glycolytic process" and "lipid A biosynthetic process'. The different encodings used are seen on the $x$ axis.

# Part 2

## Combining encodings with neural networks

Now that we had found some promising methods for encoding our peptide sequences, it was time to use some more sophisticated models. The recent results published by Hamid et al. showed neural models using Word2Vec embedding with high enough performance to challenge traditional bioinformatic classification methods. Our hope was to develop a model outperforming theirs by taking advantage of the more powerful and recent ELMo embeddings.
In addition to the better embedding model we also had ambitions to find a neural network architecture that could further increase performance. However, to better understand how these different architectures work there are a few more key topics to unravel.

### Neural networks, continued

*A convolutional neural network* is a type of network often used for image classification. This layer takes an input vector and runs a sliding window over it to obtain snippets of the original vector. A single number is then obtained by taking the inner product between each of these snippets and the weight vector of the layer (Figure 8A). Afterwards is is put through an activation function as we have seen with the other types of layers. This can be thought of as a way of "summarizing" the information of a certain region of the vector. A single convolutional layer can have multiple filters with each their own weights and thus identify different patterns in the input. Convolutional layers are typically followed by pooling layers. This is a way of compressing the outputs of the different filters of the convolutional layer, as more filters mean more new dimensions added. Pooling works similarly to convoluting and runs a sliding window over the activated filter output. Typically it takes either the maximum or the average within each sliding window. Lastly there is a flattening layer that takes each of the vectors from the pooling output and concatenates them to obtain a single long vector.

*A gated recurrent unit* (GRU) is another type of recurrent cell, very similar to the LSTM cell. It also has the ability to selectively "forget" information that is not important for classification. But unlike an LSTM cell it only takes 2 inputs - one from the previous cell in the network and the input $x$ itself. Furthermore, only a single output goes from the current cell to the next in the network.
A potential benefit of using RNNs, which was not discussed before, comes with their ability to take sequential inputs. This means that the input can be of varying lengths, which is highly useful when classifying input of this kind such as sentences in a book. It also means that biological sequences of varying lengths can be classified by the same model. This might be preferred to having to limit oneself to a maximum sequence length by padding each sequence as discussed previously. However, platforms such as TensforFlow only allow matrices with rows of the same lengths because of the computational efficiency that comes from a fixed shape. For this reason we stuck with padding our sequences. Yet, there are ways of dealing with this problem. One could for example split the input sequences into batches of different maximum lengths[17].
To deal with encountering longer sequences than those observed in the training data one could save the weights of the old model. Afterwards one could reload them into a model that takes an input with the desired max length[18].

*Preventing overfitting of DNNs* means to somehow limit the ability of the DNN to fit the data. This might seem counterintuitive but in practice it prevents the model from overfitting the training data and be better at generalizing to samples it has not seen before. It is necessary because the vast amount of parameters in a DNN makes it able to fit any data set perfectly. This includes the noise of the given sample, which is what decreases the performance when new samples with new noise is observed. Three of the most common techniques for limiting overfitting are regularization, dropout and early stopping.
Regularization, which is also frequently used for other methods such as linear regression, is a way of penalizing relatively larger weights of the model. This is done by adding a penalty term doing exactly this to the cost function that is minimized during training.
Early stopping means to observe the validation performance as training is carried out. By doing this one can stop the training once the validation set performance no longer improves (Figure 12). This in turn has the downside of possibly overfitting the validation data, which is why the test set is important to verify the performance.
Dropout means to randomly remove nodes from a layer during training. This is a way of forcing the network to to

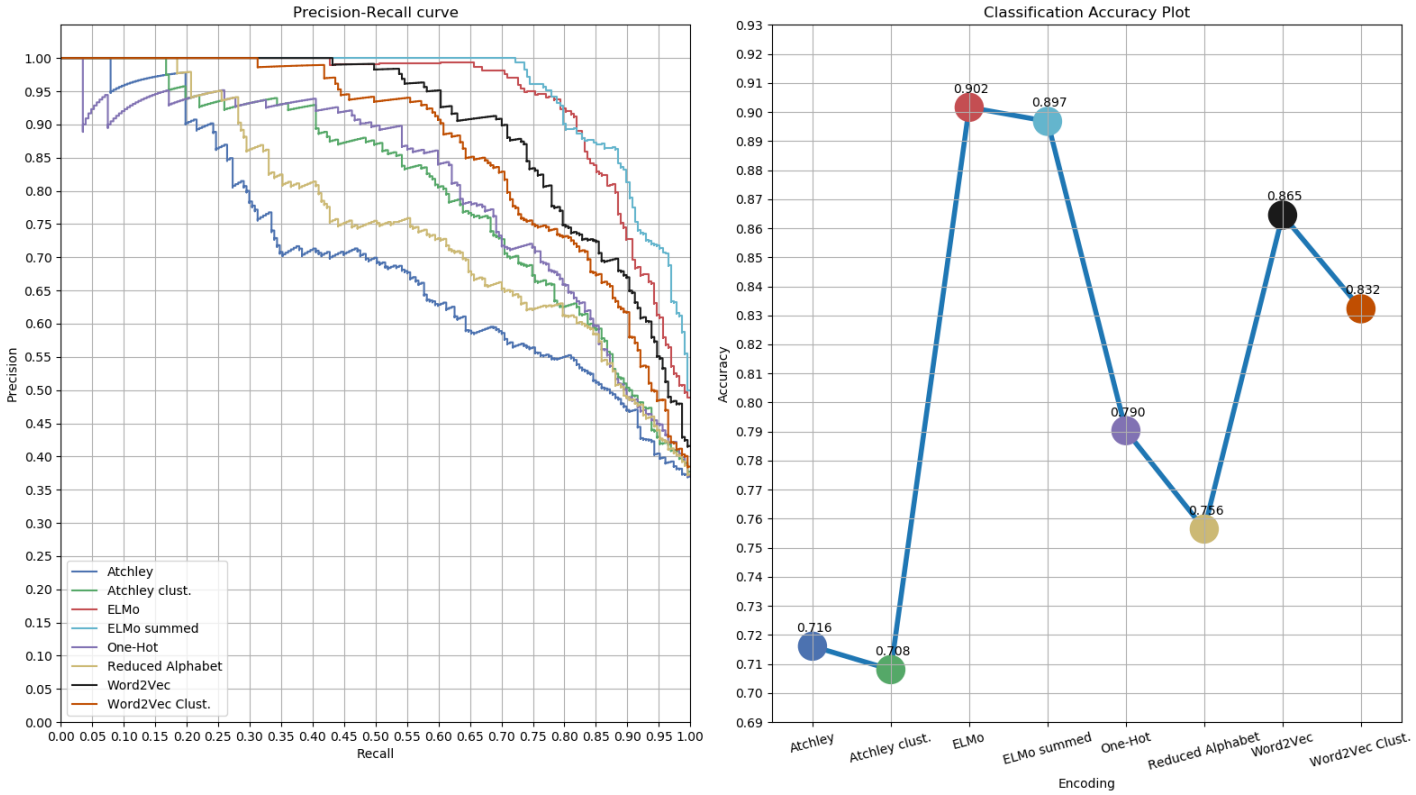



Figure 11: Test set result of SVM classifiers trained on peptide data with the GO-terms "glycolytic process" and "lipid A biosynthetic process". **Left** Precision recall curves for all SVM classifiers with precision on the $y$ axis and recall on the $x$ axis. Encoding type is shown by color in the legend. **Right** Test set accuracy scores of SVM classifiers. The different encodings used are seen on the $x$ axis.

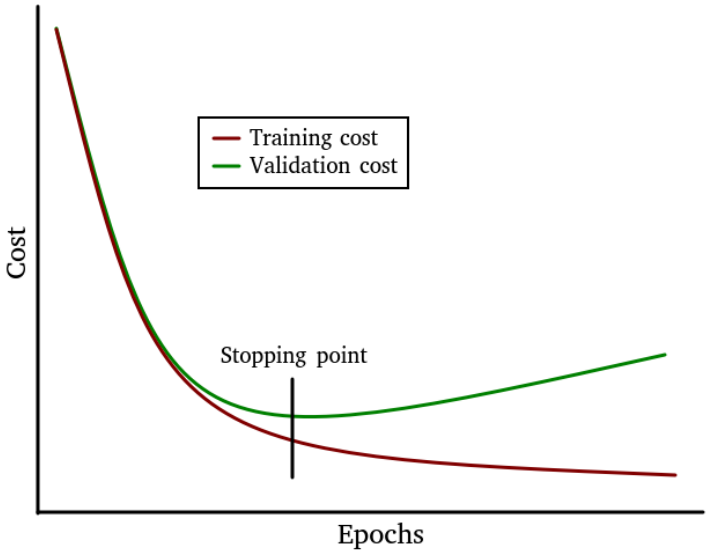



Figure 12: Early stopping. By observing the validation score as training proceeds, one can stop as soon as the validation performance stops improving.

rely less on individual nodes, as it cannot count on the node consistently being available[19]. It is implemented by having a dropout layer in the neural network with the same amount of nodes as the preceding layer. In this layer a fraction $p$ of the nodes are dropped at each iteration.

## Part 2 results

### Bidirectional recurrent GRU network architecture

The architecture utilized by Hamid et al., referred to here as BIDGRU (Figure 13), consist of two bidirectional GRU layers with 32 nodes each. These are followed by a dense layer (meaning feed forward fully connected and is neither an RNN or CNN) for the final classification. Though it does not show in the figure, there is a 50% dropout layer after each of the LSTM layers.

Before looking into additional architectures we tried classifying the UniProt data set from before with GO terms "glycolytic process" and "lipid A biosynthetic process" with BIDGRU. We used the same cross-validation strategy as the one described in Part 1 to obtain validation scores for both the ELMo and the Word2Vec embedding strategy. The results in Figure 14 show that ELMo outperforms Word2Vec on this data set. The results also show that the accuracy is higher with the neural network than with the SVM from Part 1.

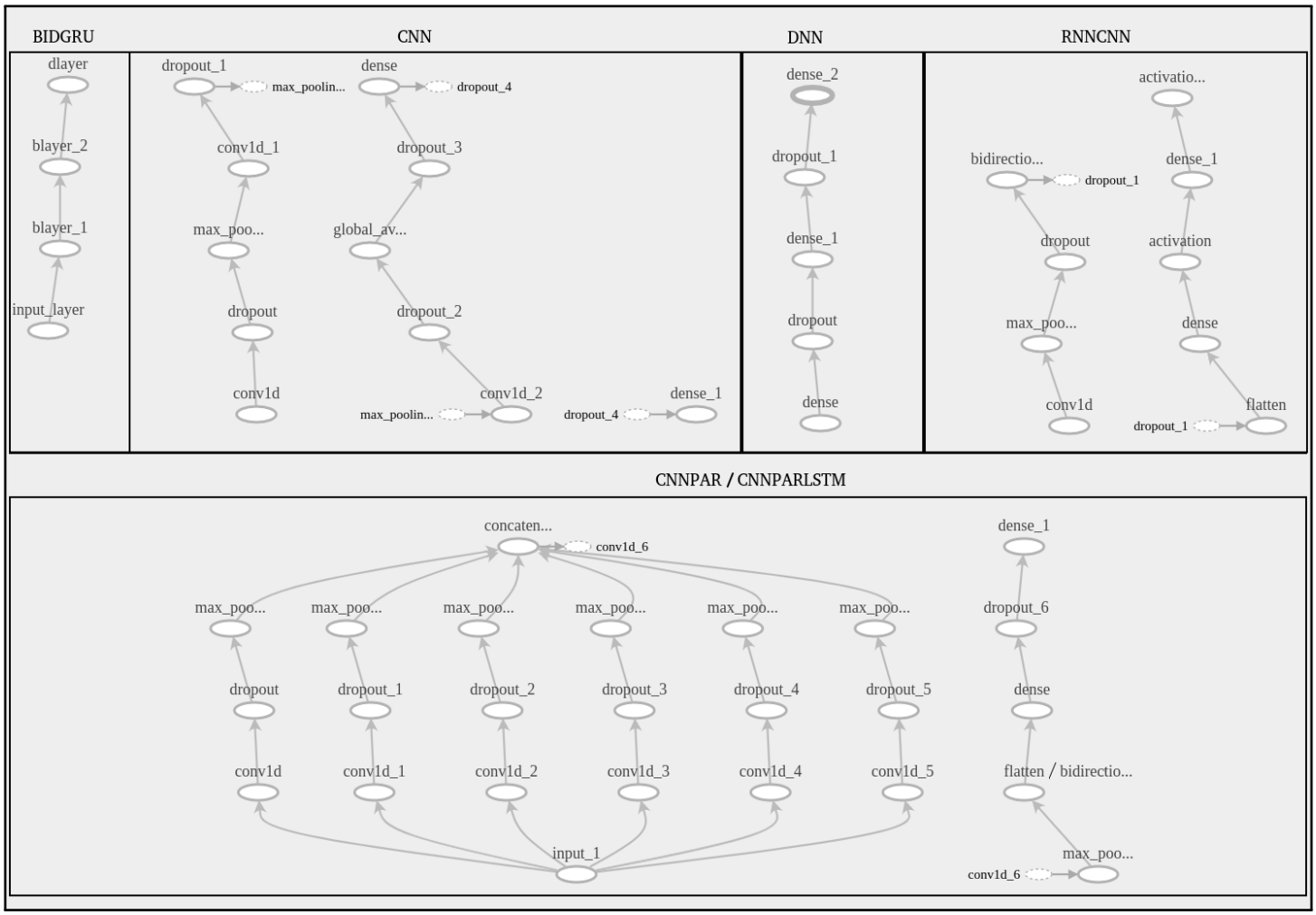



Figure 13: All six neural network architectures used in our experiments. Dropout and activation is not always shown. And when it is, it is not always applied. This is caused by Keras automatically generating the graph based on the way the neural net is implemented. CNNPAR and CNNPARLSTM is shown in the same graph because the only difference is the flattening/bidirectional recurrent layer indicated by a forward slash.

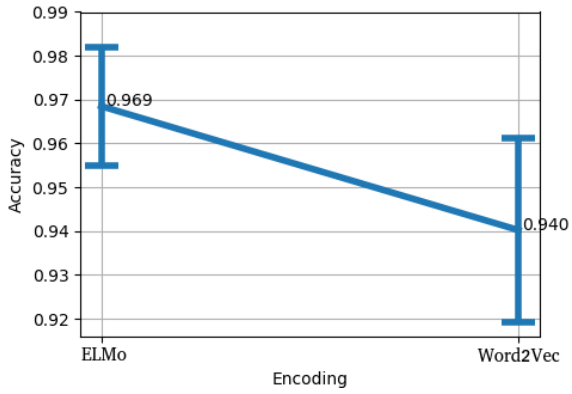



Figure 14: Cross validation accuracy scores of ELMo and Word2Vec BIDGRU classifiers trained on peptide data with the GO-terms "glycolytic process" and "lipid A biosynthetic process'.

### Training and validating bacteriocin model

*Obtaining bacteriocin data set*

With all the preceding work, we now felt confident that we had the tools to make a decent bacteriocin classifier. The first step was to obtain a data set containing bacteriocins. We downloaded the databases BAGEL, CAMP and Bactibase. With the exception of CAMP these databases exclusively store bacteriocins. By filtering away non-bacteriocin sequences of the CAMP databases we got a total of 1003 bacteriocins between 10 and 359 amino acid residues long. To get a set of non-bacteriocin sequences we searched UniProt Swiss-Prot for bacterial peptides of length 10 to 359, clustered with UniRef50. As done by Hamid et al.[12] we included the keywords "not antibiotic", "not antimicrobial" and "not plasmid" in our query. This got us 286.070 sequences which we length-matched with the bacteriocins in a 1:1 ratio. In this way we had a balanced and equally length-distributed data set.

*Finding new neural network architectures*
Next we wanted to find new network architectures that could possibly increase performance relative to BIDGRU. A review by Li et al. provides implementations of eighth different architectures previously used for other bioinformatics tasks[4]. We found three of these to be relevant for our experiment. Furthermore a paper by de los Santos compared a handful of architectures for classifying ribosomally synthesized and post-translationally modified peptides[20]. Of these we picked an additional 2 architectures, giving us a total of 5 new ones. These are all shown in Figure 13 and are described in more detail below. Preceding their classification layers the different architectures consist of:

- Deep neural network (DNN)[4]
    1. Two fully connected layers with 32 and 64 nodes, respectively, each followed by 50% dropout layers.
- Convolutional neural network (CNN)[4]
    1. Three sequential convolutional layers with 128, 128 and 256 filters, respectively. Each followed by a 25% dropout and a pooling layer.
    2. A dense layer with 128 nodes and a dropout of 50%.
- Recurrent LSTM convolutional neural network (RNNCNN)[4]
    1. 640 filter convolutional layer with pooling and 20% dropout rate.
    2. Bidirectional LSTM layer without (or 0%) dropout.
    3. Flattening layer.
    4. Dense 925 node layer.
- Parallel convolutional neural network (CNNPAR) and CNNPAR recurrent LSTM network (CNNPARLSTM)[20]
    1. Six parallel convolutional layers with 75 filters each of varying sliding window sizes. Each with a max pooling layer, but no dropout (or dropout of 0%).
    2. Concatenation layer.
    3. 300 filter Convolutional layer followed by max pooling.
    4. A flattening layer for CNNPAR and a 50 node bidirectional LSTM layer for CNNPARLSTM.
    5. 120 node dense layer with no dropout (or dropout of 0%).

*Hyperparameter tuning procedure*
To get the best model possible we tried tuning the hyperparameters of the 5 new architectures with ELMo embedded input. We left the BIDGRU architecture with Word2Vec embedding unchanged as a benchmark. We have already covered a number of hyperparameters such as the size of each layer, number of convolutional filters and sliding window size. But more exist such as the size of each batch used for learning and for how long the network should train. The length of the training can be dialed up or down by changing the number of "epochs". An epoch refers to evaluating and backpropagating through the training data once, and is typically done multiple times until the cost function converges. Another hyperparameter is learning rate. Once the weight gradient has been computed, as discussed in Part 1, the learning rate decides how large the "step" should be towards minimizing the cost function.
With so many dials to turn it quickly becomes computationally infeasible to try out all hyperparameter values. Instead we chose a few values for each hyperparameter around the default value of the original implementation. By randomly combining these hyperparameter settings, we could cover a lot of ground with far fewer training sessions. Instead of validating each of these hyperparameter combinations on all ten folds of the cross validation we did it only on the first fold. This saved us time, so that we could try out more different hyperparameter values. However, this also meant that the settings had only been validated once each and are thus less trustworthy. After obtaining the best hyperparameter values we trained and validated the model on all 10 folds. We selected the amount of epochs with the highest mean validation accuracy over all folds for each architecture and embedding pair (Figure 15).

*ELMo vs Word2Vec on 6 architectures*
Figure 16 shows the results from the cross-validation. With the BIDGRU architecture we saw a significant increase in validation accuracy when using ELMo embedding. Exploring more architectures we found that both of the models from de los Santos[20] had a very high accuracy but with the lowest deviation for CNNPAR. Surprisingly enough it was thus CNNs that performed best, though RNNs are the primary architectures used in NLP. This might indicate that the translatability from sentence analysis to biological sequence analysis is not straightforward. On the other hand, we did not try an architecture purely recurrent other than the BIDGRU architecture. Also, when we tried a pure recurrent architecture it was using GRU cells and not LSTM cells, which is probably the most common of the two.
Overall we see significantly better mean performance when using ELMo embedding as well as smaller deviations. Furthermore, when looking at Figure 15 ELMo seems to be have a more stable convergence than Word2Vec as the training progresses through the epochs. With these results we decided to move forward with CNNPAR and ELMo embedding.

*Test set accuracy*
We downloaded the freely available "NeuBI" model made by Hamid et al.[12] and applied it on the test set to obtain a score of 85.96%. In this evaluation we excluded 3 sequences longer than 302 amino acids because of their model's limitations.
Afterwards, we trained the CNNPAR model on the full

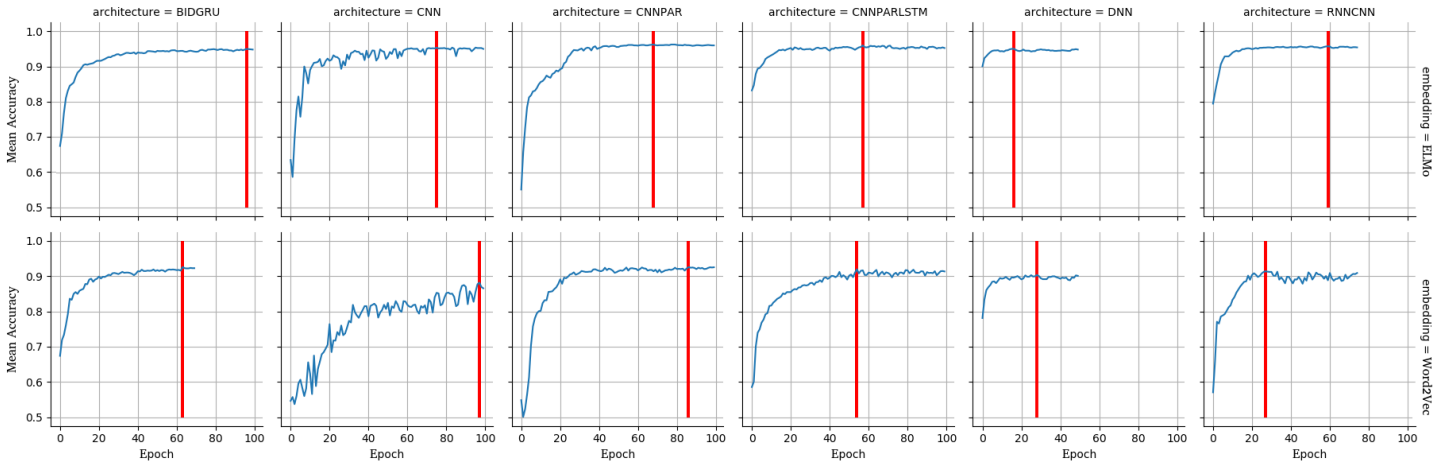



Figure 15: Mean validation accuracy development on bacteriocin training set. Epochs are shown on the $x$ axis and the epoch with the highest mean accuracy is indicated with a red vertical line for each architecture-embedding pair.

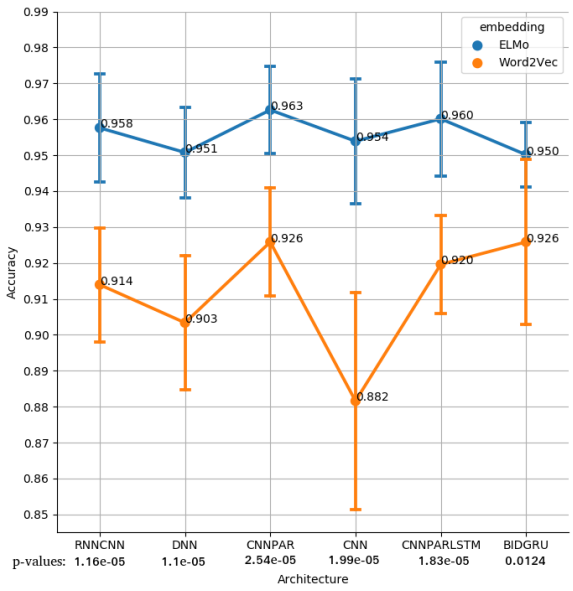



Figure 16: Cross validation accuracies of all embedding-architecture pairs on bacteriocin training set. Standard deviation is indicated with error bars, and Welch's independent t-test p values with equal mean null hypothesis is seen on $x$ axis.

ELMo embedded training set and evaluated it on the test set to obtain the results in Figure 17. The test accuracy of 0.948 which is lower than the validation accuracy indicates that we have overfitted the validation set. However, along with the output label, the model also provides a certainty score indicating its confidence. When looking more closely it is clear that there are some peptides in which the model has very low confidence in classifying (red density line in Figure 17). When looking at the bulk of the test set peptides where the model certainty is above 80% the classification accuracy actually reaches $> 98\%$. Nevertheless, as we do not know if this is also the case for the validation set, we cannot exclude the possibility of overfitting. In any case, we have built a model with higher accuracy than its predecessor, NeuBI.

# Part 3

## Applying the model

To apply our new model to discover potential bacteriocins we thought a great place to look would be the human microbiome. Microbiomes such as the human one are rich in bacteria competing for nutrients and territory. This makes it a very good place to look for potential bacteriocins.

### New small proteins database

Recently, Sberro et al. released a large body of small proteins sampled from the human microbiome of four different anatomical sites[21]. They argue that short proteins (<50 amino acids long) have thus far been ignored in the literature. This is mainly due to the fact that open reading frames (ORFs) this small are likely to be confused with randomly occurring non-coding ORFs. Using techniques to discriminate between functional and spurious ORFs they identified ∼467 thousand small proteins which could be

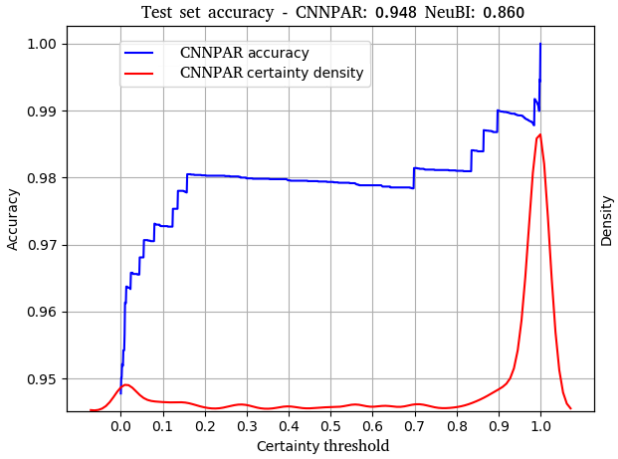



Figure 17: Test set accuracy of CNNPAR model with ELMo embedding. The blue line indicates the accuracy when only looking at predictions with certainty above the $x$ axis threshold. The red line shows the kernel density of the distribution of certainty values of the predictions. Above the plot the test set accuracy is seen for all test samples for both the CNNPAR model and the NeuBI model.

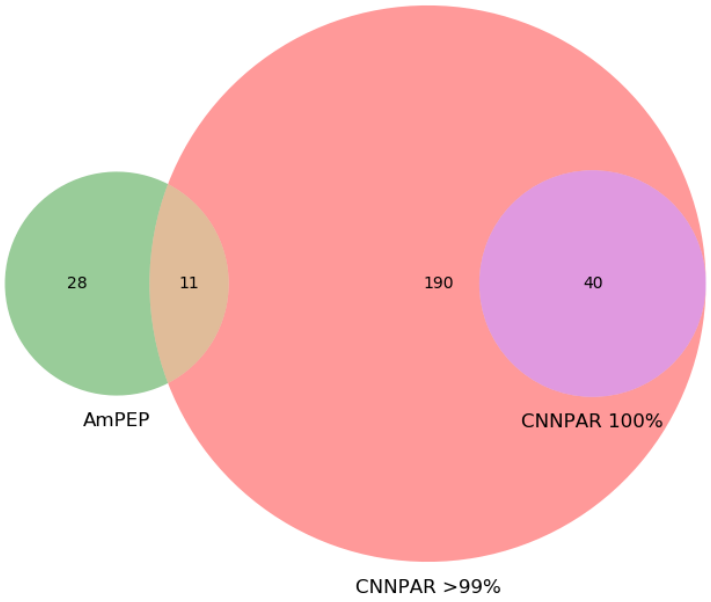



Figure 18: Venn diagram of positive predictions made by CNNPAR with >99% and 100% certainty as well as AmPEP on the $\sim 4,500$ small protein families

clustered into what they refer to as ∼4.5 thousand "small protein families". Of these small proteins families very few had previously been annotated.
Sberro et al. also screened their newly found small proteins for antimicrobial activity. Of the ∼4.5 thousand protein families they found 39 potential antimicrobial peptides. They used software written in MATLAB called AmPEP, a tool that uses random forests for classification[22]. To compare the AmPEP model to ours, we classified all the peptides of our bacteriocin test set. AmPEP had an accuracy of 65.42% compared to the 94.8% of our model. To the best of our knowledge we have executed the AmPEP code correctly and followed the instructions they provide. This shows us that AmPEP might not be the optimal model to scan the small protein database for potential bacteriocins.

*Classification results*
We applied our model on the small protein families and found 241 potential bacteriocins with a model certainty of >99% (Figure 18). There was an overlap of 11 proteins also identified by AmPEP. However, the model score of AmPEP was only in the range of 50.3-86.0%. When we only looked at 100% certainty bacteriocin hits we found 40 proteins but without any overlap with the AmPEP result.

## Discussion

*State of the art word embedding models*
Recently the field of NLP has had an overwhelming amount of breakthroughs, each outcompeting the previous one. These include everything from the release of the groundbreaking ELMo in early-2018 to BERT in late-2018[23], XLNET in June 2019[24] and the most recent (that we are aware of) Ernie 2.0 in July 2019. It's worthy to note that we have not yet seen the full capability of these models in NLP. According to the lead author of the ELMo paper they haven't yet capped out performance-wise as they become larger and are trained on bigger and bigger corpora of data[25].

*NLP in bioinformatics*
With such a rapid change of the state of the art it is difficult to stay on top of the field of NLP. Yet we in the bioinformatics field need to keep our eyes on this exceptionally fast-moving ball. There might be very important discoveries awaiting the adoption of these NLP techniques in biological sequence analysis. Embedding models such as BERT and ELMo has already been used in biomedical text mining. The applications comprise named entity recognition, question answering, relation extraction[26, 27], identifying detection methods from molecular interaction experiments and classifying papers containing new interaction data[28]. But other than text mining applications we are not aware of cases of bioinformatics analysis where embedding models newer than Word2Vec has been used besides Heinzinger et al.[29].

Recently Hamid et al. showed that using neural networks and NLP word embedding techniques could challenge existing methods[12]. Others have shown the utility of Word2Vec and its closely related sentence level embedding method Doc2Vec[11, 30, 24]. We have showed that updating the embedding method to ELMo can increase the performance of these types of neural nets. We used this new high-accuracy model to find 40 putative bacteriocins undetected by an older model.

New inventions happen on an almost monthly basis in NLP. If we utilize the power of this rapid development we might see in the near future that older biological sequence classification methods become obsolete. But that can only happen if papers like this one and others cited here succeed in calling attention to the power of word embedding in biological sequence analysis. We do not yet know what kind of impact it would have to use a state of the art embedding model to classify bacteriocins. It is our hope to see large models such as Google's BERT being trained on the vast amounts of sequence data available, to further push the performance of sequence classifiers as well as the bioinformatics field.

# References


[1] Zheng Pang, Renee Raudonis, Bernard R. Glick, Tong-Jun Lin, and Zhenyu Cheng. Antibiotic resistance in Pseudomonas aeruginosa: mechanisms and alternative therapeutic strategies. *Biotechnology Advances*, 37(1):177–192, jan 2019.

[2] Hedvig E. Jakobsson, Cecilia Jernberg, Anders F. Andersson, Maria Sjölund-Karlsson, Janet K. Jansson, and Lars Engstrand. Short-Term Antibiotic Treatment Has Differing Long-Term Impacts on the Human Throat and Gut Microbiome. *PLoS ONE*, 5(3):e9836, mar 2010.

[3] Veeresh Juturu and Jin Chuan Wu. Microbial production of bacteriocins: Latest research development and applications. *Biotechnology advances*, 36(8):2187–2200, 2018.

[4] Yu Li, Chao Huang, Lizhong Ding, Zhongxiao Li, Yijie Pan, and Xin Gao. Deep learning in bioinformatics: Introduction, application, and perspective in the big data era. *Methods*, apr 2019.

[5] Sebastian Spänig and Dominik Heider. Encodings and models for antimicrobial peptide classification for multi-resistant pathogens. *BioData Mining*, 12(1):7, dec 2019.

[6] L R Murphy, A Wallqvist, and R M Levy. Simplified amino acid alphabets for protein fold recognition and implications for folding. *Protein engineering*, 13(3):149–52, mar 2000.

[7] Niclas Thomas, Katharine Best, Mattia Cinelli, Shlomit Reich-Zeliger, Hilah Gal, Eric Shifrut, Asaf Madi, Nir Friedman, John Shawe-Taylor, and Benny Chain. Tracking global changes induced in the CD4 T-cell receptor repertoire by immunization with a complex antigen using short stretches of CDR3 protein sequence. *Bioinformatics*, 30(22):3181–3188, nov 2014.

[8] Sebastian Ruder. https://nlpprogress.com/.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. jan 2013.

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. oct 2013.

[11] Ehsaneddin Asgari, Alice C. McHardy, and Mohammad R. K. Mofrad. Probabilistic variable-length segmentation of protein sequences for discriminative motif discovery (DiMotif) and sequence embedding (ProtVecX). *Scientific Reports*, 9(1):3577, dec 2019.

[12] Md-Nafiz Hamid and Iddo Friedberg. Identifying antimicrobial peptides using word embedding with deep recurrent neural networks. *Bioinformatics*, 35(12):2009–2016, jun 2019.

[13] UniProt Consortium. UniProt: a worldwide hub of protein knowledge. *Nucleic Acids Research*, 47(D1):D506–D515, jan 2019.

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

[15] Mihail Eric. Deep Contextualized Word Representations with ELMo. *mihaileric.com*, 2018.

[16] Martin Steinegger and Johannes Söding. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026–1028, 2017.

[17] Kbrose. Training an RNN with examples of different lengths in Keras. *https://datascience.stackexchange.com/a/27879*, 2018.

[18] Aneesh Joshi. Extract learned weights from old model in Keras. *https://datascience.stackexchange.com/a/26505*, 2018.

[19] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. jul 2012.

[20] Emmanuel LC de los Santos. NeuRiPP: Neural network identification of RiPP precursor peptides. *bioRxiv*, page 616060, 2019.

[21] Hila Sberro, Brayon J. Fremin, Soumaya Zlitni, Fredrik Edfors, Nicholas Greenfield, Michael P. Snyder, Georgios A. Pavlopoulos, Nikos C. Kyrpides, and Ami S. Bhatt. Large-Scale Analyses of Human Microbiomes Reveal Thousands of Small, Novel Genes. *Cell*, 178(5):1245–1259.e14, aug 2019.

[22] Pratiti Bhadra, Jielu Yan, Jinyan Li, Simon Fong, and Shirley W. I. Siu. AmPEP: Sequence-based prediction of antimicrobial peptides using distribution patterns of amino acid properties and random forest. *Scientific Reports*, 8(1):1697, dec 2018.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. oct 2018.

[24] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. jun 2019.

[25] Matthew E. Peters and Kyle Polich. ELMo podcast. *https://dataskeptic.com/ time:18:35*, 2019.

[26] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. jan 2019.

[27] Yuqi Si, Jingqi Wang, Hua Xu, and Kirk Roberts. Enhancing clinical concept extraction with contextual embeddings. *Journal of the American Medical Informatics Association : JAMIA*, jul 2019.

[28] Gully A Burns, Xiangci Li, and Nanyun Peng. Building deep learning models for evidence classification from the open access biomedical literature. *Database*, 2019, jan 2019.

[29] Michael Heinzinger, Ahmed Elnaggar, Yu Wang, Christian Dallago, Dmitrii Nechaev, Florian Matthes, and Burkhard Rost. Modeling the Language of Life – Deep Learning Protein Sequences, 2019.

[30] Dhananjay Kimothi, Pravesh Biyani, James M Hogan, Akshay Soni, and Wayne Kelly. Learning supervised embeddings for large scale sequence comparisons. *bioRxiv*, page 620153, 2019.