Predicting Human-Pathogen Protein-Protein Interactions using Natural Language Processing methods

An exploration of HP-PPI using Deep Learning models that specialize in sequential data

Nikhil Mathews
Master's in Computer Science
University at Albany, New York
nik007_me@live.com

***Abstract*—We use multiple Natural Language Processing (NLP) methods available in deep learning and apply them to predict the interaction of proteins between Humans and Yersinia pestis by examining their respective amino acid sequences. Without using any biological knowledge, a model is developed that gives a cross validation AUC score of 0.91 and an independent test score of 0.92, which rivals the reference research paper[1]that uses amino acid sequence and network data as well as extensive use of bio-chemical properties, both sequential and network related, to make their predictions. The same model gave a score of 0.94 for predicting Humans-Virus PPI. This is done by combining advanced tools in neural machine translation into an integrated end-to-end deep learning framework as well as methods of preprocessing that are novel to the field of bioinformatics.**

***Index Terms*—Natural Language Processing, NLP, LSTM, RNN, HP-PPI, Protein-Protein Interactions, Deep Learning, Bidirectional, neural networks, CNN, TCN, Attention, Transformer, 3X**

I. Introduction

The unseen battle between humans and pathogens have been fought since the dawn of time, much of which happened, and still happens, at a molecular level. As the most important type of host-pathogen interaction, protein-protein interactions (PPI) between host and pathogen play an important role in infection and disease progression. One such organism is the *Yersinia pestis*, a rod-shaped bacteria and plague pathogen classified as a potential agent of bio terrorism responsible for three pandemics that have killed tens of millions of people. The reference research paper on which this work is based on uses data from Human-Y.pestis PPI.

In 2019, Xianyi Lian, Shiping Yang, Hong Li, Chen Fu, and Ziding Zhang from China Agricultural University and Hainan University authored a paper[1] where they developed a new machine-learning-based predictor of human Yersinia pestis PPIs. First, three conventional sequence-based encoding schemes and two host network-property-related encoding schemes (NetTP and NetSS) were introduced. The individual predictive models for each encoding scheme were inferred by Random Forest. The first sequence encoding scheme, AC, employed seven physico-chemical properties of amino acids, including hydrophobicity, hydrophilicity, volumes of side chains, polarity, polarizability, solvent-accessible surface area, and net charge index of side chains, to infer the AC feature vector using an equation. This model yielded an impressive AUC of 0.88. Then, CKSAAP encoding considers 400 amino acid pairs that can be extended to the k-spaced amino acid pairs (i.e., the pairs separated by k other amino acids). Here, the CKSAAP encoding considered the k-spaced amino acid pairs, with k = 0, 1, 2, and 3. Finally, the PseTC encoding uses the tripeptide composition to represent a protein sequence by dividing the 20 amino acids into 13 groups and then calculate the group-based tripeptide composition. Regarding network based encoding schemes, they designed NetTP to systematically characterize the host proteins' network topology properties and designed NetSS to reflect the molecular mimicry strategy used by pathogen proteins. Finally, through the noisy-OR algorithm, 5 individual models were integrated into a final powerful model with an AUC value of 0.922 in the 5-fold cross-validation, as well as 0.924 on independent testing and could achieve a better performance than two state-of-the-art human bacteria PPI predictors. It must be noted that among network based encoding models, no model exceeded an AUC of 0.82, while among sequence based encoding models, none went beyond 0.88.

We will use sequence based deep learning models to exceed this with the help of neural machine translation. Deep learning is part of a family of machine learning methods based on artificial neural networks that mimics the workings of the human brain in processing data for applications such as speech recognition and translation, decision making, object detection, etc. One important use of this would be to find patterns in sequential or temporal data which is what we will use to examine amino acid sequences of protein pairs that interact as well as those that do not, and use these methods to predict interactions with accuracy that exceeds all 5 previously mentioned model's individual performance while not using any physico-chemical or biological properties. The implementation can be found in my GitHub page.

II. Methodology

The input data, both training and testing is given in the format of:

$$Human.P.id \qquad Yersinia.P.id \qquad +1/-1$$

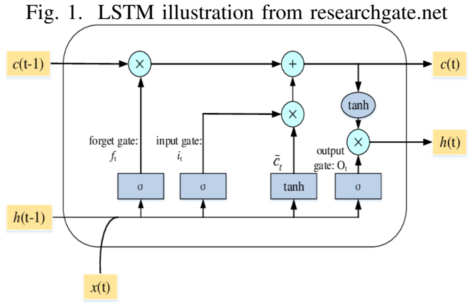
where +1 denotes positive interaction and -1, negative interaction. There are 6270 datapoints in the training set and 1514 datapoints in the testing set. The test/train split is stratified, which is to say both have a True False ratio of 1:1. Since we are looking for amino acid sequences, web scraping is used to look up the uniprot website[2] by their respective protein id to get a dataset that looks like this:

$$[M, Y, V, T, M...R] \quad [M, T, G, P, Q...A]$$

Keep in mind that each protein peptide chain here is at least 35 units long and consists of 20 types of amino acids that are represented by the letters of the alphabet. From here NLP methods are used to train the model and testing accuracy is measured using the AUC - ROC curve available from the sklearn package in python. The results for each model shown are based on AUC score by testing them with the independent dataset.

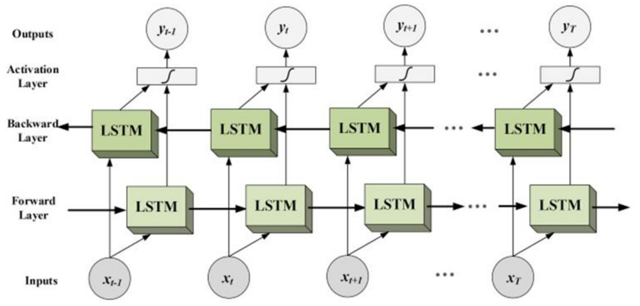
The first method that will be used is called Recurrent Neural Network (RNN). Based on a slight modification of the simple feed forward architecture, they are a class of neural networks that allow previous outputs to be used as inputs while having hidden states, that makes it ideal for prediction of temporal data. At a high level, it remembers the past and makes predictions based on what it learnt from it. The Long short-term memory (LSTM) is based on the RNN architecture uses a combination of various gates to retain context during predictions which makes it more powerful than standard RNN or its modified counterpart, the GRU. However, this may

Fig. 1. LSTM illustration from researchgate.net



not be enough since we are dealing with sequences that are thousands of units long. An LSTM would give more priority to "words" that are towards the end during its decision calculus while it is entirely possible that amino acid sequences at the beginning are just as important in determining PPI. To handle this, we employ a modification of the LSTM called Bidirectional LSTM which essentially creates an identical LSTM configuration to examine sequences from the opposite end. The additional hidden layer weights created are simply concatenated to the original. A GlobalMaxPool1D layer is used to get the highest among the concatenated hidden layers, although in some cases, a Dense layer has been added in between.

Fig. 2. Bidirectional LSTM



The encoding or "words" we feed to the NLP model as input will vary each time. Assuming the network takes each amino acid sequence from Human and Yersinia as input, we could give a simple first degree sequence input as shown earlier, or we could convert it into second degree (2D) as shown below and feed it to the network:

$$[MY, YV, VT, ...] \quad [MT, TG, GP, ...]$$

or we could convert it into a third degree (3D) input:

$$[MYV, YVT, VTM, ...] \quad [MTG, TGP, GPQ, ...]$$

This is done so that we account for every two or three sequences of amino acids could play an important role in determining interaction and is represented as "words" during neural machine translation. If we assume that these common sequence bits from both Human and Yersinia play a role in prediction of interaction, we could concatenate them and give a single third degree input as:

$$[MYV, YVT, VTM, ..., MTG, TGP, GPQ, ...]$$

The next phase is to find a way for the neural network to understand these "words". Usually categorical data is processed by one hot encoding them. For language, however, since there are several hundred thousand words and every sentence usually uses a word once, we tend to get an extremely sparse matrix making it computationally expensive. So, encoding is used where each word is assigned an index value to an embedding matrix, and its actual value in the neural network is the embedding vector whose position with respect to other words can be understood in n-dimensional space where n is the embedding dimension. This is exactly what we will use here.

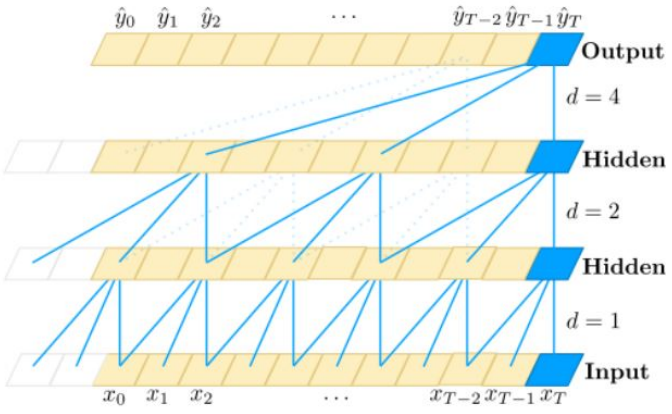
For example, considering the 3D join input shown above, it is first converted into a 'sentence' or a sequence of 'words' as follows:

$$[MYV\ YVT\ VTM\ ...MTG\ TGP\ GPQ\ ...]$$

The number assigned to each 'word' will be decided by the tokenizer which is then used to convert the data (a list of 'sentences') to a purely integer sequence matrix of width equal to a pre designated maximum sequence length. By default, in NLP, 'sentences' longer than this maximum length are pre-truncated and those lesser than the maximum length are pre-padded, which is to say, zeros are added on the left side. Also, when you create a tokenizer, it may be important to specify a maximum vocabulary size above 3D input data since the combinations of amino acids or 'words' go up polynomially each time D increases by 1. The tokenizer in that case will include words with the highest frequency. Now that the input data is uniform, each word in a sentence is expressed as a vector in n-dimensional space with the help of an embedding layer. The position of these vectors change based on the training of the neural network during back propagation.

The next tool employed will be a very versatile one: the convolution neural network, where feature extraction is done using several pairs of Convolution layers or Kernels, and pooling filters. The convolution operation extracts high-level features such as edges, from the input image after which the pooling layer reduces its spatial size to decrease the computational power required to process the data through dimensionality reduction. It also helps us find dominant features. CNN is one of the most widely used Deep Learning methods in computer vision. It has also been found to be useful in sequential data analysis and is called "Temporal Convolutional Network" (TCN) when used in this capacity.

Fig. 3. TCN illustration from Medium.com


$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ ... $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 ... x_{T-2} x_{T-1} x_T

According to Ceshine Lee[3], the distinguishing characteristics of TCNs are: 1) the convolutions in the architecture are causal, meaning that there is no information "leakage" from future to past; 2) the architecture can take a sequence of any length and map it to an output sequence of the same length, just as with an RNN. The most important component of TCNs is dilated causal convolution. "Causal" simply means a filter at time step t can only see inputs that are no later than t. A residual block stacks two dilated causal convolution layers together, and the results from the final convolution are added back to the inputs to obtain the outputs of the block. What TCNs do is simply stacking a number of residual blocks together to get the receptive field that we desire. If the receptive field is larger or equal to the maximum length of any sequences, the results of a TCN will be semantically equivalent to the results of a RNN. In fact, in some cases, we will prove than TCN is more powerful and efficient than the RNN.

Other than replacing the Bidirectional LSTM with CNN, the architecture is still largely similar and the approach to solve the problem remains identical as explained earlier.

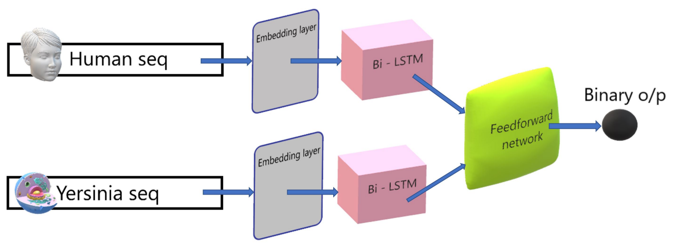
III. Platform used

The machine learning models are created and trained in TensorFlow using Keras. TensorFlow is an open-source machine learning environment designed and developed by Google. It is very popular and supports libraries that can allow software to run without changes on regular CPU. Its popularity should not come as a surprise since it offers good computational graph visualizations, a varied library backed by Google, good scalability, pipelining and performance. But by far the greatest asset in its arsenal regarding designing deep neural networks is Keras. It is a high-level framework that hides the backend computation and allows us to quickly build a neural network model. If we tried to implement the same functionality in another framework like Pytorch, much of our time would be consumed in writing the tiniest details of code. Keras, on the other hand, offers profound levels of abstraction and encapsulation which allows the user to focus almost entirely on neural net design and data preprocessing in exchange for computational efficiency and architecture customization. Implementation of the code is carried out using Google Colab Pro which offers high speed 25 GB RAM and Tesla P100-PCIE-16GB GPU and is strongly recommended for anyone looking to train these models.

IV. Bi-LSTM

We start with the most common method used to solve this problem, which is, send each Human and Yersinia sequence to separate embedding matrices and then connect to a Bidirectional LSTM to predict interaction. For this paper, we will call this the *doubleip* configuration.

Fig. 4. The 'doubleip' configuration



The most obvious observation while using *doubleip* is that accuracy increases with increase in complexity (size) of the

TABLE I
AUC SCORES FOR *doubleip* CONFIGURATION

D1	D2	D3	D4	D5	D6
0.812	0.835	0.870	0.891	0.890	0.903

words. However, keep in mind that there are some peptide chains with just 35 amino acids which will not be represented adequately as D increases because of the vocabulary cap imposed by the tokenizer which makes the chances of 'words' from smaller peptide chains to make the cut go down drastically. In 5D input for example, the total possible combinations of amino acids are $20^5 = 3,200,000$, and even if just half of that are in the training data, assuming the vocabulary cap to be 500,000 many of them would be left out causing several peptide chains to be single digits long, while in 6D, several sequences are not represented at all.

The other method as discussed earlier will be to concatenate the two sequences and evaluate them through a single embedding matrix and Bi-LSTM. For the purpose of this paper, we will call this the 'join' configuration.

Fig. 5. The 'join' configuration

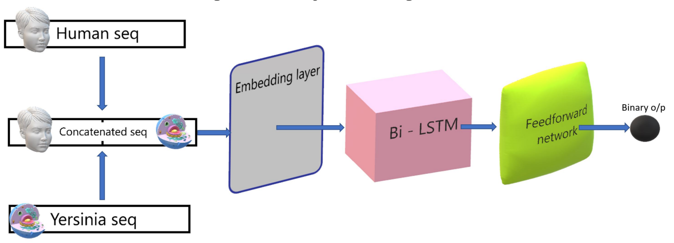


TABLE II
AUC SCORES FOR *join* CONFIGURATION

D1	D2	D3	D4	D5	D6
0.766	0.784	0.820	0.847	0.867	0.849

This is one of the methods used in NLP for sentiment analysis. Just like earlier, we observe that accuracy increases with word size but the results are not as good as for *doubleip* configuration.

Now, we create an end to end configuration that combines the two discussed above so that the neural network can account for the separate nature of both Human and Yersinia (doubleip) as well as account for the possibility that common sequence bits or "words" from both species play a role in prediction of interaction (join). Let us call this *combine* configuration.

From the results it becomes obvious that our theory was correct. *Combine* configuration seems to give us the best results.

Fig. 6. doubleip-join comparison

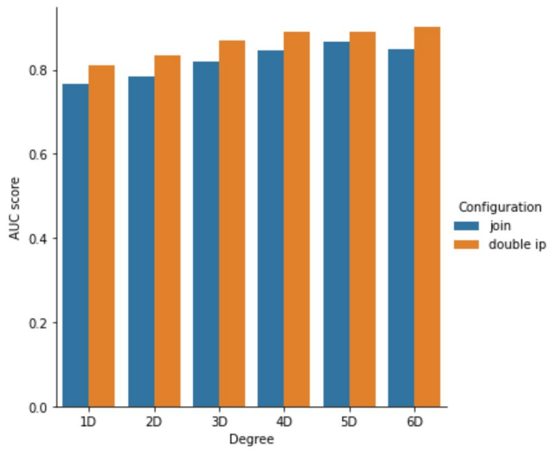


Fig. 7. The 'combine' configuration

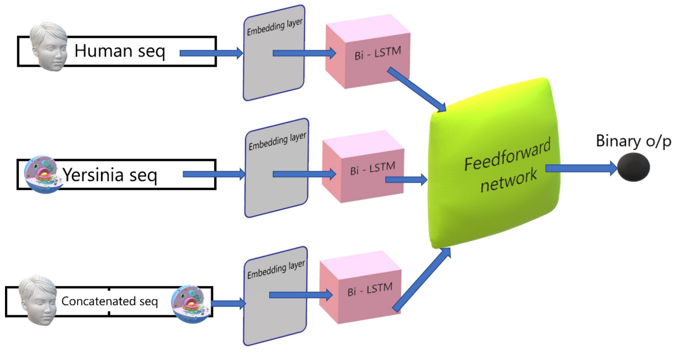
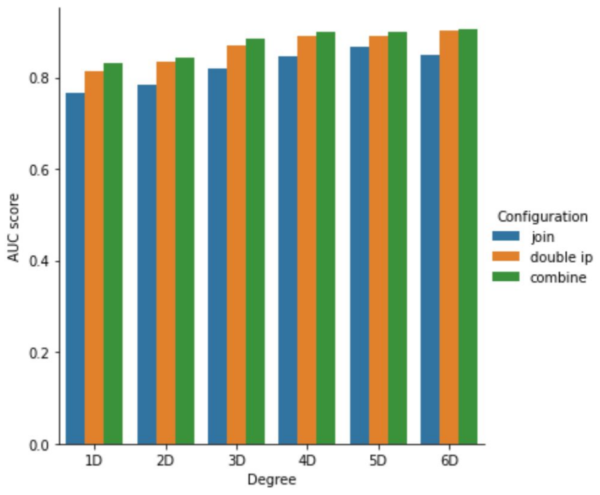


TABLE III
AUC SCORES FOR *combine* CONFIGURATION

D1	D2	D3	D4	D5	D6
0.830	0.843	0.884	0.899	0.900	0.906

Fig. 8. doubleip-join-combine comparison



V. CNN

The convolutional neural network, or to be more specific, the Temporal Convolutional Network is often not the standard approach since it does not appear 'instinctive'. However, overlooking them can be a serious mistake as shown in this case. The configuration used here has 32 filters, 3 kernels followed by Max pooling. We ignore 6D inputs due to reasons mentioned earlier as well as the fact that the gains appear negligible from there, probably because of those very reasons.

Fig. 9. The 'combine' configuration using CNN

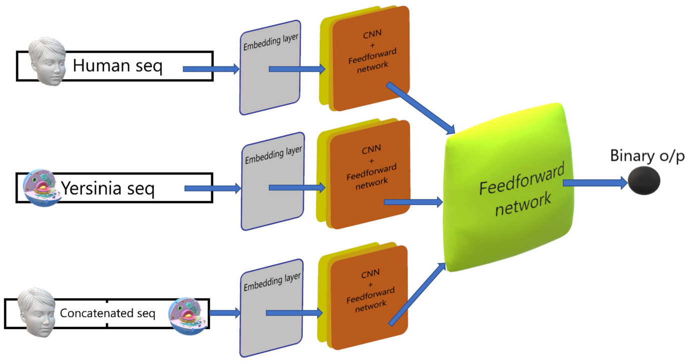


TABLE IV
AUC SCORES WHILE USING CNN

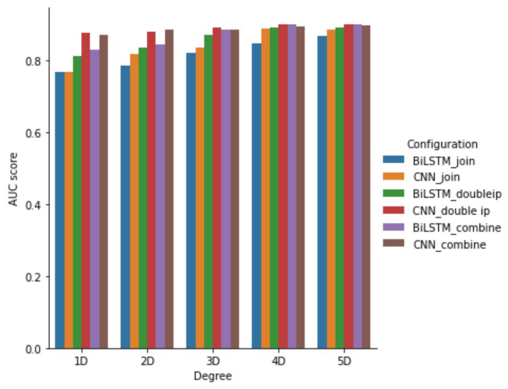
Config.	D1	D2	D3	D4	D5
join	0.766	0.818	0.833	0.886	0.883
doubleip	0.876	0.880	0.889	0.898	0.899
combine	0.870	0.884	0.883	0.894	0.897

We observe that using CNN instead of Bi-LSTM gives us higher accuracy at lower degrees and as D increases it plateaus to approximations of those given by the Bi-LSTM. The *doubleip* configuration seems to give significant gains from lower D inputs compared to Bi-LSTMs, so much so in fact, that they are comparable to 5D Bi-LSTM results. To understand the significance of this, keep in mind that 1D CNN models train in about 1% of the time taken to train a 5D Bi-LSTM model. In fact, it has been observed that CNN based models overall take significantly less time to train than their counterparts which plays an important role in aspects that can test and shape them such as hyper parameter tuning. Another interesting observation is that unlike for Bi-LSTMs, the *combine* configuration seems to give little to no improvement on CNN based models.

VI. Advanced Preprocessing

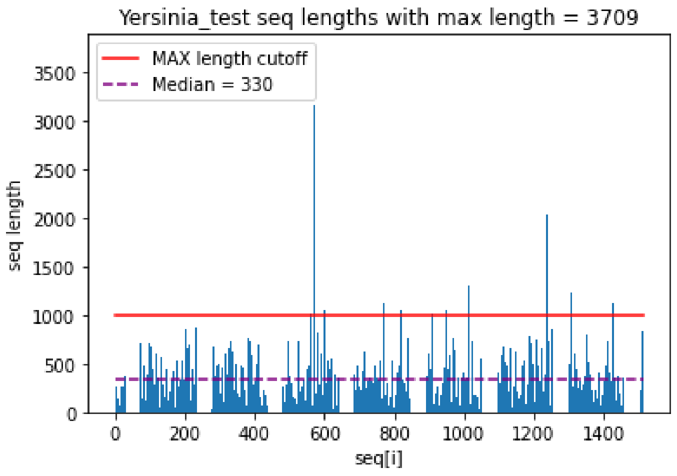
So far we have been able to apply simple methods in neural machine translation to exceed every individual model in the reference paper. However, everything we have used till now applies default NLP procedures, which as mentioned earlier, pre-truncates and pre-pads sequences to a MAX length so that

Fig. 10. BiLSTM vs CNN



they can be processed by the neural network. During pre-truncation, words (or rather, their index numbers given by the tokenizer) from the left are cut off to fit MAX length. The problem in this case is that amino acid lengths in each peptide chain vary from 35 to nearly 9000 which put us in a quandary when setting MAX length. If it is too low, you miss out on too many 'words' reducing the accuracy of the model. The next obvious move would be to make it as high as possible to include the maximum number of words. But this causes drastic reduction in accuracy as well. To understand this, consider the sequence length distribution shown below.

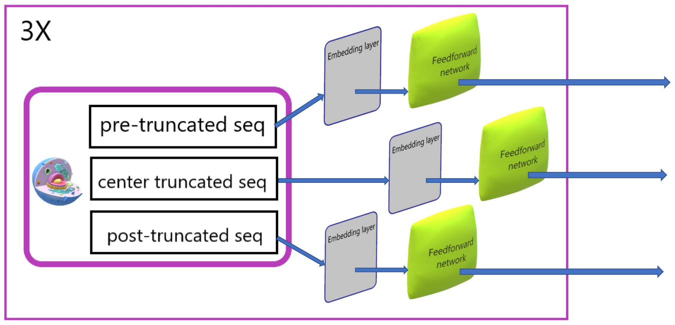
Fig. 11. Sequence length distribution in Yersinia test data



Here, the MAX length covers most datapoints which means most words are included. However, you also see a lot of empty space below that red line. When this is translated to a sequence matrix, we end up with a lot of zeros due to padding, in other words we have a sparse matrix. Both CNN and Bi-LSTMS do not work well in this scenario. Till now, we have selected MAX length as close to the median as possible to give us best results, at the cost of losing significant data just because they happen to be on the left side. That is about to change.

Fig.12 shows us a '3X' preprocessing configuration. Input sequences are pre truncated as well as post truncated in order

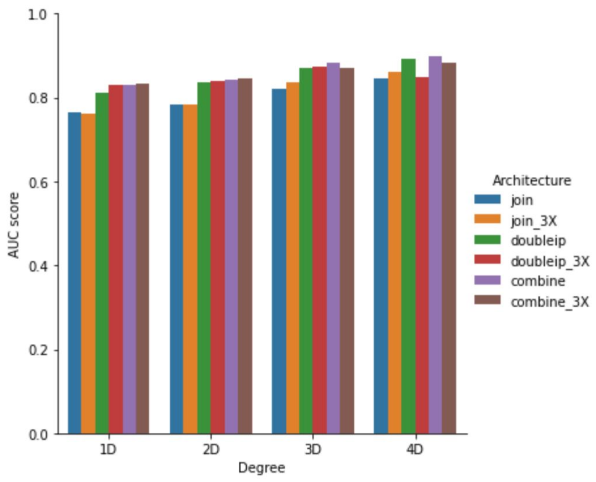
Fig. 12. 3X preprocessing for Yersinia sequences



to get the words from both sides. We also add center truncation to get as many words in the middle. These three inputs are put through separate embedding matrices which are connected in an end to end configuration so as to let the neural network decide their importance.

We start by testing this in Bi-LSTMs to get the following results: It can be observed that 3X preprocessing offers little

Fig. 13. 3X preprocessing using Bi-LSTMs



to no improvement. So we do not proceed any further and instead try this for CNN models. The results are as follows:

TABLE V
AUC SCORES WHILE USING CNN WITH 3X PREPROCESSING

Config.	D1	D2	D3	D4	D5
join	0.824	0.838	0.855	0.881	0.897
doubleip	0.891	0.899	0.900	0.901	0.905
combine	0.874	0.889	0.903	0.906	0.913

We see that the impact of 3X on CNN is significant when compared to default preprocessing. This shows us that CNN with *doubleip* and *combine* configurations are very effective.

This method has brought us closer to the final AUC score of the integrated model in the reference paper. Unfortunately, from this point on wards, we will have to deal with the principle of diminishing returns. That means we have now hit the later end of the curve and any major changes to the model can only give us marginal improvements in our score.

Fig. 14. 3X preprocessing using CNN

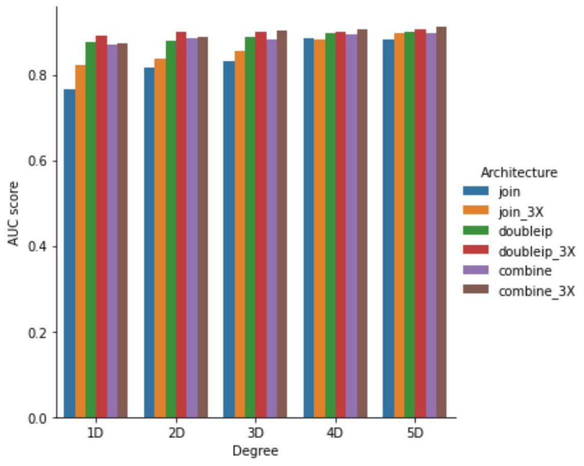
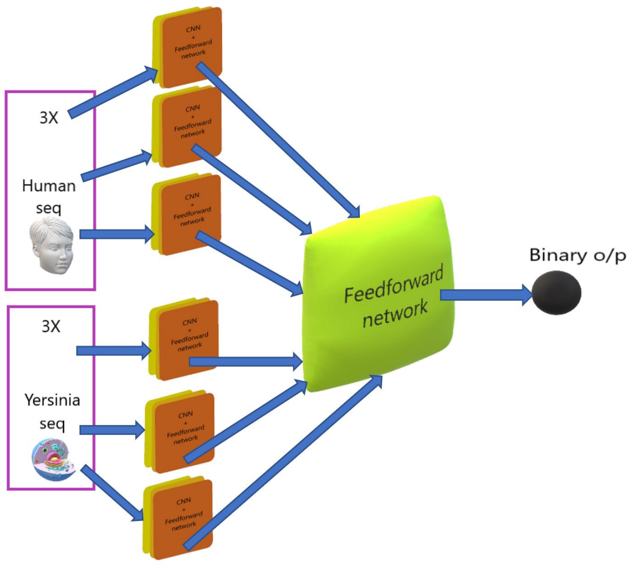


Fig. 15. 3X *doubleip* configuration using CNN



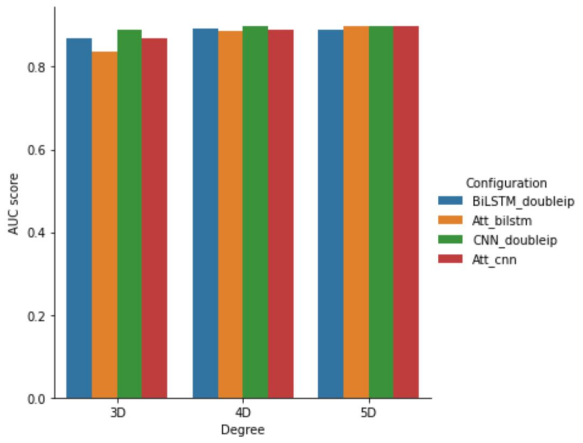
VII. Other Methods

There are several tools available in neural machine translation that have been developed in the last few years. We will apply two of them to our problem.

A. Attention

This was introduced to improve seq2seq performance in NLP which till then had a standard encoder-decoder configuration, where the encoder would go through the input sequence and send it as a hidden state vector to the decoder which uses it to create another sequence word by word. This decreases accuracy when the input sequence is too long because a fixed size hidden state vector may not successfully summarize it because RNN prioritizes words towards the end. In 2014 a paper was written[4] that employs what we call today the attention layer. In this model the every hidden layer created by the Bi-LSTM in the encoder is considered to create a context vector that determines the next word in the decoder. For our implementation, we use code available from the Keras API[5] that accepts values sequences and query sequences which are encoded using a CNN or Bi-LSTM before sending it to the attention layer.

Fig. 16. Attention using CNN and Bi-LSTM with *doubleip* configuration

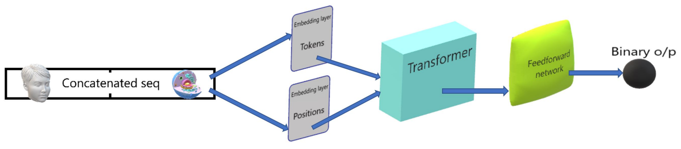


Attention does not seem to make an improvement at lower degrees but starts to prove itself as D increases. However, it does not seem to exceed the models used earlier. Using 3X inputs for attention also does not seem to make any difference.

B. Transformers

In Jun 2017,in a paper called *Attention Is All You Need*[6], the authors introduced a new architecture called Transformers which has an solely attention-based encoder-decoder configuration where the encoder turns a sequence a continuous representation while the decoder uses it to generate a word step by step. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. For our implementation, we use code available from the Keras API[7] that implements a Transformer block as a layer and uses two separate embedding layers, one for tokens, one for token index (positions) and analyzes a sequence for sentiment analysis. So naturally this will be used in the *join* configuration. In this approach, as we mentioned earlier,

Fig. 17. Transformers using *join* configuration



common amino acid sequences in both species are processed in the same n-dimensional space. One way to get around this and get an approximation of *doubleip* results would be using something I call *differential join*. Here, each amino acid is represented by a different letter for Yersinia sequences. Assume a Human and Yersinia sequence pair:

$$[M, M, L, G, T...] \quad [M, V, M, K, K...]$$

We can change Yersinia sequence letters:

$$[M, M, L, G, T...] \quad [\tilde{a}, \grave{\imath}, \tilde{a}, \acute{a}, \acute{a}...]$$

Now we can create words based on the degree, for instance, when we turn it into 2D input,

$$[MM, ML, LG, GT, ...] \quad [\tilde{a}\grave{\imath}, \grave{\imath}\tilde{a}, \tilde{a}\acute{a}, \acute{a}\acute{a}, ...]$$

and concatenate it to make it suitable for Transformer's *join* configuration,

$$[MM, ML, LG, GT, ...\tilde{a}\grave{\imath}, \grave{\imath}\tilde{a}, \tilde{a}\acute{a}, \acute{a}\acute{a}, ...]$$

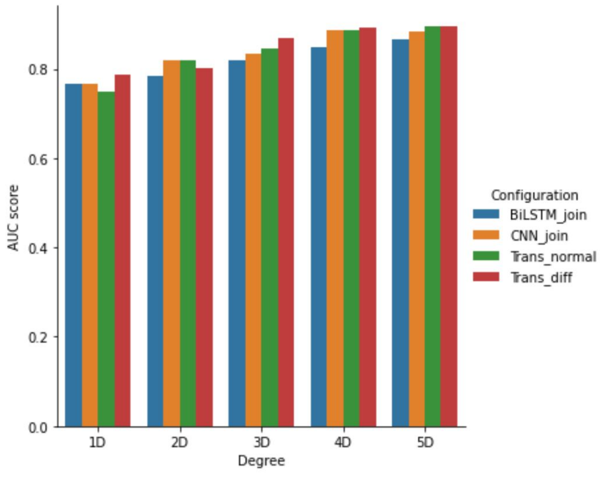
This way, every word that represents each species in the embedding matrix can never be the same.

The AUC scores for Transformers using default inputs (pre truncated) are shown below:

TABLE VI
AUC SCORES FOR TRANSFORMERS

Config.	D1	D2	D3	D4	D5
normal join	0.750	0.819	0.846	0.887	0.896
diff join	0.786	0.801	0.869	0.892	0.895

Fig. 18. Transformers vs Bi-LSTM vs CNN



It looks like Transformers give us the best performance for *join* configuration. We will be using this property next. The *differential join* inputs give us better results until we get to higher degrees.

VIII. Final Model

To get the best AUC score we will combine the best methods examined above. In short, we will use a *combine* configuration where CNN will use 3X *doubleip* along with 3X *normal join* using Transformers. We do not use *differential join* because its functionality is carried out by the CNN *doubleip* configuration which has proved to be of high performance. As mentioned earlier, *combine* architecture needs to account for the separate nature of both Human and Yersinia, as well as account for the possibility that common sequence bits or "words" from both species play a role in prediction of interaction which is where *normal join* using Transformers comes into play. The degree of the input sequences will be 5D since that gives us the best results while representing every input sequence, even if it is by single digits. This is in spite of the fact that MAX vocabulary size for *doubleip* is at 500,000 while for *join* is 1,000,000. The width of the input matrix, which is the MAX sequence length, is kept at 1000 for *doubleip* and 2000 for *join*.

Implementation of this is available as the final model at GitHub and the pseudo-code below should give an idea regarding what happens:

1. Verification of input: Dataframe with Human and Yersinia sequence each in a list format as well as a concatenated list of Human and Yersinia sequence called Joined.

2. Convert each of these three sequences to 5D format using the combine_AC() function.

3. If this is new training data:

- convert the list of 5D sequences to a single sentence containing a string of words. Do this for every data point.
- for Human and Yersinia sentences, create separate tokenizers with a maximum vocabulary size of 500000, have them create index numbers of words for each species by frequency of occurrence, and save them (the tokenizers).
- for Joined sentences, create one tokenizer with a maximum vocabulary size of 1000000, have it create index numbers of words for all sentences in Joined by frequency of occurrence, and save it.

4. Convert the list of 5D sequences of Human, Yersinia and Joined to a single sentence containing a string of words. Do this for every datapoint.

5. Load the tokenizer for Joined sequences and apply it to Joined sentences to create three integer input matrices of width 2000 that are pre padded and pre truncated, post padded and post truncated, and center truncated.

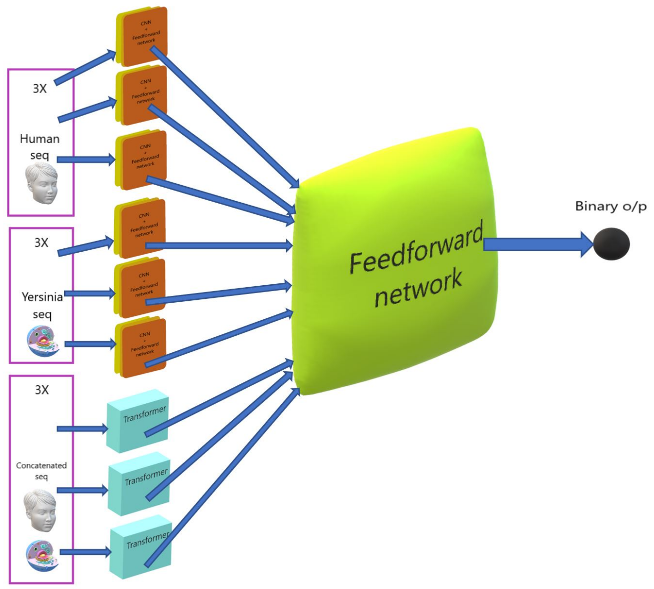
6. Load the tokenizer for Human sequences and apply it to Human sentences to create three integer input matrices of width 1000 that are pre padded and pre truncated, post padded and post truncated, and center truncated. Do the same for Yersinia sequences using its tokenizer.

7. Send the 6 matrices created by Human and Yersinia sequences to separate Embedding-CNN layers and the 3 matrices created by Joined sequences to separate dual embedding-Transformer layers as shown in Fig 19.

8. Concatenate the outputs of all of them to a single dense layer before sending it to a single output neuron with a sigmoid activation.

The drop rates for CNN are kept at 50% while for the neural network in the transformers with two attention heads are at 90%. Despite that, we have over fitting issues once we go beyond a single epoch. The embedding dimensionality at 25 seems to give the best results which is surprising because for NLP processing with English language, usually it is kept at around 300 despite having much lesser MAX vocabulary size. The Adam optimizer seems to give best results although for Bi-LSTM models, its learning rate had to be increased. The batch size is 32 to give some generalization but had to be lowered to 16 during cross validation because of decrease in training data.

Fig. 19. Final Model

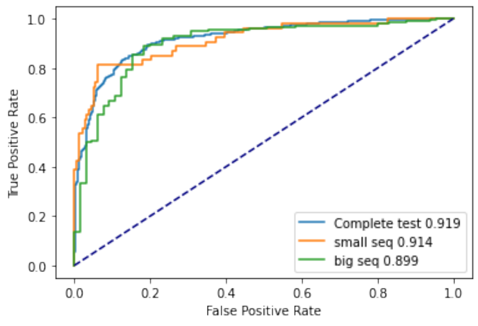


IX. Results

A. Human-Yersinia PPI

This integrated end to end neural network gives us the best performance so far. We now have an AUC score on the independent test data set of 0.919, as well as a five-fold cross validation score of 0.910 on the training set. The impact of 3X inputs have also been validated. In the test data, isolating 233 smallest sequences gave an AUC score of 0.914 while 280 longest sequences gave 0.899. Interestingly, looking at Fig.20, small sequences give us better thresholds with more 'pure' true positives while big sequences give us higher and more 'impure' true positives.

Fig. 20. ROC curve comparison



B. Human-Virus PPI

All the work done so far is pointless if this process cannot be applied to other scenarios. So, we now test the versatility of the model by trying to solve a different problem. In December 2019, a paper[8] was submitted by faculty members from Universities of China and Miami, that predicted human-virus PPI through a sequence embedding-based machine learning method (doc2vec) to represent protein sequences, a method similar to our final model. A Random Forest (RF) classifier was also trained and included that obtained excellent accuracy outperforming various combinations of machine learning algorithms and commonly-used sequence encoding schemes. A high AUC score of 0.954 was obtained through this method. Our final model was applied to this problem, without any examination of data or hyper parameter tuning, and gave an AUC score of 0.938.

The source paper gave us three train-test pairs with over 50,000 common data points across all tests and trains. So all of them were concatenated, duplicates removed, randomized, and split into a single train-test pair in a 80-20 ratio. Even though web scraping did not get the sequences of all proteins, we now have a dataset that is half a million rows long and this was the biggest challenge to execute. The training data had to be divided into 50 subsections and saved, the model would then be trained by loading each one of the subsection into the RAM one by one. The *doubleip* embedding dimensionality was increased to 50 to account for greater number of words in the vocabulary that was inevitable considering the large dataset. Creation of tokenizers was also a challenge since 25 GB RAM could only hold half the training data while executing *create-tokenizers* function at the same time. To overcome this challenge, the model was applied using the train-test groups as per source data and gave an average AUC of above 0.939. It must be noted that for any PPI usage of this model, the column names used must be [Human, Yersinia, Label, Joined] as shown in the GitHub page.

X. Comparison to other sequence based encoding methods

A. Conjoint Triad (CT)

Based on the physicochemical properties of their side chains, 20 amino acids are clustered into seven groups, replacing each amino acid in a protein sequence with the corresponding group number, the frequency of each conjoint triad in the protein sequence is determined through a sliding window. As a consequence, a protein pair is finally represented by a 686-dimensional vector [9]. Our model on the other hand, uses every 5 combinations of amino acids as words (5D) to be represented as an input matrix.

B. Local Descriptor (LD)

Similar to CT encoding, the seven groups of amino acids are also used in LD. A protein sequence is divided into ten local regions to further extract features of each subregion, mainly reflecting local characteristics of the underlying protein. Each region is represented by three features that reflect the characteristics of seven amino acid groups. The three features are Composition (C), Transition (T), and Distribution (D), where C represents the composition of each amino acid group, T reflects the composition of any two amino acid groups, and D represents the distribution of the first, 25%, 50%, 75%, and 100% of the total number of amino acids.[10] Our model uses none of this or any knowledge in bio-chemistry.

C. Auto Covariance (AC)

AC encoding accounts for correlations and interactions between different position sequences, and uses seven residue physicochemical properties to represent the protein. They take into account the distance between different sequences in the chain.[11] Just like before, our model uses no knowledge in bio-chemistry, but it does in a way consider the distance between sequences because each sequence is represented by a word and we use NLP techniques, which means their order is important.

D. Doc2vec+RF

This is similar to the final model implemented here. Used in the Human-Virus PPI prediction paper described earlier[8], it uses 4D and 5D words to train the embedding matrix. But this was done using distributed-memory (DM) model instead of CNN+Transformers and the words were non-overlapping unlike the ones used here which were overlapping words. Moreover, The model was trained using Random Forest unlike ours which was trained using end to end deep learning. Also, from what I have read, they employed the standard 'doubleip' configuration instead of 'combine' configuration which uses another embedding matrix to represent sequences of both species due to reasons described previously. Moreover, I do not think they accounted for every part of the sequence like it was done here using '3X' configuration.

XI. Conclusion

The results compare favourably with the first paper[1] that has a model which gives independent test and cross validation scores of 0.924 and 0.922 respectively after processing both sequential and network data unlike our model that only takes in sequential data. The results vis-à-vis the second paper[8] is also impressive considering the fact that no analysis of any sort has been carried out. Moreover, as mentioned previously, no knowledge in Biology has been used during the development of our model. The fact that similar results can be obtained with a fraction of the manpower and no funding should be a testament to deep learning. That being said, there are a few shortcomings here. Considering that it has nearly 100 million parameters, the model is about 1GB in size. The entire preprocessing algorithm is not optimized and so takes some time, also during which, if the number of datapoints are above 50,000, you risk crashing the system since the RAM goes above 20 GB which forces the user to create tokenizers separately, split the data for preprocessing and concatenate them in the end or fit the model in batches of training data as it was done for the Human-Virus PPI problem. Then there is the fact that this has been developed using high end Google GPUs which means training the model on an average computer could take upto an hour for the Human-Yersinia case. It would be inconceivable to train the model for Human-Virus PPI in an ordinary computer since it took 8 hours by Google GPUs considering the massive size of our dataset and the fact the parts of it had to be loaded and removed from the RAM frequently. Then there was a problem that whenever resources allotted by the virtual machine were exceeded, it would reset. So often we would have to save after a cell execution and load it when the VM reconnected. Another issue with the Human-Virus PPI problem was that only half the training data could be analysed (once again, due to RAM limitations) to create tokenizers which means it could not get the most frequently used 'words'. But perhaps the biggest issue that impacted performance was the loss of around 50,000 datapoints during web scraping. The model also has more scope for improvement since there were issues during hyper parameter tuning because Keras would not automate for AUC metric leaving manual grid optimization as the only choice (which was not carried out for Human-Virus PPI given the previously mentioned time constraints). Coupling this with better network architecture could take the score to above those obtained by the two reference papers.

XII. Acknowledgement

I would like to thank Prof. Chinwe Pamela Ekenna Ph.D., Assistant Professor, Computer Science, University at Albany for her encouragement and support in this endeavour and giving me the opportunity to work on this fascinating project. I would also like to dedicate this work to my friends and family, both in the US and at home, without whom none of this would be possible.

References

[1] Xianyi Lian, Shiping Yang, Hong Li, Chen Fu, and Ziding Zhang, "Machine-Learning-Based Predictor of Human–Bacteria Protein–Protein Interactions by Incorporating Comprehensive Host-Network Properties" Journal of Proteome Research 2019 18 (5), 2195-2205
[2] Uniprot's Basic Local Alignment Search Tool (BLAST)
[3] Ceshine Lee Medium.com [Tensorflow] Implementing Temporal Convolutional Networks.
[4] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio CU Neural Machine Translation by Jointly Learning to Align and Translate
[5] Attention layer Keras Dot-product attention layer, a.k.a. Luong-style attention
[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin CU Attention Is All You Need
[7] Text classification with Transformer Keras Implement a Transformer block as a Keras layer and use it for text classification
[8] Xiaodi Yang, Shiping Yang, Qinmengge Li, Stefan Wuchty and Ziding Zhang sciencedirect Prediction of human-virus protein-protein interactions through a sequence embedding-based machine learning method
[9] Sun T, Zhou B, Lai L, Pei J. Sequence-based prediction of protein protein interaction using a deep-learning algorithm. BMC Bioinf 2017;18:277.
[10] Davies MN, Secker A, Freitas AA, Clark E, Timmis J, Flower DR. Optimizing amino acid groupings for GPCR classification. Bioinformatics 2008;24:1980–6.
[11] Yang KK, Wu Z, Bedbrook CN, Arnold FH. Learned protein embeddings for machine learning. Bioinformatics 2018;34:2642–8