Predicting Human–pathogen Protein–protein Interactions Using Natural Language Processing Methods

Nikhil Mathews
University at Albany, SUNY, NY

Tuan Tran
University at Albany, SUNY, NY

Banafsheh Rekabdar
Portland State University, rekabdar@pdx.edu

Chinwe Ekenna
University at Albany, SUNY, NY

Citation Details

Mathews, N., Tran, T., Rekabdar, B., & Ekenna, C. (2021). Predicting human–pathogen protein–protein interactions using Natural Language Processing methods. Informatics in Medicine Unlocked, 26, 100738.

Contents lists available at ScienceDirect

Informatics in Medicine Unlocked

journal homepage: www.elsevier.com/locate/imu

Predicting human–pathogen protein–protein interactions using Natural Language Processing methods

Nikhil Mathews [a], Tuan Tran [a,*], Banafsheh Rekabdar [b], Chinwe Ekenna [a]

[a] Department of Computer Science, University at Albany, SUNY, NY 12206, USA
[b] Portland State University, USA

ARTICLE INFO

Keywords:
Natural Language Processing
protein–protein interactions
Deep learning
Neural networks

ABSTRACT

In this paper, we predict the interaction of proteins between Humans and Yersinia pestis via amino acid sequences. We utilize multiple Natural Language Processing (NLP) methods available in deep learning in a unique format and produce promising results. Our developed model gives a cross-validation AUC score of 0.92 and is comparable with other work that utilizes extensive biochemical properties i.e, network and sequence in conjunction. We achieve this 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. We show that our proposed approach is robust to different protein–protein interactions between host and pathogen data.

1. Introduction

The unseen battle between humans and pathogens have been fought since the dawn of time, much of which 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.

In 2019, Lian et al. in [1] developed a new machine-learning-based predictor of human-Yersinia pestis PPIs. Three conventional sequence-based encoding schemes (NetSS) and two host network-property-related encoding schemes (NetTP) were introduced. The individual predictive models for each encoding scheme were inferred by Random Forest. The first sequence encoding scheme, Auto Covariance (AC), employed seven physio-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, the composition of k-spaced amino acid pairs (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 mimic 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 physio-chemical or biological properties.

* Corresponding author.
E-mail addresses: nnikhiltittymathews@albany.edu (N. Mathews), ttran3@albany.edu (T. Tran), rekabdar@pdx.edu (B. Rekabdar), cekenna@albany.edu (C. Ekenna).

https://doi.org/10.1016/j.imu.2021.100738
Received 6 June 2021; Received in revised form 8 September 2021; Accepted 15 September 2021
Available online 25 September 2021

2. Related work

The use of machine learning began in earnest after the explosion of computational power in the last two decades. Consequently, its use in biological sequence predictions is relatively recent.

Ahmed et al. [2] extracted triplet and quadruplet features to train 4 different models in total that used Support Vector Machines as well as neural networks. The best one was chosen for prediction. In the neural network, 4 layers and a varying number of nodes in the input and hidden layers were used. This setup along with 60,000 data points to train and test gave an Area under the curve analysis of 0.92. Another one is by Li [3] which is similar to this work and both inputs go through encoding and embedding, a convolution filter-pooling layer pair for feature extraction, and a Long short-term memory (LSTM) before they are joined by a dense layer to binary output. Having a dataset of several hundred thousand gave it an accuracy of up to 98%.

Similar to our model, Tsukiyama et al. [4] first developed the LSTM model (an artificial recurrent neural network architecture) with word2vec to predict PPIs between humans and viruses, named LSTM-PHV, by using amino acid sequences alone. The LSTM-PHV effectively learned the training data with a highly imbalanced ratio of positive to negative samples and achieved an AUC of 0.976 with an accuracy of 98.4% using 5-fold cross-validation. In predicting PPIs between human and unknown or new viruses, the LSTM-PHV presented higher performance than the existing predictors when they were trained by multiple host protein-including datasets. Interestingly, using only amino acid sequence contexts as “words” presented remarkably high performances. The use of uniform manifold approximation and projection demonstrated that the LSTM-PHV clearly distinguished the positive PPI samples from the negative ones. The structure of the model here is similar to what is described in our paper as ‘4D Bi-LSTM doubleip’.

Zhou et al. [5] used knowledge bases (KBs) that contain huge amounts of structured information of protein entities and their relations, which can be encoded in entity and relation embeddings to help PPI extraction. However, the prior knowledge of protein–protein pairs must be selectively used so that it is suitable for different contexts. Specifically, the work proposes a Knowledge Selection Model (KSM) to fuse the selected prior knowledge and context information for PPI extraction. Firstly, two Transformers encode the context sequence of a protein pair according to each protein embedding, respectively. Then, the two outputs are fed to mutual attention to capture the important context features of the protein pair. Next, the context features are used to distill the relation embedding by a knowledge selector. Finally, the selected relation embedding and the context features are concatenated for PPI extraction. Experiments on the BioCreative PPI dataset show that KSM achieves a new state-of-the-art performance (38.08% F1-score) by adding knowledge selection.

By including another important aspect of PPI, Tsubaki et al. [6] predicted the compound–protein interactions (CPIs) where data are provided as discrete symbolic data, i.e. compounds are represented as graphs where the vertices are atoms, the edges are chemical bonds, and proteins are sequences in which the characters are amino acids. In this study, they investigated the use of end-to-end representation learning for compounds and proteins, integrate the representations, and develop a new CPI prediction approach by combining a graph neural network (GNN) for compounds and a convolutional neural network (CNN) for proteins. They demonstrated that the proposed end-to-end approach achieves competitive or higher performance as compared to various existing CPI prediction methods. In addition, the proposed approach significantly outperformed existing methods on an unbalanced dataset. This suggests that data-driven representations of compounds and proteins obtained by end-to-end GNNs and CNNs are more robust than traditional chemical and biological features obtained from databases.

Finally, Yang et al. [7] applied an unsupervised sequence embedding technique (doc2vec) to represent protein sequences as rich feature vectors of low dimensionality. Training a Random Forest (RF) classifier through a training dataset that covers known PPIs between human and all viruses, they obtained excellent predictive accuracy outperforming various combinations of machine learning algorithms and commonly used sequence encoding schemes to provide competitive and promising performance, suggesting that the doc2vec encoding scheme effectively captures context information of protein sequences, pertaining to corresponding protein–protein interactions. Looking at most published journals, perhaps the biggest challenge for this work will be the paucity of data, having to work with around 7000 data points for both training and testing.

3. Methodology

3.1. Dataset

Our dataset is extracted from the work in [1]. 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 [8] by their respective protein id to get a dataset that looks like:

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

In our work, each protein peptide chain is at least 35 units long and consists of 20 types of amino acids that are represented by the letters of the alphabet. NLP methods is used to train the model and testing accuracy is measured using the receiver operating characteristic (ROC) curve and the area under the ROC Curve (AUC) available from the sklearn package [9] in python. The results for each model shown are based on AUC score by testing them with the independent dataset.

3.2. Recurrent neural network

The first method that will be used is named Recurrent Neural Network (RNN) [10]. 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, which makes it ideal for the prediction of temporal data. At a high level, it remembers the past and makes predictions based on what it learned. The LSTM [11] 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 Gated Recurrent Unit [12].

However, this may not be enough since sequences can be 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 then concatenated to the original.

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:

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

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

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

This is done so that we account for every two or three sequences of amino acids that 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 the prediction of interaction, we could concatenate them and give a single third degree input as:

$[MYV, YVT, VTM, \ldots, MTG, TGP, GPQ, \ldots]$

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 what we will apply in our work.

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

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

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 pure 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 changes based on the training of the neural network during back propagation.

3.3. Convolution neural network

The next tool employed will be 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) [13] when used in this capacity. Fig. 1 shows the overview of TCN architecture with dilation factors $d = 1, 2, 4$ and filter size of 3.

According to the work in [14], 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 an RNN. Other than replacing the Bidirectional LSTM with CNN, the architecture is still largely similar and the approach for the problem remains identical as explained earlier.

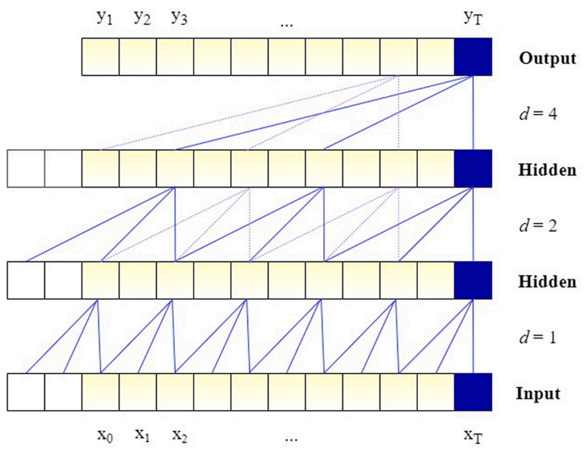

Fig. 1. TCN architecture.

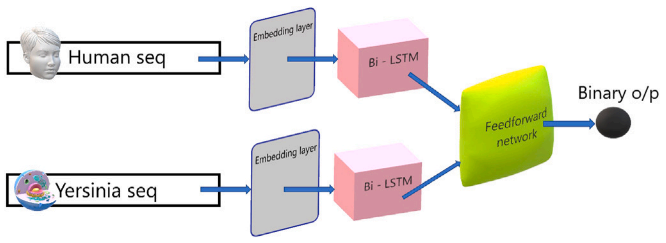

Fig. 2. The 'doubleip' configuration using Bi-LSTM.

4. Implementation

The machine learning models are created and trained in TensorFlow [15] using Keras [16]. TensorFlow is an open-source machine learning environment designed and developed by Google. It is very popular and supports libraries that can allow the software to run without changes on a regular CPU. It offers good computational graph visualizations, a varied library, good scalability, pipelining, and performance. Keras 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. The model's implementation is carried out using Google Colab Pro which offers high-speed 25 GB RAM and Tesla P100-PCIE-16 GB GPU. The implementation detail, neural network architectures, and dataset are available at our GitHub.

5. Experimental results

5.1. 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 work, we call this the *doubleip* configuration, shown in Fig. 2.

The other method as discussed earlier will be to concatenate the two sequences and evaluate them through a single embedding matrix and Bi-LSTM. In this work, we call this the 'join' configuration, shown in Fig. 3. The 'join' configuration is one of the methods used in NLP for sentiment analysis.

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

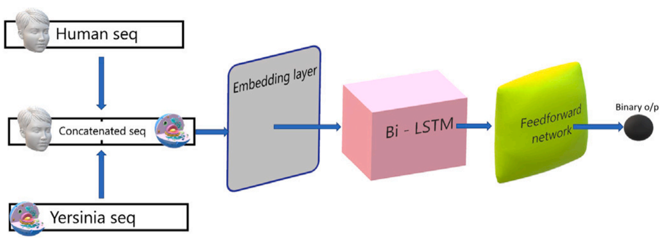

Fig. 3. The 'join' configuration using Bi-LSTM.

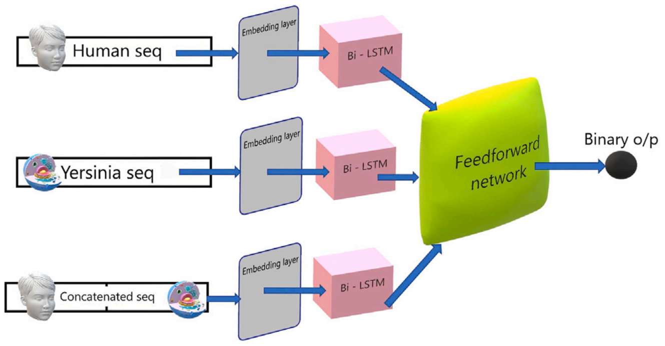

Fig. 4. The 'combine' configuration using Bi-LSTM.

Table 1
AUC scores while using Bi-LSTM.

Config.	D1	D2	D3	D4	D5	D6
Join	0.766	0.784	0.820	0.847	0.867	0.849
Doubleip	0.812	0.835	0.870	0.891	0.890	0.903
Combine	0.830	0.843	0.884	0.899	0.900	0.906

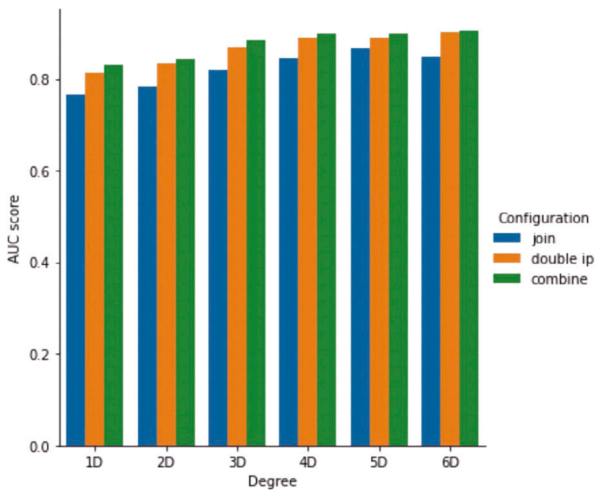

Fig. 5. Doubleip-join-combine configuration comparison.

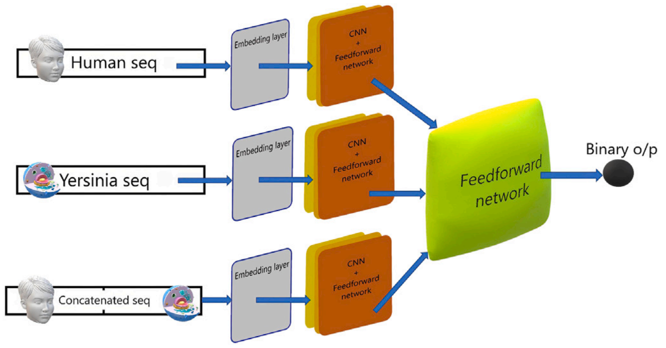

Fig. 6. The 'combine' configuration using CNN.

play a role in the prediction of interaction (join). Let us call this *combine* configuration, shown in Fig. 4.

Table 1 and Fig. 5 shows the AUC for our 3 configurations using Bi-LSTM. The most obvious observation while using *doubleip* is that accuracy increases with an increase in complexity (size) of the words. However, keep in mind that there are some peptide chains with just 35 amino acids which will not be represented adequately as the dimension (D) increases because of the vocabulary cap imposed by the tokenizer making the chances of 'words' from smaller peptide chains to be represented 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 is in the training data and 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. For *join* configuration, we observe that accuracy increases with word size but the results are not as good as for *doubleip* configuration. Additionally, it becomes obvious that our theory was correct, *Combine* configuration seems to give us the best results. Overall, the best accuracy is 99% for all 6D models.

5.2. 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, there are some advantages to including a CNN as shown in our case. Fig. 6 shows the combine configuration using CNN. The configuration used here has 32 filters, 3 kernels followed by Max pooling. We exclude 6D inputs due to reasons mentioned earlier as well as the fact that the gains appear negligible from there.

Table 2
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

Table 2 shows the AUC for our 3 configurations using CNN. Similarly, the model performance improves as the D increases when using CNN. Fig. 7 shows the performance comparison using BiLSTM and CNN. 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 gives significant gains for lower D inputs compared to Bi-LSTMs, and they are shown to be 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 play an important role in hyperparameter tuning. Another interesting observation is that, unlike Bi-LSTMs, the *combine* configuration seems to give little to no improvement on CNN based models. Overall, the best accuracy is 97% for all 6D models.

5.3. Advanced preprocessing

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

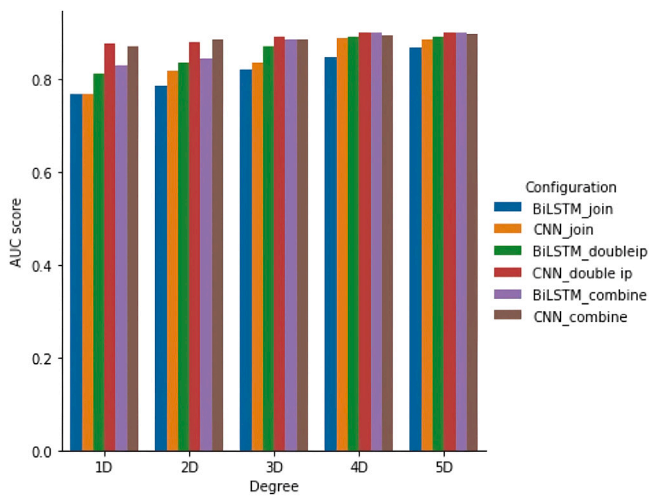

Fig. 7. BiLSTM vs CNN.

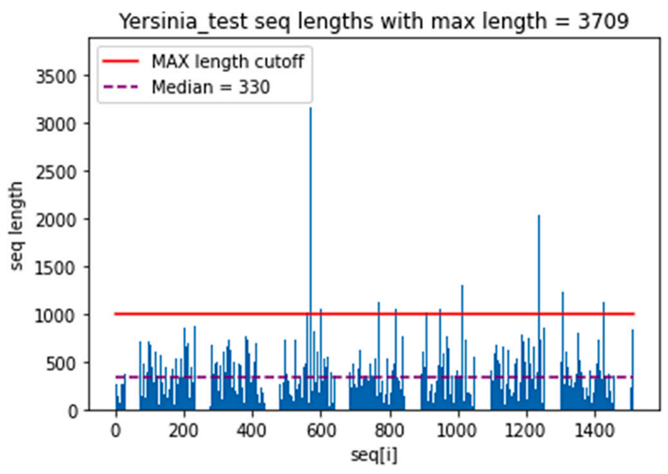

Fig. 8. Sequence length distribution in Yersinia test data.

5.3.1. *Sequence truncation*

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 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. In this case, the problem 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 a drastic reduction in accuracy as well.

To better illustrate that problem, consider the sequence length distribution shown in Fig. 8. The MAX length (red line) covers most data points, meaning 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. We initially selected MAX length as close to the median as possible to give us the best results, at the cost of losing significant data. To address this problem, we propose the following configuration.

Fig. 9 shows us a '3X' preprocessing configuration. Input sequences are pre-truncated as well as post-truncated in order to get the words from both sides. We also add center truncation to get as many words

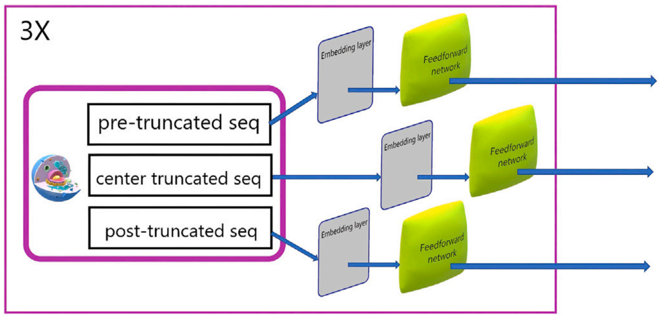

Fig. 9. 3X preprocessing for Yersinia sequences.

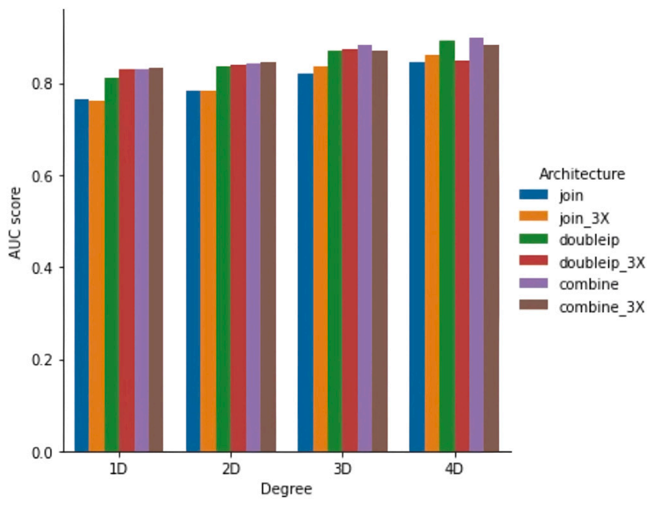

Fig. 10. 3X preprocessing using Bi-LSTMs.

Table 3
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

in the middle. These three inputs are put through separate embedding matrices which are connected in an end-to-end configuration to allow the neural network to decide their importance.

We start by testing this in Bi-LSTMs to get the following results shown in Fig. 10. It can be observed that 3X preprocessing offers little to no improvement. So we do not proceed any further and instead use the CNN models.

Table 3 and Fig. 11 show the result for 3X preprocessing using CNN. 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. Overall, the best accuracy is 99% for all 5D models.

This method has brought us closer to the final AUC score of the integrated model in the reference work [1]. From this point onward, we began to deal with the principle of diminishing returns. That means we hit the later end of the curve and any major changes to the model only gave us marginal improvements in our score.

5.3.2. *Attention mechanism*

Traditionally, seq2seq processing in NLP 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 then

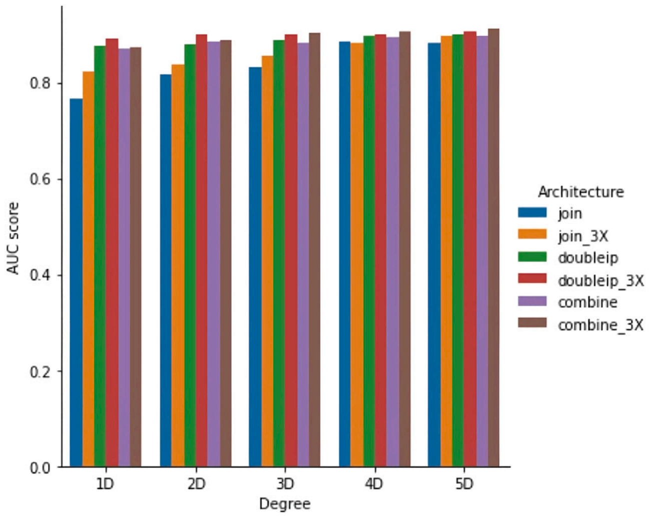

Fig. 11. 3X preprocessing using CNN.

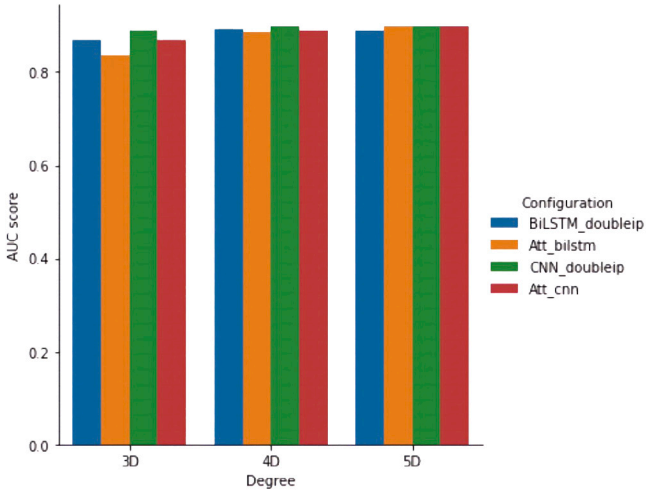

Fig. 12. Attention using CNN and Bi-LSTM with *doubleip* configuration.

uses that hidden state to create another sequence word by word. When the input sequence is too long, the model would be inaccurate because RNN would prioritize words towards the end. To address this drawback, we applied the attention layer [17]. In this model, 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.

Fig. 12 shows the comparison when applying the attention layer for CNN and Bi-LSTM. The attention mechanism does not seem to improve 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. Overall, the accuracy is 98% for Bi-LSTM with attention and 99% for CNN with attention.

5.3.3. Transformers

Recently, the work [18] introduced a new architecture called Transformers which has a solely attention-based encoder–decoder configuration where the encoder turns a sequence into a continuous representation while the decoder uses it to generate a word step by step. In this approach, 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 we call *differential join*. Here, each amino acid is represented by a

Table 4
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

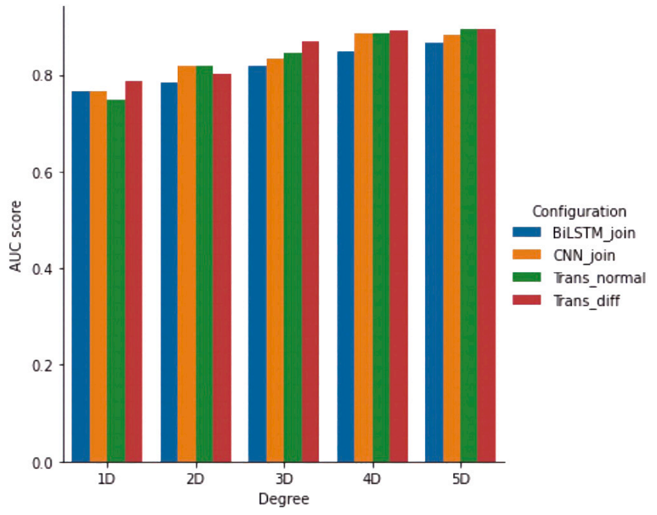

Fig. 13. Transformers vs Bi-LSTM vs CNN.

different letter for both Human and Yersinia sequences. 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 in Table 4:

Fig. 13 shows the comparison using 3 different approaches. Transformers give us the best performance for *join* configuration with accuracy about 99% for D5. We will be using this property next. The *differential join* inputs give us better results until we get to higher degrees.

5.4. Final model

To get the best AUC score we will combine the best methods examined above. Fig. 14 shows the overview of our model. 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 the 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*. The following steps provide an overview of our process:

1. Input preparation: Human sequence, Yersinia sequence, and a concatenated Human and Yersinia sequence called Joined.

2. Convert each of these three sequences to 5D format as mentioned above.

3. For each new training data:

- Convert the list of 5D sequences to a single sentence containing a string of words.

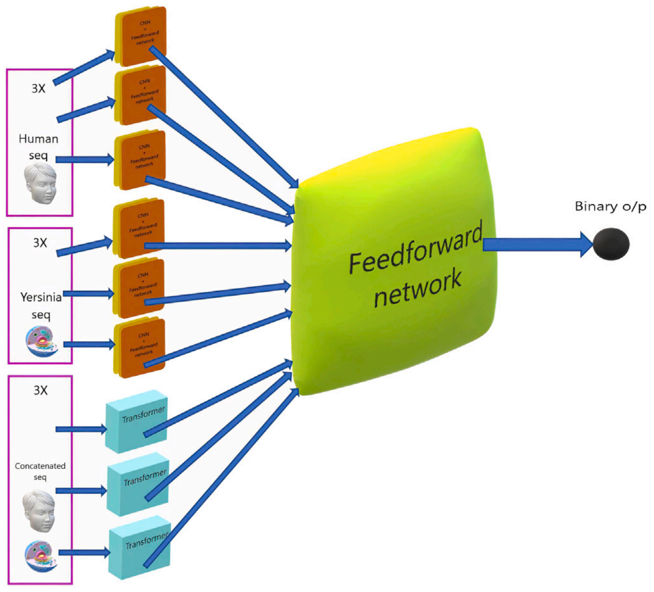

Fig. 14. Final model.

- For Human and Yersinia sentences, create separate tokenizers with a maximum vocabulary size of 500,000, then create index numbers of words for each species by frequency of occurrence.
- For Joined sentences, create one tokenizer with a maximum vocabulary size of 1,000,000, then create index numbers of words for all sentences in Joined by frequency of occurrence.

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

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.

8. Concatenate all the outputs to a single dense layer before sending it to a single output neuron with 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%. The embedding dimensionality at 25 seems to give the best results which is surprising because for NLP processing with the English language, usually it is kept at around 300 despite having a much lesser MAX vocabulary size. The Adam optimizer seems to give the 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 a decrease in training data.

6. Discussion

6.1. 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 dataset of 0.919, as well as a five-fold cross-validation score of 0.910 on the training set. The impact of 3X inputs has also been validated.

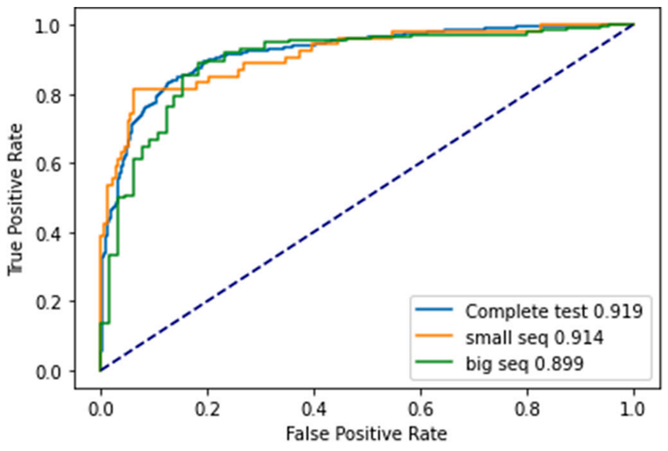

Fig. 15. ROC curve comparison.

In the test data, isolating 233 smallest sequences gave an AUC score of 0.914 while 280 longest sequences gave 0.899. Overall, we achieve better results than NetSS (0.88) and NetTP (0.82) in [1]. Interestingly, from Fig. 15, small sequences give us better thresholds with more ‘pure’ true positives while big sequences give us higher and more ‘impure’ true positives.

6.2. Human-Virus PPI

We applied this process to other scenarios to test the versatility of the model by trying to solve a different problem. Recently, Yang et al. [7] proposed to predict 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. Our final model was applied to this problem, without any examination of data or hyperparameter tuning.

We used the train–test dataset from [7] 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 an 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 subsections into the RAM one by one. The *doubleip* embedding dimensionality was increased to 50 to account for the greater number of words in the vocabulary that was inevitable considering the large dataset. The 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 which is comparable to 0.954 from the work in [7]. It must be noted that for any PPI usage of this model, the column names used must be [Human, Yersinia, Label, Joined].

6.3. Comparison to other sequence based encoding methods

6.3.1. 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 [19]. On the other hand, our model uses every 5 combinations of amino acids as words (5D) to be represented as an input matrix.

6.3.2. 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 sub-region, 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 [20]. Our model uses none of this or any knowledge in biochemistry.

6.3.3. 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 [21]. Just like before, our model uses no knowledge in biochemistry, 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.

6.3.4. Doc2vec+RF

This is similar to the final model implemented here. Used in the Human–Virus PPI prediction work described earlier [7], it uses 4D and 5D words to train the embedding matrix. But this was done using a 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 we understand, 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, they did not account for every part of the sequence like it was done here using '3X' configuration.

6.4. Future work

As described earlier, the deep learning architecture used here shows the versatility of our approach. However, improvements can be made to the transformers used for this data like the residual network combined with hyperparameter tuning for different datasets this will be in future works. Once all sequence related work has been done, the next step would be to train a Graphical Neural Network that accounts for PPI by protein ID with the help of an adjacency matrix and Node Embeddings. Another factor to take into account is Protein folding which is a process by which a polypeptide chain folds to become a biologically active protein and is crucial to its function. Protein folding prediction has been done with relatively good accuracy with AlphaFold [22] using amino acid sequences as input. We plan to test the folding landscape of these proteins or the 3D structure representation as an additional input in the end-to-end framework.

7. Conclusion

In this work, we applied multiple Natural Language Processing methods to predict the interaction of proteins between Human and Yersinia pestis by examining their respective amino acid sequences. The results compare favorably with the work in [1] that has a model which gives the 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 work [7] is also impressive considering the fact that little analysis of any sort has been carried out. The main strength of our approach is that our design provides comparable performance for other datasets using innovative methods with little to none pre-processing and hyperparameter tuning demonstrated with human–virus PPI.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Lian X, Yang S, Li H, Fu C, Zhang Z. Machine-learning-based predictor of human–bacteria protein–protein interactions by incorporating comprehensive host-network properties. J Proteome Res 2019;18(5):2195–205.
[2] Ahmed I, Witbooi P, Christoffels A. Prediction of human-bacillus anthracis protein–protein interactions using multi-layer neural network. Bioinformatics 2018;34(24):4159–64.
[3] Li H, Gong X-J, Yu H, Zhou C. Deep neural network based predictions of protein interactions using primary sequences. Molecules 2018;23(8):1923.
[4] Tsukiyama S, Hasan MM, Fujii S, Kurata H. LSTM-PHV: Prediction of human-virus protein-protein interactions by LSTM with word2vec. BioRxiv 2021.
[5] Zhou H, Li X, Yao W, Liu Z, Ning S, Lang C, et al. Improving neural protein-protein interaction extraction with knowledge selection. Comput Biol Chem 2019;83:107146.
[6] Tsubaki M, Tomii K, Sese J. Compound–protein interaction prediction with end-to-end learning of neural networks for graphs and sequences. Bioinformatics 2019;35(2):309–18.
[7] Yang X, Yang S, Li Q, Wuchty S, Zhang Z. Prediction of human-virus protein-protein interactions through a sequence embedding-based machine learning method. Comput Struct Biotechnol J 2020;18:153–61.
[8] UniProt: The universal protein knowledgebase in 2021. Nucleic Acids Res 2021;49(D1):D480–9.
[9] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30.
[10] Rumelhart DE, Hinton GE, Williams RJ. Learning internal representations by error propagation. Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science; 1985.
[11] Schmidhuber J, Hochreiter S. Long short-term memory. Neural Comput 1997;9(8):1735–80.
[12] Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. 2014, arXiv preprint arXiv:1406.1078.
[13] Lea C, Vidal R, Reiter A, Hager GD. Temporal convolutional networks: A unified approach to action segmentation. In: European conference on computer vision. Springer; 2016, p. 47–54.
[14] Bai S, Kolter JZ, Koltun V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. 2018, arXiv preprint arXiv:1803.01271.
[15] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015, Software available from tensorflow.org. [Online]. Available: http://tensorflow.org/.
[16] Chollet F, et al. Keras. 2015, https://keras.io.
[17] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. 2014, arXiv preprint arXiv:1409.0473.
[18] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. 2017, arXiv preprint arXiv:1706.03762.
[19] Sun T, Zhou B, Lai L, Pei J. Sequence-based prediction of protein protein interaction using a deep-learning algorithm. BMC Bioinformatics 2017;18(1):1–8.
[20] Davies MN, Secker A, Freitas AA, Clark E, Timmis J, Flower DR. Optimizing amino acid groupings for GPCR classification. Bioinformatics 2008;24(18):1980–6.
[21] Yang KK, Wu Z, Bedbrook CN, Arnold FH. Learned protein embeddings for machine learning. Bioinformatics 2018;34(15):2642–8.
[22] Senior AW, Evans R, Jumper J, Kirkpatrick J, Sifre L, Green T, et al. Improved protein structure prediction using potentials from deep learning. Nature 2020;577(7792):706–10.