End-to-end prediction of protein-protein interaction based on embedding and recurrent neural networks

Francisco Gonzalez-Lopez, Juan A. Morales-Cordovilla, Amelia Villegas-Morcillo, Angel M. Gomez, Victoria Sanchez
Dept. of Signal Theory, Telematics and Communications and CITIC-UGR, University of Granada, Spain
E-mail: {fgonzalezlopez,jamc,ameliavm,amgg,victoria}@ugr.es

***Abstract*—Protein-Protein interactions (PPIs) are key to many important life processes, such as cancer replication or DNA transcription. While *in vivo* or *in vitro* methods for PPI screening exist, they are expensive and computational approaches have been proposed to address PPI prediction. Previous computational methods rely on hand crafted features to capture the underlying information of the protein data. In this work we present a deep neural network architecture leveraging embedding techniques and recurrent neural networks to extract features and predict interaction between protein pairs. The results achieved are similar to those obtained by other state-of-the-art computational approaches to the problem but without *any* feature engineering involved, directly using the raw amino acid sequences.**

***Index Terms*—protein-protein interaction, deep neural network, recurrent neural network, embedding**

I. Introduction

Protein-protein interactions (PPI) are the driving forces behind many molecular processes which define the cellular activity at various levels. These interactions are physical contacts between proteins driven by the electromagnetic properties of their components. Interactions define important functions such as DNA transcription and replication and cell signaling. Protein presence or absence caused by genetic mutations has been linked to certain diseases, such as Alzheimer's [1] or cancer [2]. Further understanding how proteins interact in such diseases may prove paramount to find more effective and less invasive methods for treating them, be it in the form of prevention, risk identification based on genetic profiles, or better drug design.

The problem associated with PPI is that *in vivo* or *in vitro* procedures to determine interaction, such as those described in [3], are relatively expensive and are affected by uncertainty. In other problems computational methods usually serve as a first screening and PPI can benefit as well from an inexpensive computational screening of candidates to interaction, in order to reduce costs and increase success rates of experimental methods.

Several computational PPI prediction approaches have been proposed such as [4], [5] or [6]. These methods rely on hand crafted representations of the proteins involved in the interactions, using different kinds of data (sequence data, physico-chemical properties of the amino acids, etc.). Such features are the result of a complex and time consuming feature engineering process, that requires domain knowledge, as they depend on underlying properties of the data. A similar problem is present in the computer vision field; the features representing an image must capture the underlying complexity of the data relevant for the task. Crafting such features can be very time consuming, and nearly impossible in some cases. Thus, more modern approaches to computer vision problems do not rely on hand crafted features. Instead, they revolve around learning features from the data that perform well on a certain task using deep neural networks (DNNs), as currently the amount of data and computational power are enough to successfully apply these methods.

This work has been supported by the Spanish MINECO/FEDER Project TEC2016-80141-P. We also acknowledge the support of NVIDIA Corporation with the donation of a Titan Xp GPU.

In a similar vein, rather than relying on hand crafted features to represent a protein pair for PPI prediction, we propose a deep learning approach to the problem, where the features are learned through an optimization process, leveraging the increasing amount of available PPI data. In [5] the approach taken is to compute an extensive set of features, and then apply deep learning techniques to the classification task, but they still rely on an initially established set of features. We take the idea one step further, learning low level features directly from the raw protein sequence data. In order to do so, we tackle the problem by combining Natural Language Processing (NLP) techniques, such as embedding [7], and recurrent neural networks (RNNs) [8] that have been shown to individually perform well in other tasks, like protein function prediction.

In this paper, we present the results obtained by our method (called DeepSequencePPI) in four PPI datasets obtained from the Database of Interacting Proteins (DIP) [9], two of them containing *S. cerevisiae* PPI, one containing *H. pylori* PPI [10], and the last one containing *H. sapiens* PPI obtained from the Human Protein References Database (HPRD) [11]). We will show that relying solely on the raw sequence data, no feature engineering is needed to achieve state-of-the-art results on these datasets.

II. Materials and Methods

A. Datasets

The main protein interaction dataset used in the experiments is described in [5]. It contains both positive and negative interactions between protein pairs from the yeast species *"Saccharomyces cerevisiae"*. The positive interactions are obtained from the Database of Interacting Proteins (DIP) [9], discarding those proteins whose chains are shorter than 50 amino acids and those that are highly similar in sequence (compared using CD-HIT [12] with sequence identity greater than 40%). The negative interactions are selected by pairing proteins from different subcellular locations (obtained from SwissProt [13]) that do not appear in the positive data set [5]. It contains 17257 and 48594 interactions (positive and negative, respectively). This dataset will be referred to as *"S. cerevisiae full"* dataset.

In addition, other 3 datasets [5] are used to validate the method:

1) *S. cerevisiae core subset*, containing 5594 positive pairs and 5594 negative pairs from *"Saccharomyces cerevisiae"*. This dataset will be referred to as *"S. cerevisiae core subset"* to differentiate it from the *"S. cerevisiae full"* dataset.
2) *H. pylori* dataset [10], containing 1458 positive pairs and 1458 negative pairs from *"Helicobacter pylori"*.
3) *H. sapiens* dataset [14], containing 3899 positive pairs and 4262 negative pairs from *"Homo sapiens"*.

B. Method

The proposed method has two distinct parts. First, data preparation, where each protein is converted into a suitable data format for the prediction model, using a technique known as tokenization. The result is a numeric vector for each protein amino acid sequence. It is also worth noting that this is not a feature matrix, only a fixed-length equivalent representation of the protein sequence data. Second, neural network training and testing, where each sequence is passed through a neural network, modifying its parameters (i.e. the weights of the network) by back propagating the errors in the training set, optimizing both classification and feature extraction together. Finally, the test data is predicted using the resulting classification model, and its performance is assessed.

1) Data preparation: Protein-protein interaction datasets describe each sample using three attributes, protein A code, protein B code, and interactivity, encoded as 0 or 1 (1 for interaction, 0 otherwise). Once the amino acid sequences are obtaind from the Uniprot [15] database then the data is prepared to be fed to the neural network. Intuitively, we are going to interpret this problem as a Natural Language Processing (NLP) problem, where the amino acid sequence is viewed as a text or phrase, and amino acids are akin to words. Moreover, instead of interpreting each amino acid in the sequence as a separate unit, we are going to model the protein as a *"3-gram"* sequence, where 3 contiguous amino acids are joined and form a single unit or "word". The strategy adopted does not include overlapping adjacent n-grams. Also, if a sequence cannot be split exactly by 3, the first amino acids in the chain are discarded. This helps to characterize each amino acid not only by itself, but also by its context. Thus, our basic unit is an amino acid "X" preceded by amino acid "Y", and followed by amino acid "Z", forming a triplet, "YXZ". This modeling strategy was adopted because it has been shown to perform well with similar approaches to other problems, such as [7]. The 3-gram extraction process is shown in Fig. 1

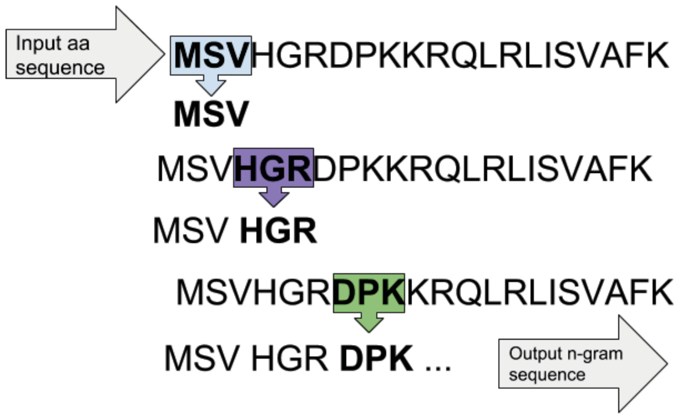

Fig. 1. The 3-grams extracted from an amino acid sequence.

Since the neural network requires a numerical input, we apply a **tokenization** process (Fig. 2), where every triplet in the sequence is encoded as an integer (token), obtaining a sequence of tokens. The special token "0" is reserved for padding. Thus, the number of different tokens will be 8001.

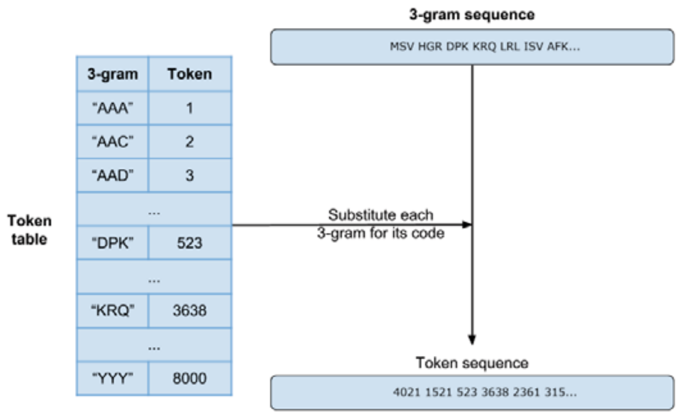

Fig. 2. Token extraction process where an integer is assigned to each unique 3-gram. 0 is reserved.

Finally, in order to obtain a fixed length representation, we pad the token sequence vector with the value "0" to obtain a vector with a fixed length of 1000 elements. If a token sequence vector is longer than that length, it is truncated discarding the leftmost elements (i. e., the beginning of the amino acid chain) in the sequence. On the other hand, if a token sequence vector is shorter than the fixed length, we add "0" tokens to the left of the vector until the required length is reached.

We now have a fixed length numerical vector representation of each protein, suitable for feeding the neural network. The whole process is summarized in Fig. 3.

2) Deep neural network model and training: Deep neural networks (DNN) are a type of neural networks with a high number of layers. These models rely on non-linear operations

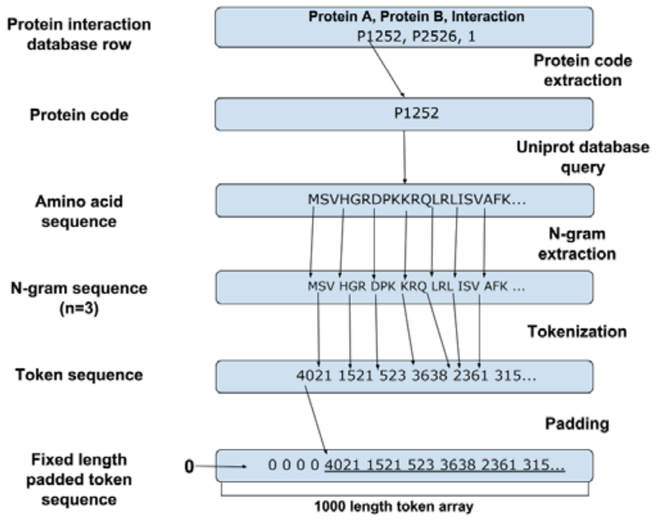

Fig. 3. Summarized pre-processing example of an amino acid sequence, from the raw sequence to its fixed length numerical representation.

to learn complex relations between the inputs. The proposed model takes two proteins as its inputs, represented as described in the former section, and decides whether the protein pair interacts or not. In order to do so, the parameters of the network are optimized by passing the training pairs through the network, and comparing the predicted label against the true label.

The design of the architecture of the network is a crucial step in the DNN approach. If the network is too simple to capture the relations among the data, or there are not enough data to tune the parameters sufficiently (the weights of the network don't converge to a solution), the performance of the model suffers. Likewise, if the network is too complex, the network may start learning the training data itself. This is known as *overfitting*; it is somewhat similar to ”memorizing” information, rather than ”understanding” it. Another related issue is that there is no way to know what network complexity or what type of architecture fits a problem. There is some intuition from similar problems, but there are no guarantees that a given approach will perform better or worse than any other, unless both of them are tested.

The proposed neural network architecture, shown in Fig. 4, consists of 2 basic parts: feature extraction, and classification. Since the data is composed by a protein pair and a label (interaction), the network receives two different inputs (each protein of the pair). The format of each input was described in the former section. The proteins are processed separately in two branches of the network that share the same architecture, whose task is to learn the features that will describe the protein. Each branch is composed by an *embedding layer*, a *recurrent layer* (with GRU units) and finally a *fully connected layer* to rearrange and recombine the information extracted by the two previous layers. Note that the embedding layer is shared among the two branches, since any context dependencies between the sequences will be the same regardless of the branch they enter. Then, the features that each extractor branch has computed are merged into a single vector by simply concatenating them. Finally, a fully connected layer is used to combine the features of both proteins. The last layer of the network is a 2-neuron fully connected layer, and its activation function is a 2-class softmax, allowing us to assign the output of each neuron as the score for each class. This is then interpreted as the probability of a sample (a protein pair) belonging to a certain class (interaction or no interaction), assigning the final label to the most likely class (the one with the highest score).

Each layer has a specific architecture and role in the network. The layers used in the architecture are as follows:

- *Embedding layer*: The embedding layer is tasked with learning a compact representation of the input data (sequence of tokens), attending to the context (i.e. the tokens that appear in the same sequence) of each input token. The result is a compact vector representation of the input, that depends on the contexts of each token in it. The proposed length of the compact context vector is 512. This process adds another dimension to the output, generating for each position in the sequence a vector representing the context given the token and position. In our case, this would be a 1000x512 matrix. A basic scheme is shown in Fig. 5.

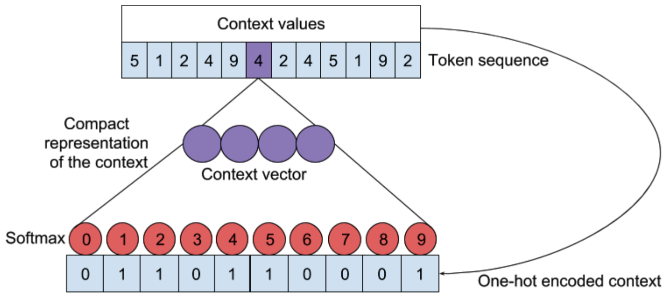

Fig. 5. Basic embedding scheme. Note that we use the compact representation layer, and discard the softmax layer, as it is only for training the weights. This is an example using a set of 10 tokens to illustrate the idea behind the embedding.

- *Recurrent layer*: The recurrent layer receives a matrix (1000x512) as an input, processes each position in the input using information from previous positions, and outputs a 64 feature vector. The layer uses information from previous elements in the sequence. Intuitively, it processes the input sequentially and carries a sort of *”memory”* by feeding each output as an additional input to the next unit. This kind of process is similar to how reading works; instead of processing each word separately, we retain certain information that helps us give context and understand what we read next, as illustrated in Fig. 6. The particular RNN unit proposed is the GRU (gated recurrent unit). The recurrent layer uses 2 activation functions; first, the recurrent activation, applied to the output of each recurrent unit (to the *”memory”*),

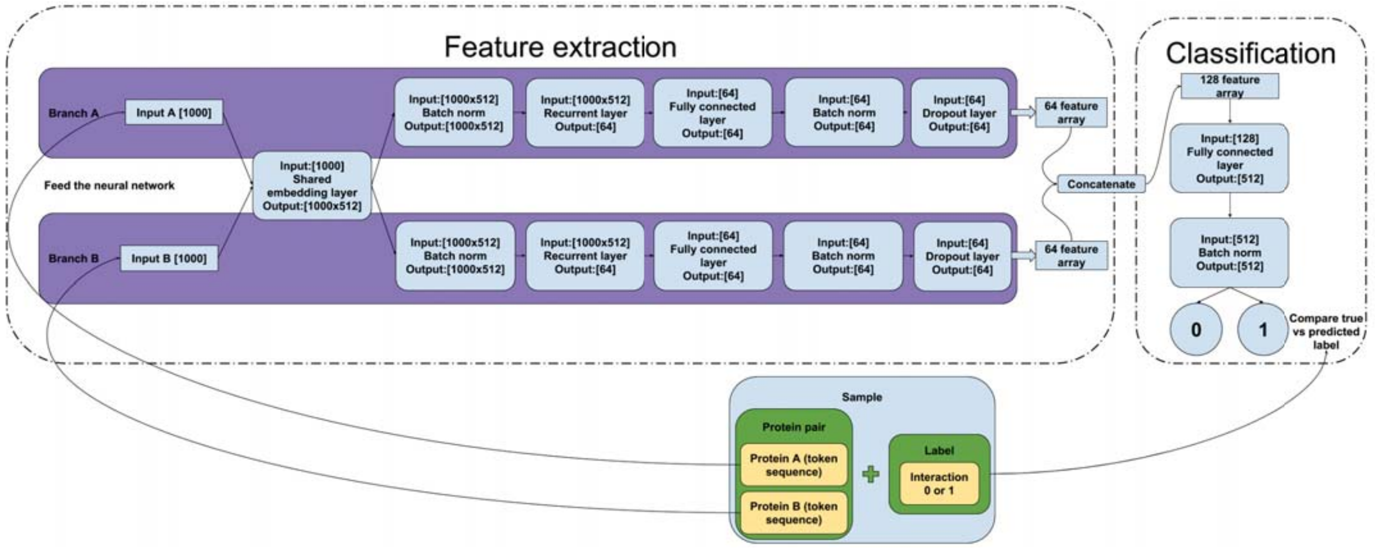
Feature extraction
Classification
Branch A
Input A [1000]
Feed the neural network
Input:[1000] Shared embedding layer Output:[1000x512]
Branch B
Input B [1000]
Input:[1000x512] Batch norm Output:[1000x512]
Input:[1000x512] Recurrent layer Output:[64]
Input:[64] Fully connected layer Output:[64]
Input:[64] Batch norm Output:[64]
Input:[64] Dropout layer Output:[64]
64 feature array
Concatenate
128 feature array
Input:[128] Fully connected layer Output:[512]
Input:[512] Batch norm Output:[512]
0
1
Compare true vs predicted label
Sample
Protein pair
Protein A (token sequence)
Protein B (token sequence)
Label
Interaction 0 or 1

Fig. 4. Architecture of the neural network used to extract features and then classify the protein pairs. First, the input is passed through an embedding layer, that condenses the contextual relations between amino acid 3-grams in a vector for each position of the chain (that is, if the fixed input length is 1000, the output of the embedding using 512 features will be a 1000x512 matrix). Then the recurrent layer attempts to capture temporal relationships and patterns among these features, outputting a 64 feature vector. That vector is passed on to a fully connected layer to combine these features before merging the branches. Then, the two 64-feature vectors are merged by concatenating them into a single 128-feature vector (branch A vector always goes first). After the merge, the data is passed through 2 more fully connected layers that subsequentially reduce the dimensionality of the data by combining it, until finally a 2 neuron layer acts as the output of the network, giving a score to each class.

and second, the activation that is applied to the final output of the layer. We use the hard sigmoid function for the recurrent activation and the hyperbolic tangent function for the final activation. The final output is the output of the last position in the sequence, which is computed using information from the rest of the sequence (along with the last element itself).

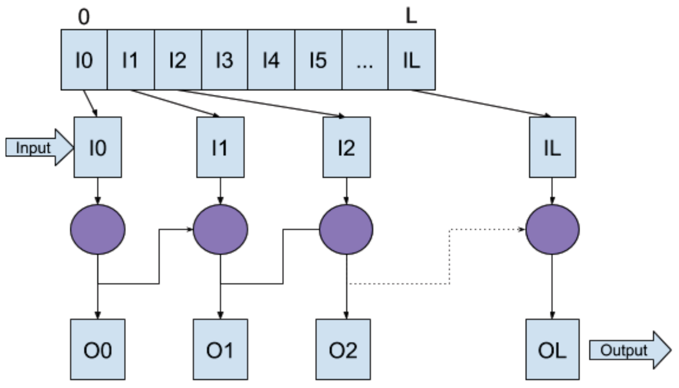

Fig. 6. Simplified scheme of the connections between units in a recurrent layer. Bear in mind that each element in the input sequence is usually a feature vector, rather than a single value. In our architecture, each element is a 512 feature vector from the previous embedding layer.

- *Fully connected layer*: Fully connected layers are used in 2 fashions. First, they learn non-linear relationships of their inputs, where every single unit in the layer is connected to all units in the previous layer. Intuitively, fully connected layers learn a higher level representation of the input, thus, they are used to combine the features produced by the previous layers. Although the most popular activation is the rectified linear unit (ReLU), we use the exponential linear unit (ELU) [16], because it is harder to saturate and has been successfully applied [17]. The second task they fulfill is serving as an output 2-class softmax, where the output of each unit represents the score for a corresponding class. Since there are 2 classes, we will have two output units, one being ”negative interaction” and the other being ”positive interaction”.
- *Dropout*: The dropout technique [18] consists in ”dropping” (set to 0) a randomly selected portion of the layer’s inputs. In practice, this has been shown to reduce overfitting. Dropout is applied after each layer in the network, except for the last fully connected layer (as it is meant to output the resulting scores).
- *Batch normalization layer*: The batch normalization layer standarizes the input (i.e. substracts the mean and scales by the standard deviation, thus leaving the mean at 0 and the standard deviation at 1). This technique requires keeping ”mean” and ”standard deviation” parameters that are adjusted during training. This also has a slight regularization effect, further reducing the possibility of overfitting.

During training, two strategies are used to further guide the process, both involving a validation dataset. The validation set is a small portion (10%) of the training set. The purpose of this set is serving as a guide to check how the model is performing during training, and guide it, but it is never used to directly train the weights of the network. The applied strategies are the following:

- Early stopping [19]. This strategy simply stops the training when the loss (categorical cross entropy) does not improve (in the validation set) for 5 consecutive training epochs. This is just an aid to avoid wasting resources on a model that does not improve. Instead of keeping the last weights of the network, we roll back to a point where the model performed the best on the validation set, in order to reduce overfitting.
- Reduce learning rate when stagnating. This strategy consists in reducing the learning rate in order to explore a part of the solution space more in depth, and achieve a better local minimum.

The procedure for the neural network training, where the weights or parameters of the network are optimized to perform the classification tasks, is done by minimizing the value of a loss function that we define. We use the categorical cross entropy, paired with the RMSProp gradient descent optimization algorithm.

After the training phase, the performance of the model is measured on a test set. In all cases, the performance metrics used are fairly standard: accuracy, precision, recall, specifitiy, Mathews Correlation Coefficient (MCC), F-1 score, and ROC AUC.

The software tools used for this task are the *Python* libraries *Keras* [20] and *Tensorflow* [21]. *Keras* serves as a simpler front end to *Tensorflow*, abstracting most of the network building details and allowing quick prototyping. The *Python* biological data processing toolkit *Biopython* [22] has been used to parse the sequence data. The implementation used for the experiments will be made available on GitHub. For reference, a single repetition of experiment 1 takes about 8 minutes to complete using a *Nvidia GeForce GTX 1080ti*.

III. Results

In this section we present the results of two sets of experiments. Apart from the method *"DeepPPI"* by Du et al. [5], all numerical results are taken directly from their respective papers. For *"DeepPPI"* all results have been generated using the code they provide online. These sets of experiments are conforming with the procedure that most other works use to evaluate their methods on the respective dataset, and thus allow us to compare the results of our proposed method to the rest. The comparison with *"DeepPPI"* is particularly interesting, as the approaches are fairly similar, the key difference being that we *learn* a feature extraction process, rather than selecting features out of an extensive set.

3) First experiment (S. cerevisiae dataset): The first experiment is a comparison of the results obtained by [5] for the *"S. cerevisiae full"* dataset, training and testing using a randomly selected portion of the dataset. This selection consists of creating a new dataset where both classes (interacting and non interacting pairs) are balanced. In order to achieve this, all positive interaction pairs are selected, and out of all the negative samples, we randomly select negative ones until the subset has the same amount of positive and negative pairs. This will result in a dataset containing all 17257 positive pairs and 17257 negative pairs chosen at random, totaling 34514 pairs.

Since the experiment requires the generation of a dataset using a random subset of the full data, the standard procedure involves repeating the experiment to aleviate the effects the selection might have on the results. Instead of repeating the experiment 5 times as in [5], in our experimental framework 30 repetitions are considered to further reduce the effects of random selection. Although the datasets are randomly generated, they are obtained using the same random number generator seed to always train both techniques with the same random sets of samples and have a fairer comparison.

Once a subset has been selected, we reserve 25% of the data for the test set that will be used to evaluate the quality of the prediction method, and train the model using the remaining 75%. Out of the training data, a 10% is used as a validation set to guide the learning process using the early stopping and learning rate reduction strategies.

By comparing the results obtained by our method and DeepPPI [5], we can see that their most striking feature is their similarity, differing less than a percentual point. In order to assess if both methods yield performances that are statistically different, we conducted a McNemar's test [23], where we compared the performances of both classiffiers for all test samples (a total of 258870 test samples across the 30 iterations). The result of the test is that both classifiers are not statistically different using a confidence interval of 95%, with a *p-value* of 0.984. The averaged results of all 30 experiments are presented in Table I.

4) Second set of experiments:: To further validate the performance of the method, three more datasets have been used separately. In each of the three experiments, a 5-fold cross validation has been used to assess performance. The resulting numbers in all cases are the mean values for each metric. All folds contain the same label balance (i.e. the relative amount of samples of each class) as their respective full datasets.

- *"S. cerevisiae core subset"*. 5594 positive samples, 5594 negative samples. Results are presented in Table II. As we can see in the table, the results obtained are very similar to the other state-of-the-art methods, meaning that while the available data for training is significantly smaller, the results are still competitive.
- *"H. pylori dataset"*. 1458 positive samples, 1458 negative samples. Results are presented in Table III. In this case the results fall behind other works, such as You *et al.* [6]. This is possibly due to the lack of data to learn a good feature extraction process.
- *"H. sapiens dataset"*. 3899 positive samples, 4262 negative samples. Results are presented in Table IV. No data balancing or class weighting is applied despite containing more negative than positive pairs. Again, when given enough data the model is able to yield similar results to other methods.

To sum up the results of the three experiments, the performance of our DeepSequencePPI method is again similar to other state-of-the-art methods for all datasets, falling behind

TABLE I
EXPERIMENT 1: *"S. cerevisiae full"*. MEAN VALUES OF THE METRICS FOR THE TEST PORTIONS OF ALL 30 DATASETS GENERATED.

Method	Accuracy	Precision	Recall	Specificity	MCC	AUC	F1
Our method	$92.59 \pm 0.31\%$	$93.65 \pm 0.53\%$	$91.4 \pm 0.46\%$	$91.59 \pm 0.49\%$	0.852 ± 0.0063	0.974 ± 0.0016	$92.51 \pm 0.31\%$
Du [5]	$92.41 \pm 0.26\%$	$94.19 \pm 0.52\%$	$90.4 \pm 0.48\%$	$90.75 \pm 0.48\%$	0.849 ± 0.0052	0.973 ± 0.0015	$92.25 \pm 0.25\%$

only where the proposed method is expected to: where there are not enough samples to learn a robust feature extraction process. This is the main dissadvantage of these kind of approaches, they require more data to perform the task in exchange for automating the feature extraction.

IV. CONCLUSIONS

In this paper we proposed an end-to-end deep neural network architecture based on RNNs and combined with embedding to predict protein-protein interactions directly from their amino acid sequences. Our approach levarages the high volumes of data available nowadays to bypass any requirement of feature engineering (associated with prediction problems), changing it to a neural network architecture selection problem. The results obtained are shown to be statistically similar to those achieved in previous works, suggesting that the underlying information extracted from the sequences is very similar to the information other methods use (hand-crafted features), given a sufficient amount of data to allow the neural network to properly model such information.

REFERENCES

[1] J. Murrell, Farlow, Ghetti, and Benson, "A mutation in the amyloid precursor protein associated with hereditary alzheimer's disease," *Science*, Oct 1991.

[2] K. Honda, T. Yamada, R. Endo, Y. Ino, M. Gotoh, H. Tsuda, Y. Yamada, H. Chiba, and S. Hirohashi, "Actinin-4, a novel actin-bundling protein associated with cell motility and cancer invasion," *The Journal of Cell Biology*, vol. 140, no. 6, pp. 1383-1393, 1998.

[3] S. Xing, N. Wallmeroth, K. W. Berendzen, and C. Grefen, "Techniques for the analysis of protein-protein interactions in vivo," *Plant Physiol*, vol. 171, pp. 727–758, Jun 2016.

[4] Y. Ding, J. Tang, and F. Guo, "Identification of protein–protein interactions via a novel matrix-based sequence representation model with amino acid contact information," *International Journal of Molecular Sciences*, vol. 17, no. 10, p. 1623, 2016.

[5] X. Du, S. Sun, C. Hu, Y. Yao, Y. Yan, and Y. Zhang, "DeepPPI: Boosting prediction of protein–protein interactions with deep neural networks," *Journal of Chemical Information and Modeling*, vol. 57, no. 6, pp. 1499–1510, 2017.

[6] Z.-H. You, Y.-K. Lei, L. Zhu, J. Xia, and B. Wang, "Prediction of protein-protein interactions from amino acid sequences with ensemble extreme learning machines and principal component analysis," *BMC Bioinformatics*, vol. 14, no. Suppl 8, 2013.

[7] E. Asgari and M. R. K. Mofrad, "Continuous distributed representation of biological sequences for deep proteomics and genomics," *Plos One*, vol. 10, Oct 2015.

[8] X. Liu, "Deep Recurrent Neural Network for Protein Function Prediction from Sequence," *ArXiv e-prints*, Jan. 2017.

[9] I. Xenarios, D. W. Rice, L. Salwinski, M. K. Baron, E. M. Marcotte, and D. Eisenberg, "Dip: the database of interacting proteins," *Nucleic Acids Res*, vol. 28, pp. 289–291, Jan 2000. gkd005[PII].

[10] S. Martin, D. Roe, and J.-L. Faulon, "Predicting protein-protein interactions using signature products," *Bioinformatics*, vol. 21, no. 2, pp. 218–226, 2004.

[11] "Human protein reference database." http://www.hprd.org/.

[12] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li, "CD-HIT: accelerated for clustering the next-generation sequencing data," *Bioinformatics*, vol. 28, pp. 3150–3152, Dec 2012.

[13] A. Bairoch and R. Apweiler, "The SWISS-PROT protein sequence database: its relevance to human molecular medical research," *J. Mol. Med.*, vol. 75, pp. 312–316, May 1997.

[14] Y.-A. Huang, Z.-H. You, X. Gao, L. Wong, and L. Wang, "Using weighted sparse representation model combined with discrete cosine transformation to predict protein-protein interactions from protein sequence," *BioMed Research International*, vol. 2015, pp. 1–10, 2015.

[15] T. U. Consortium, "Uniprot: the universal protein knowledgebase," *Nucleic Acids Research*, vol. 45, no. D1, pp. D158–D169, 2017.

[16] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *ArXiv e-prints*, Nov. 2015.

[17] S. Sinai, E. Kelsic, G. M. Church, and M. A. Nowak, "Variational autoencoding of protein sequences," *ArXiv e-prints*, Dec. 2017.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.

[19] R. Caruana, S. Lawrence, and L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, (Cambridge, MA, USA), pp. 381–387, MIT Press, 2000.

[20] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[22] P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and M. J. L. de Hoon, "Biopython: freely available python tools for computational molecular biology and bioinformatics," *Bioinformatics*, vol. 25, no. 11, pp. 1422–1423, 2009.

[23] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, pp. 153–157, Jun 1947.

[24] Y.-A. Huang, Z.-H. You, X. Gao, L. Wong, and L. Wang, "Using weighted sparse representation model combined with discrete cosine transformation to predict protein-protein interactions from protein sequence," *BioMed Research International*, vol. 2015, pp. 1–10, 2015.

[25] Z.-H. You, L. Zhu, C.-H. Zheng, H.-J. Yu, S.-P. Deng, and Z. Ji, "Prediction of protein-protein interactions from amino acid sequences using a novel multi-scale continuous and discontinuous feature set," *BMC Bioinformatics*, vol. 15, no. Suppl 15, 2014.

[26] L. Wong, Z.-H. You, Z. Ming, J. Li, X. Chen, and Y.-A. Huang, "Detection of interactions between proteins through rotation forest and local phase quantization descriptors," *International Journal of Molecular Sciences*, vol. 17, no. 1, p. 21, 2015.

[27] Y. Guo, L. Yu, Z. Wen, and M. Li, "Using support vector machine combined with auto covariance to predict protein-protein interactions from protein sequences," *Nucleic Acids Res*, vol. 36, pp. 3025–3030, May 2008. gkn159[PII].

[28] Y. Z. Zhou, Y. Gao, and Y. Y. Zheng, "Prediction of protein-protein interactions using local description of amino acid sequence," pp. 254–262, 2011.

[29] L. Yang, J. F. Xia, and J. Gui, "Prediction of protein-protein interactions from protein sequence using local descriptors," *Protein Pept. Lett.*, vol. 17, pp. 1085–1090, Sep 2010.

[30] L. Nanni, "Hyperplanes for predicting protein-protein interactions," *Neurocomputing*, vol. 69, no. 1, pp. 257 – 263, 2005. Neural Networks in Signal Processing.

TABLE II

EXPERIMENT 2: *"S. cerevisiae core subset"*. MEAN VALUES OF THE METRICS ACROSS ALL 5 FOLDS.

Method	Accuracy	Precision	Recall	Specificity	MCC	AUC	F1
Our method	$94.65 \pm 0.43\%$	$96.76 \pm 0.68\%$	$92.42 \pm 1.34\%$	$92.77 \pm 1.15\%$	0.894 ± 0.008	0.984 ± 0.002	$94.53 \pm 0.48\%$
Du [5]	$94.67 \pm 0.33\%$	$96.73 \pm 0.67\%$	$92.47 \pm 0.46\%$	$92.80 \pm 0.40\%$	0.894 ± 0.007	0.984 ± 0.002	$94.55 \pm 0.32\%$
Huang [24]	$96.28 \pm 0.52\%$	$99.92 \pm 0.18\%$	$92.64 \pm 1.00\%$	n/a	0.928 ± 0.010	0.963 ± 0.007	n/a
You [6]	$87.00 \pm 0.29\%$	$87.59 \pm 0.32\%$	$86.15 \pm 0.43\%$	n/a	0.774 ± 0.004	n/a	86.8%
You [25]	$91.36 \pm 0.36\%$	$96.55 \pm 0.61\%$	$90.67 \pm 0.69\%$	n/a	0.842 ± 0.006	n/a	91.3%
Wong [26]	$93.92 \pm 0.36\%$	$96.45 \pm 0.45\%$	$91.10 \pm 0.31\%$	n/a	0.886 ± 0.006	0.94 ± 0.002	n/a
Guo [27]	$89.33 \pm 2.67\%$	$88.87 \pm 6.16\%$	$89.93 \pm 3.68\%$	n/a	n/a	n/a	89.4%
Zhou [28]	$88.56 \pm 0.33\%$	$89.50 \pm 0.60\%$	$87.37 \pm 0.22\%$	n/a	0.772 ± 0.007	0.951 ± 0.004	n/a
Yang [29]	$86.15 \pm 1.17\%$	$90.24 \pm 1.34\%$	$81.03 \pm 1.74\%$	n/a	n/a	n/a	85.39%

TABLE III

EXPERIMENT 3: *"H. pylori dataset"*. MEAN VALUES OF THE METRICS ACROSS ALL 5 FOLDS.

Method	Accuracy	Precision	Recall	Specificity	MCC	AUC	F1
Our method	$85.18 \pm 1.27\%$	$82.79 \pm 1.42\%$	$88.89 \pm 2.34\%$	$88.06 \pm 2.17\%$	0.706 ± 0.026	0.921 ± 0.011	$85.71 \pm 1.29\%$
Du [5]	$85.88 \pm 2.11\%$	$83.57 \pm 2.76\%$	$89.85 \pm 2.28\%$	$88.77 \pm 2.28\%$	0.720 ± 0.041	0.929 ± 0.009	$86.56 \pm 1.92\%$
You [6]	87.50%	86.15%	88.95%	n/a	0.781	n/a	87.53%
You [25]	84.91%	86.12%	83.24%	n/a	0.744	n/a	84.66%
Wong [26]	$89.47 \pm 1.05\%$	$89.63 \pm 1.77\%$	$89.18 \pm 1.42\%$	n/a	0.810 ± 0.009	0.900 ± 0.015	n/a
Huang [24]	86.74%	87.01%	86.43%	n/a	0.770	n/a	86.72%
Zhou [28]	84.20%	83.30%	85.10%	n/a	n/a	n/a	84.19%
Nanni [30]	84.00%	84.00%	86.00%	n/a	n/a	n/a	n/a
Martin [10]	83.40%	85.70%	79.90%	n/a	n/a	n/a	82.70%
Nanni [31]	86.60%	85.00%	86.70%	n/a	n/a	n/a	85.84%

TABLE IV

EXPERIMENT 4: *"H. sapiens dataset"*. MEAN VALUES OF THE METRICS ACROSS ALL 5 FOLDS.

Method	Accuracy	Precision	Recall	Specificity	MCC	AUC	F1
Our method	$97.98 \pm 0.36\%$	$98.9 \pm 0.43\%$	$96.85 \pm 0.36\%$	$97.17 \pm 0.36\%$	0.960 ± 0.007	0.996 ± 0.002	$97.86 \pm 0.38\%$
Du [5]	$98.08 \pm 0.3\%$	$99.11 \pm 0.19\%$	$96.85 \pm 0.67\%$	$97.18 \pm 0.57\%$	0.962 ± 0.006	0.996 ± 0.001	$97.96 \pm 0.32\%$
Huang [24]	$96.30 \pm 0.1\%$	$99.59 \pm 0.29\%$	$92.63 \pm 0.44\%$	n/a	0.928 ± 0.002	0.965 ± 0.005	n/a
Nanni [30]	63.00%	63.00%	64.00%	n/a	n/a	n/a	n/a
Nanni [31]	70.00%	67.00%	70.80%	n/a	n/a	n/a	n/a

[31] L. Nanni and A. Lumini, "An ensemble of K-local hyperplanes for predicting protein-protein interactions," *Bioinformatics*, vol. 22, pp. 1207–1210, May 2006.