Original Paper

Human Heredity

Hum Hered 2017/2018;83:163–172
DOI: 10.1159/000493215

Published online: January 25, 2019

tRNA-DL: A Deep Learning Approach to Improve tRNAscan-SE Prediction Results

Xin Gao[a] Zhi Wei[a] Hakon Hakonarson[b, c]

[a]Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA; [b]The Center for Applied Genomics, The Children's Hospital of Philadelphia, Philadelphia, PA, USA; [c]Department of Pediatrics, University of Pennsylvania School of Medicine, Philadelphia, PA, USA

Keywords
Deep learning · tRNA prediction · Multilayer neural network · Genomics · Machine learning · tRNAscan-SE improvement

Abstract
Background: tRNAscan-SE is the leading tool for transfer RNA (tRNA) annotation, which has been widely used in the field. However, tRNAscan-SE can return a significant number of false positives when applied to large sequences. Recently, conventional machine learning methods have been proposed to address this issue, but their efficiency can be still limited due to their dependency on handcrafted features. With the growing availability of large-scale genomic datasets, deep learning methods, especially convolutional neural networks, have demonstrated excellent power in characterizing sequence patterns in genomic sequences. Thus, we hypothesize that deep learning may bring further improvement for tRNA prediction. ***Methods:*** We proposed a new computational approach based on deep neural networks to predict tRNA gene sequences. We designed and investigated various deep neural network architectures. We used the tRNA sequences as positive samples, and the false-positive tRNA sequences predicted by tRNAscan-SE in coding sequences as negative samples, to train and evaluate the proposed models by comparison with the conventional machine learning methods and popular tRNA prediction tools. ***Results:*** Using the one-hot encoding method, our proposed models can extract features without involving extensive manual feature engineering. Our proposed best model outperformed the existing methods under different performance metrics. ***Conclusion:*** The proposed deep learning methods can substantially reduce the false positive output by the state-of-the-art tool tRNAscan-SE. Coupled with tRNAscan-SE, it can serve as a useful complementary tool for tRNA annotation. The application to tRNA prediction demonstrates the superiority of deep learning in automatic feature generation for characterizing sequence patterns.

Introduction

Transfer RNA (tRNA) plays a central role in protein translation [1]. Thus, accurate prediction and annotation of tRNA are fundamental for interpreting and improving protein translation in all living cells. The most popular computational algorithm for tRNA prediction, tRNAscan, can be dated back more than 20 years [2]. It develops a seminal covariance model to describe the sec-

KARGER

E-Mail karger@karger.com
www.karger.com/hhe

Zhi Wei
Department of Computer Science, New Jersey Institute of Technology
323 Dr. Martin Luther King Jr. Blvd
Newark, NJ 07102 (USA)
E-Mail zhiwei@njit.edu

ondary structural profile and primary sequence consensus of an RNA sequence. The covariance models essentially are probabilistic secondary structure profiles based on stochastic context-free grammars. Since then, quite a few improved software tools have been developed for tRNA annotation and prediction, such as tRNAscan-SE [3, 4], ARAGORN [5], and ARWEN [6]. Based on homology with recognized tRNA consensus sequences, ARAGORN develops heuristic algorithms to search in silico for tRNA genes and predict the secondary tRNA structure as a cloverleaf diagram [5]. ARWEN focuses on detecting tRNAs from metazoan mitochondrial nucleotide sequences, which are usually degenerate in sequence and structure compared with tRNAs in their bacterial ancestors. It employs a heuristic algorithm that searches for specific hairpin structures [6]. tRNAscan-SE represents a practical application of RNA covariance models which can detect a remarkable number of tRNAs in DNA sequences with good accuracy. After extracting tRNA candidate segments from the input using a modified version of tRNAscan and a Pavesi algorithm [7], tRNAscan-SE passes them to the covariance models [2] for predicting the secondary structural profile of tRNAs. Due to its superior performance, tRNAscan-SE remains the de facto tool for identifying tRNA genes encoded in genomes [3]. More recently, tRNAscan-SE 2.0 [4] has been released, which employs improved covariance model search technology enabled by the Infernal 1.1 software [8].

Although the sensitivity of the covariance model is high in detecting tRNAs, it is not powerful enough to separate tRNAs from pseudo-tRNAs. As a result, a significant number of false positives can still be output from tRNAscan-SE. To address this problem, Zou et al. [9] proposed a machine learning-based method and built an ensemble classifier, LibMutil, to improve the tRNAscan-SE annotations and predictions. They used all the true tRNAs as the positive samples, and the pseudo-tRNAs predicted from coding region sequences by tRNAscan-SE as the negative samples, for training their model. Then they employed a three-feature extraction method to construct a compressed feature set consisting of local sequence, structure sequence, and multilevel features. Using these features, they constructed an ensemble method for classification consisting of eight common classifiers, namely, AdaBoost.M, Bagging, Naive Bayes, Logistic, Random Forest, Random Tree, J48, and K-nearest neighbors (KNN). They showed that their methods could improve the tRNAscan-SE prediction results substantially. However, we note that their methods required in-depth domain knowledge as it involved extensive manual feature engineering in the feature extraction and construction step. In addition, the loss of each nucleotide's essential position information in the sequence would affect the performance of prediction. Thus, in this work, we propose to use the sequences themselves as direct inputs for building a discriminative model to distinguish functional tRNAs from pseudo-tRNAs using a training set comprising all the acknowledged true and pseudo-tRNAs. Deep learning models are the candidates for our consideration.

With the growing availability of advanced computational techniques and large-scale genomic datasets, deep learning-based methods have been proposed to automatically identify and understand genomic regulatory regions directly from raw DNA or RNA sequences, and predict the unknown sequences' profile [10]. Amongst a set of deep neural networks, convolutional neural networks (CNNs) have been seen in many applications for the analysis of genomic sequences, for example, motif discovery [11], DNA binding site prediction [12], HLA class I peptide binding prediction [13], and functional effect prediction of noncoding variants [14, 15], among others [16, 17].

In this paper, we propose a computational method based on deep neural networks for predicting tRNA gene sequences. Our main contributions are summarized as follows:

1. We propose a deep learning approach (tRNA-DL)[1] to separate tRNAs from pseudo-tRNAs; to the best of our knowledge, our proposed model is the first application of deep learning to the tRNA prediction problem
2. We illustrate that the proposed model can automatically acquire problem-related expertise and extract predictive tRNA-related features and patterns in a data-driven manner without requiring researchers' prior knowledge and extracting handcrafted features
3. We investigate various deep learning architectures extensively to evaluate their performance in predicting tRNA
4. We show that the proposed model, without manual feature engineering, can outperform conventional machine learning models, including LibMutil, Support Vector Machine, Adaboost, KNN, and Random Forest; it substantially improves the prediction results of tRNAscan-SE

[1] Codes are freely available on GitHub: https://github.com/stella-gao/trna-dl.

Methods

Background

Recent researches on deep learning have shown that it can solve genomic problems in a more accurate way than traditional machine learning approaches. A deep convolutional network is a category of multilayer neural networks which can model nonlinear relationships and are always organized in a sequential layer-by-layer structure executing a sequence of functional transformations [10].

Deep learning architectures include many variants, of which the two most prominent domains are CNNs and recurrent neural networks (RNNs). CNNs are widely used for computer vision, and RNNs are mainly used for natural language processing [18, 19]. Recent works based on CNNs allowed training directly with the DNA sequences rather than extracting features beforehand [11, 14, 16]. The connections between units inside RNNs can form a directed graph along a sequence, which allowed RNNs to provide another effective and efficient way of extracting features from DNA sequences [15, 20].

In CNNs, the number of model parameters is significantly reduced compared to fully connected neural networks, via convolution operation and parameter sharing. In addition, convolutional layers can extract high-level features from raw sequences. The neurons in the convolutional layer scan for motifs and patterns, which is similar to traditional position weight matrices [21]. The max-pooling layer can be applied to summarize the activations of some specific adjacent neurons by their maximum value and can help reduce overfitting. Fully connected layers are always applied after feature extraction on top of all kinds of layers to create final nonlinear combinations of features.

The two popular networks based on RNNs for language modeling are gated recurrent units (GRU) and long short-term memory units (LSTMs) [22, 23]. GRU layers and LSTM layers are similar to each other. The difference between them is that a GRU has two gates and an LSTM has three gates. Both architectures yield comparable performance but utilize distinct methods to prevent the vanishing gradient problem. GRUs have no memory unit and fewer parameters, and thus they expose the full hidden content without any control and may train a bit faster or need fewer data to generalize. On the other hand, LSTMs may lead to better results when the dataset is large. Sometimes bidirectional LSTMs or bidirectional GRUs, where the two directional neurons do not have any interactions, are used to preserve information from both the past and the future.

Architecture of the tRNA-DL Deep Learning Model

To find out which neural network model worked best for the tRNA prediction task, we examined several different types of model. Based on popular deep learning architectures, we built models of three kinds: a CNN, an RNN, and a hybrid combination of a CNN and an RNN (CNN-RNN). Table 1 shows the abbreviations of the different types of deep neural network layer. The Conv1D, Conv2D, MaxPool1D, MaxPool2D, AvgPool1D, and AvgPool2D layers are deployed for building CNNs. The LSTM, GRU, BDLSTM, and BDGRU layers are deployed for building RNNs. The FC layer can be used in both CNNs and RNNs.

To find out which deep neural network architecture can achieve satisfying performance results, we constructed thirteen different deep neural network architectures. The deployment and connections between layers are shown in Table 2, with their abbreviations listed. Among all these models, the CNNs are CF, CCMF, CMCMF, CCMCAF, CMCMCMF, CCCMF, CMCMCF2, and CCCMF2. CMCMCF2 and CCCMF2 apply two-dimensional CNNs, while the others apply one-dimensional neural networks. Two-dimensional convolution can be considered as sliding one function on top of another, multiplying and adding. Although they are mostly used for image processing, they can also be used in feature extraction from genomic sequences, such as in DeepSEA [14]. RNNs are seldom used alone for training gene sequences; to address this issue, we built one RNN model (LLLF) for evaluation comparison with other the models. The remaining four architectures (CMBLF, CCMBLF, CMBGF, and CCMBGF) belong to the CNN-RNN architecture, for which we got the idea from DanQ [15].

Table 1. Abbreviations of different types of deep neural network layer

Abbreviation	Type of layer
Conv1D	One-dimensional convolutional layer
Conv2D	Two-dimensional convolutional layer
MaxPool1D	Max-pooling layer for temporal data
MaxPool2D	Max-pooling layer for spatial data
AvgPool1D	Average pooling for temporal data
AvgPool2D	Average pooling for spatial data
FC	Fully connected layer
LSTM	Long short-term memory layer
GRU	Gated recurrent unit layer
BDLSTM	Bidirectional wrapper for LSTM layer
BDGRU	Bidirectional wrapper for GRU layer

For the tRNA prediction problem, we selected the model that demonstrated the best prediction potential among the abovementioned architectures. The following section shows that CCMBLF performed best of the thirteen different model architectures. Thus, we selected CCMBLF as our proposed model and named it "tRNA-DL." The proposed model tRNA-DL is shown in Figure 1; the model consists of two convolutional layers and one max-pooling layer to identify predictive motifs from the context of DNA sequences, as well as one bidirectional LSTM layer and one fully connected hidden layer with a rectified linear unit activation function to model motif interactions [10].

Our proposed model takes raw DNA sequences (or RNA sequences) as inputs, which are encoded into bit matrices using the one-hot encoding method, with each nucleotide represented as a 4-element binary vector. The DNA nucleotides are numerically encoded as binary vectors with only one entry set to 1: A = [1, 0, 0, 0], C = [0, 1, 0, 0], G = [0, 0, 1, 0], and T = [0, 0, 0, 1]. Then, a DNA sequence will be represented as an encoded bit matrix, with columns corresponding to A, C, G, and T. With the one-hot encoding method, we can preserve the vital position information of each nucleotide in the DNA sequences. Because we read each sequence in its entirety as input into the model, we can preserve the position of each nucleotide without missing the relationship information between any two nucleotides.

Table 2. Abbreviations of the deep neural network architectures

No.	Abbreviation	Prediction deep learning model architectures
1	CF	Conv1D + FC + SGD
2	CCMF	Conv1D + Conv1D + MaxPool1D + FC + SGD
3	CMCMF	Conv1D + MaxPool1D + Conv1D + MaxPool1D + FC + SGD
4	CCMCAF	Conv1D + Conv1D + MaxPool1D + Conv1D + AvgPool1D + FC + SGD
5	CMCMCMF	Conv1D + MaxPool1D + Conv1D + MaxPool1D + Conv1D + MaxPool1D + FC + SGD
6	CCCMF	Conv1D + Conv1D + Conv1D + MaxPool1D + FC + SGD
7	CMCMCF2	Conv2D + MaxPool2D + Conv2D + MaxPool2D + Conv2D + FC + SGD
8	CCCMF2	Conv2D + Conv2D + Conv2D + MaxPool2D + FC + SGD
9	LLLF	LSTM + LSTM + LSTM + FC + SGD
10	CMBLF	Conv1D + MaxPool1D + BDLSTM + FC + RMSprop
11	CCMBLF	Conv1D + Conv1D + MaxPool1D + BDLSTM + FC + RMSprop
12	CMBGF	Conv1D + MaxPool1D + BDGRU + FC + RMSprop
13	CCMBGF	Conv1D + Conv1D + MaxPool1D + BDGRU + FC + RMSprop

Table 3. tRNA-DL model layer architecture and output layer dimension

Layer No.	Layer type	Size	Output
0	INPUT	–	4×134
1	CONV	16×4×4	16×131
2	RELU	–	16×131
3	DROPOUT	–	16×131
4	CONV	64×4×4	64×128
5	RELU	–	64×128
6	POOL	4×2	64×64
7	BDLSTM	64×4×6	128×64
8	RELU	–	128×64
9	FC	128	128
10	DROPOUT	–	128
11	FC	1	1
12	SIGMOID	1	1

The size column describes the kernel size of the convolutional layer, the window size of the max-pooling layer, and the size of the fully connected layer.

Table 3 shows the proposed architecture with the hyperparameter setting based on the evaluation of the performance on a validation dataset. We set the number of kernels as 16 and 64 in the first and second convolutional layers, respectively. We first utilize a convolutional layer that directly operates on the encoded bit matrix to scan the motif. Next, we apply a max-pooling layer to reduce dimensionality and a second convolutional layer to model the interactions between motifs generated by the previous layers, where the outputs of the convolutional layers are then flattened into vectors and fed to the bidirectional LSTM and the fully connected layer for further extraction. The final results are then converted into probabilities via a "sigmoid" function.

Experiments and Results

Here, we will illustrate the data processing, the libraries and packages used for the training models, the experimental environment, and the numerical evaluation performance of the proposed model compared with both the deep learning models and baseline classic machine learning models.

Data Source and Data Preprocessing

Our goal was to further improve the tRNAscan-SE software's [3] tRNA prediction results with our proposed tRNA-DL deep learning model. Deep learning models mostly rely on their training dataset. Choosing a high-quality negative dataset for our model was significant. Following Zou et al. [9], we adopted the same positive and negative samples they used. We chose false-positive tRNA sequences as negative samples. The same tRNA sequence datasets, consisting of both positive and negative samples selected from the baseline machine learning method-based literature [9], were utilized for our experiments. The lengths of the selected tRNA sequences were in the range of 54–134 bp. The positive samples had 623 tRNA sequences of 104 species, which were the sequences obtained by random sampling from the tRNAdb 2009 database [24]. The database has over 12,000 tRNA gene sequences of 577 species in total. The negative samples were obtained by random sampling from the predicted false-positive tRNA sequences by tRNAscan-SE in the EMBL CDS database [25], which consists of known and predicted coding sequences. The extracted negative samples contained 1,183 false-positive tRNA sequences, where re-

dundant sequences had been discarded through CD-HIT software with a parameter c = 0.9 [26].

We divided the entire dataset into training, validation, and test sets and made sure there was no overlap between them. The ratio between them was 7:2:1. We used the training and validation datasets to select the best-performing deep neural network model. The test dataset was used later for prediction performance comparison with the baseline machine learning methods and existing programs and tools. Note that the validation dataset was then partitioned into a sub-validation dataset and a sub-test dataset for evaluating the performance of the different deep learning models. During the training process, the sub-validation dataset helped us tune the weights and parameters that the training dataset learned. Later, the sub-test dataset could show the performance results based on 90% of the entire datasets, after which the best model was selected among them. The test dataset was adopted to evaluate the performance of our selected tRNA-DL model against the baseline machine learning methods and other prediction tools.

Model Training and Testing

In this part, we introduce how we applied and tuned different hyperparameters to train our deep learning models so that we could find the model weights with the best evaluation performance on the validation dataset during model training. We aimed to discover the weights that can minimize the objective function for the models and further evaluate the performance against the other machine learning models and popular deep learning models on the test dataset. We applied standard stochastic gradient descent, Adam [27], and RMSprop optimizers to train the model. The training performance, as well as the training speed, significantly rely on parameter initialization, the learning rate, and the batch size of stochastic gradient descent. The optimum batch size has a relationship to the learning rate; that is, larger batch sizes commonly require smaller learning rates. We also employ a low learning rate decay in model training, which helps explore confined parameter regions of the objective function and avoids overshooting. Diverse learning rates were explored on a logarithmic scale, such as 0.1, 0.01, and 0.001, with 0.01 as the initial training value. Through preliminary experiments, a learning rate of 0.001, a batch size of 128, a momentum rate of 0.9, and an advanced adaptive learning rate according to Adam [27] were found to be suitable for our application. Techniques such as dropout [28], batch normalization [29], and "early stopping" were employed to prevent overfitting. Dropout is

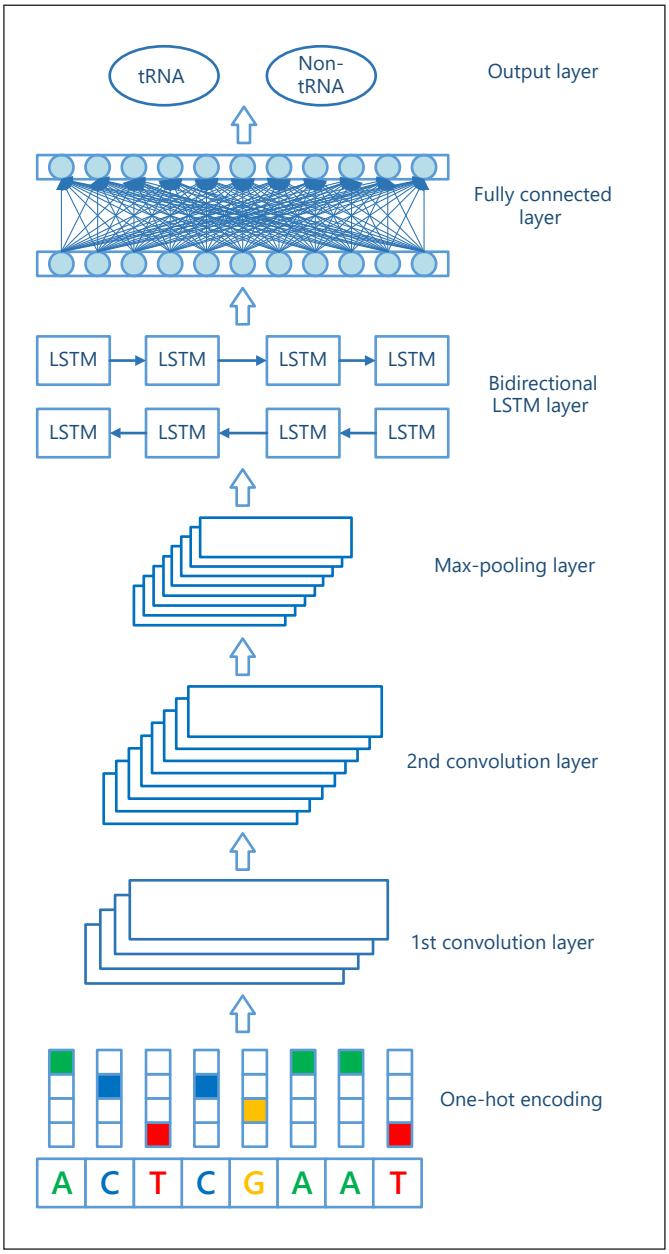

Fig. 1. The chosen deep learning model with the best performance on the validation set. tRNA, transfer RNA; LSTM, long short-term memory.

the most common regularization technique and often used to reduce overfitting. Another popular regularization method we used for training the models is early stopping, which will stop the training process as soon as the performance on a validation dataset starts to saturate. During the training process, the parameters will be updated when a better performance on the validation set

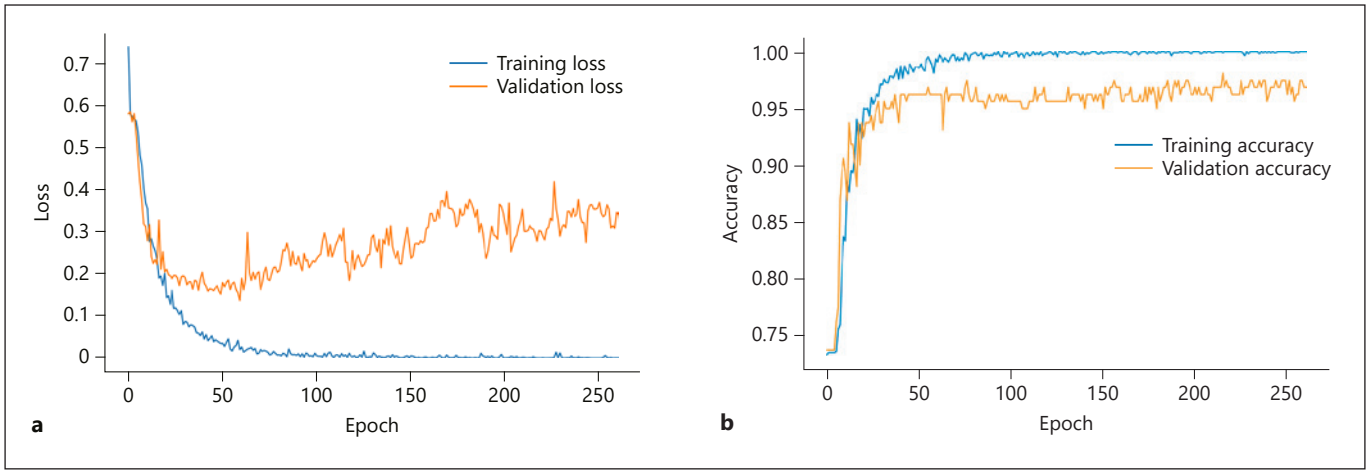

Fig. 2. Loss values (**a**) and accuracy values (**b**) during model training.

shows up. Then, as soon as training has stopped, the latest weights that have the best evaluation performance on the validation dataset are saved for further evaluation.

Hyperopt [30] is a Python library for model selection and hyperparameter optimization which is based on Bayesian optimization. We applied the Hyperopt package to tune the hyperparameters for our models. We first randomly picked 5–6 values within a large range, with the same interval between each value, and then narrowed down the range as much as possible. We repeated this procedure to finally find the hyperparameters that were suitable for our models. Figure 2 illustrates the loss and accuracy values, respectively, during model training. The model weights were updated and saved as soon as there was a better performance. The training process stopped at about 265 epochs, since the best weights were obtained at 65 epochs. There were no updates on the weights after 200 epochs. Then, to avoid overfitting and to save time, we stopped the training and evaluated our models.

We implemented all the deep learning models using the Keras library (version 2.0.8) [31] in the Python programming language based on a Theano (version 0.9.0) [32] back end, which optimizes mathematical computations and provides efficient low-level implementations. Keras is a dominant high-level Python API built on top of Theano [32] or TensorFlow [33] for developing and training deep learning models. The experiments were run on a Linux server with two Intel E5-2620 CPUs and 4-T K80 GPUs (4992 NVIDIA CUDA cores and 24 GB of GDDR5 memory).

Prediction Assessment

Metrics, including accuracy, Matthew's correlation coefficient, sensitivity or recall, specificity, F-score, and area under the receiver operating characteristic curve (AUC), are used to evaluate competing methods. We considered the tRNA sequences as positive samples and the false-positive tRNA sequences as negative samples. "True positive" and "true negative" are the numbers of correctly predicted tRNA sequences and pseudo-tRNA sequences, respectively; "false positive" and "false negative" are the numbers of incorrectly predicted tRNA sequences and pseudo-tRNA sequences, respectively. Sensitivity (true positive rate) measures the fraction of the true-positive samples that are correctly predicted. Specificity (true negative rate) measures the fraction of the predicted tRNA samples that is correct amongst those predicted. Accuracy measures an average performance on the positive and negative samples. However, this measure does not consider the possible difference between positive and negative samples. Matthew's correlation coefficient is especially useful when the two classes are of very different sizes and can measure the relation between correctly predicted positives and negatives as well as false positives and negatives. Considering all possible thresholds, the AUC is commonly used as a baseline and useful metric for binary classification. We adopted the AUC to assess the evaluation, performance, and usability of the different deep neural network models. The F-score can be interpreted as the harmonic mean of precision and recall. The higher the F-score, the higher are both precision and recall.

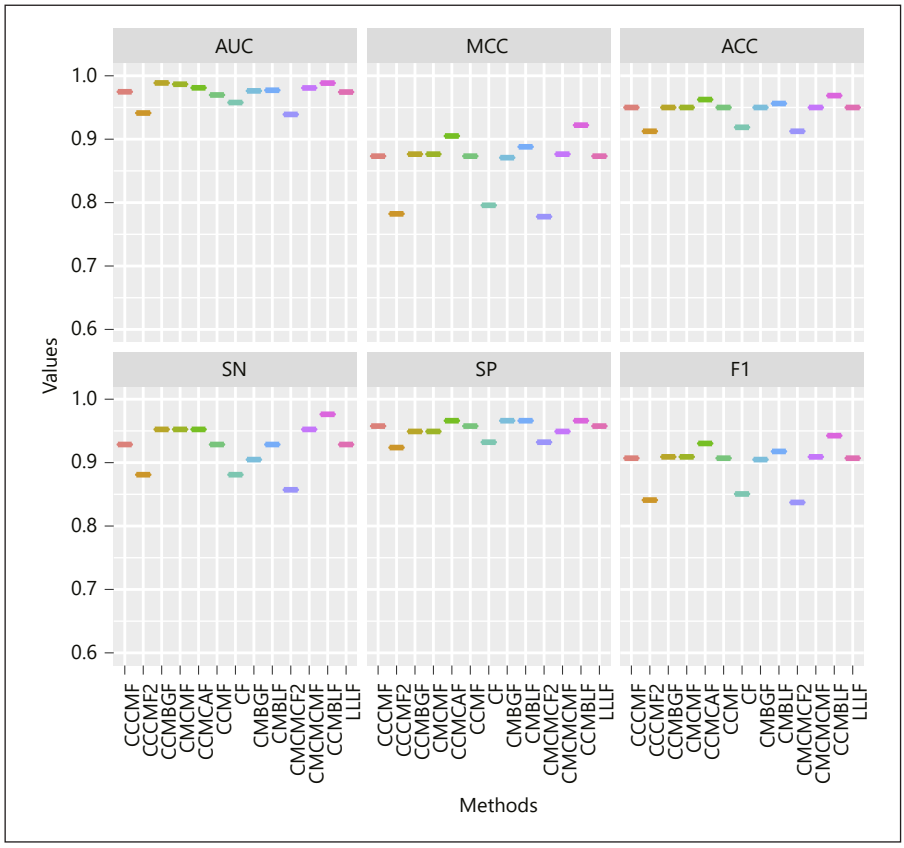

Fig. 3. Prediction performance of various deep learning models. Performance was measured using boxplots by 6 evaluation metrics, including the area under the receiver operating characteristic curve (AUC), accuracy (ACC), recall (SN), specificity (SP), Matthew's correlation coefficient (MCC), and F-score (F1). The lower and upper hinges are the 25th and 75th percentiles, respectively. For the model abbreviations, see Table 2.

Experiments on Deep Learning Model Selection

Figure 3 shows the prediction performance of all the thirteen deep neural network models mentioned above. In the performance assessments of the thirteen constructed deep neural network methods, we discovered that the CCMBLF model performed best, which is why it was selected as our proposed model tRNA-DL.

The CNN layers were used to extract sequence patterns which are related to the vital position information. The RNN layers, especially the bidirectional LSTM layers, were applied on top of the CNN layers so as to extract the inner relation between adjacent nucleotides or even between the beginning part and ending part of the entire sequence. Figure 4 illustrates the AUC values of all the thirteen deep neural network models and also of the classic machine learning models. It is clear that our tRNA-DL model achieved a higher AUC value than the other models. From this, we can deduce that a hybrid combination of CNN and RNN models demonstrates the best performance and will always achieve a better performance than CNN models. The models that used bidirectional LSTM layers performed better than the models using one-directional LSTM layers, one-directional GRU layers, or bidirectional GRU layers.

Comparison with Existing Machine Learning Methods

We compared our proposed tRNA-DL method with the state-of-the-art machine learning methods, including the Support Vector Machine algorithm with a linear kernel, the Random Forest algorithm with 50 trees, the KNN algorithm with K = 4, the AdaBoost algorithm with a J48 base classifier, and a LibMutil ensemble classifier [9]. All the machine learning methods are implemented in the Weka [34] software. As shown in Table 4, our selected deep learning model tRNA-DL achieved a remarkable and stable prediction performance on the test dataset that outperformed all the classic machine learning methods (including the baseline method LibMutil) in all metrics.

Comparison with Existing Software and Tools

To validate our proposed tRNA-DL model and evaluate its effectiveness, we compared the performance results with those of three other tools: tRNAscan-SE, ARWEN, and ARAGORN. The results based on six evaluation met-

Table 4. Prediction performance (%) of tRNA-DL compared with existing machine learning methods

Metrics	SVM (linear)	KNN (K = 4)	RF (n = 50)	AdaBoost (J48)	LibMutil	tRNA-DL
AUC	82.3	82.5	97.6	95.4	97.5	99.8
MCC	63.7	44.2	75.7	77.9	81.1	96.4
ACC	84.3	74.7	89.8	90.7	92.0	98.4
SN	84.3	74.7	89.8	90.7	92.0	96.8
SP	80.2	71.5	80.0	84.8	88.7	99.2
F1	84.4	75.3	89.4	90.6	92.0	97.6

SVM, Support Vector Machine; KNN, K-nearest neighbors; RF, Random Forest; AUC, area under the receiver operating characteristic curve; MCC, Matthew's correlation coefficient; ACC, accuracy; SN, recall/sensitivity; SP, specificity; F1, F-score.

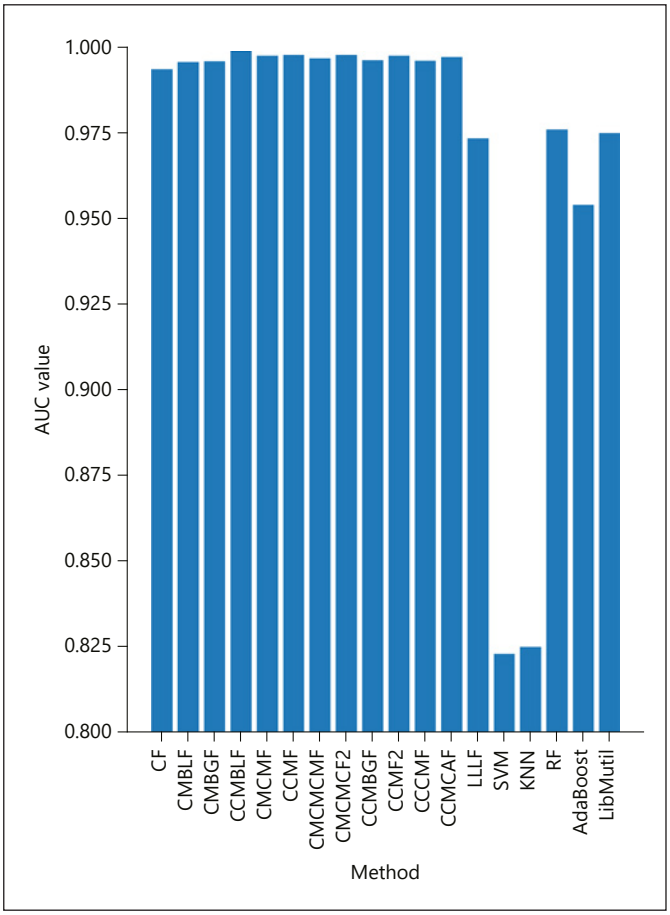

Fig. 4. Area under the receiver operating characteristic curve (AUC) values for all models. For the model abbreviations, see Table 2. SVM, Support Vector Machine; KNN, K-nearest neighbors; RF, Random Forest.

Table 5. Prediction performance (%) of tRNA-DL compared with existing tRNA prediction tools

Metrics	tRNAscan-SE	ARWEN	ARAGORN	tRNA-DL
AUC	53.1	52.1	57.7	99.8
MCC	10.4	10.0	20.6	96.4
ACC	40.1	37.9	46.2	98.4
SN	95.2	98.4	95.2	96.8
SP	10.9	5.9	20.2	99.2
F1	52.4	52.3	52.4	97.6

AUC, area under the receiver operating characteristic curve; MCC, Matthew's correlation coefficient; ACC, accuracy; SN, recall/sensitivity; SP, specificity; F1, F-score.

rics are shown in Table 5. We discovered that tRNA-DL dramatically outperformed the other three tools when it came across false-positive tRNA sequences. Overall, the results indicate that our tRNA-DL model can detect false-positive tRNA sequences and may help improve the tRNAscan-SE tool's prediction results a lot.

Conclusions and Future Work

Because of the unacceptable false positive rate of tRNAscan-SE for a significant amount of sequences, we introduced a deep learning approach to predict tRNAs so as to further improve the annotation results of tRNAscan-SE. The benefit of introducing a deep learning approach is the automatic and accurate extraction of satisfactory features without involving extensive manual feature engineering for a better prediction performance.

We obtained positive samples from the tRNAdb 2009 database and false-positive tRNA sequences predicted by tRNAscan-SE from coding sequences as negative samples. The positive and negative samples we used were the same as those used by Zou et al. [9]. After removing redundant data, we transferred the tRNA genomic sequences into encoded bit matrices using the one-hot encoding method. We designed thirteen competing deep learning models of various architecture, trained these models, and chose the one with the best evaluation performance as our model.

We discovered that bidirectional LSTM layers performed better than others in identifying tRNA sequence patterns when comparing the classification performance with that of baseline machine learning methods across several meaningful metrics. The evaluation results demonstrate that the chosen model outperformed the competing machine learning methods as measured by its higher accuracy and AUC score.

In summary, the proposed deep learning method is a promising tool for tRNA classification and prediction. Pursuing this approach shows great potential for solving other related problems in computational biology, which will be considered in our follow-up studies.

Disclosure Statement

The authors declare that they have no competing interests.

Author Contributions

Z.W. and H.H. conceived the study. X.G. designed the algorithms, implemented the models, and conducted the experiments. All authors wrote and approved the final manuscript.

References

1 Sprinzl M, Hartmann T, Weber J, Blank J, Zeidler R: Compilation of tRNA sequences and sequences of tRNA genes. Nucleic Acids Res 1989;17(suppl):r1–r172.
2 Eddy SR, Durbin R: RNA sequence analysis using covariance models. Nucleic Acids Res 1994;22:2079–2088.
3 Lowe TM, Eddy SR: tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. Nucleic Acids Res 1997; 25:955–964.
4 Lowe TM, Chan PP: tRNAscan-SE On-line: integrating search and context for analysis of transfer RNA genes. Nucleic Acids Res 2016; 44(W1):W54–W57.
5 Laslett D, Canbäck B: ARAGORN, a program to detect tRNA genes and tmRNA genes in nucleotide sequences. Nucleic Acids Res 2004;32:11–16.
6 Laslett D, Canbäck B: ARWEN: a program to detect tRNA genes in metazoan mitochondrial nucleotide sequences. Bioinformatics 2007; 24:172–175.
7 Pavesi A, Conterio F, Bolchi A, Dieci G, Ottonello S: Identification of new eukaryotic tRNA genes in genomic DNA databases by a multistep weight matrix analysis of transcriptional control regions. Nucleic Acids Res 1994;22:1247–1256.
8 Nawrocki EP, Eddy SR: Infernal 1.1: 100-fold faster RNA homology searches. Bioinformatics 2013;29:2933–2935.
9 Zou Q, Guo J, Ju Y, Wu M, Zeng X, Hong Z: Improving tRNAscan-SE annotation results via ensemble classifiers. Mol Inform 2015;34: 761–770.
10 LeCun Y, Bengio Y, Hinton G: Deep learning. Nature 2015;521:436–444.
11 Alipanahi B, Delong A, Weirauch MT, Frey BJ: Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nat Biotechnol 2015;33:831–838.
12 Zeng H, Edwards MD, Liu G, Gifford DK: Convolutional neural network architectures for predicting DNA-protein binding. Bioinformatics 2016;32:i121–i127.
13 Vang YS, Xie X: HLA class I binding prediction via convolutional neural networks. Bioinformatics 2017;33:2658–2665.
14 Zhou J, Troyanskaya OG: Predicting effects of noncoding variants with deep learning-based sequence model. Nat Methods 2015;12:931–934.
15 Quang D, Xie X: DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. Nucleic Acids Res 2016;44:e107.
16 Kelley DR, Snoek J, Rinn JL: Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. Genome Res 2016;26:990–999.
17 Zhou J, Lu Q, Xu R, Gui L, Wang H: CNNsite: prediction of DNA-binding residues in proteins using convolutional neural network with sequence features. Proceedings (IEEE Int Conf Bioinformatics Biomed) 2016;2016: 78–85.
18 Krizhevsky A, Sutskever I, Hinton GE: ImageNet classification with deep convolutional neural networks. Adv Neur Inform Proc Syst 2012;2012:1097–1105.
19 Mikolov T, Karafiát M, Burget L, Černockỳ J, Khudanpur S: Recurrent neural network based language model. 11th Annu Conf Int Speech Commun Assoc 2010;2010:1045–1048.
20 Liu X: Deep recurrent neural network for protein function prediction from sequence. arXiv:1701.08318v1 (Jan 28, 2017).
21 Stormo GD, Schneider TD, Gold L, Ehrenfeucht A: Use of the "Perceptron" algorithm to distinguish translational initiation sites in *E. coli*. Nucleic Acids Res 1982;10:2997–3011.
22 Hochreiter S, Schmidhuber J: Long short-term memory. Neural Comput 1997;9:1735–1780.
23 Chung J, Gulcehre C, Cho K, Bengio Y: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv: 1412.3555v1 (Dec 11, 2014).
24 Jühling F, Mörl M, Hartmann RK, Sprinzl M, Stadler PF, Pütz J: tRNAdb 2009: compilation of tRNA sequences and tRNA genes. Nucleic Acids Res 2008;37(database issue):D159–D162.
25 Stoesser G, Baker W, van den Broek A, Camon E, Garcia-Pastor M, Kanz C, et al: The EMBL Nucleotide Sequence Database. Nucleic Acids Res 2002;30:21–26.
26 Li W, Godzik A: Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. Bioinformatics 2006;22:1658–1659.
27 Kingma DP, Ba J: Adam: a method for stochastic optimization. arXiv:1412.6980v9 (Jan 30, 2017).

28 Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR: Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580v1 (Jul 3, 2012).
29 Ioffe S, Szegedy C: Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167v3 (Mar 2, 2015).
30 Bergstra J, Cox DD: Hyperparameter optimization and boosting for classifying facial expressions: how good can a "Null" model be? arXiv:1306.3476v1 (Jun 14, 2013).
31 Chollet F, et al: Keras. 2015. https://github.com/fchollet/keras.
32 Theano Development Team: Theano: a Python framework for fast computation of mathematical expressions. arXiv: 1605.02688v1 (May 9, 2016).
33 Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al: TensorFlow: large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467v2 (Mar 16, 2016).
34 Holmes G, Donkin A, Witten IH: WEKA: a machine learning workbench. Proceedings (IEEE 2nd Aust NZ Conf Intelligent Inf Syst) 1994:1994:357–361.