Deep Learning for Sequence Pattern Recognition

Xin Gao
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ, USA
xg54@njit.edu

Jie Zhang
Data Science
Adobe Systems Inc.
San Jose, CA, USA
jiezhan@adobe.com

Zhi Wei
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ, USA
zhi.wei@njit.edu

***Abstract*—Deep Learning is a superb way to solve remote sensing related problems, which mainly cover four perspectives: image processing, pixel-based classification, target recognition and scene understanding. In this paper, we focus on target recognition by building deep learning models, and our target is sequence pattern. Accurate prediction of sequence pattern would help identify significant characters from text sequence. Despite considerable advances in using machine learning techniques for sequence pattern recognition problem, its efficiency is still limited because of its involving extensive manual feature engineering in the process of features extraction from raw sequences. Thus, we apply a deep learning approach in sequence pattern recognition problem. The sequences of the datasets we used are self-generated genomic format sequences, and each dataset is generated based on a kind of pattern. We then investigate and construct various deep neural network models (such as convolutional networks, recurrent networks and a hybrid of convolutional and recurrent networks). The one-hot encoding method that preserves the vital position information of each character is presented to represent sequences as inputs to the models. The sequence patterns are then extracted from the input and output the probabilities of the existence of sequence patterns. Experimental results demonstrate that the deep learning approaches can achieve high accuracy and high precision in sequence pattern recognition. In addition, a saliency-map-based method is applied to visualize the learned sequence patterns. In view of the simulation results, we believe that we can find an appropriate deep learning model for a certain sequence sensing problem.**

***Index Terms*—deep learning; machine learning; sensing; sequence pattern; feature extraction**

I. Introduction

Traditional machine learning methods become popular and have been used in many research areas [1]–[3]. However, they use predefined features to represent sequences, which requires in-depth domain knowledge and involves extensive manual feature engineering. In addition, the loss of each character's essential position information in the sequence would affect the performance of prediction [4], [5]. With the growing availability of large-scale datasets and advanced computational techniques, a lot of research works apply deep learning models to understand genome regulatory instructions directly from gene sequences without pre-defined features, to predict the unknown sequences profile [6]–[8]. In view of this, we consider utilizing deep learning models for a more general problem: sequence pattern recognition. We intend to use the sequences themselves as inputs for training. We generated several datasets of sequences with different patterns and trained the three constructed deep learning models based on the simulated datasets.

Deep learning has been widely used and has become one of the effective methods to detect complicated patterns from feature-rich datasets. A special deep learning model - convolutional neural network (CNN) can achieve superb results in computer vision, natural language processing (NLP), and speech recognition because of its effective and efficient feature extraction capability on highly challenging datasets [9], [10]. CNN architectures are appropriate for multi-dimensional and high-dimensional data and have already been applied as the premier model in piles of genomics problems, such as motif or functional activity discovery [6], [7], [11], [12]. Alternative deep learning architectures could also be considered, such as recurrent neural network (RNN) model and the hybrid combination of CNN and RNN (CNN-RNN) model. For example, DanQ is a hybrid framework that combines convolutional and bi-directional long short-term memory recurrent neural network for sequence pattern discovery [8]. In RNN, the connections between units can form a directed cycle, which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs allow modeling arbitrary sequences of inputs and capturing long-range interactions within the sequence, such as text, speech, protein or genomic sequences. Suitable for modeling sequential data, RNNs have been widely applied to solve a variety of machine learning problems in natural language processing(NLP), such as translation, named entity recognition and sentiment analysis. It is more challenging to train RNNs than CNNs because that an incorrect parameter initialization can lead to vanishing or exploding gradients. Various complex models can be built by combining different architectures. RNNs and CNNs can be combined in many kinds of ways. For example, CNNs can be combined with RNNs for training image data, where CNNs encode the images and RNNs generate the corresponding image description. Here, we present a general architecture with convolutional feature extractors applied on the input followed by RNNs applied on top of the output of CNNs and then followed by fully connected layers on top of the output of RNNs. In the following paragraphs, we show the performance of these models based on several datasets of different sequence patterns.

The models we applied do not separate the feature extraction

978-1-5386-5053-0/18/$31.00 ©2018 IEEE

and model training. They learn predictive sequence patterns in a data-driven manner. In this work we make three contributions: firstly, we apply deep learning to recognize different kinds of sequence patterns, which is a novel approach for sequence pattern recognition. Secondly, it is shown that deep learning models could automatically learn patterns without involving extensive manual feature engineering, which really saves a lot of efforts. The models first obtain low-layer patterns from sequences and then form high-level complex features through nonlinear transformation layers. Furthermore, we apply a saliency map to visualize the learned sequence patterns. The third contribution is shown that our proposed models can achieve high accuracy and high AUC. We first train the model on the datasets of self-generated sequences and then fine-tune the model parameters to obtain optimal models. We then evaluate the models by six performance measurements: AUC, MCC, accuracy, Sn, Sp and F-score. This can help us find an appropriate deep learning model for a special sequence pattern.

The rest of the paper is organized as follows: in Section II, we introduce the background, present the architectures and explain technical details. Furthermore, the raw data preprocessing method, experimental settings and results are reported in Section III. Finally, Section IV concludes the paper and provides an outlook to future work.

II. Methods

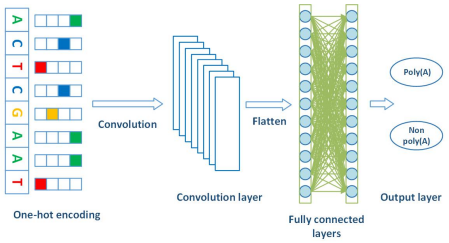
A. Background

CNN is a kind of multilayer neural networks, which executes a sequence of functional transformations through a sequential layer-by-layer structure. In CNNs, the number of parameters is less than a fully connected network by applying convolutional operations to several small parts of the input with parameters shared between them. The neurons of the convolutional layer can search for pattern segments and combinations. The deeper layers can inform the convolutional layer which patterns are most meaningful and relevant. A convolutional layer consists of multiple maps of neurons, which are called feature maps or kernels. Distinctive feature maps are integral components of the layered architecture and might capture short and recurring sequence patterns. Generally, a kernel refers to an operator applied to the entirety of the processed input sequence data such that it transforms the encoded information. There are a set of kernels convolved across the input volume to gain a lower-dimensional output, which is called "activation maps". The distance that the kernel is moved across the input data from the previous layer is the stride, which is often equal to the size of the receptive field to avoid any overlap. They follow a sequence of one or more convolutional layers and are intended to consolidate the features learned and expressed in the previous layers' feature map. The max pooling layer is used to summarize the activations of a number of adjacent neurons by their maximum value, which can help reduce the overfitting of the training data for the model. Fully connected layers are utilized at the end of the network after feature extraction and consolidation, which are performed by the convolutional and pooling layers to create final nonlinear combinations of features. Batch size is the number of samples' each time we fetch from the training set. More training data are generated by looping through the complete genomic sequence data with multiple passes, which are called "epochs". Using dropout [13] as regularization, the network can be less sensitive to the specific weights of neurons so that multiple independent internal representations can be learned by the network. This makes the network better generalized and less likely to overfit the training data. During the training processing, dropout can be applied to reduce overfitting through inactivating a proportion of neurons. Usually, a sigmoid function is used to model binary outputs, while a softmax function is to model categorical outputs. RNNs are a network of neuron-like nodes with each one has a directed connection to every other node. Long Short Term Memory network (LSTM) [14] is a special kind of RNN, which can learn long-term dependencies. Just like all RNNs, LSTM has the form of a chain of repeated modules of the neural network. But the module has a different structure.

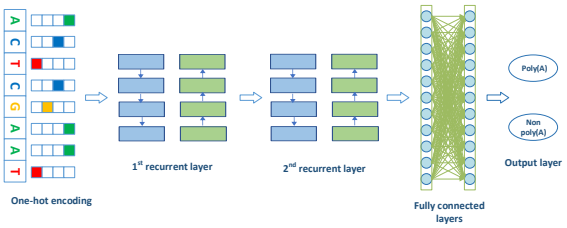
B. Model Construction

For the sequence pattern recognition problem, we illustrate three deep learning models: a CNN model, which consists of one convolutional layer to identify predictive characters from the context of genomic sequences and two fully connected hidden layers to model their interactions; a RNN model, which consists of two LSTM layers and two full connected layers with each LSTM layer contains 16 neurons; a CNN-RNN model, which consists of two convolutional layers, one max-pooling layer, one LSTM layer and two full connected layers. For example, in CNN, the convolutional layer operates directly on the bit matrices, where the convolutional filters scan for motifs. Fully-connected layers with dropout perform a linear transformation to a vector of 64 elements that represents the target cells. While in CNN-RNN model, the number of convolutional-pooling layers is a hyperparameter that can be tuned according to the performance evaluated on the validation set. With several convolutional and pooling operations, the model is designed to extract features from high-dimensional inputs while keeping the number of model parameters tractable. Each of these layers is accompanied by a sub-sampling layer, which is for extracting features from representation matrixes of sequences. Subsequent max-pooling layer and batch normalization layer are used for dimension reduction and convergence acceleration. We set the pooling size as 2 and the stride as 2 to ensure that there is no overlap. Additional convolutional layers model the interaction between motifs in previous layers and obtain high-level features. The extracted features were then transformed into a vector that represents the targets of a fully connected neural network layer which contains 64 hidden neurons with a rectifier activation function. A task-specific activation function "sigmoid" maps this vector to the range [0,1], where the elements serve as probability predictions of

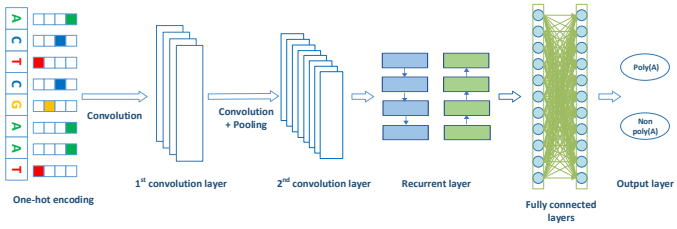
sequence patterns existence, to be compared via a loss function to the true value vector. The encoded bit matrix is passed to a one-dimensional convolutional layer, which acts as a motif scanner across the input matrix and computes the activation of multiple convolutional filters at every position within the DNA sequence window. Then an output matrix with a row for each one-dimensional convolutional kernel is produced. Fig 1 shows the graphical illustration of these models. Model parameters are initialized randomly following the approach in the paper [15]. Model hyperparameters were optimized on the validation set by random sampling. Validation loss is measured by each epoch's training to monitor convergence. Dropout with different dropout rates for the sequence was used for additional regularization Dropout is not suitable to be located in the last fully connected layer, which can lead to the loss of some significant features. Thus we applied dropout between the hidden layers.



(a) CNN model



(b) RNN model



(c) CNN-RNN model

Fig. 1. A graphical illustration of the CNN, RNN and CNN-RNN model

III. Experiments and Results

In this section, we describe the experimental settings, the platform, and packages used for building models as well as the evaluation performance comparing with the acknowledged state-of-the-art approaches.

A. Data Source and Data Preprocessing

Most of the successful deep learning methods need sufficient labeled training samples for supervised because better performance can usually be achieved based on more data with informative features. The sequence datasets utilized in our experiments are generated based on three kinds of patterns, as shown in Table 1. For example, the first pattern can be "AAAAAAAA" with a fixed starting position for all the sequences. The second pattern can be "AAAAAAAA" with a different starting position among all the sequences. The starting position is randomly generated within the range [0, 86]. The third pattern can be "ATGCCGTA" or "TACGTAATGCAT" with a fixed starting position for all the sequences. But the characters are not fixed and they can be any palindrome format string (each character can only be A, C, G or T), which means the palindrome pattern in each sequence can be totally different. To generate the sequence datasets, we first generate 400k sequence samples for each sequence pattern, which is for generating positive samples. Positive samples are generated by random sampling from each 400k dataset with a certain sequence pattern. We also generate 400k sequence samples randomly with no patterns, from which negative samples are generated by random sampling. The number of both positive samples and negative samples are 200k. All experimental samples are sequences with a length of 100 base pairs and belong to "Positive" or "Negative" class. Samples in "Positive" class contain regions wrapping around sequence patterns. In contrast, samples in "Negative" class do not contain them.

Using one-hot encoding method, the raw sequence is then encoded into a bit matrix with each character in the sequence represented as a four-element binary vector. To make it simple, in our simulation experiment, we assume the sequence is genomic format sequence, which means each sequence has four kinds of characters: A, C, G, T. We encoded each character numerically as one of the binary vectors: A=[1,0,0,0], C=[0,1,0,0]; G=[0,0,1,0]; T=[0,0,0,1]. Then each sequence can then be represented as an encoded bit-matrix 4×100, with columns corresponding to A, C, G and T. With this method, the vital position information of each character can be preserved in sequences.

In machine learning, datasets need to be split as training data, validation data and test data to avoid overfitting and assure that the model will generalize to new data. We holdout validation dataset and partition our entire dataset into a training set, a validation set, and a test set. Weights and parameters of the models are both learned from the training set and evaluated on the validation set. The model which performs best will then evaluated on the test set to quantify the performance.

B. Model Training and Testing

After data preparing, we applied the designed deep learning models with different architectures in Section 2 on our prepared data. The goal of model training is to find the parameters that can minimize the objective function. However, it is challenging due to its high-dimensional and non-convex. The objective function can measure the fit between the predictions and the actual observations. In general, model parameters should be initialized randomly to avoid local optima determined by a fixed initialization [15], [16]. To train the models,

TABLE I
SETTING OF THE THREE KINDS OF SEQUENCE PATTERNS

No.	Pattern Parameters		
	Patterns' lengths	***Characters***	***Positions***
1	length [6,14]	consecutive same characters (e.g. AAAAAA)	starting position of pattern is fixed
2	length [6,14]	consecutive same characters (e.g. AAAAAA)	starting position of pattern is randomly generated
3	length [8,14]	randomly generated palindrome string (e.g. AGTCCTGA)	starting position of pattern is fixed

we need to tune the parameters (such as the number of hidden neurons, the number of kernels, the kernel size, the dropout rate, the learning rate, etc.) based on the performance of validation dataset. During the training, dropout regularization, ReLU activations, and batch normalization were typically used to optimize the update procedure. Then the model with the best performance on the validation set is chosen. Thus, the models we trained using the stochastic gradient descent optimizer with a minibatch size of 128 to minimize the average multi-task binary cross entropy loss function on the training set. The models will evaluate the average multi-task cross entropy loss on the validation set at the end of each epoch to monitor the progress of training. We set the parameters to schedule the training times of the models for 1000 epochs but may early stop if the validation loss did not decrease over 100 consecutive epochs. Dropout is the most common regularization technique and often one of the key ingredients to training deep models. The dropout rate corresponds to the probability that a neuron is dropped out, where 0.5 is a sensible default value. Between these two convolutional layers, we used dropout to randomly exclude 40% of neurons in the layer in order to reduce overfitting. The training speed will be increased as the batch size increased and the memory usage decreased as the batch size decreased. It is important for training complex models on memory-limited GPUs. Various learning rates are usually explored on a logarithmic scale such as 0.1, 0.01, or 0.001, with 0.01 as initial training value [17]. The optimal learning rate and batch size usually affect each other, for instance, larger batch sizes always require smaller learning rates. Learning rate decay over each update during training. In our simulation experiment, the loss function was optimized by mini-batch stochastic gradient descent with a batch size of 128 and a global learning rate of 0.01. The learning rate was adopted by SGD and decayed by a default factor of 0.9 after each epoch.

Here, training is stopped as soon as the validation performance starts to saturate or deteriorate, and the parameters with the best performance on the validation set are chosen. Since the best hyperparameter configuration is data and application dependent, models with different configurations should be trained and their performance should be evaluated on a validation set. As the number of configurations grows exponentially with the number of hyperparameters, trying all of them is impossible in practice. It is therefore recommended to optimize the most important hyperparameters such as the learning rate, batch size, or length of convolutional filters independently via line search (trying different values while keeping all other hyperparameters constant). The refined hyperparameter space can then be further explored by random sampling, and settings with the best performance on the validation set are chosen. Hyperopt [18] can automatically explore the hyperparameter space using Bayesian optimization, which we applied to tune the parameters for our models.

Several popular deep learning frameworks, such as Theano, TensorFlow, Caffe, and Torch, have been developed for building neural networks from existing module on a high level. The frameworks differ from each other by their modularity, ease of use and the way models are defined and trained. Our approach was built in Keras 2.0.8, running on top of a Theano 0.9.0 backend. Keras [19] is a a neural networks python library which runs on top of Theano or TensorFlow. Keras provides a highly modular neural networks API to develop and evaluate deep learning models. Keras is mainly designed for CNN and RNN. Theano [20] is a linear algebra compiler which optimizes mathematical computations and provides efficient low-level implementations. With Keras, we can quickly define and implement different complex deep learning models. Our prediction models were implemented in Python.

C. Prediction Assessment

Six major performance metrics are used to evaluate the prediction method. They are Sensitivity/ Recall (Sn) [21], Specificity (Sp), Precision, Accuracy (ACC), Matthew's correlation coefficient (MCC), and F-score. By applying the recognition function we identify correctly TP (true positive) samples and TN (true positive) samples. At the same time, FP (false positive) positive samples were incorrectly classified as negative samples and FN (false negative) negative samples were incorrectly classified as positive samples. S_n measures the fraction of the true positive samples which are correctly predicted. S_p measures the fraction of the predicted positive samples which are correct amongst those predicted. Accuracy (ACC) measures the average of positive and negative samples, without considering the possible difference inside them. A better measurement MCC considers the relation between correctly predictive positives and negatives as well as false positives and negatives. MCC measures a correlation coefficient between trained and tested datasets. In addition, we adopt AUC (area under the receiver operating characteristic curve) to assess the performance as well. AUC is the expectation that a uniformly drawn random positive is ranked before a uniformly drawn random negative, which is commonly used as a baseline to determine if the model is useful. F-score is applied as the weighted harmonic mean of precision and recall. The higher

f-score is, the higher both precision and recall are. For each performance metric, we considered the sequences containing sequence patterns as the positive class and the sequences with no sequence patterns as the negative class. As a binary classification, positive sequences specificity corresponds to the negative sequences sensitivity (and conversely).

$$S_n = \frac{TP}{TP + FN} \tag{1}$$

$$S_p = \frac{TN}{TN + FP} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Accuracy = \frac{TP + TN}{TN + FP + TP + FN} \tag{4}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TN + FN)(TN + FP)(TN + FP)}} \tag{5}$$

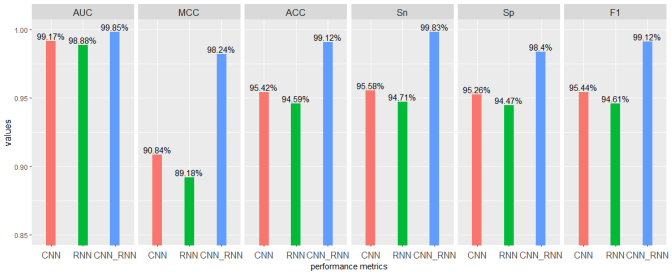
$$AUC = 0.5 \times (S_n + S_p) \tag{6}$$

$$F_{score} = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{7}$$

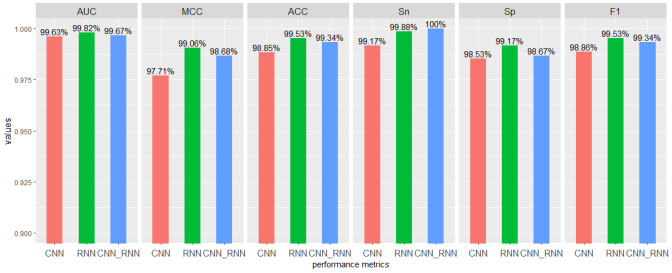
D. Experiments on Model Comparison

To achieve a fair comparison, we used the same datasets for training and compared the performance among the three proposed deep learning models: a regular CNN model, a regular RNN model and a CNN-RNN combined model. All models were fit on the training set, hyperparameters were optimized on the validation set, and the final model performance was reported on the test set. Since all the three models can get an accuracy and precision at 100% on the first sequence pattern, we only show the performance results of the second pattern and the third pattern. The summary of experimental results evaluated based on the prediction assessments is shown in Fig 2. To see the results clearly, we set the y-axis from 0.85 in the 2nd pattern and 0.9 in the 3rd pattern. From the Fig. 2 (a) we can find that the CNN-RNN combined model performs better over the other two models. Thus, the CNN-RNN model is suitable for the consecutive same characters pattern. CNN and help find the characters of the pattern while RNN can help find the position of it. From the Fig. 2 (b) we can find that the RNN model may be more appropriate for the palindrome string format sequence, since it may help find the connection between the previous and next characters.

To make it more understandable, Fig 3 illustrates the saliency map of the entire sequence with extracted sequence pattern based on CNN model, where a saliency map can only be visualized based on convolutional neural networks. It is observed that the signal is strong around the sequence pattern. In Fig. 3 (a), the sequence pattern of consecutive characters with fixed starting position can be easily extracted. In Fig. 3 (b), since the starting position of the pattern is randomly set, the consecutive characters pattern can be at different the positions. The signal is stronger near the consecutive characters and forms the pattern as the figure shows. In Fig. 3 (c), the starting position of the palindrome string format pattern is fixed, which can be easily extracted. Since the length is randomly generated and the sequences in the dataset should have different length of the pattern. Thus, the signal near the ending position is not that strong. Since the pattern string is a palindrome, the signal near the ending position also affects the starting position.



(a) 2nd pattern

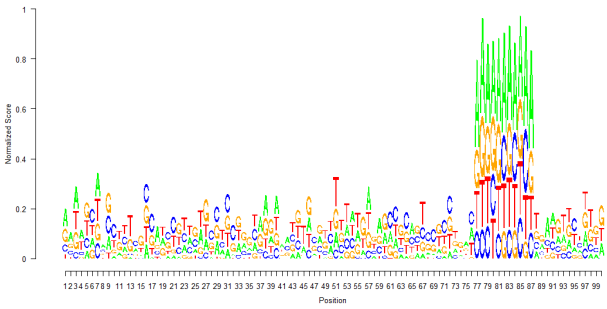


(b) 3rd pattern

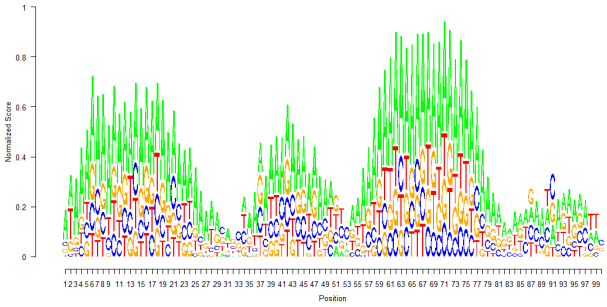
Fig. 2. Prediction performance comparison of three deep learning models: CNN, RNN, and CNN-RNN.

IV. Conclusion and Future Works

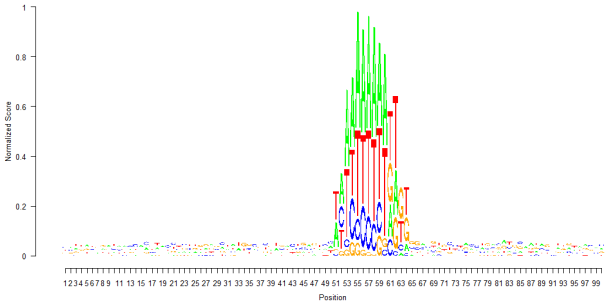
Deep learning has been widely used for pattern recognition and scene sensing problem. To our knowledge, we are the first to implement deep learning models on the task of sequence pattern recognition problem. To accurately characterize the sequence patterns and to design a prediction tool considering all the relative regions, we proposed three deep learning models for learning the sequence pattern which could generalize to new sequence data for recognition. Using one-hot encoding method for sequences representation, the deep learning methods can gain high accuracy and high precision without involving extensive manual feature engineering. We provided experimental evidence showing that performance comparison among three deep learning models: CNN, RNN and CNN-RNN models, which can help us find an appropriate deep learning model for a special sequence pattern. In conclusion, this approach is a powerful way to recognize special sequence patterns with only raw sequences provided. Encouraged by our simulation results, we intend to evaluate the models'



(a) 1st pattern



(b) 2nd pattern



(c) 3rd pattern

Fig. 3. A graphical illustration of the saliency map of the entire sequence for three extracted sequence patterns based on CNN model

performance on more sequence patterns in our future work. Furthermore, we would optimize the model by tuning a variety of hyperparameters related to training process such as batch size and learning rate, as well as that for model building such as the type of hidden layers, the number of hidden layers, the number of kernels, kernel size, and dropout rate. With the carefully generated dataset and the accurate prediction performance on sequence pattern, we believe it will be useful to investigate learned patterns and create an interface showing them.

REFERENCES

[1] J. Zhang, K. Du, R. Cheng, Z. Wei, C. Qin, H. You, and S. Hu, "Reliable gender prediction based on users' video viewing behavior," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 2016, pp. 649–658.

[2] J. Zhang, Z. Wei, Z. Yan, and A. Pani, "Collaborated online change-point detection in sparse time series for online advertising," in *Data Mining (ICDM), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1099–1104.

[3] J. Zhang, Z. Wei, Z. Yan, M. Zhou, and A. Pani, "Online change-point detection in sparse time series with application to online advertising," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2017.

[4] J. Zhang and Z. Wei, "An empirical bayes change-point model for identifying 3' and 5' alternative splicing by next-generation rna sequencing," *Bioinformatics*, vol. 32, no. 12, pp. 1823–1831, 2016.

[5] J. Zhang, Z. Zhao, K. Zhang, and Z. Wei, "A feature sampling strategy for analysis of high dimensional genomic data," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, no. 1, pp. 1–8, 2017.

[6] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, "Predicting the sequence specificities of dna-and rna-binding proteins by deep learning," *Nature biotechnology*, vol. 33, no. 8, pp. 831–838, 2015.

[7] J. Zhou and O. G. Troyanskaya, "Predicting effects of noncoding variants with deep learning–based sequence model," *Nature methods*, vol. 12, no. 10, p. 931, 2015.

[8] D. Quang and X. Xie, "Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences," *Nucleic acids research*, vol. 44, no. 11, pp. e107–e107, 2016.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[10] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[11] H. Zeng, M. D. Edwards, G. Liu, and D. K. Gifford, "Convolutional neural network architectures for predicting dna–protein binding," *Bioinformatics*, vol. 32, no. 12, pp. i121–i127, 2016.

[12] J. Zhou, Q. Lu, R. Xu, L. Gui, and H. Wang, "Cnnsite: Prediction of dna-binding residues in proteins using convolutional neural network with sequence features," in *Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on*. IEEE, 2016, pp. 78–85.

[13] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[17] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[18] J. Bergstra and D. D. Cox, "Hyperparameter optimization and boosting for classifying facial expressions: How good can a "null" model be?" *arXiv preprint arXiv:1306.3476*, 2013.

[19] F. Chollet, "Keras: Deep learning for python," 2015.

[20] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *Proc. 9th Python in Science Conf*, 2010, pp. 1–7.

[21] J. Zhang, Z. Wei, and J. Chen, "A distance-based approach for testing the mediation effect of the human microbiome," *Bioinformatics*, 2018.