Deep Recurrent Neural Network for Protein Function Prediction from Sequence

Content

1. Paper Description

As high-throughput biological sequencing becomes faster and cheaper, we can obtain far more datasets of biological sequences than ever, which inspires the applications of machine learning algorithm in protein function prediction.

1.1 Model Design and Architecture

In this paper, the author applied artificial recurrent neural networks(RNN) models, containing long-short-term-memory(LSTM) units, towards classification of protein function directly from the amino-acid sequences, as shown in figure 1.

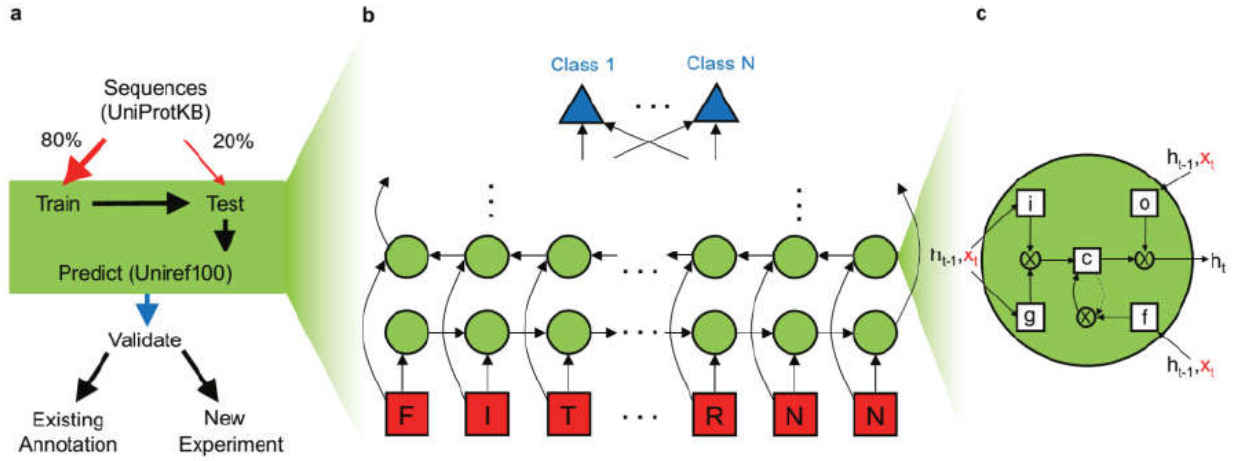

Figure 1. (from paper page 32) Model and Architecture.

a) The left part in the figure shows the whole procedure of this design. We firstly trained the model over set of protein sequences with certain functions as labels. Once evaluating the performance of the trained model, predictions are made over unannotated sequence and validated by experimental assay.

b) The middle part shows the bi-directional RNN model which could utilize the context of both sides instead of only focusing on one direction. After obtaining the outputs from the bi-directional RNN model, we feed them to a fully-connected neural network to predict the actual function of this sequence.

c)The right part shows the structure of a single LSTM neuron in the bi-directional RNN model. LSTM neuron is an architecture which could connect previous information to the present task resulting in the persistence of information. A good description can be found at: ***http://colah.github.io/posts/2015-08-Understanding-LSTMs/.***

1.2 Experiment Design

Here in our experiment, we test the “Ferritin” protein function prediction with datasets from UniProt. The RNN model showed superiority over other machine learning algorithms, such as logistic regression and SVM. It achieved excellent performance in in-class prediction.

1.3 Models Introduction

1.3.1 RNN

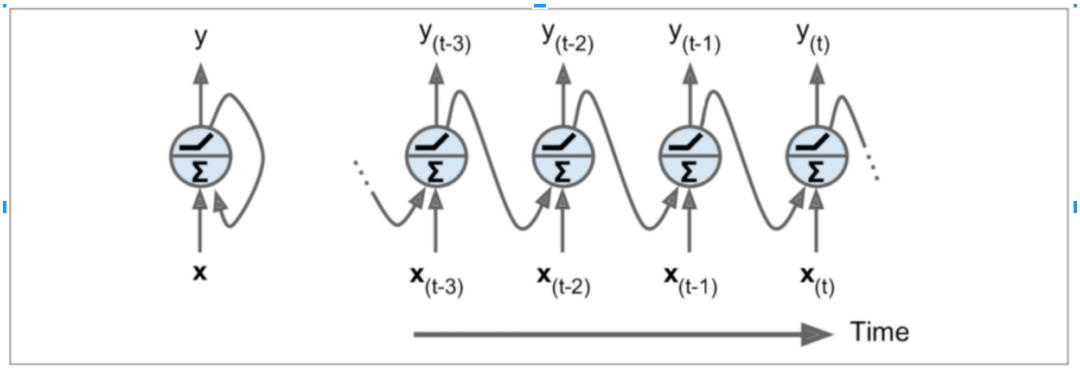

Figure 2. A recurrent neuron(left), unrolled through time(right)

A recurrent neural network is very similar to a feedforward neural network, although [1], different from traditional artificial neural networks which don't take into account of the connections among a continuous datas, RNN feeds neuron with not only the current data, but also the output from the last data recurrently, as shown in Fig 2, in order to find out the relationship of the whole data of one example as a sequence. So RNN performs very well in data with time series such as text, stock and so on. However, traditional RNN is mainly limited by two problems: vanishing gradient problem and memory lost. Long-term network training may result in the weights disappear and memory of connection for early data would gradually fade away during a long training time. To solve these problems, we can make use of LSTM and GRU rather than normal RNN cell.

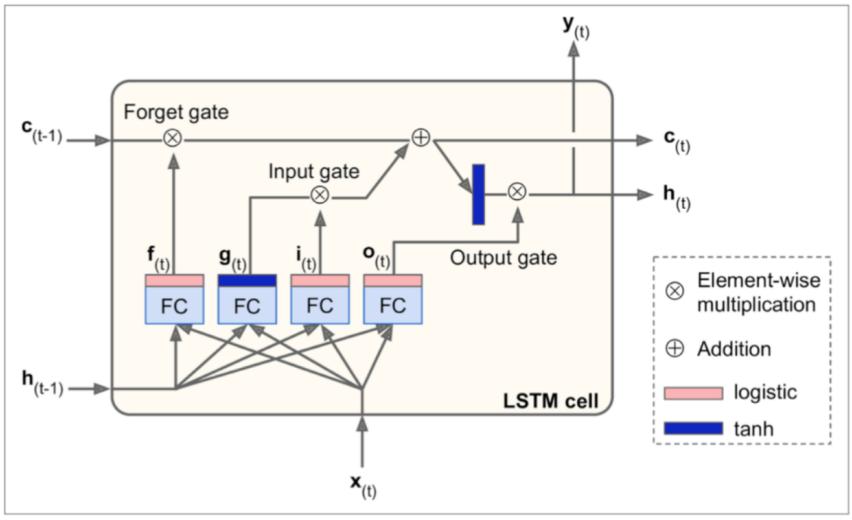

Figure 3. LSTM cell

1) Long Short-Term Memory

Sepp Hochreiter et.al [2] introduce long short-term memory(LSTM) to overcome the vanishing gradient problem and result in a long memory network.

As Fig 3 shows, The Long Short-Term Memory(LSTM) cell improved the traditional cell with three gates: forget gate, input gate and output gate. LSTM decides which data to remember which to drop. Inside the LSTM cell, it uses some tanh activation to void vanishing gradient problem and after every timestep training the cell output both a long-term state and a short-term state to help RNN remember history.

2) BRNN

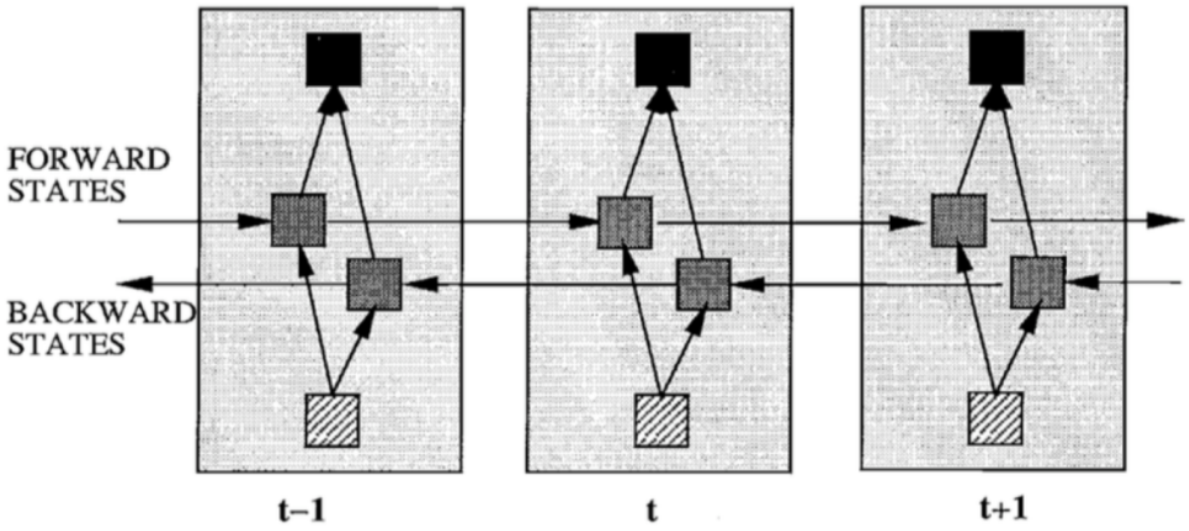

Figure 4. General structure of the BRNN shown unfolded in time for three time steps

Based on recurrent neural network, Mike Schuster et, al. [3] proposed bidirectional recurrent neural network (BRNN). BRNN takes account not only the forward connections but also the backward connections of protein sequence data, which makes more sense in human typical recognitions. The forward stream and backward stream can be regarded as two separate network to find out each series' relations in opposite direction. Then, outputs from the two streams can be fused together with a fusion method. In this project, we used a fully connected layer to fulfill it.

1.3.2 SVM

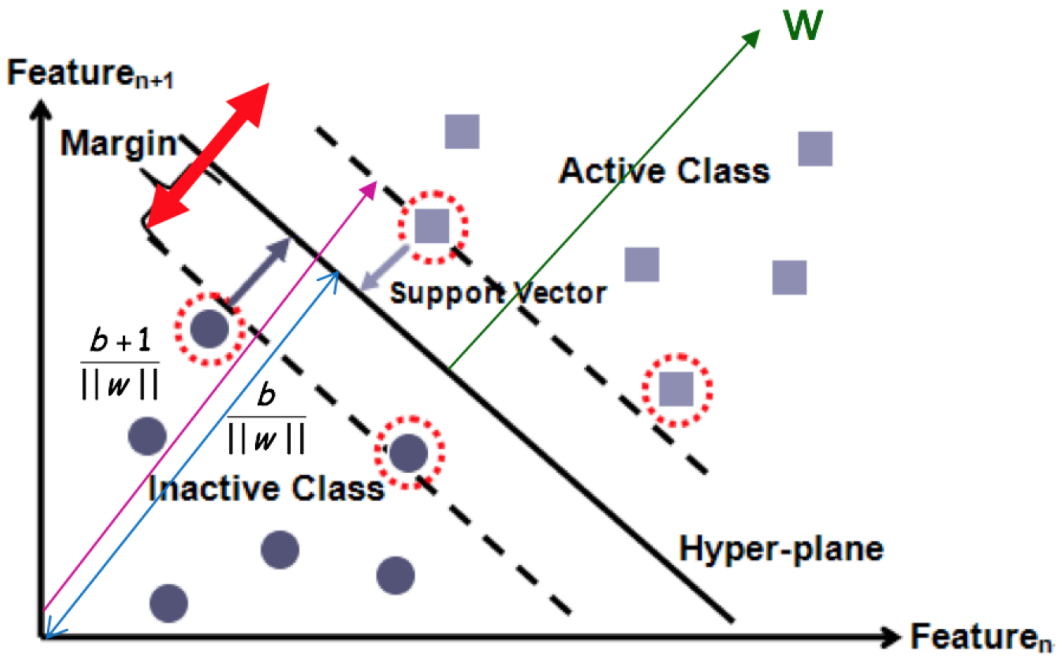

Figure 5. SVM illustration

Support Vector Machine(SVM) plays an important role in traditional machine learning algorithms. As shown in Fig 4, the whole dataset can be mapped into points falling into space. The SVM then try to find a hyper-plane that can separate the active class and inactive class into two subspace with max margin through the support vectors. However, the raw datasets are often not linearly-separable which makes it impossible to find such a hyper-plane. In case, we can take advantages of kernel tricks, mapping the raw features into a higher dimensional space, to solve this problem. In this project, we use the radial basis function(RBF) kernel.

2. Experiments

2.1 configuration

Since there is no source code online for this paper, we implemented it by ourselves. The configuration of this implementation is listed as follows:

1) Language and Package

Here we use python to implement the whole experiment, with utilizing some packages like cPickle, pandas, numpy, sklearn.

2) Deep Learning Framework

And we use the tensorflow framework to implement the bidirectional RNN model with LSTM units.

3) Computation Resources

As sequence data training is very slow, and the converted one-hot vector data is quite

large, we just apply from Palmetto the conservative resources which can run the program successfully.

<1> Cpus: 16 cores

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
On-line CPU(s) list:   0-15
Thread(s) per core:    1
Core(s) per socket:    8
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 45
Model name:            Intel(R) Xeon(R) CPU E5-2665 0 @ 2.40GHz
Stepping:              7
CPU MHz:               2400.000
CPU max MHz:           2400.0000
CPU min MHz:           1200.0000
BogoMIPS:              4788.14
Virtualization:        VT-x
Hypervisor vendor:     vertical
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              20480K
NUMA node0 CPU(s):     0-7
NUMA node1 CPU(s):     8-15
```

Figure 6. Cpu configuration

<2> Gpus: 2 cores

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.98                 Driver Version: 384.98                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K20m          Off  | 00000000:24:00.0 Off |                    0 |
| N/A   27C    P0    46W / 225W |      0MiB /  4742MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K20m          Off  | 00000000:27:00.0 Off |                    0 |
| N/A   23C    P0    51W / 225W |      0MiB /  4742MiB |     98%      Default |
+-------------------------------+----------------------+----------------------+
```

Figure 7. GPU configuration

<3> Memory: 100GB

2.2 implementation

2.2.1 Data Source

This project focused on a binary classification between protein sequences of ferritin from reviewed Uniprot and protein sequences of non-ferritin from unreviewed Swiss-Prot, 178743 samples of positive and 556164 samples of negative.

Uniprot is a professional website of biological science, which provides comprehensive, high-quantity and easy accessible dataset of various protein sequences for scientific community.

2.2.2 Data Preprocessing

The raw data represents a protein sequence with different letters. In this project, we read data files with pandas lib and translated the character sequences into digit format in two ways: one is using continuous numbers stand for different character, the other is using “one-hot” vector indicates a specific letter. To avoid memory error, we also implemented two ways to feed the model: saved the processed data into disk as different chunks and read these chunk data when training the model; read and processed a chunk of data only before training.

Besides, we divided processed data into train dataset(80%) and test dataset(20%). And we tried a series of sequence lengths, including (10, 35, 80, 120, 150, 200, 240, 333) to make a comparison.

2.2.3 different methods

1）BRNN

We built this model using tensorflow framework. In order to create the model graph, first, construct a forward RNN layer with LSTM cell; Second, construct a backward RNN layer with LSTM cell; Then package them into static_bidirectional_rnn wrapper. To avoid the problem of overfitting, a dropout wrapper was applied above rnn wrapper with dropout rate equals to 0.5. Outputs from the two opposite directions layers will be fused by a fully connected layer. Finally, evaluate the model by comparing the results of prediction with real labels. Because we were using the static_bidirectional_rnn, we had to unstack the train data before feeding. And this model applied AdamOptimizer during the backpropagation course with 0.001 learning rate to reduce losses. The number of hidden neurons units is 128, batch size is 150 and number of epochs is 5.

2) Other

Except BRNN, we also implemented SVM model and logistic regression model by scikit-learn lib to train the data and then make comparisons. In SVM model, we select rbf kernel and C=1.

3. Results and Analysis

When analyzing the results, we separate it into 2 parts. One part is comparison of statistical values of different sequence length based on the same RNN model. The other part is the comparison of RNN, SVM and Logistic Regression. We just do comparison based on 1 single protein function--“ferritin”.

3.1 RNN model

3.1.1 Basic Statistical values comparison

From the following figure, we can see that the RNN model presented in paper did performs well in predicting protein sequence function as the sequence length grows to a relatively large number. And this means it is a success in utilizing RNN model to do prediction with known protein sequence data.

In this figure, when length = 333, the statistical values seems weaker than those statistical values of length = 80,120,150,200,240. We think this might be the overfitting problem. With unseen test data, the model of sequence length = 333 has too many parameters and is more easier to overfit when training. Here in this paper, we just want to prove the effectiveness of RNN model on prediction of protein sequence data, so we didn’t use any method to reduce the effect of overfitting. Maybe in the 3rd step of improving the experiment, we will consider a more effective model and take the overfitting problem into consideration.

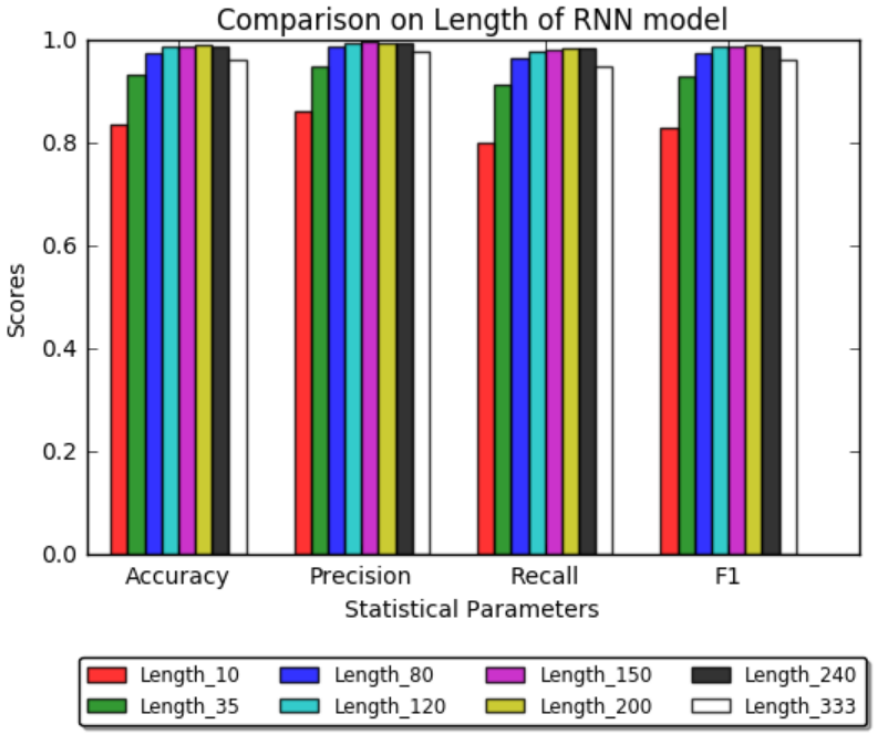

Figure 8. Performance comparison of the same RNN-LSTM model based on different sequence length

3.1.2 ROC Curves

From the roc curve, we can find that those rnn roc curves are much more different from the diagonal line y=x, which means our model does have high accuracy in prediction. One thing needs to be mentioned is that we use the function of sklearn.metrics.roc_curve(yTrue, yPrediction) to get the false positive points and true points to draw ROC curves, but that function just gave 3 points back for this binary class problem, which leads to this rough figure.

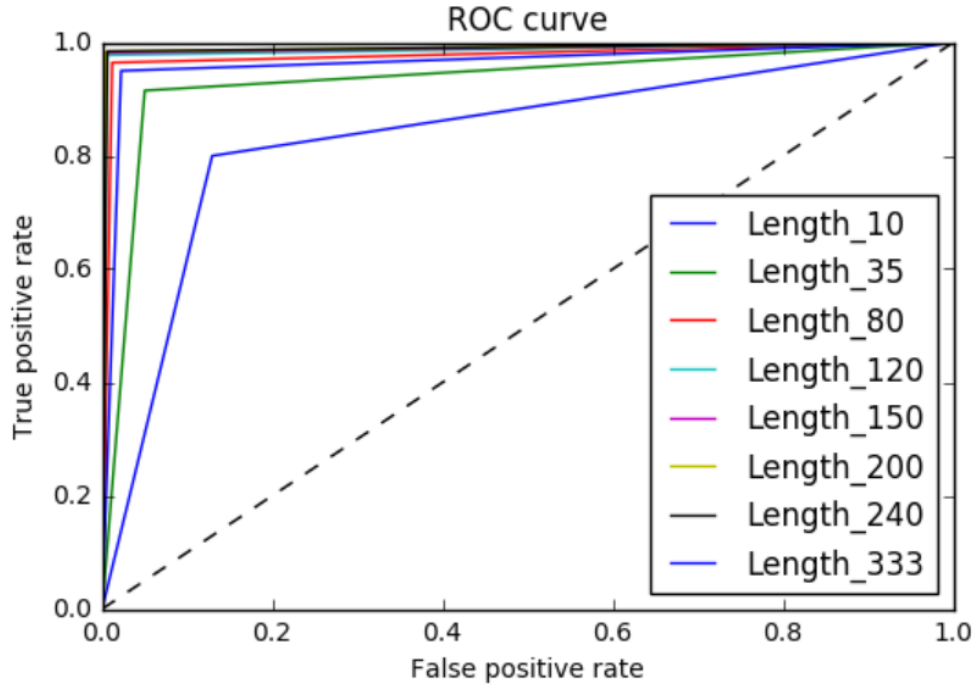

Figure 9. ROC curve of the same RNN-LSTM model based on different sequence length

3.1.3 Precision and Recall

The following figures show the precision and recall of the rnn model based on different sequence length. From these two, we can get the sme results of the 3.1.1 that this RNN model can get pretty good prediction results. And the last lower points of these two pictures show the overfitting problem mentioned in 3.1.1.

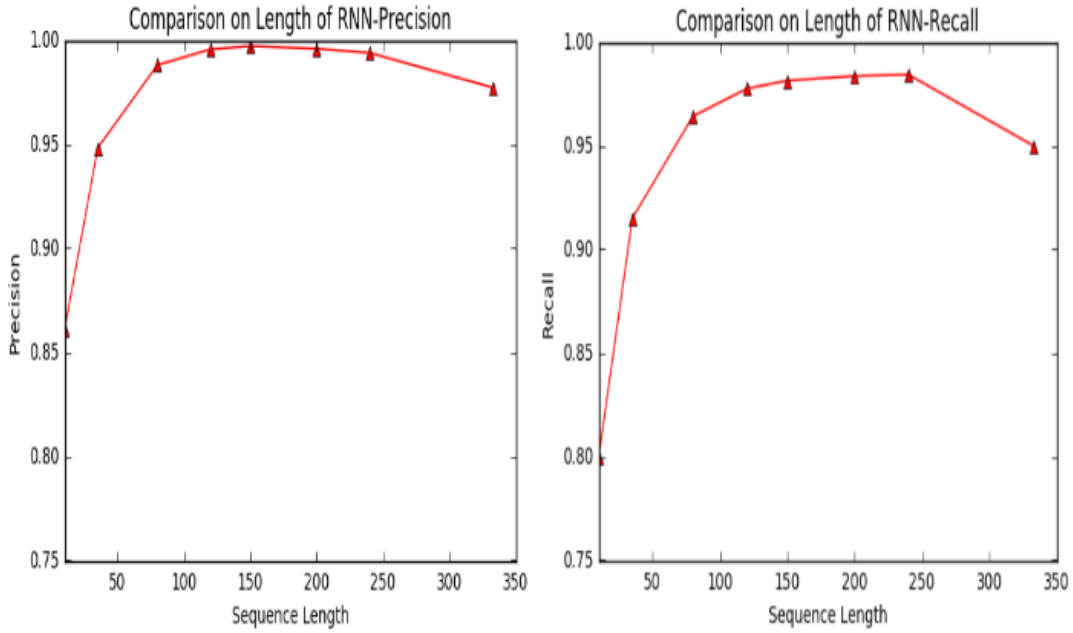

Figure 10. Precision and Recall of RNN-LSTM model based on different sequence length

3.2 RNN model vs. SVM, Logistic Regression

3.2.1 Comparison based on sequence length=150

From the following figure, we can see that our RNN model outforms heavily the SVM and LR models. Most of the statistical values of RNN model shows much better results than SVM and LR models, which means this RNN model used in predicting protein sequence function is quite trustable.

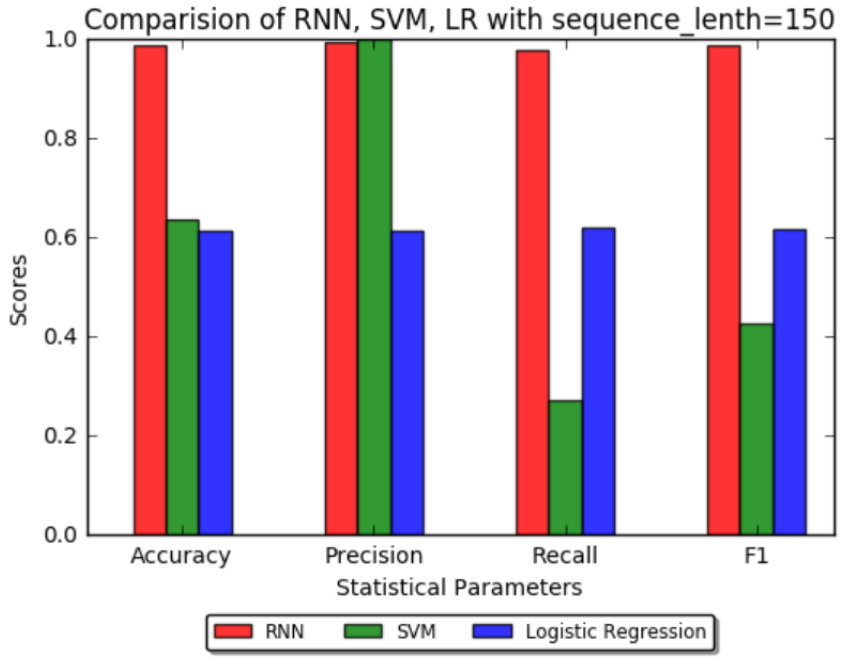

Figure 11. Performance comparison between RNN-LSTM, SVM, LR based on the same sequence length 150

3.2.2 The whole scope comparison

Here we give the whole scope comparison of RNN model vs. SVM, LR mode based on different sequence length. And seeing from the following 8 pictures, we can conclude that no matter what the sequence length is, the RNN model outperforms SVM and LR.(Here the red bar corresponding to RNN model, Green bar corresponding to SVM model, blue bar corresponding to Logistic Regression model)

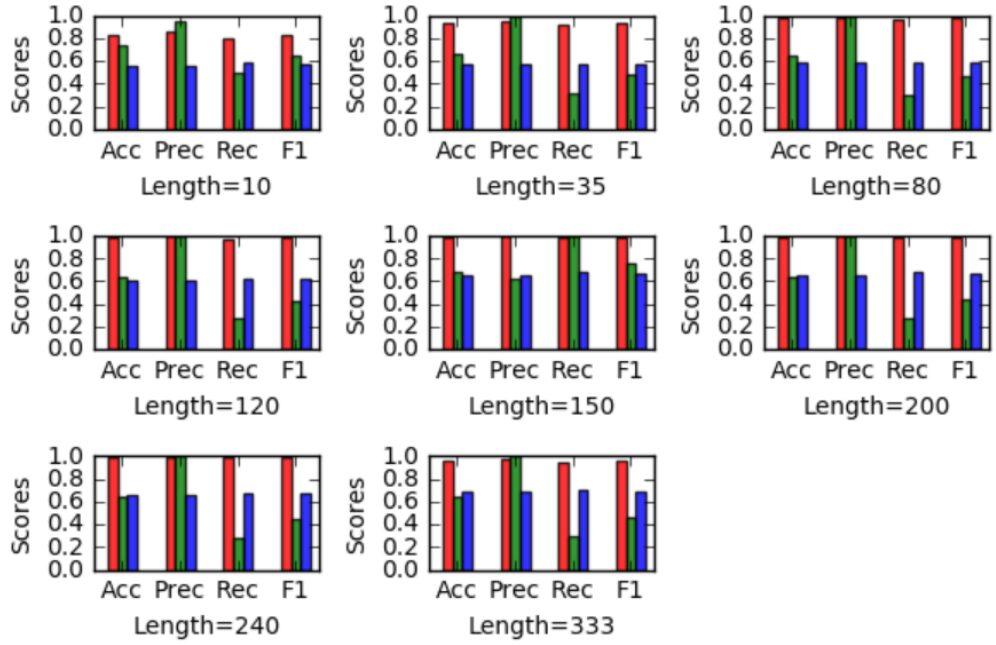

Figure 12. Comparison of RNN-LSTM, SVM, LR based on different sequence length

4. Trials

4.1 One-hot vector Input vs. Integer Input

In this project, we transferred each amino acid into an integer and then convert that integer into a one-hot vector. Easy to find that using one-hot vector as input will cost more computation resources and time consumption. Besides, the dataset is large engouh(positive over 170k, negative over 560k). So we just wonder whether just using integer as input could save time efficiently as well as keeping the same accuracy.
Our results/comparisons are as follows.

4.1.1 Time Comparison

As we can see from the below picture, using integer as input does reduce the time consumption but it's just a decent result. From this picture, we can indicate that the matrix multiplication does occupy a computation time. But the main time consumption is due to the values transmission

during the forward propagation and backward propagation. This gives us a deeper understanding of RNN model. And now we could understand why SRU[4] developed for parallel computing will improve the efficiency a lot.

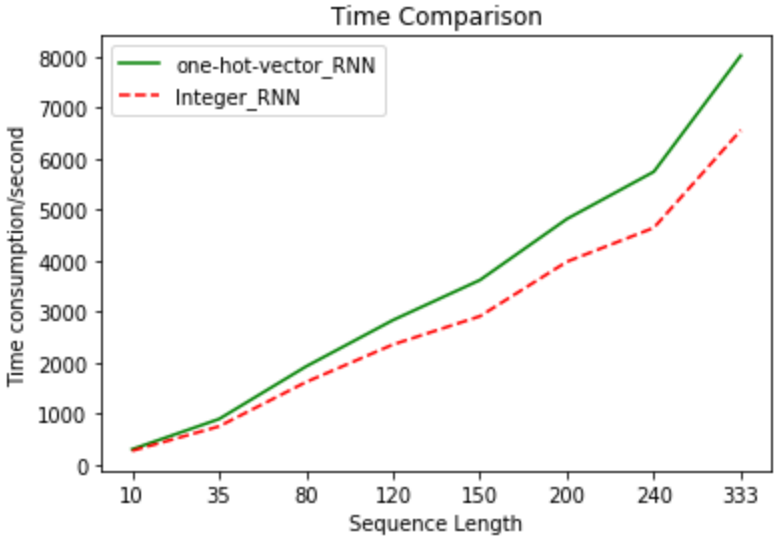

Figure 13. Time Comparison

4.1.2 Accuracy Comparison

From the picture below, we can see that there still exist differences of accuracies based on these two kinds of input. And we could find that using one-hot vector as input will outperforms that of using integer as input.

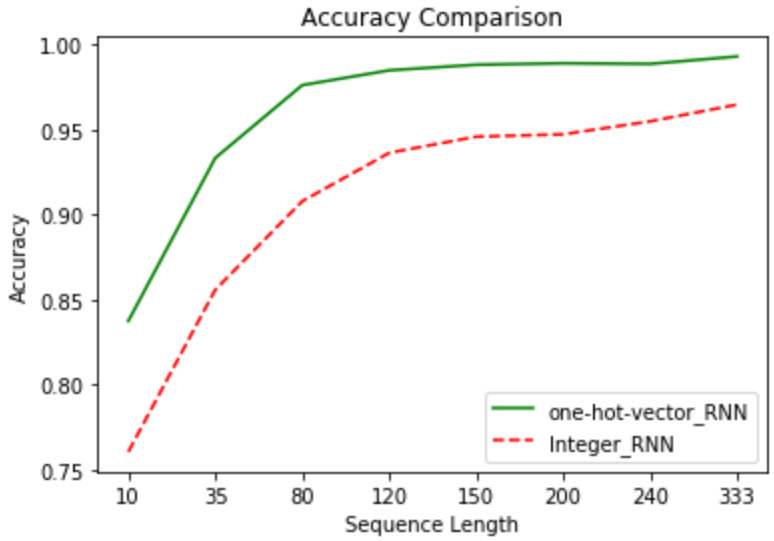

Figure 14. Accuracy Comaprsion

To sum up, combine 4.1.1 and 4.1.2, we could say using one-hot vector as input is much better

than using integer as input even though using integer as input does run a little faster than using one-hot vector as input.

5. Reference

[1] Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.".
[2] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.
[3] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673-2681.
[4] Lei, T., & Zhang, Y. (2017). Training RNNs as Fast as CNNs. *arXiv preprint arXiv:1709.02755*.
[5] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.