An Analysis for the Prediction of Protein Subcellular Localization

Group 7
Wei Meng, Yaoxiang Li, Zewei Xiong

Introduction

Background

- As a result of large-scale genome sequencing efforts in recent years, protein data has accumulated in public data banks at an increasing rate. Subcellular localization is a key functional attribute of a protein
- Knowledge of the subcellular localization of a protein can significantly improve the target identification during the drug discovery process
- Usually considered as a multi-label classification problem

Introduction

Popular methods

- Basic Local Alignment Search Tool (BLAST)
- This method gathers the homologous proteins' function information by searching the query sequence against the existing protein databases as it contains experimentally determined protein function information

- Network based methods
- These methods are under the assumption that interacted proteins share similar functions

Methods

Data description

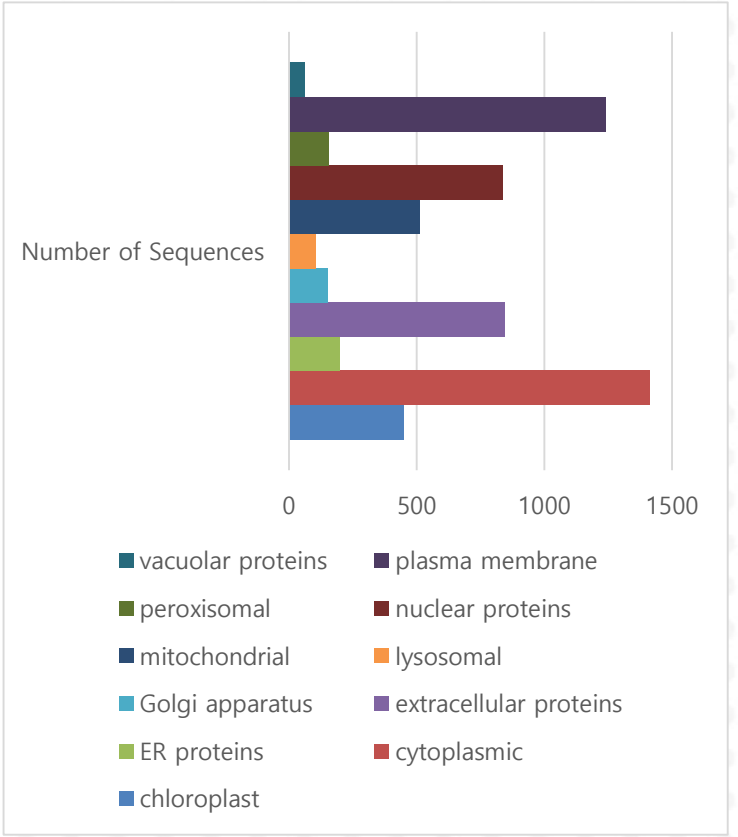
- The proteomics dataset was summarized by the SWISS-PROT database by which obtained extracting all animal, fungal and plant protein sequences

- The dataset contains ***5959*** proteins annotated to one of ***11*** different subcellular lo-cations

Protein Type	Number of Sequences	Protein Type	Number of Sequences
chloroplast	449	mitochondrial	510
cytoplasmic	1411	nuclear proteins	837
ER proteins	198	peroxisomal	157
extracellular proteins	843	plasma membrane	1238
Golgi apparatus	150	vacuolar proteins	63
lysosomal	103	Total	5959

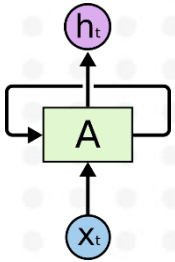
Methods

- Data pre-process
  - protein sequences were truncated to length 4 01 to reduce computational time
  - truncated by removing from the middle of the protein as both the N- & C-terminal regions
  - Binary classification; three classes classificati on; four classes classification
  - 80% train data and 20% test data
  - 100-fold cross validation

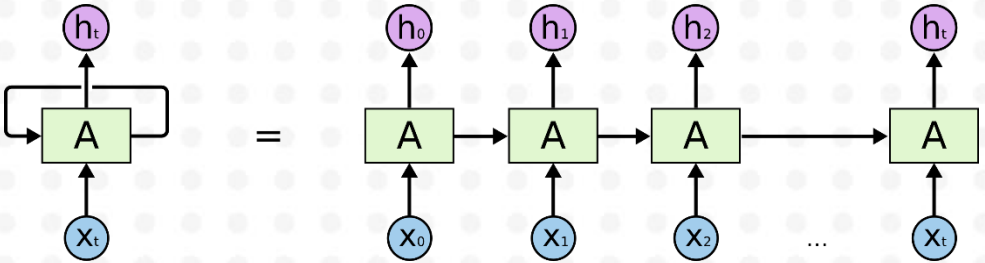


Methods

- Recurrent Neural Networks have loops

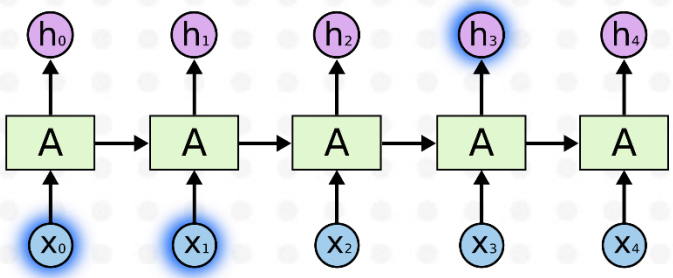


- An unrolled recurrent neural network



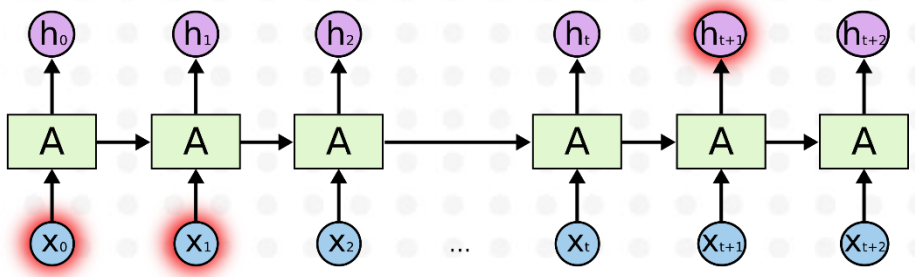
Methods

e.g., If we are trying to predict the last word in “the clouds are in the *sky*,” we don’t need a ny further context – it’s pretty obvious the next word is going to be sky. In such cases, whe re the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information.



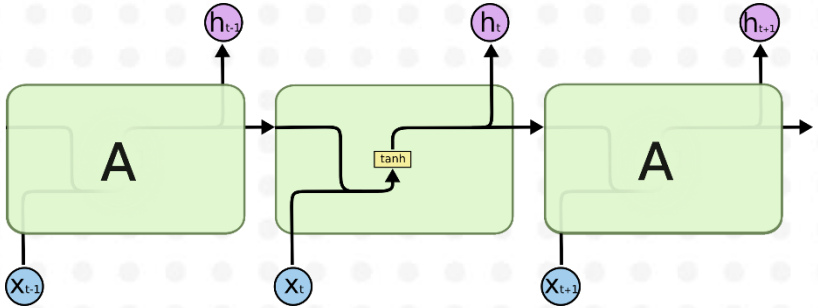
Methods

In theory, RNNs are absolutely capable of handling such “long-term dep endencies”. the problem was explored in depth by Hochreiter (1991) wh o found some pretty fundamental reasons why it might be difficult.

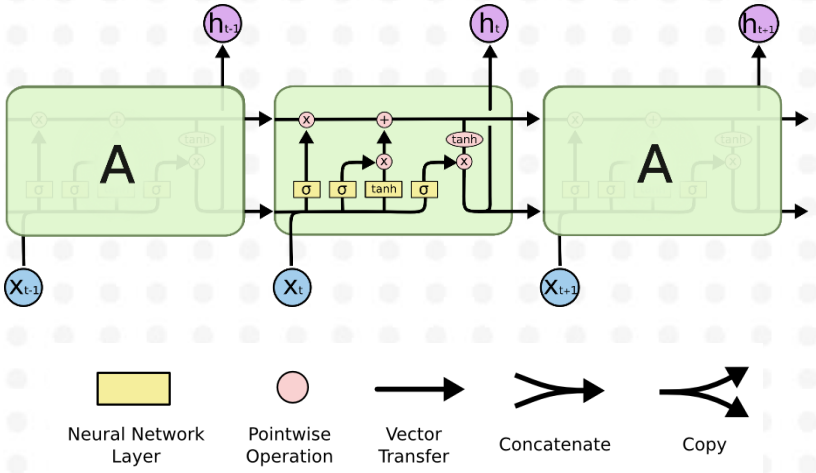


Methods

The repeating module in a standard RNN contains a single layer.

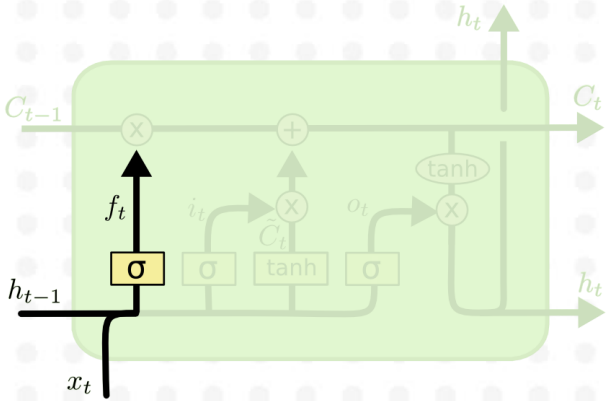


Methods


h_{t-1}
h_t
h_{t+1}
A
A
tanh
σ
σ
tanh
σ
X_{t-1}
X_t
X_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

Methods

- The first step in our LSTM is to decide what information we're going to throw away from the cell state.

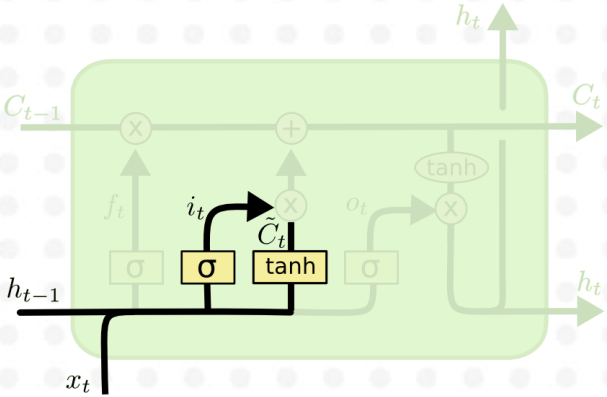


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Methods

- This step is to decide what new information we're going to store in the cell state. This has two parts:

1. a sigmoid layer called the "input gate layer" decides which values we'll update
2. a tanh function creates a vector of new candidate values that could be added to the state. In the next step, we'll combine these two to create an update to the state



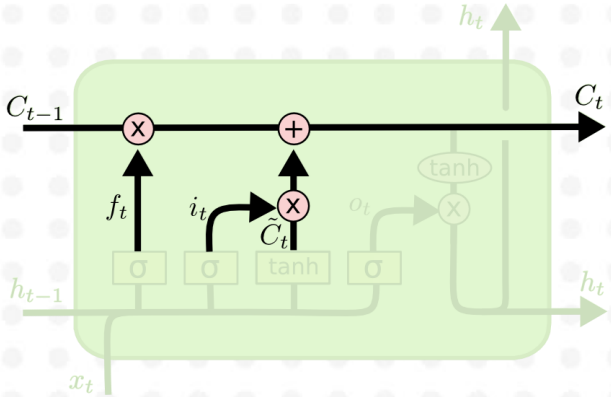
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Methods

- It's now time to update the old cell state, C_{t-1}, into the new cell state C_t.The previous steps already decided what to do, we just need to actually do it

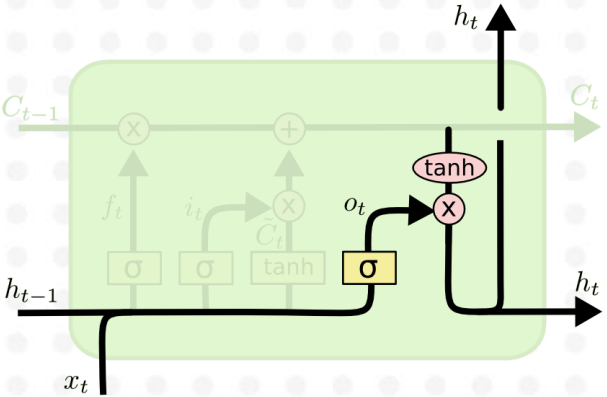
- We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Methods

- Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version
- we run a sigmoid layer which decides what parts of the cell state we're going to output
- we put the cell state through tanh and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

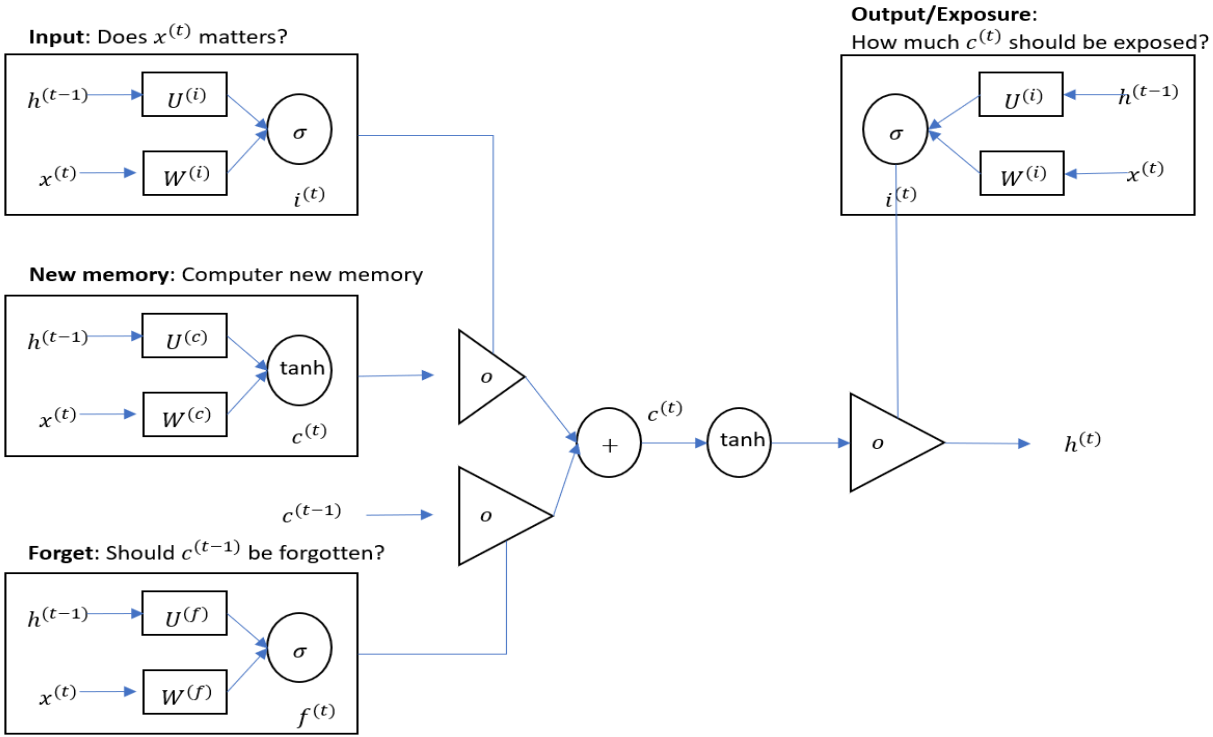
Methods

- See how LSTM works

  - input gate: $i_t = \sigma(W^{(i)} x_t + U^{(i)} h_{t-1})$
  - forget gate: $f_t = \sigma(W^{(f)} x_t + U^{(f)} h_{t-1})$
  - output/exposure gate: $o_t = \sigma(W^{(o)} x_t + U^{(o)} h_{t-1})$
  - new memory cell: $\tilde{c}_t = \tanh(W^{(c)} x_{t\_} + U^{(c)} h_{t-1})$
  - final memory cell: $c_t = f_t \circ c_{t-1} + i_t \circ c_t$
  - $h_t = o_t \circ \tanh(c_t)$

Methods

- The detailed internals of a LSTM

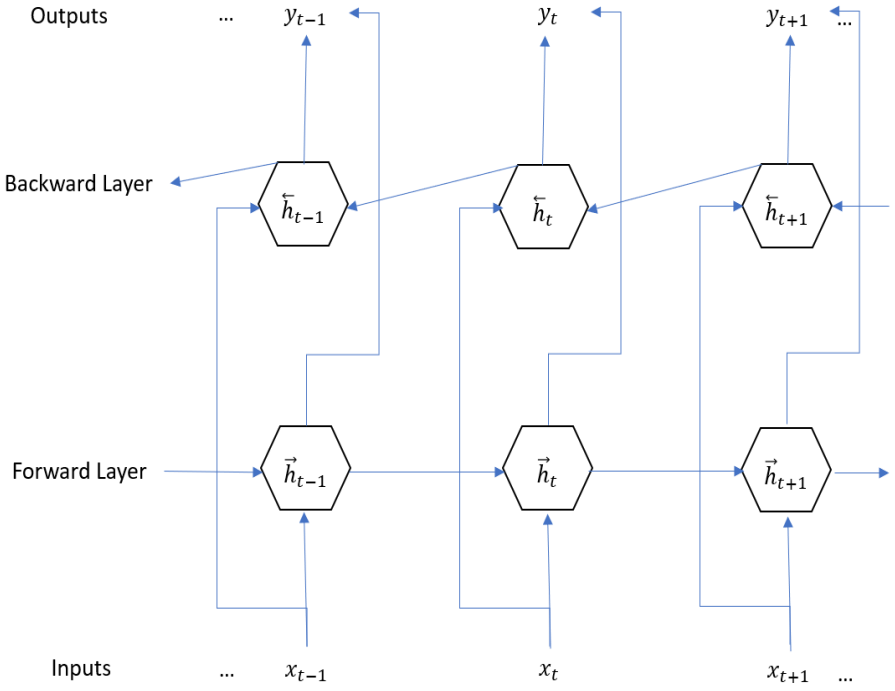


Methods

- Bi-directional Long Short Term Memory network (Bi-LSTM)
  - extend the unidirectional LSTM networks by introducing a second layer
  - the hidden to hidden connections flow in opposite temporal order
  - these two sub-layers compute forward and backward hidden sequences $\overrightarrow{h}$ and $\overleftarrow{h}$ respectively

Methods

- Architecture of a Bi-LSTM



- $\vec{h}_t = H(W_{x\underset{h}{\rightarrow}} x_t + W_{\underset{h}{\rightarrow}\underset{h}{\rightarrow}} \vec{h}_{t-1} + b_{\underset{h}{\rightarrow}})$

- $\overleftarrow{h}_t = H(W_{x\underset{h}{\leftarrow}} x_t + W_{\underset{h}{\leftarrow}\underset{h}{\leftarrow}} \vec{h}_{t+1} + b_{\underset{h}{\leftarrow}})$

- $y_t = W_{\underset{h}{\rightarrow}\underset{y}{}} \vec{h}_t + W_{\underset{h}{\leftarrow}\underset{y}{}} \overleftarrow{h}_t + b_y)$

Methods

- Gated recurrent units (GRU)
  - although RNNs (Recurrent Neural Network) can theoretically capture long-term dependencies, they are very hard to actually train to do this
  - GRU are designed in a manner to have more persistent memory
  - make it easier for RNN to capture long-term dependencies

Methods

- See how GRU works

  - $h_t = \sigma(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$: the relationship to compute the hidden layer output features at each time-step t
  - $x_{1,}...., x_{t-1}, x_t, x_{t+1},...x_T$: the word vectors corresponding to a corpus with T words
    - $x_t \in R^d$: input word vector at time t
  - GRU uses h_{t-1} and x_t to generate the next hidden state h_t

    - update gate: $z_t = \sigma(W^{(z)} x_t + U^{(z)} h_{t-1})$
    - reset gate: $r_t = \sigma(W^{(r)} x_t + U^{(r)} h_{t-1})$
    - new memory: $\tilde{h}_t = \tanh(r_t \circ U h_{t-1} + W x_t)$
    - hidden state: $h_t = (1 - z_t) \circ h_t + z_t \circ h_{t-1}$

Methods

- Experimental Setup
  - Loss function

$$Loss_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) = -f_{y_i} + \log\sum_j e^{f_j}$$

•Optimization method and learning rate: Adam algorithm, learning rate=0.1, beta1=0.9, beta2=0.999, epsilon=1e-08

•Size of RNN hidden state: 2000, number of layers in RNN is 1 or 4

•Dropout design and probability: p=0.6; initial weights were sampled uniformly distributed from the interval [-0.2, 0.2]

•Environment:

•NVIDIA Graphics Drivers v375.66, NVIDIA CUDA Toolkit v 8.0, NVIDIA cuDNN v5.1, Python3 v3.5.2, Tensorflow v1.4, NumPy v1.13.3, pandas v0.20.3, Matplotlib v2.1.0

Results

1. Proteins visualization

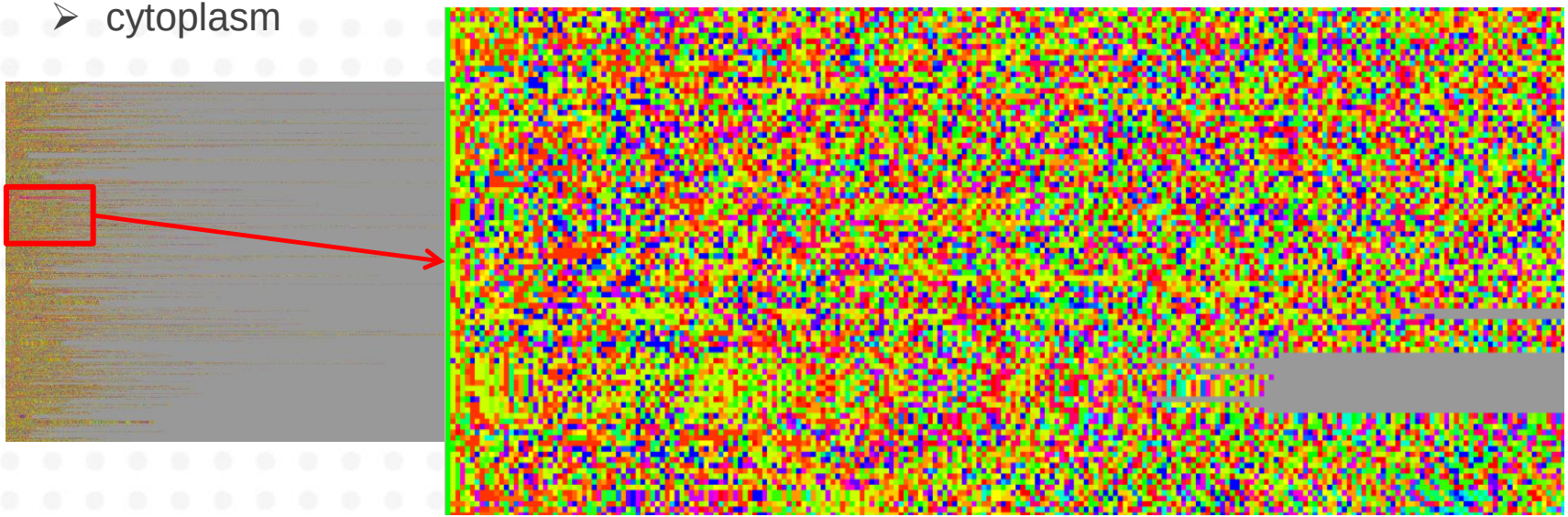
- visualizing the 11 types of protein to see the different distribution pattern of amino acids for each protein

- here we put the visualization plots of ***cytoplasm***, ***nucleus, plasma membrane*** and ***extracellular proteins***

Results

1. Proteins visualization

- cytoplasm

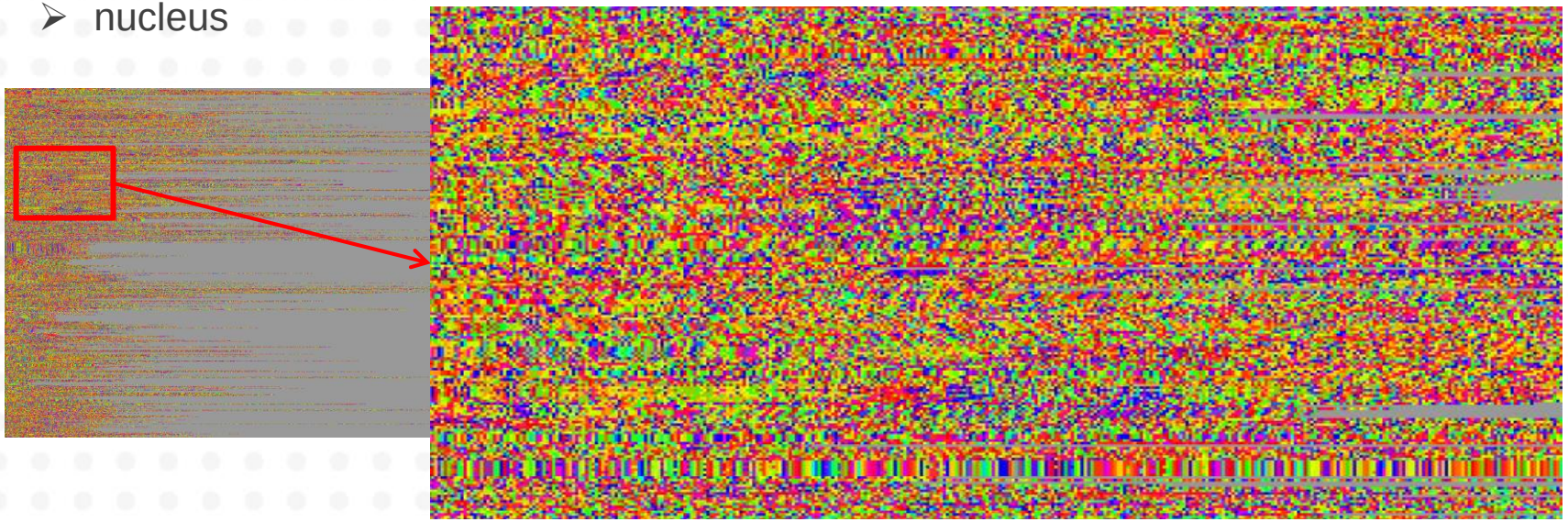


- magnify the selected part
- the proportion of yellow color is large

Results

1. Proteins visualization

- nucleus

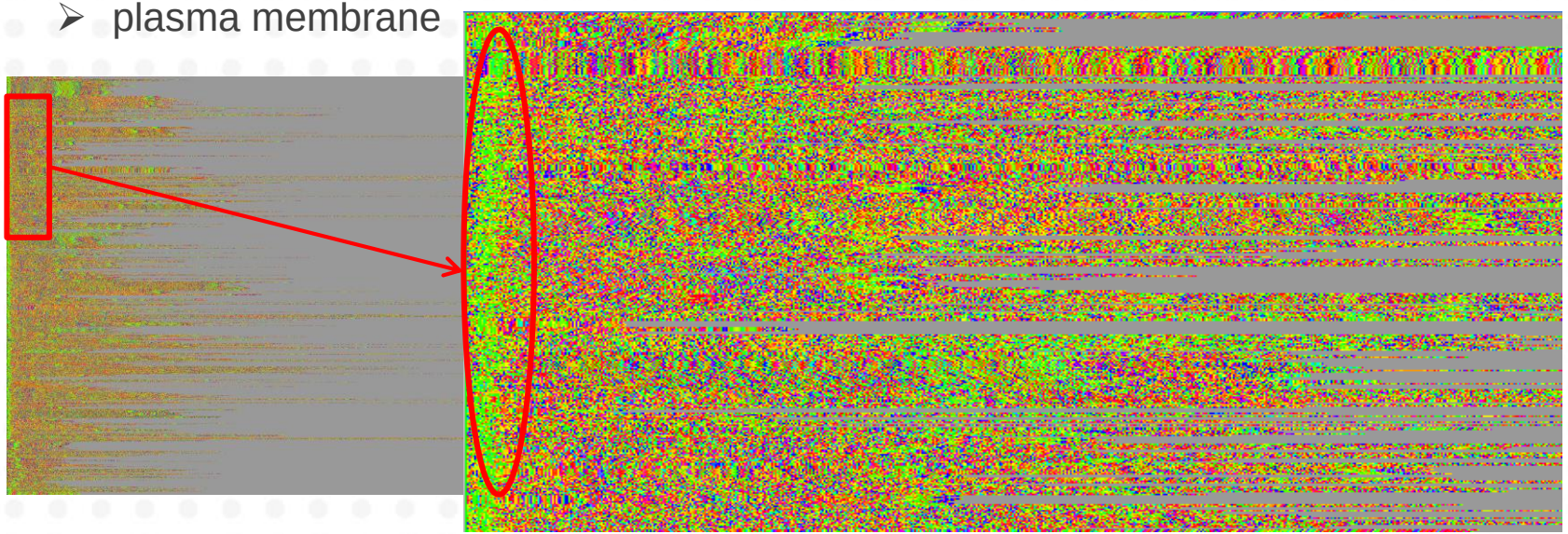


- magnify the selected part
- the proportion of pink color is large

Results

1. Proteins visualization

- plasma membrane

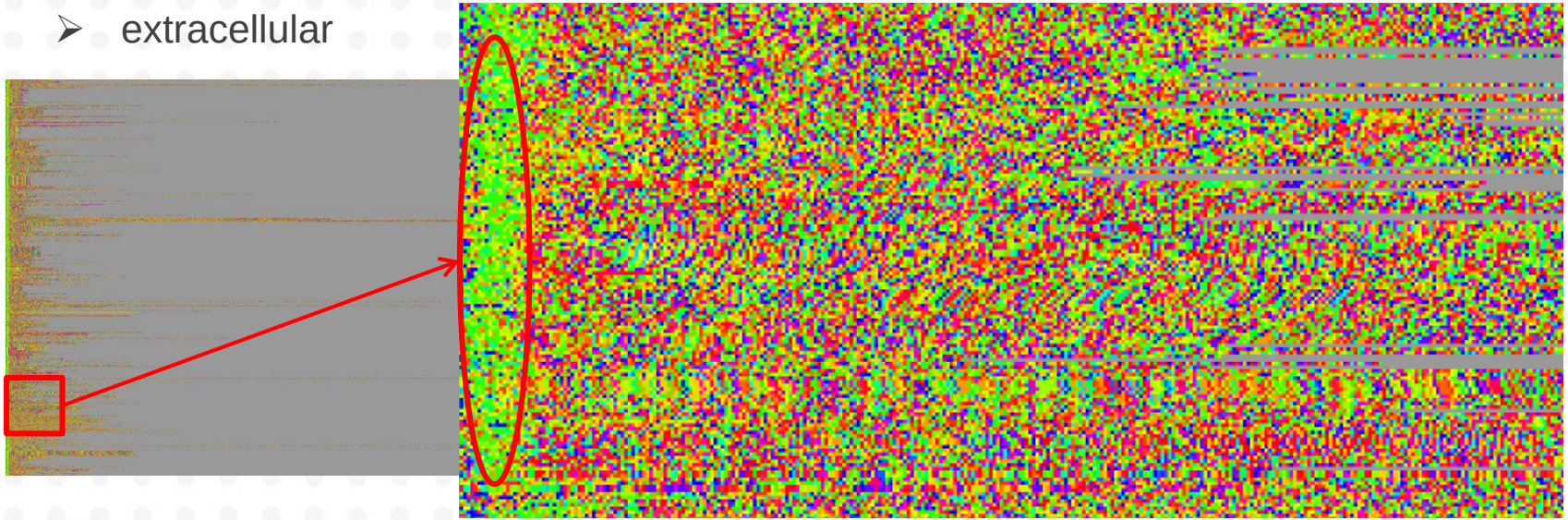


- magnify the selected part
- the proportion of green color is large; color green concentrates on the start end

Results

1. Proteins visualization

- extracellular

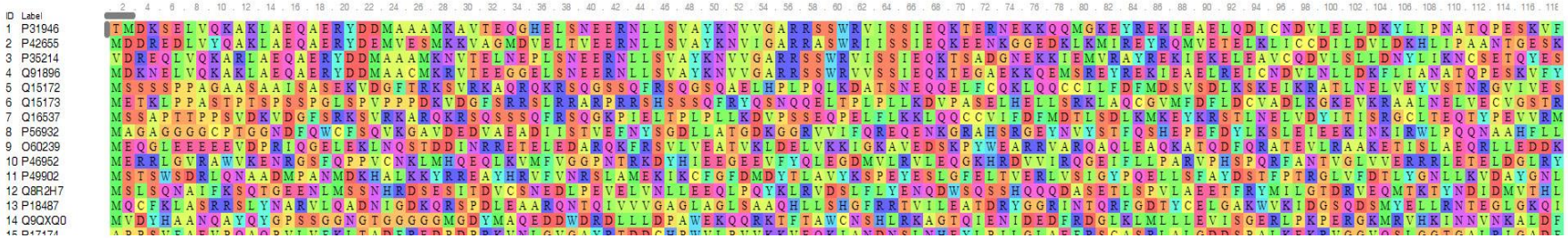


- magnify the selected part
- the proportion of green and pink is large; color green concentrates on the start end

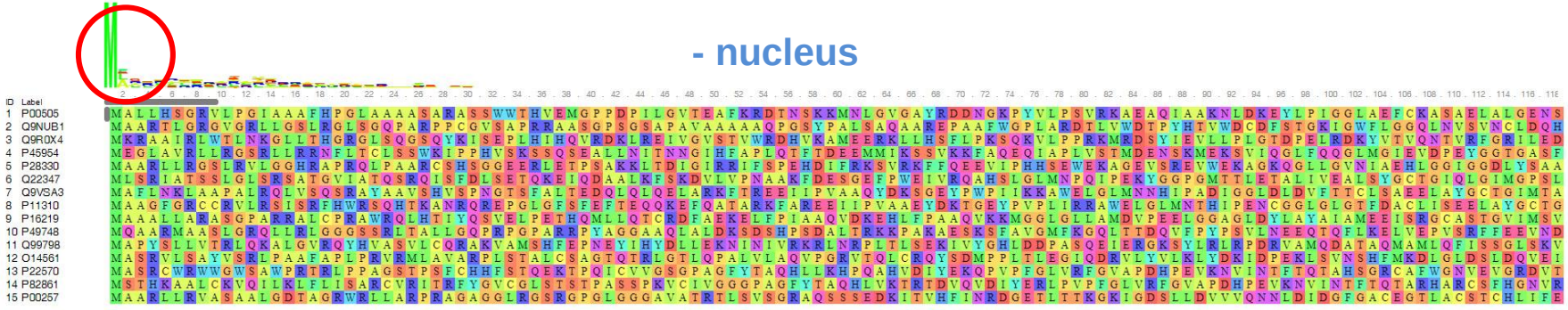
Results

1. Proteins visualization

- cytoplasm



- nucleus



Results

2. Performance of GRU, LSTM, Bi-LSTM model on test dataset

- **Binary** classification
- In general, all models performed well in this case as the high accuracy, precision, recall and F1 score which were around 0.9 and the highest value can be reached is 0.986
- classify chloroplast vs. plasma membrane and chloroplast vs. cytoplasmic: Bi-LSTM model performed better than the other two
- classify chloroplast vs. nuclear, ER vs. plasma membrane, ER vs. cytoplasmic and ER vs. nuclear, GRU performed better than the other two

	Binary classification	Accuracy	Precision	Recall	F1 Score
LSTM	chloroplast & plasma membrane	0.899	0.818	0.744	0.780
Bi-LSTM		**0.985**	**0.958**	**0.933**	**0.945**
GRU		0.896	0.818	0.900	0.857
LSTM	chloroplast & cytoplasmic	0.950	0.953	0.976	0.964
Bi-LSTM		**0.967**	**0.973**	**0.986**	**0.979**
GRU		0.912	0.945	0.841	0.890
LSTM	chloroplast & nuclear	0.856	**0.964**	0.769	0.856
Bi-LSTM		0.928	0.931	0.909	0.920
GRU		**0.943**	0.931	**0.929**	**0.930**
LSTM	ER & plasma membrane	0.768	0.748	0.777	0.762
Bi-LSTM		0.882	0.899	**0.865**	0.882
GRU		**0.916**	**0.951**	0.858	**0.902**
LSTM	ER & cytoplasmic	0.922	0.959	0.878	0.917
Bi-LSTM		0.923	0.938	0.882	0.909
GRU		**0.957**	**0.979**	**0.959**	**0.969**
LSTM	ER & nuclear	0.910	0.979	0.909	0.943
Bi-LSTM		0.907	0.952	0.866	0.907
GRU		**0.946**	**0.979**	**0.936**	**0.957**

Results

2. Performance of GRU, LSTM, Bi-LSTM model on test dataset

- **Three** classes classification
- In general, all models performed well in this case as precision, recall and F1 score were around 0.85
- LSTM model performed a bit better than the other two

	cytoplasm				plasma membrane	
LSTM	precision*	**0.875**	precision	**0.842**	precision	**0.833**
	recall	**0.827**	recall	**0.809**	recall	0.791
	F1 score	**0.851**	F1 score	**0.830**	F1 score	**0.812**
Bi-LSTM	precision	0.857	precision	0.825	precision	0.817
	recall	0.792	recall	0.753	recall	0.751
	F1 score	0.823	F1 score	0.787	F1 score	0.782
GRU	precision	0.860	precision	0.841	precision	**0.841**
	recall	0.815	recall	0.777	recall	0.779
	F1 score	0.837	F1 score	0.808	F1 score	0.809

*In multi-label classification, precision and recall refers to the micro precision and micro recall. F1 score is the harmonic average of them.

Results

2. Performance of GRU, LSTM, Bi-LSTM model

- **Four** classes classification
- In general, all models performed well in this case as most precision, recall and F1 score values were around 0.9
- In this case, Bi-LSTM model perfromed a bit better than the other two

	cytoplasm		nucleus		plasma membrane		extracellular	
LSTM	Precision*	0.791	precision	0.805	precision	0.791	precision	0.868
	recall	0.834	recall	0.788	recall	0.778	recall	0.753
	F1 score	0.812	F1 score	0.796	F1 score	0.784	F1 score	0.807
Bi-LSTM	precision	**0.886**	precision	**0.870**	precision	**0.870**	precision	**0.890**
	recall	0.849	recall	**0.815**	recall	**0.817**	recall	**0.783**
	F1 score	**0.867**	F1 score	**0.842**	F1 score	**0.843**	F1 score	**0.833**
GRU	precision	0.817	precision	0.830	precision	0.817	precision	0.886
	recall	**0.856**	recall	0.814	recall	0.806	recall	0.783
	F1 score	0.836	F1 score	0.822	F1 score	0.811	F1 score	0.831

*In multi-label classification, precision and recall refers to the micro precision and micro recall. F1 score is the harmonic average of them.

Conclusions & Discussion

Conclusion

- protein visualization is an effective way to see the distribution pattern of amino acids for each protein
- GRU model, LSTM model, and bi-directional LSTM model are practical approaches to perform the prediction of the subcellular localization of proteins for binary classification, three categories classification and four categories classification

Discussion

- Did not consider 11 categories due to computational complexity
- consider batch-normalization in further study to avoid over-fitting problem
- small data size compared with the number needed for general deep learning model - consider data augmentation mechanism/transfer learning

Thank you!