Spring 2022

Graph Neural Network and Phylogenetic Tree Construction

Gaofeng Pan

Graph Neural Network and Phylogenetic Tree Construction

by

Gaofeng Pan

Bachelor of Science
Tianjin Normal University 2009

Master of Science
Tianjin University 2018

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Computer Science

College of Engineering and Computing

University of South Carolina

2022

Accepted by:

Jijun Tang, Major Professor

Yan Tong, Committee Member

Ramtin Zand, Committee Member

Qi Zhang, Committee Member

Yan Guo, Committee Member

Tracey L. Weldon, Interim Vice Provost and Dean of the Graduate School

Dedication

This dissertation is dedicated to my wife Chao Sun who encouraged me to pursue my dreams and gave me support to finish my work.

Acknowledgments

I would first like to thank my advisor, Dr. Jijun Tang, whose expertise was invaluable in formulating the research questions and methodology. His positive attitude gave me support and his cautious attitude help me overcome difficulties during my research. His persistent help makes it becomes available.

I also would like to thank my dissertation committee members, Dr. Yan Tong, Dr. Ramtin Zand, Dr. Qi Zhang and Dr. Yan Guo. Thanks for their kindness to serve on my dissertation committee and their insightful feedback on my research work. Their comments and suggestions pushed me to sharpen my thinking and brought my work to a new level.

I would also like to acknowledge the members in my research group for their wonderful collaboration and peaceful research environment.

At last, I would like to thank my family members that I value most in my life for their wise counsel and strong support. My wife, my parents and my brother, they are always there for me and always understand my thoughts. Without them, I could not have completed this dissertation or have current achievements.

Abstract

Deep Learning had been widely used in computational biology research in past few years. A great amount of deep learning methods were proposed to solve bioinformatics problems, such as gene function prediction, protein interaction classification, drug effects analysis, and so on; most of these methods yield better solutions than traditional computing methods. However, few methods were proposed to solve problems encountered in evolutionary biology research. In this dissertation, two neural network learning methods are proposed to solve the problems of genome location prediction and median genome generation encountered in phylogenetic tree construction; the ability of neural network learning models on solving evolutionary biology problems will be explored.

Phylogenetic tree represents the evolutionary relationships among genomes in intuitive ways, it could be constructed based on genomics phenotype and genomics genotype. The research of phylogenetic tree construction methods is meaningful to biology research. In the past decades, many methods had been proposed to analyze genome functions and predict genome subcellular locations in cell. However these methods have low accuracy on multi-subcellular localization. In order to improve prediction accuracy, a neural network learning model is defined to extract genome features from genome sequences and evolution position matrix in this research. Experiment results on two widely used benchmark datasets show that this model has significant improvements than other currently available methods on multi-subcellular localization; deep neural network is effective on solving genome location prediction.

Phylogenetic tree construction based on genomics genotype has more accurate

results than construction based on genomics phenotype. The most famous phylogenetic tree construction framework utilizes median genome algorithms to filter tree topology structure and update phylogenetic ancestral genome. Currently, there are several median genome algorithms which could be applied on short genome and simple evolution pattern, however when genome length becomes longer and evolution pattern is complex these algorithms have unstable performance and exceptionally long running time. In order to lift these limitations, a novel median genome generator based on graph neural network learning model is proposed in this research. With graph neural network, genome rearrangement pattern and genome relation could be extracted out from internal gene connection. Experiment results show that this generator could obtain stable median genome results in constant time no matter how long or how complex genomes are; its outstanding performance makes it the best choice in GRAPPA framework for phylogenetic tree construction.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Phylogenetic tree shows the evolutionary relationships between species or genomes in an intuitive way. The leaf nodes in the phylogenetic tree which are called taxa represent species or genome sequences while the path between leaf nodes shows the relation of genome sequence pairs, weighted tree path also shows the similarity between pairs. The taxa in a phylogenetic trees could be species, organisms or genomes [1]. The research of phylogenetic tree construction and analysis is meaningful to biology research. For example, it could help understanding the mechanism of evolution, tracing the origins of organisms, predicting the development of virus or diseases, and so on [2]. Phylogenetic tree could be constructed based on two categories of genomics information, genomics phenotype and genomics genotype. According to the phenotype of genomes, such as their appearance, functions or locations in a cell, they could be clustered into multiple groups; genomes in the same group have similar function and they are close to each other on the evolution path. Another category of phylogenetic tree construction methods are based on genotype, such as genome sequence similarity, genome variants amount, genome evolution distance, and so on.

In the past decades, a lot of methods were proposed to analyze or predict functions or locations of genomes in a cell [3]. One major branch of genome function analysis is subcellular localization. Subcellular localization plays an important role in the research of genome function analysis. Traditional protein subcellular localization methods could accurately locate genomes in a cell by using fluorescent agent and electron microscope device, however this kinds of methods are laborious and time-

consuming because a lot of wet-lab experiments and expensive equipment should be deployed to tag and trace genome development path in a cell. Although a lot of computational methods had been proposed to reduce the cost of traditional experimental methods, they still have shortages on multiple subcellular prediction accuracy [4]. Many computational methods currently available take only gene sequence into consideration and use feature extraction algorithms to define features. Machine learning algorithms predict the locations of genome sequences based on their extracted features. They treat each subcellular as an independent organelle, which results in high accuracy of single subcellular prediction but low accuracy of multi-subcellular prediction [5]. In order to improve the prediction accuracy, a neural network learning model which utilizes convolutional neural network and recurrent neural network as feature extractor from genome sequence and evolution statistics is defined and evaluated in this research. Discussions about significance of genome information category and importance of learning model parts are also developed in this dissertation, which could be helpful to future subcellular localization research.

In order to construct accurate phylogenetic trees, many computation models, in purpose of generating ancestral genomes based on gene distance and optimizing phylogenetic tree topology structures according to evolution costs, were proposed in the past decades [6, 7, 8]. However, they have limitations on problem size because their computing complexity is very high when genomes become longer. Currently, these methods obtain the optimal ancestral for three genomes by utilizing heuristic search strategy, which takes exceptional time to do computation when genome length is long or evolution pattern is complex. In this research, I will try to lift these limitations by using a graph neural network learning model. The major concern of phylogenetic tree construction is computing the similarity or difference between each pair of genome sequences. The relations among genes in a genome sequence couldn't be defined as a Euclidean space data structure because each gene has different number

of neighbors and genome distance don't obey Euclidean rules; graph data structure is a good solution to represent genome sequences. With graph data structure, a node could represent a data sample and an edge could represent relation between a pair of samples; nodes holds their own features and trainable parameters, edges could be directed, undirected, weighted or unweighted.

Graph neural networks are good at graph structure data learning; different from convolutional neural network, messages pass through graph paths and nodes update their features according to adjacency local neighbors [9]. This is very useful for biological data because the connections between each data sample are complex in biology problems. Nodes update their states and parameters based on the difference between generated states and target states at each iteration of graph convolution [10]. GNNs could do node level, edge level and graph level learning tasks. Node level learning models predict the states of nodes after training with a lot of graphs and their target node states [11]. Edge level learning models can discover the connections between pairs of nodes in a graph [12]. Graph level learning tasks focus on graph encoding and dynamic graph generation [13]. In order to generate median genome for phylogenetic tree construction, a graph level learning model will be defined and discussed in this research. There are three major parts to fulfil this learning model, the first one is the generation of training data samples which are pairs of three source genomes and their corresponding median genome, the second one is the construction of learning model framework composed of multiple graph convolution layers and graph encoder-decoder layers, the third part is the definition of algorithm which translates probability matrix back to genome sequence.

There are five chapters in the following part of this dissertation. Chapter 2 gives out a brief literature review about computational methods on phylogenetic tree construction and genome location prediction. Chapter 3 proposes a neural network learning model for subcellular localization, evaluation and discussion about this model are

also included. Chapter 4 fulfils the three parts of median genome generator, which are median genome training samples generation strategy, graph learning model framework and probability matrix conversion algorithm. Multiple evaluation processes are applied on this median genome generator and evaluation results are listed in detail. Even more, the advantages and disadvantages of this generator and other median genome algorithms are discussed based on these evaluation results. Chapter 5 talks about the key points of phylogenetic tree construction framework GRAPPA and how the median genome generator helps to improve its performance. At last, chapter 6 makes a conclusion of this dissertation. The major contributions of this research are listed out and several future work directions are discussed in this chapter.

Chapter 2

Brief Literature Review of Computational Phylogenetic and Methods

2.1 Phylogenetic Tree

Phylogenetic tree is a relation tree of different species, organisms or genes which have the same ancestral during the evolution process. It represents the evolutionary relationship between each leaf node, as shown in figure 2.1. Each leaf node represents a specie, an organism or a genome and all the nodes have the same ancestral. They developed and acquired different features and functions during the process of biological evolution with random mutations to their genomes and different phenotypes are appeared. With the phylogenetic tree of specific species or specific genes, we could gain insights about character obtain and character loss of their evolution process and understand the underlying evolution mechanisms of these taxa[1].

There are two types of phylogenetic tree, rooted tree and unrooted tree, according to whether there exists an ancestral taxa in the tree. As shown in figure 2.1(a), the rooted phylogenetic tree has a group of taxa, which are different species or different genomes and which we know their phenotypes or their genotypes in current environment, as the leaf nodes of the tree. The internal nodes of the tree are unknown and they are inferred from the leaf taxa. Each internal node is assumed to be the common ancestral of its children in the tree and there exists a root node which is the ancestral of all the leaf nodes. For example, the rooted phylogenetic tree in figure 2.1(a) has 4 species and the root specie R is the ancestral of the species S_1, S_2, S_3 and S_4.

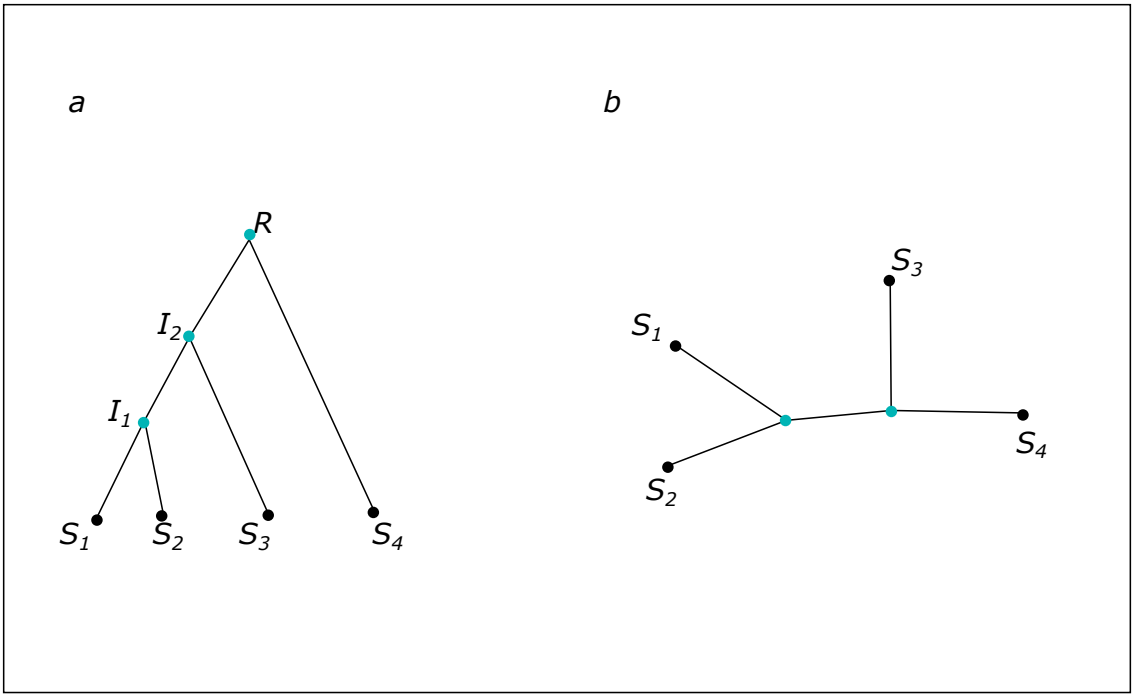

Figure 2.1 An example of rooted phylogenetic tree and unrooted phylogenetic tree with four genomes

The species S_1 and S_2 are similar with each other and they are developed from an ancestral I_2 which has the same ancestral I_1 with specie S_3. The rooted phylogenetic tree shows the relationships between each species and shows the evolution steps from their ancestral to themselves. Figure 2.1(b) is the unrooted phylogenetic tree of the 4 species S_1, S_2, S_3 and S_4. The connections between each node in the tree represents the distance between two species, which is a variable to evaluate the difference between them. There are no information about the ancestral of each specie and the underlying biological evolution process isn't clear. From this figure, we can see that species S_1, S_2 are similar with each other and specie S_4 has more difference to S_1, S_2 than specie S_3.

The phylogenetic tree of a group of taxa have many different forms and the number of possible phylogenetic trees is very large. Assume we only consider the situation that the internal node only has two descendants in the rooted phylogenetic tree and the

intersection only has three neighbors in the unrooted phylogenetic tree, the number of possible rooted phylogenetic trees for a group of distinguishable species or genomes is

$$\frac{(2n-3)!}{2^{n-2}(n-2)!},\ (n \geq 2)$$

and the total number of possible unrooted phylogenetic trees of the same taxa is

$$\frac{(2n-5)!}{2^{n-3}(n-3)!},\ (n \geq 3)$$

in both of which, n is the number of leaf nodes[14]. For example, figure 2.2 shows several different topology structures and node labels of the phylogenetic tree of four taxa. Figure 2.2(a) lists out four out of the 15 possible rooted phylogenetic trees in two different topology structures and figure 2.2(b) lists out the three possible unrooted phylogenetic trees of these four taxa. When there are only specie or two species, obviously the shape of the phylogenetic tree is identify. As the number of species in the phylogenetic tree becomes larger, the topology structure and the node label becomes complex. The topology structure of the rooted phylogenetic tree only has one form when there are three taxa, however the node labels could have different forms. The unrooted phylogenetic tree of three taxa is in the form of star tree which has one joint node and the three taxa are the neighbors of this node.

Traditional phylogenetic tree construction methods based on the phenotype of species or organisms. As the development of gene sequencing technology, more and more genomics data becomes available and a bunch of computational methods based on the organism genotypes were proposed in the past several years. These kind of methods construct the phylogenetic trees according to the variation among a group of genomes or gene sequences. The variation of two genomes could be the distance defined for the genomes or the character which exists in the gene sequences[6].

Phylogenetic tree construction methods such as neighbor joining [15], Unweighted Pair Group Method with Arithmetic mean(UPGMA) [16], Fitch Margoliash Algorithm [17], ClustalW [18] and the variation algorithms based on these methods, such

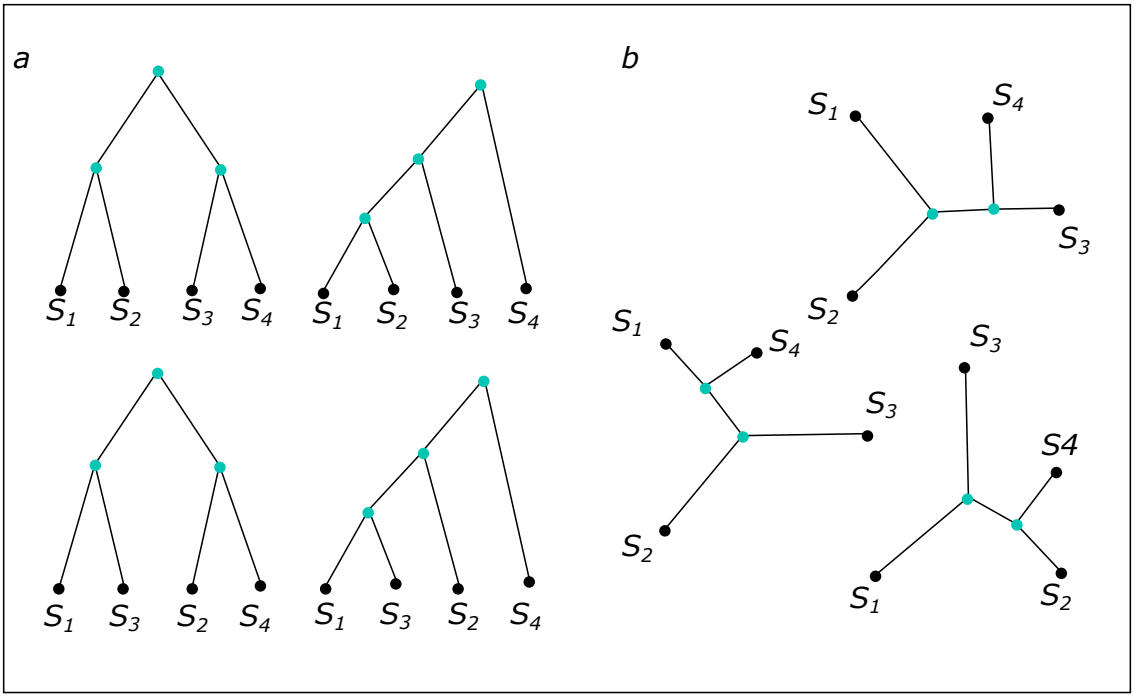

Figure 2.2 Different topology structure examples of rooted phylogenetic tree and unrooted phylogenetic tree with four genomes

as BIONJ [19], MISHIMA [20], SeaView [21] and so on, are distance dependent. These methods make multiple sequence alignment on the target genomes to figure out the same part of the sequences and then use the size of different pairwise as the distance between genomes. Based on the calculated distance matrix, the genomes are clustered together as a tree in an iterative manner[8]. Because multiple sequence alignment couldn't be estimated in an efficient way and the cluster algorithms don't consider the optimization of the whole generated tree, the distance based methods have some limitations when they are applied on distantly related genomes.

Different from distance based phylogenetic tree construction methods applying only the gene positions in a genome into consideration, character based phylogenetic tree construction methods takes the statistic features of genomes as the classification criteria which provide more meaningful information to help generating better phylogenetic trees. There are three categories of character based methods which are Max-

imum Parsimony(MP), Maximum Likelihood(ML) and Bayesian Inference(BI). The maximum parsimony method considers multiple possible phylogenetic tree topologies which are generated under certain assumptions about the evolution mechanisms and takes the one which has the least evolutionary events as the target phylogenetic tree topology. It ignores the evolution feature that there are different rates for the different evolution event and there exists diverse transitions rate among genome sites. The Maximum Likelihood method builds up a statistical model with a group of adjustable parameters to represent the evolutionary structure and then find out the values of parameters with which the evolutionary structure gains the highest possibility to occur. A lot of phylogenetic tree construction methods take this strategy to achieve the optimal phylogenetic tree topology because of the interpretability of model, such as FastTree, IQ-Tree, PAML, Phyml, MP-EST and so on [22, 23, 24, 25, 26]. The Bayesian Inference method also takes a statistical model and tries to maximize the likelihood of the phylogenetic tree model with a series of Markov Chain Monte Carlo processes. The methods MrBayes, PhyloBayes, BEAST and RevBayes belong to this category [27, 28, 29, 30]. All of them are trying to select out the maximum posterior probability tree from a group of simulation trees based on existing genome features. The selected tree is the most reasonable phylogenetic tree given the genomes and the Bayesian Inference model.

2.2 Subcellular Localization

The basic functional unit in an organism is cell and cells perform their function by using a bunch of subcellulars enclosed in it. The genomes or proteins have different expression when they exist in different subcellulars. Figuring out the locations of genomes and proteins in a cell is critical to understand their function and their evolutionary process in the formation of cells. Traditional experimental methods to identify the subcellular location of proteins with fluorescent agents and fluorescence

microscopes by tracking the fluorescent radiation emitted from proteins. This kind of methods are expensive and time-consuming because of there is very high requirement of equipment and operators to operate the experiments.

In past decades, many computational methods to identify the subcellular location of genomes and proteins were proposed in the purpose of reducing the cost of traditional experimental methods[4]. Computational methods predict the location of a protein or a genome in the cell according to the amino acid components and sequence order of these amino acids in the protein or genome by applying machine learning models or deep learning models. Besides the amino acid components and their position information, some computational methods takes other kind of information into consideration, such as Gene Ontology(GO) annotations, physical-chemical properties, evolution characteristics and so on.

The computational subcellular localization methods could be divided into two categories, machine learning model based methods and deep learning model based methods. Most of these methods take amino acid sequences as the only information source to make predictions and other methods take extra property source of the target protein to help improving the accuracy of prediction results[5].

The machine learning model based methods extract features from the gene sequences or the amino acid sequences and classify the extracted features into different category. Based on the sequence features, the target sequence falls into one or more subcellular in which this sequence has great chances exists. The subcellular localization methods SubLoc, MKSVM, FSVM-KNR, IMMMLGP, mGoF-Loc are machine learning model based[31, 32, 33, 34, 35]. The general procedure of these methods is that, applying several feature extraction functions to extract features from original sequence inputs and then using kernel function as the inner product of the extracted feature space to condense the features, at last classify the processed features into different category.

The method SubLoc uses Support Vector Machine(SVM)[36] to classify the basic sequence vector of protein[3]. This method takes the encoded sequence vector as the extracted features without other feature extraction methods and it only has one kernel function to make classification. In order to improve the accuracy of prediction results, the method MKSVM applies multiple kernel functions with the SVM classifier and adds PsePSSM and PsePP as extra feature extraction functions[32]. With the PsePSSM and PsePP, the type of features expand to 6 categories and the number of total extracted features increase to 7931, which make the prediction accuracy a great improvement. Variations of SVM classifier, such as Fuzzy Support Vector Machine(FSVM) [37] and Bayesian Support Vector Machine(BSVM) [38], could play the rule as classifiers to predict subcellular location. Adjustments to the weights of samples in these variation classifiers could help the methods to make more accurate predictions, for example the method FSVM-KNR which generates the membership values of amino acid sequences and classifies their position with Kernelized Neighborhood Representation function and Fuzzy Support Vector Machine performs better than other SVM based methods[33].

The GO annotations of genes or proteins are another form of gene description along with the gene sequence. It has been utilized in the research of gene expression, gene function prediction, gene classification and so on. The GO description of a gene sequence is more specific, includes gene function and gene expression. Since the location and function of genes have correlation, classifiers could easily identify where a gene exists in a cell based on its function. There are many subcellular localization methods utilize the GO annotations of genes to supplement the gene sequence information in the prediction process. The methods mGOASVM include GO annotations as extracted features for classification [39]. Sometimes the specific GO annotation of a gene doesn't exist in the GO database, then the GO annotation of its homologs which could be obtained by using BLAST algorithm will be used as a replacement.

The method mGOASVM takes this strategy to avoid the situation that some genes absent in the GO database. Other subcellular localization methods utilized GO annotation have mLASSO-Hum[40], Hum-mPLoc[41, 42] and iLoc-Hum[43]. All these methods have the similar procedure but with different properties in the classification feature space. Although GO annotations have descriptions about gene function to solve the difficulty encountered in gene feature recognition, they weren't widely applied in prediction of gene subcellular locations since a great number of genes exists in nature environment are not available in the GO annotation database at current time.

Physicochemical Property is another very popular gene feature in the design and implementation of subcellular prediction methods. Every nucleotide has its own specific physical properties and chemical properties, amino acid is the same. These properties include molecular mass, hydrophobicity, polarity and so on. They could act as the features of a gene sequence or a protein sequence in the prediction of its subcellular location. When two molecular combined together to form a larger molecular, they exhibit other characteristics besides their own characteristics. Six physical structural properties of two molecular combination had been identified, they are twist, tilt, roll, shift, slide and rise. There are 16 types of possible combination of the four nucleotide, so there are 16 possible values for each element of the six physical structural properties. With physicochemical properties as the sequence feature, a gene sequence could be transformed into a $6 \times (l-1)$ matrix. This feature matrix expands the feature space and helps classifiers to distinguish the function and location of gene sequences.

Feature extraction algorithms, such as Discrete Wavelet Transform, Pseudo Amino Acid Composition and Average Blocks, have been tested in subcellular prediction methods. The experiment results of these methods shown that these feature extraction algorithms have great ability to condense the raw features of gene sequences and

they are good at filtering out the insignificant properties from a large amount of raw features. The Discrete Wavelet Transform (DWT) algorithm discretely sample the raw feature matrices with temporal resolution and decompose them into two categories, approximation coefficients and detailed coefficients [44]. The approximation coefficients will be transformed to cosine coefficients and the detailed coefficients will get next level of wavelet transform. In a multiple level DWT, each level of transform will generate a sampled feature vector. The features have high coefficients with gene expressions could be filtered out at each level of transform. These sampled features highlights the properties of gene sequence related to gene location, so the subcellular localization method which make classification based on these properties has high prediction accuracy.

The Pseudo Amino Acid Composition (PseAAC) algorithm combines features located at different position of a gene sequence into one feature [45]. It considers higher level relations exist between distantly separated genes in genome and includes the whole genome structure information in the classification features. For a $m \times n$ raw feature matrix, the PseAAC algorithm could condense it into a $m \times (t+1)$ feature matrix in which the t is the parameter to adjust the density level. The transformed feature matrix is like:

$$P(M_m^n) = \begin{bmatrix} p^0, & p^1, & p^2, & \cdots, & p^t \end{bmatrix}$$

in which M is the raw feature matrix and t is the distance parameter to control the density of feature matrix. The element p^0 and $p^k (1 \leq k \leq t)$ are defined as:

$$p^0 = \frac{1}{n} \left[\sum_{i=1}^{n} M_1^i, \ \sum_{i=1}^{n} M_2^i, \ \sum_{i=1}^{n} M_3^i, \ \cdots, \ \sum_{i=1}^{n} M_m^i \right]^T$$

and

$$p^k = \frac{1}{n-k} \left[\sum_{i=1}^{n-k} \left| (M_1^i - M_1^{i+k}) \right|, \sum_{i=1}^{n-k} \left| (M_2^i - M_2^{i+k}) \right|, \cdots, \sum_{i=1}^{n-k} \left| (M_m^i - M_m^{i+k}) \right| \right]^T.$$

The experience value of t according to several previous experiment results of subcellular location prediction is around 10 as the distance of gene expressions in the

secondary structures. Small t values couldn't cover the distantly located genes and too large t values would bring noise information into the condensed features.

The Average Block (AvBlock) algorithm has the similar idea as the algorithm PseAAC but with different condense function [46]. This algorithm divides the features into groups and combines the features in each group together to obtain their average value. It keeps the local properties of a specific gene site while considering the global properties of the whole gene sequence at the same time. For a raw feature matrix M which has $m \times n$ elements, the condensed feature matrix generated by AvBlock algorithm is

$$A(M_m^n) = \left[a^1, \quad a^2, \quad a^3, \quad \cdots, \quad a^t\right]$$

in which t the number of blocks the features divided into and the element $a^k (1 \leq k \leq (t-1))$ is defined as:

$$a^k = \frac{1}{l}\left[\sum_{i=1}^{l} M_1^{i+s};\ \sum_{i=1}^{l} M_2^{i+s};\ \cdots;\ \sum_{i=1}^{l} M_m^{i+s};\right]^T, \left(l = \left\lfloor \frac{n}{t} \right\rfloor, s = (k-1) \times l\right)$$

and a^t is defined as:

$$a^t = \frac{1}{n-r}\left[\sum_{i=1}^{n-r} M_1^{i+r}; \sum_{i=1}^{n-r} M_2^{i+r}; \cdots; \sum_{i=1}^{n-r} M_m^{i+r};\right]^T, \left(l = \left\lfloor \frac{n}{t} \right\rfloor, r = (t-1) \times l\right).$$

By adjusting the value of t, the raw feature matrix could be condensed in different densities. It is important to use a suitable t value in subcellular localization methods. The influence of single site will be enlarged with small t values while with large t values some significant properties in gene sequence would be concealed from the location classifiers. Both of these situations will reduce the accuracy of subcellular site prediction.

2.3 Convolutional Neural Network

Convolutional Neural Network (CNN) is a multiple layer perceptron algorithm with regularization functions and activation kernels which could extract and condense features automatically from raw data like gene sequences or protein sequences. In the

past decades, the CNN algorithm had been successfully applied in computer recognition such as computer vision, image processing, medical image analysis, natural language translation, time series data prediction and so on [47, 48, 49, 50]. As the genome sequencing technology developed and plentiful biological datasets were assembled in the past several years, utilization of CNN algorithm in computational biology research becomes available.Some biological problems, such as non-coding gene sequence function prediction [51], genome sequence methylation sites classification [52], protein-protein interaction site identification [53] and raw Hi-C data noise reduction [54], could gain accurate solution with the assistance of CNN algorithm.

The general framework of computational methods implemented based on CNN algorithm in purpose of solving the biological site prediction problems is that, extracting features from raw biological data samples with multiple CNN layers and filtering out the significant features with activation functions, then condensing the features together and generating the probability of each site by passing the condensed features through Sigmoid function which has output value between 0 and 1. The learning process adjusts the kernel parameters of CNN layers with an appropriate optimization algorithm based on the difference between target values and prediction values. The target values of site prediction is a vector of binary values which represent the existence or not of each site and the prediction values correspond to the probability of each site [55].

The difference between target values y and prediction values $\hat{y}$ are evaluated with a binary cross entropy equation which is defined as:

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \times log\hat{y}_i + (1 - y_i) \times log\left(1 - \hat{y}_i\right)\right)$$

in which n is the number of sites should be predicted, y_i is the binary value at site i and $\hat{y}_i$ is the prediction value of site i. During each learning epoch, function $L(y, \hat{y})$ is passed back to CNN layers using backpropagation algorithm and optimization algorithms could adjust the kernel parameters of each CNN layer according to the

derivative of L to reduce the difference between prediction values and target values [56]. A bunch of optimization algorithms, such as Stochastic Gradient Descent (SGD) [57], Averaged Stochastic Gradient Descent (ASGD) [58], Adaptive Gradient (AdaGrad) [59], Adaptive Moment Estimation (Adam) [60], Root Mean Square Propagation (RMSProp) [61], AdaDelta [62] and so on,could optimize the CNN parameters with a selection of suitable hyperparameters.

There are a lot of subcellular localization methods take CNN layers as learning kernel to pursue higher accuracy on the prediction of gene sequence location, some of them utilize Recurrent Neural Network (RNN) and attention network structure as a supplement. For example, the method DeepLoc uses attention mechanism to identify the important regions in a gene sequence [63]. The method ImPLoc [64] applies CNN layers on immunohistochemistry (IHC) images to predict protein subcellular locations [64]. Some methods will also apply pre-processing like PseAAC transform on the raw gene sequence in order to make up the limitation of local region convolution, such as the methods pLoc_Deep-mHum, pLoc_Deep-mEuk and pLoc_Deep-mvirus [65, 66, 67]. Other transformations appeared in subcellular localization methods have dipeptide composition of gene sequence used by DeepPSL [68], Gamma correction on microscopy images used by DeepYeast [69] and human protein subcellular localization [70, 70]. The multiple CNN layers framework and data processing strategy could be applied in the methods of messenger RNA prediction and the methods of long non-coding RNA prediction. The methods like IncLocator [71], RNATracker [72] and DeepLncRNA [73] show that it is feasible to use CNN to predict subcellular locations.

2.4 Genome Rearrangement

Biological evolution processes consists of a series of genome rearrangement events in which the chromosomes of an organism acquire or loss a segment of genes. By

these genome rearrangement events, the organisms could gain different genotypes and phenotypes. Genome rearrangement events occurred on single genome could be inversion, transposition and fission and the events occurred on two genomes include transposition and fusion. The inversion event is that a segment of genes in a genome is changed to the reverse order and the direction of genes in the segment is turned around. Like the example shown in figure 2.3(a), the segment between point g_a and point g_b in the sample genome is reversed as the head of the segment is connected to point g_b and the tail of the segment is connected to point g_a. Figure 2.3(b) shows an example of transposition event in which two segments s_a and s_b swap with each other in the genome while the gene order and gene sequence of each segment keep their original states. The translocation event exchanges the segments in two genomes with each other as shown in figure 2.3(c). The fusion event and the fission event are inverse operations of each other, as shown in figure 2.3(d). The fusion event merges two separate genomes into one while the fission event split one genome into two parts.

Double-cut-and-join(DCJ) model defines the genome rearrangement events into four categories [74]. The four categories, as shown in figure 2.4a, are break, merge, replace, cross. The break operation breaks an adjacency into two end points, the merge operation connects two separate end points into one adjacency point, the replace operation breaks an adjacency point then connects one of the generated end points with a new end point and the cross operation breaks two adjacency points then connects the end points with each other. When considering the genome as a linear sequence of genes and only one genome involved in the rearrangement process, there are two types of genome rearrangement events, which are inversion operation and transposition operation, as shown in figure 2.4b. These two operations corresponds to one or two DCJ cross operations.

The inverse rearrangement event changes the direction of a segment of genes into the opposite direction. For example, there is a genome which consists of five genes

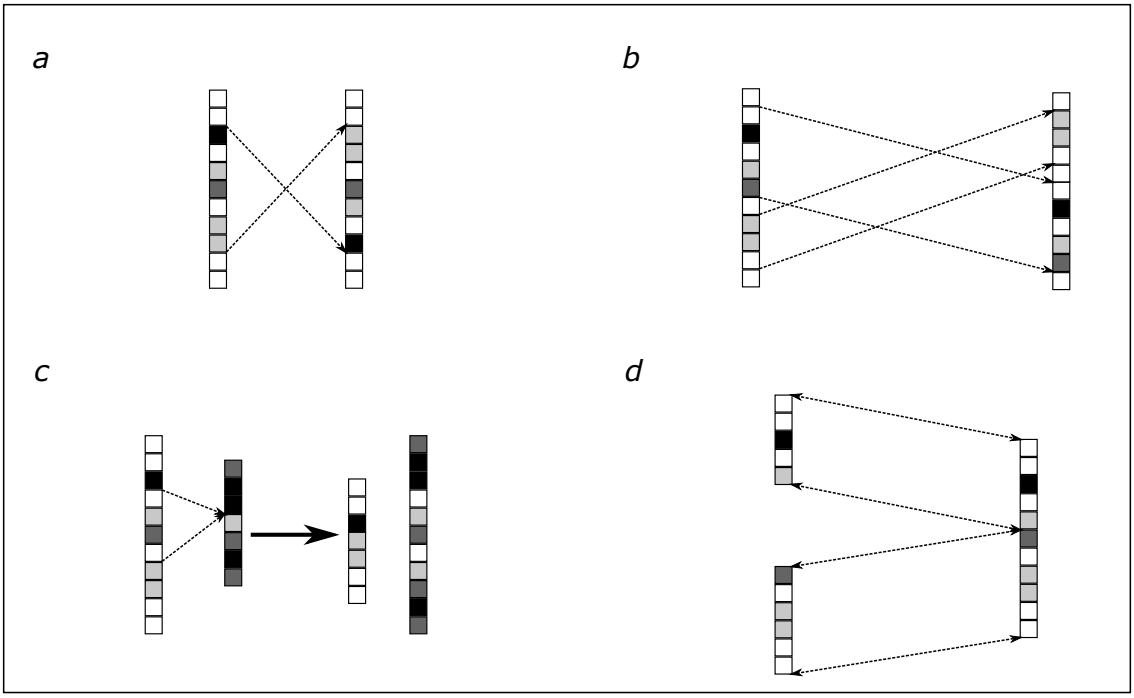

Figure 2.3 Genome rearrangement operations: inversion, transposition, translocation, fusion and fission

$\{g_1, g_2, g_3, g_4, g_5\}$ and the direction of those three genes are from left to right. When an inverse rearrangement event happened on gene segment g_2g_3, the direction of the gene segment g_2g_3 will change to right to left. In order to represent the direction of a gene in genome, symbol '+' and '−' will be append to gene name. The symbol '+' means the direction of the gene is from left to right and the symbol '−' means the direction of the gene is from right to left. So the inversion of g_2g_3 is like

$$[+g_1, +g_2, +g_3, +g_4, +g_5] \Rightarrow [+g_1, -g_3, -g_2, +g_4, +g_5]$$

The inversion not only changes the direction of genes but also changes the position of genes.

The inverse rearrangement event has one DCJ cross operation. For example, in order to inverse the segment g_2g_3 in genome $[+g_1, +g_2, +g_3, +g_4, +g_5]$, two adjacency, adjacency between g_1 and g_2 and adjacency between g_3 and g_4, are broken, then g_1

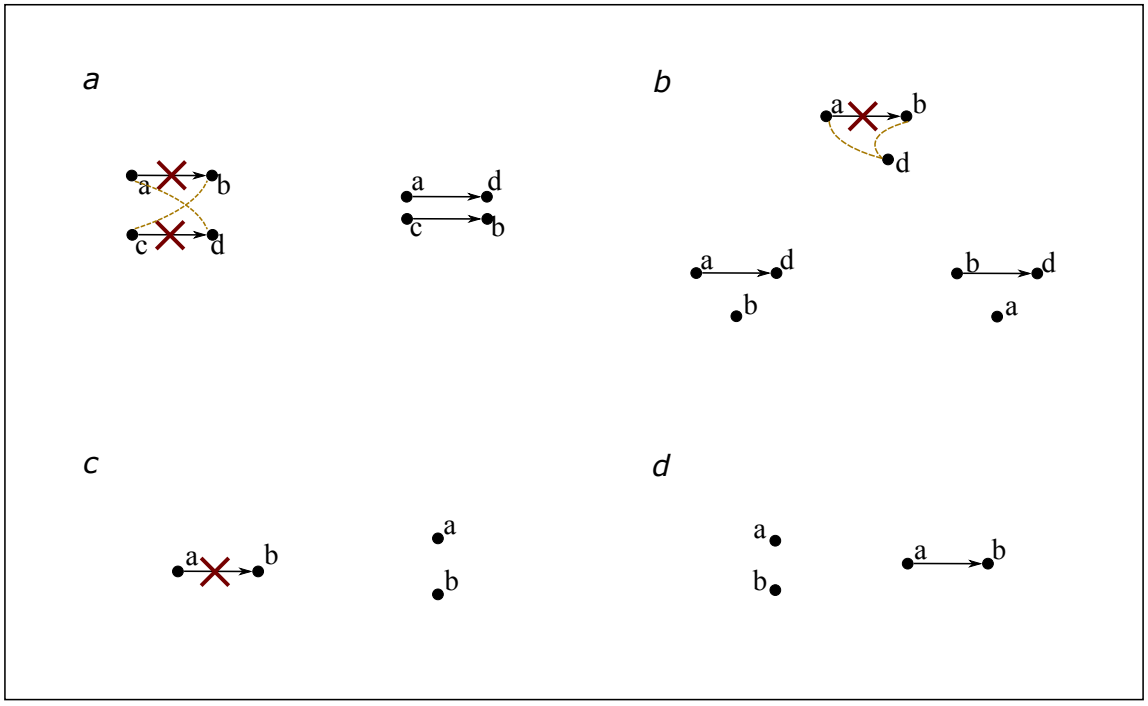

Figure 2.4 Double cut and join model operations demonstration

and g_3 are connected together, g_2 and g_4 are connected together.

The transposition rearrangement event changes the position of two separate segments in a genome by swapping them with each other. For example, transposition of gene g_2 and g_4 will move g_2 to the fourth position and move g_4 to the second position in the genome. It looks like

$$[+g_1, +g_2, +g_3, +g_4, +g_5] \Rightarrow [+g_1, +g_4, +g_3, +g_2, +g_5]$$

After transposition, the gene segments will keep their original directions.

The transposition rearrangement event can also be fulfilled with two DCJ operations. For example, transposition of gene g_2 and g_4 consists of a DCJ cross operation between adjacency 'g_2g_3' and adjacency 'g_4g_5', in which g_2 connects to g_5 while g_3 and g_4 connect with each other in a circle, and a DCJ cross operation between adjacency 'g_1g_2' and adjacency 'g_3g_4', in which g_1 connects to g_4 while g_3 connects to g_2. The genome changes from $[+g_1, +g_2, +g_3, +g_4, +g_5]$ to $[+g_1, +g_4, +g_3, +g_2, +g_5]$.

A series of inverse rearrangement events could trans the location of two genome segments. In this research, only inverse rearrangement will be applied to simulate the genome evolution process. To trans the location of two segments which are adjacent to each other in the genome, three inverse rearrangement events are required. For example, in order to swap the locations of genes g_2 and g_3 in genome $[+g_1, +g_2, +g_3, +g_4]$, firstly inverse segment 'g_2g_3' to get genome $[+g_1, -g_3, -g_2, +g_4]$, then inverse g_3 and g_2 one by one to get genome $[+g_1, +g_3, +g_2, +g_4]$. To trans the location of two segments which are separated by another gene segment, four inverse rearrangement events could fulfill it. Firstly inverse the three segments as a whole segment to change their location, then inverse each segment one by one to change their directions.

Double-cut-and-join distance is defined as the minimum number of DCJ operations needed to transform a genome into another one. It represents the difference and distance between two genomes on the evolutionary tree. Small distance means there are few number of gene rearrangement events while one genome evolved to another one and these two genomes are close to each other on the evolutionary tree.

The distance between two genomes could be computed in linear time. Assume there are two genomes G_1 and G_2, they have the same number of genes and their gene sets are identical to each other and each gene only appears one time in each genome. To compute teh DCJ distance between genome G_1 and G_2, as illustrated in figure 2.5, firstly starting from one end point p_i^1 of g_i in G_1 draw a line between this end point and the same end point p_i^2 of g_2 in G_2, then draw line between the adjacency point of p_i^2 in G_2 and the corresponding end point in G_1, repeat this process until all the end points in G_1 are connected with their corresponding end points in G_2. The DCJ distance between G_1 and G_2 is

$$\Delta(G_1, G_2) = n - c + 1$$

in which n is the size of gene set of G_1 and G_2 and $n = |G_1| = |G_2|$, c is the number

of cycles formed by the lines between end points from G_1 and G_2. It takes linear time to connect the corresponding end points in two genomes with n genes and linear time to count the numbers of formed cycles, so the DCJ distance between two genomes could be computed in linear time.

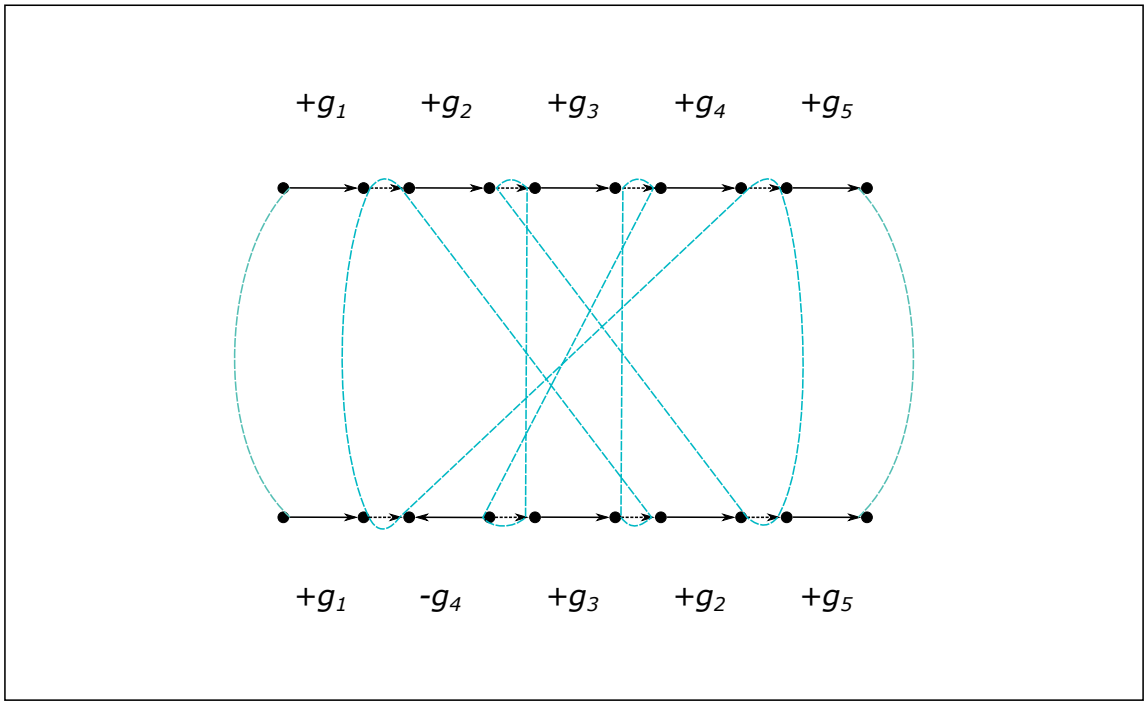

Figure 2.5 Genome distance computation method under double cut and join model

Given any two genomes $\{G_i, G_j\}$ with identical gene set, the double cut and join distance between these two genomes is $\Delta(G_i, G_j) \geq 0$. When the genome G_i and G_j are identical to each other, they have the same gene sequence and the same gene directions, the DCJ distance is 0, otherwise the DCJ distance is greater than 0. At the same time, $\Delta(G_i, G_j) \leq n$ in which n is the size of gene set. The distance from G_i to G_j is equal to the distance from G_i to G_j because all the DCJ operations are reversible. Since the DCJ distance is the minimum number of DCJ operations from one genome to another one, it obeys the triangle inequality rules. For any three genomes $\{G_i, G_j, G_k\}$ which are identical with each other on the gene set, has the

property

$$\Delta(G_i, G_j) \leq \Delta(G_i, G_k) + \Delta(G_k, G_j)$$

when $\Delta(G_i, G_k) + \Delta(G_k, G_j)$ equals to $\Delta(G_i, G_j)$, G_k is on the gene rearrangement path from G_i to G_j.

The median problem in evolution tree construction is to find the genome which is the closet to a group of genomes. Given a genome set $\mathcal{G}$ in which the genome has same gene set and same genome length, the DCJ distance between genome G_m and $\mathcal{G}$ is defined as

$$D(G_m|\mathcal{G}) = \sum_{G_i \in \mathcal{G}} \Delta(G_m, G_i).$$

The solution of median problem is $\mathcal{G}_m$ and

$$\mathcal{G}_m = argmin_{G_m} D(G_m|\mathcal{G})$$

When $\mathcal{G}$ has three elements, $\mathcal{G}_m$ is the solution to the median of three problem.

For a genome set $\mathcal{G}$ with n genes in its genome, the candidate search space for $\mathcal{G}_m$ is very large. The number of permutation of n genes is $n!$. Each gene could have 2 directions and the total number of direction combination will be 2^n. So the size of the search space is $2^n \cdot n!$; it is not possible by computing the distance between $\mathcal{G}$ and each possible genome to find the solution to median problem.

The distance $D(G_x|\mathcal{G})$ from genome G_x to genome set $\mathcal{G} = \{G_a, G_b, G_c\}$ has a lower bound. For G_x and each of $\mathcal{G}$, as shown in figure 2.6.a there is

$$\Delta(G_x, G_a) + \Delta(G_x, G_b) \geq \Delta(G_a, G_b)$$

$$\Delta(G_x, G_a) + \Delta(G_x, G_c) \geq \Delta(G_a, G_c)$$

$$\Delta(G_x, G_b) + \Delta(G_x, G_c) \geq \Delta(G_b, G_c)$$

and

$$\sum_{G_i \in \mathcal{G}} \Delta(G_x, G_i) \geq \sum_{i,j \in \{a,b,c\}} \frac{\Delta(G_i, G_j)}{2}.$$

So the half of distance sum between these three genomes in $\mathcal{G}$ is the lower bound of distance between other genomes and $\mathcal{G}$. Since the distance between genomes are integer number,

$$D(G_x|\mathcal{G}) \geq \left\lceil \sum_{i,j\in\{a,b,c\}} \frac{\Delta(G_i, G_j)}{2} \right\rceil.$$

When $D(G_x|\mathcal{G}) = \lceil \sum \Delta(G_i, G_j)/2 \rceil$, G_x is one of the medians of the genomes in $\mathcal{G}$. If G_x is on the three gene rearrangement paths between G_a, G_b and G_c at the same time, as shown in figure 2.6.b, then $\Delta(G_x, G_i) + \Delta(G_x, G_j) = \Delta(G_i, G_j)(i, j \in a, b, c)$ and G_x is a median genome of $\mathcal{G}$. According to the triangle inequality rule and the ceiling number rule, in order to reach the lower bound of distance between G_x and $\mathcal{G}$, G_x should be on two of the three gene rearrangement paths between G_a, G_b and G_c at least.

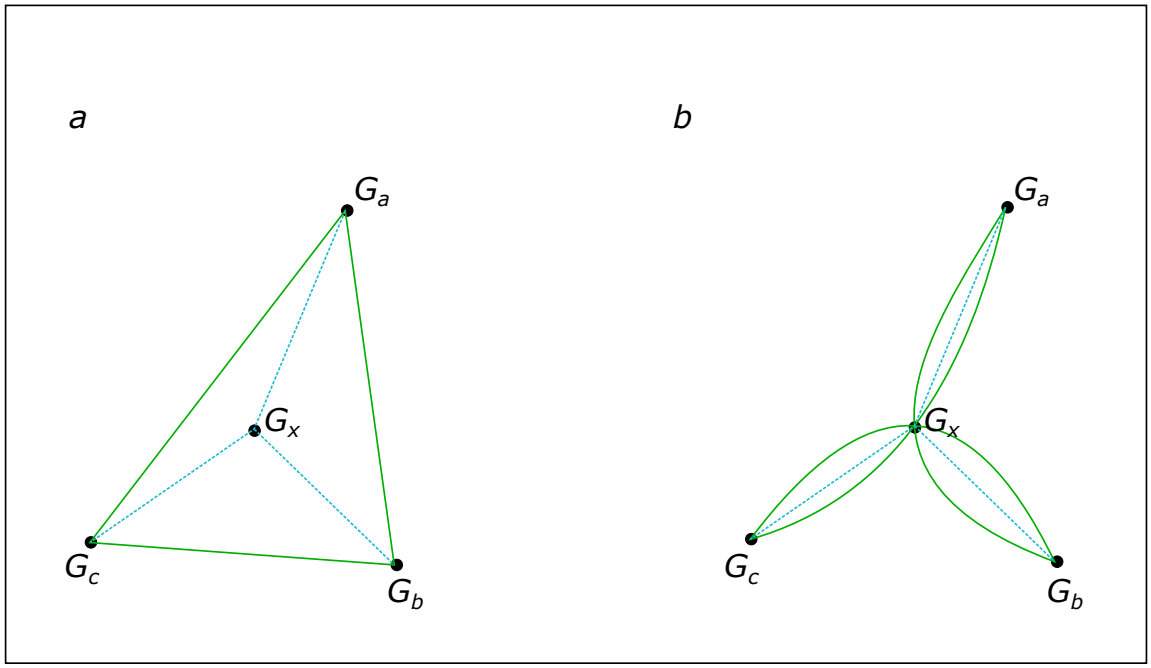

Figure 2.6 Median genome of three homologous genomes and the lower bound of median genome

2.5 Graph Neural Network

Graph Neural Networks (GNN) could learn the patterns obscured under complex graph structure data[9]. Different from CNN which operate convolution function on a local range in the Euclidean Space, GNNs take convolution along the graph edges and update node features according to the features and states of its neighbor nodes[75]. Variation networks of GNN also consider the edge weights in node feature propagation process. Figure 2.7 shows the difference between GNNs and CNNs with figure 2.7(a) is the basic protocol of GNN and figure 2.7(b) represents the convolution operation at each layer of CNN.

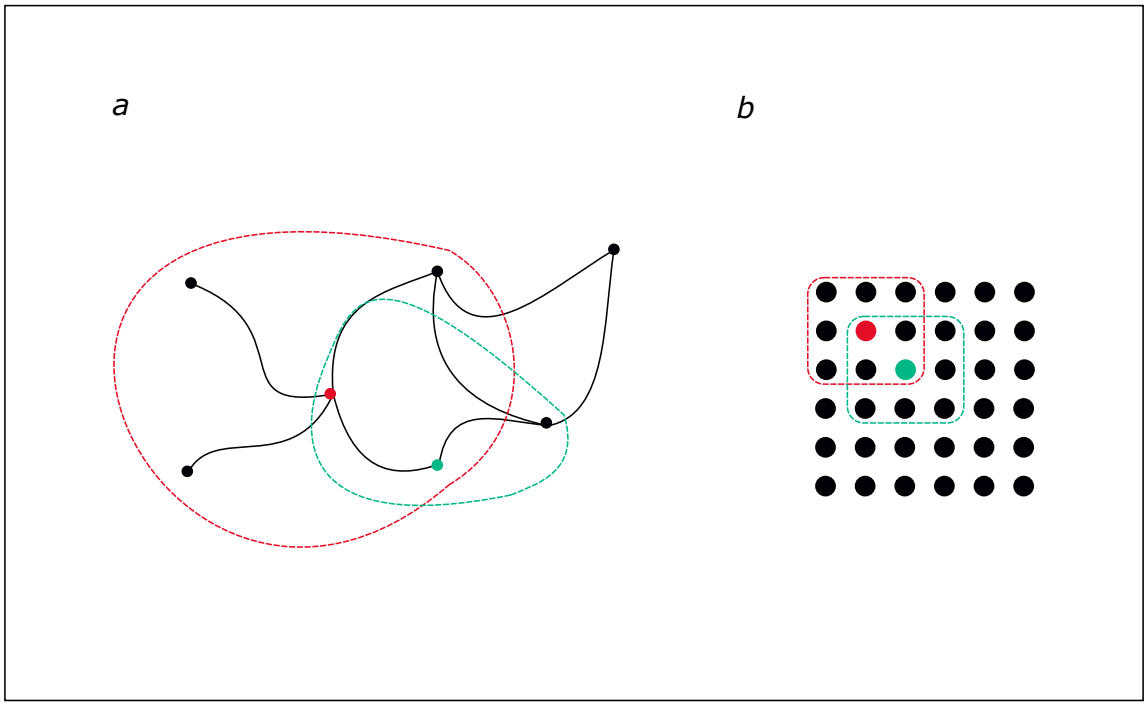

Figure 2.7 Differences between graph neural network and convolutional neural network

For a graph $G = (V, E)$ which has n nodes in $V = [v_1, v_2, \cdots, v_n]$ and m edges in E, the adjacency matrix A of this graph is a $n \times n$ matrix with $A_{ij} = 1(e_{ij} \in E)$ and other elements keeps 0. If graph G is a weighted graph, A_{ij} is the edge weight between v_i and

v_j. The node feature matrix of this graph is defined as $X^0 = [x_1, x_2, x_3, \cdots, x_n], (x_i \in R^d)$ in which d is the feature dimension of node. The label vector of this graph is $Y = [y_1, y_2, y_3, \cdots, y_n], (y_i \in R)$ with y_i is the target label of node v_i. So a graph could be defined as $G = (A, X^0, Y)$. The node feature matrix is updated after convolution operation on each node's neighbor at each layer of GNN. At layer i ($i \geq 1$) the node feature matrix

$$X^i = \hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} X^{i-1} \Theta^i$$

in which $\hat{A} = A + I$ is the adjacency matrix with self-loops, Θ^i is the trainable weight matrix at layer i and

$$\hat{D}_{ij} = \begin{cases} \sum_{j=t}^{n} \hat{A}_{it}, & i = j \\ 0, & i \neq j \end{cases}$$

is the diagonal degree matrix of adjacency matrix $\hat{A}$.

GNNs could be applied in node level prediction, edge level prediction and graph level transformation. In node level prediction, GNN is able to predict the states of each node depending on the convolutional node features[11].Edge level prediction is to discover the unrepresented connection between some pairs of nodes in a graph[12]. Graph level transformation encodes a graph into a matrix representation with the help of convolution layers and feature propagation operation, then decode the matrix into a new graph[13].

Figure 2.8 shows the procedure of node level prediction GNN. At each layer, a node collects the features of its neighbors then updates its features according its current feature and the collected features. After a number of layers of feature propagation, an activation function like Softmax function or Sigmoid function generates the node's final state by using its features. The trainable parameters at each layer could be optimized with backpropagation and loss value differentiation.

Figure 2.9 is the framework of graph level transformation GNN. The input graph is transformed into a matrix representation Z after multiple layers of convolution

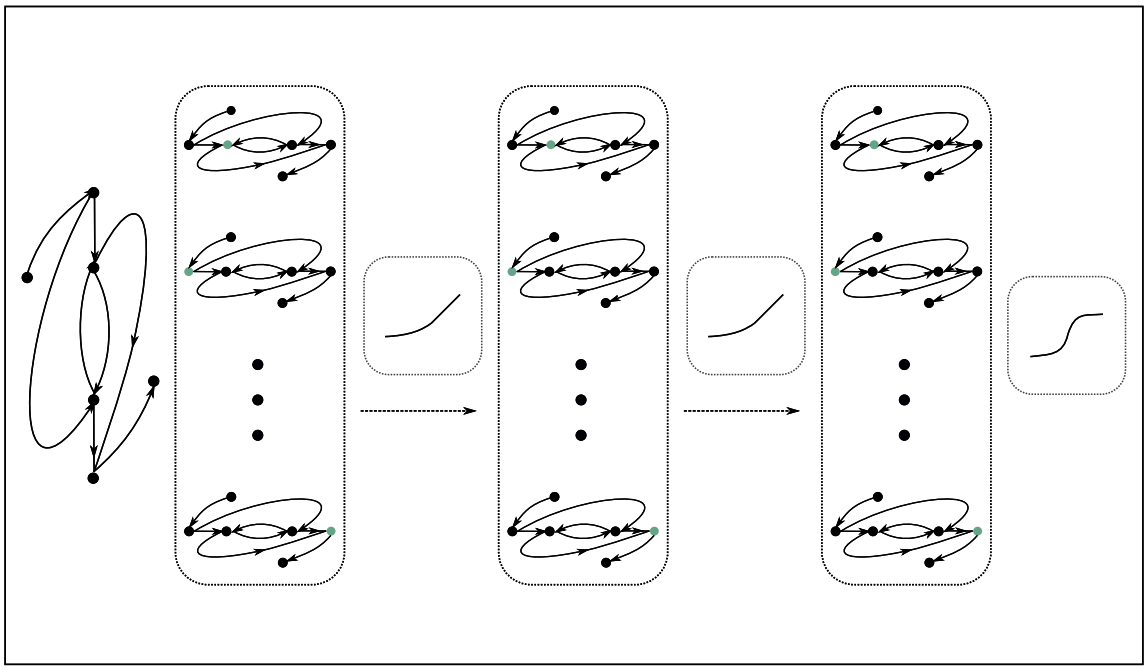

Figure 2.8 Node level graph neural network model framework

and activation. The function f in the figure is a decoder function that takes encoded matrix Z as input source and decode it into an adjacency matrix which defines the connections in the new graph.

Applications of GNN and its variants have covered a great amount of research areas, such as social network analysis, natural language processing, graph mining, physical system modeling, chemical system modeling, new material design and biomedical engineering[76, 77, 11, 78]. Especially in computational biology research, GNN has shown its advantage over other machine learning models at solving complex problems related to biology structures, such as protein-protein interaction prediction and gene function prediction. The variants of GNN have Graph Convolutional Network (GCN)[79], Graph Recurrent Network (GRN)[80], Graph Attention Network (GAT)[81], Adaptive Graph Convolution Network (AGCN)[82], Dual Graph Convolutional Network (DGCN)[83], Diffusion Convolutional Neural Network (DCNN)[84], Message Passing Neural Network (MPNN)[85] and Gated Graph Neural Network

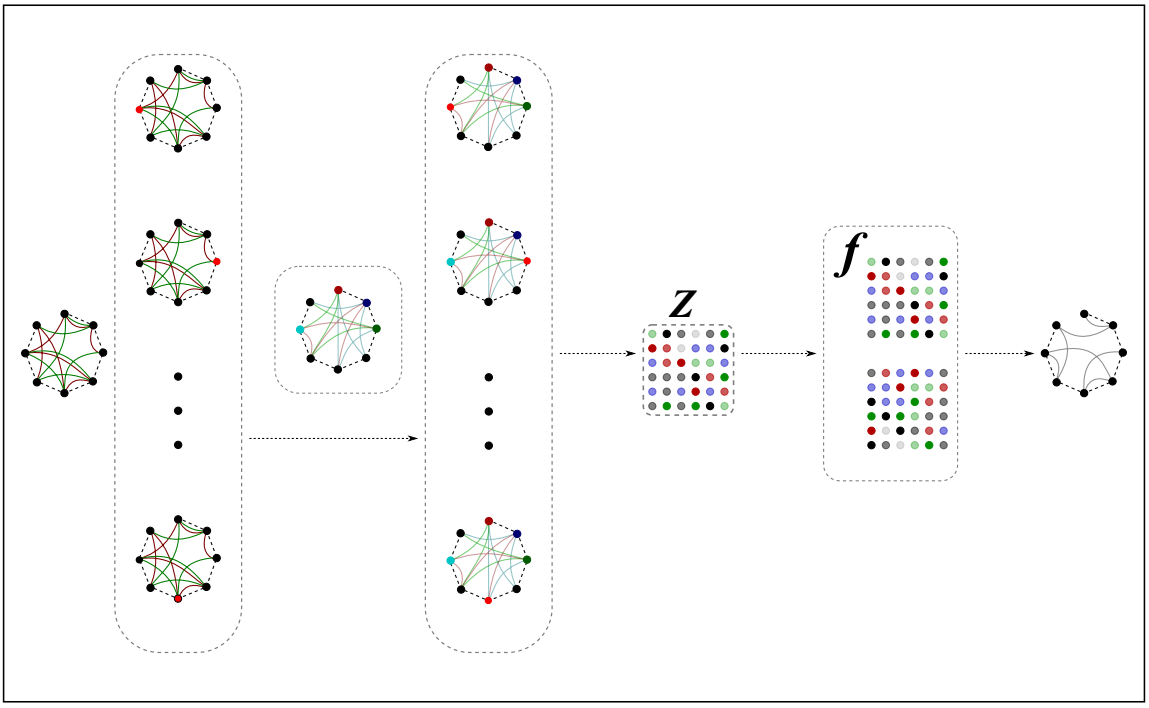

Figure 2.9 Graph level graph neural network model framework

(GGNN)[86]. All of these algorithms have shown their efficiency at specific application[10].

Chapter 3

Predict Subcellular Location Based on Genome Sequence Information

3.1 Genome Sequence Encoding

Genome sequences consist of a large number of genes and these genes perform their functions after they are translated into protein complex. So genome sequences could be transformed into protein sequences to evaluate their functions or identify their locations in a cell. DNA or mRNA coden, which consists of three nucleotides, specifies an amino acid. Since there are twenty types of amino acids defined by the genetic code rules, the unit representation form of a genome sequence are more plentiful than using nucleotides. In this research, the defined subcellular localization model will take amino acid sequences which are translated from genome sequences as its input and predict these sequences' position in a cell.

Genome sequences or protein sequences need to be encoded into a vector or matrix of numbers to be recognized by computational methods. The selection of encoding methods has great effect to the results of computational methods. Some methods takes raw genome sequences as input while some methods add feature extraction algorithms into itself to generate features as input [87]. The most commonly used encoding methods in computational biology research is one-hot-encoding. One-hot-encoding method encodes a sequence of length n into a $n \times m$ matrix with m as the encode dimension. Each unit in the sequence is converted into a m length vector and the vectors of any two different unit are orthogonal to each other.

Position Specific Scoring Matrix (PSSM) was introduced to measure the amount of consensus sequences of a large group of genome sequences or protein sequences[88, 89]. It consists of substitution probabilities of the amino acids in a protein sequence, which are computed out according to the positions of these amino acids in the sequence after many times of BLAST iteration [90]. For a protein sequence that has n amino acids in it, the PSSM of this protein sequence is like

$$PSSM = \begin{bmatrix} s_0^0 & s_0^1 & s_0^2 & \cdots\cdots & s_0^{20} \\ s_1^0 & s_1^1 & s_1^2 & \cdots\cdots & s_1^{20} \\ & & & \ddots & \\ s_n^0 & s_n^1 & s_n^2 & \cdots\cdots & s_n^{20} \end{bmatrix}$$

in which s_i^j is the logarithm of substitutions occurred at position i. Higher s_i^j indicates the amino acid at position i in this sequence has frequently substituted by amino acid j among a large amount of protein sequence statistics. The PSSM generating tool, Position Specific Iterated BLAST (PSI-BLAST), is publicly available on the National Institutes of Health website [91].

3.2 Prediction Model Framework

The architecture of model in this research is based on Bidirectional Long Short-Term Memory (LSTM) Network and CNN. Bidirectional LSTM Network encodes the sequences further to include position information into the representation vectors and CNN extracts features from the representation vectors. Sigmoid function transforms the extracted features into probabilities of each subcellular location.

3.2.1 Network Layers

The LSTM network is good at processing time series data and could cover long-term dependency in a sequence by combining previous states with current states[92]. Bidirectional LSTM (BLSTM) consists two LSTM networks. One of them processes a

sequence from beginning to end and another one processes this sequence from end to beginning[93]. In the past several years, Bidirectional LSTM had been successfully applied in natural language processing, temporal data processing and protein sequence processing[94].

With a amino acid sequence, the LSTM network could figure out the dependency relations between sites that are not adjacent with each other in this sequence. It will output a state matrix which includes position information and relation information of amino acid sites in the sequence. Because the complement sequence has reverse direction with its couple sequence, so Bidirectional LSTM network is applied in this subcellular localization model to processes amino acid sequences from two directions, which could include comprehensive states in the encoded matrix.

Multiple layers of CNN discover under-level patterns in data samples by using a lot of optimizable parameters. It had been widely utilized in many research areas, such as image processing, nature language processing, financial data analysis, biology analysis and so on[47, 48, 49, 50]. Different from traditional machine learning algorithms, CNN could extract features from raw data automatically. During training stages, convolution layers apply a bunch of kernel matrix ω on data volume to filter out significant features from raw input. During backpropagation stages, the parameters in the kernel matrix will be optimized by a gradient amount.

Activation layers in neural network transforms data in non-linear form. The active operation brought in by these layers simulate the actions happened on biological neurons, which only could be aroused when the stimulus signals are strong enough[95, 96]. For example, the rectified linear activation function (ReLU)

$$R(x) = max(0, x)$$

only pass strong signals or positive values[97]. It makes a non-linear transform to the

internal data of neural network. The Sigmoid function

$$S(x) = \frac{e^x}{1 + e^x}$$

converts x which could be any number to a value between 0 and 1. Because the special properties of Sigmoid function, the output values of it always be used as a representation of probability[98].

3.2.2 Model Architecture

The neural network model constructed in this research consists of one Bidirectional LSTM network layer, two CNN layers, several activation layers and a concatenate operation. Figure 3.1 shows the architecture of this model.

At the beginning of this model is a pre-processing layer which converts amino acids sequences into an one-hot-encoding matrix and a position-specific scoring matrix. The two converted matrices are passed to two separate neural network pipelines.

The first one takes the one-hot-encoding matrix as input and process this matrix with a Bidirectional LSTM encoder and a CNN sequentially. This Bidirectional LSTM encoder has two LSTM network which encode the one-hot-encoding matrix in parallel and convert it into a 256 values vector. It is followed by a convolutional neural network in this pipeline. This network is a stack of seven neural layers which include four convolution layers and three maxpooling layers. The numbers of each convolution layer kernels are 256, 128, 64, 32 and the kernel size of each layer is 4×3, 3×3, 3×3 and 3×3 correspondingly. These maxpooling layers filter out significant features with 2×2 window size between every two layers in the previous neural network.

The second pipeline takes the PSSM as data source and consists only one CNN. Similar to the neural network in the previous pipeline, this neural network also has four convolution layers and three maxpooling layers. The kernel size and maxpooling window size are same as the previous ones. The output feature vectors from these

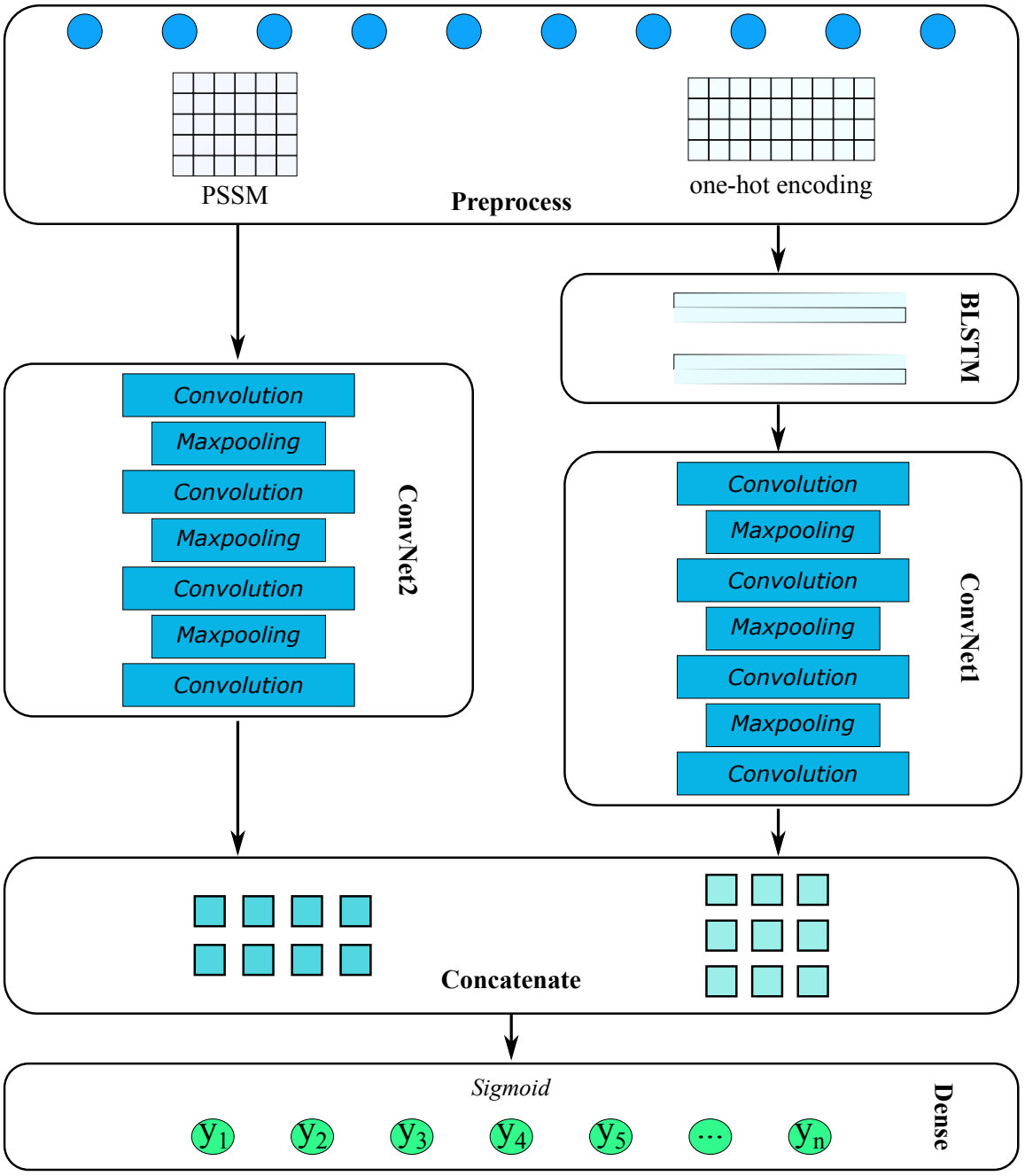

Figure 3.1 The framework of subcellular localization model

two pipelines are concatenated together and flattened into a one dimensional vector. Then features in this vector are condensed together to a small vector which has the same size as target labels. At last Sigmoid function generates the corresponding

probability that the input sequence exists at each location. According to the output probabilities, the subcellular locations of a amino acid sequences could be identified with a suitable threshold.

Every neural network in these two pipelines can work alone to predict a sequence's subcellular location. The Bidirectional LSTM network could encode a sequence and generate its location probabilities. The convolutional neural network in the second pipeline is able to extract features from position-specific score matrix and make predictions. In the following sections, the Bidirectional LSTM network will be named as BLSTM, the network in the first pipeline will be named as ConvNet1 and the network in the second pipeline will be named as ConvNet2. So the combination of two or more networks will use their names with plus symbol as its name. For example the two networks in the first pipeline will be BLSTM + ConvNet1 and the combination of all three networks will be BLSTM + ConvNet1 + ConvNet2.

3.2.3 Model Implementation

In this research, the subcellular localization model was implemented by using Keras package. It's a high-level neural network API developed based on tensorflow libraries in purpose of enabling fast experimentation [99, 100]. With well defined network layers and user friendly API, this research's network model could be easily implemented and the code of this model is available online [1].

3.3 Model Regularization

As shown in previous section, the Sigmoid function at the end of subcellular localization model transform the dense features into a vector probabilities. In order to quantify the difference between prediction and real situation, binary cross entropy loss

[1] https://github.com/Paeans/subcellular

function is utilized in the model training process to compute the loss value between the model prediction and the real sequence locations[55].

Model over-fitting could be happen after a bunch of parameter optimizations, which reduces model's ability to make accurate prediction on samples out of the training dataset. A commonly used strategy to prevent model over-fitting is that imposing a $L2$ regularization on model's parameter scale. The $L2$ regularization is appended to the loss function in the form of a L^2 norm. The loss function with $L2$ regularization applied will be

$$L = -\frac{1}{n}\sum_{i=1}^{n}(y_i \times log\hat{y}_i + (1-\hat{y}_i) \times log(1-\hat{y}_i)) + \lambda||\omega||_2^2$$

in which the first half is the binary cross entropy loss between prediction $\hat{y}_i$ and true situation y_i and the second half is the $L2$ penalty on model's parameters ω. The coefficient λ could adjust the weight of parameter penalty.

3.4 Model Evaluation

3.4.1 Benchmark Datasets

There are two benchmark datasets used to evaluate the performance of computational subcellular localization methods, dataset $D3106$ and dataset $D4802$. The benchmark dataset $D3106$ contains 14 subcellular types and 3106 protein sequences [101]. The dataset $D4802$ have 4802 protein sequences in it, which are located at 33 different subcellulars.[35]. These two benchmark datasets were widely used by subcellular localization research to do method evaluation.

Table 3.1 lists the subcellular types and number of sequences located at each subcellular location in dataset $D3106$ and dataset $D4802$. Some subcellular types are not covered by dataset $D3106$ and the corresponding values in Table 3.1 are left as blank. Since one sequence could be located at two or more positions in a cell, the protein sequences in dataset $D3106$ covers 3681 positions and the sequences in dataset $D4802$

covers 6198 positions. The ratio of position number to subcellular number shows that dataset $D4802$ has more multi-subcellular sequences than dataset $D3106$. Figure 3.2 is the histogram of multi-subcellular sequences of dataset $D3106$ and dataset $D4802$, which shows that dataset $D4802$ have a lot of two-subcellular sequences but with a small number of sequences that located at three or more subcellular locations.

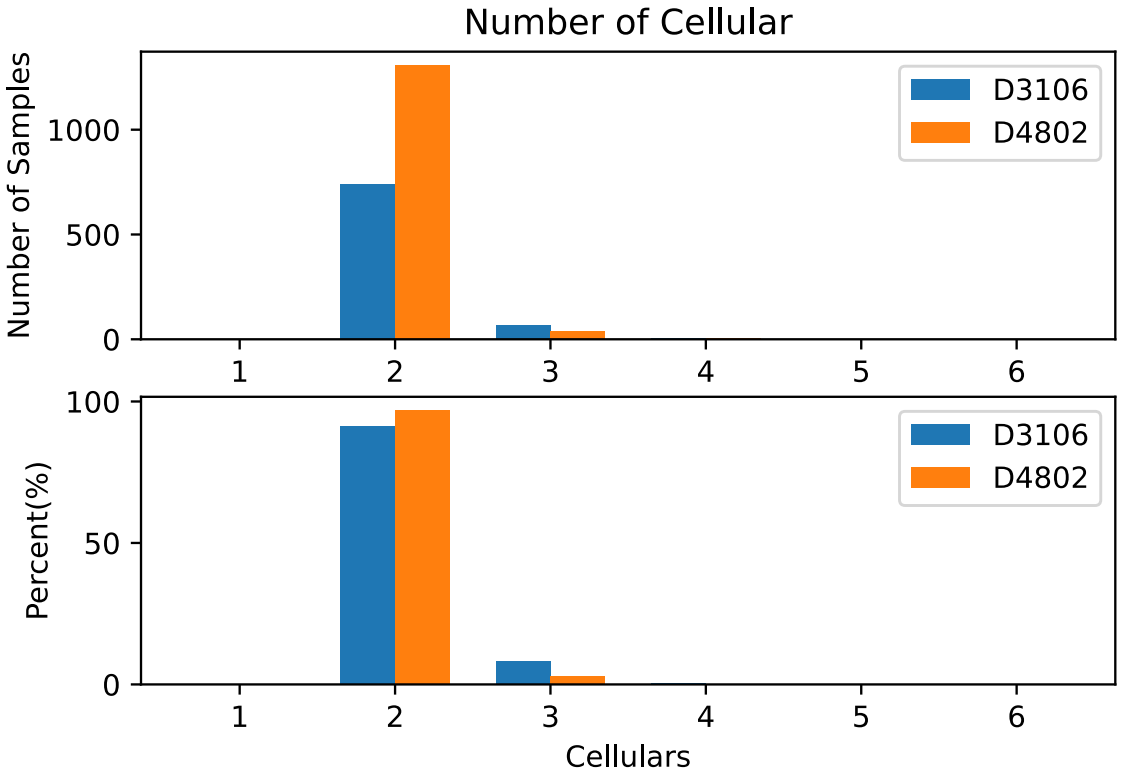

Figure 3.2 Histogram of multi-subcellular sequence numbers

3.4.2 Performance Evaluation

The prediction result of a sequence is a vector of probabilities of this sequence exists at a group of subcellular locations. The accuracy of the prediction results could be evaluated as a single-labeled prediction results or a multi-labeled prediction results. The evaluation metrics F1 score (F_1), Matthews Correlation Coefficient (MCC) and Area under ROC curve (AUC) are utilized to evaluate the accuracy of single subcellular

Table 3.1 Number of sequences in benchmark datasets $D3106$ and $D4802$

Subcellular Location	D3106	D4802	Subcellular Location	D3106	D4802
Apical Plasma		16	Golgi Apparatus	161	272
Basolateral Plasma		29	Golgi Cisterna		7
Inner Mitochondrial		4	Golgi Trans Cisterna		3
Plasma Membrane	354	836	Golgi Trans Face		11
Cellular Component		4	Golgi Medial		7
Extra Cellular	385	487	Nucleus	1021	1720
Centrosome	77	81	Nucleolus		268
Lysosomes	77	125	Nuclear Envelope		47
Melanosome		6	Synaptic Vesicles	22	28
Peroxisome	47	67	Secretory Vesicles		5
Cytoplasm	817	1050	Transport Vesicles		4
Cytoplas Vesicles		46	Tight junction		9
Early Endosomes		52	Cytoskeleton	79	89
Endosome	24	342	Endoplasmic Reticulum	229	120
Late Endosomes		16	ERGIC		4
Secretory Granule		10	Microtubule	24	26
Mitochondria	364	407			

[1] total number of sequences in $D3106$ is 3681

[2] total number of sequences in $D4802$ is 6198

location prediction and the metrics Ranking Loss (RL), Coverage (Cov) and Average Precision (AP) are applied from the aspect of sample level prediction accuracy.

The F1 score computes the harmonic mean of precision PPV and recall TPR.

The definition of F1 score is

$$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR}$$

with

$$PPV = \frac{tp}{tp + fp}, \;\; TPR = \frac{tp}{tp + fn}$$

in which tp, fp and fn are true positive, true negative and false negative respectively. For each subcellular location, the tested protein sequences have corresponding probability values and the F1 score for this subcellular location could be computed. The overall F1 score of the prediction model will be the weighted average of the F1 scores generated for each subcellular location, which reduce the effects of unbalanced class labels[102]. The MCC metric is also a balanced average of all subcellular labels [103]. The MCC of a single subcellular location is defined as

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}},$$

tn in this function is true negative of prediction. Its value ranges from -1 to 1 and higher value means the prediction results are more similar to the real results. The AUC score is the area under the ROC curve which is drawn over true positive rate and false positive rate with a bunch of classification thresholds. It evaluates the ability to separate true instances and false instances of a classification model[104].

Ranking Loss, Coverage and Average Precision evaluates whether the subcellular localization model finds out all the locations a sequence could be at and whether the predict probabilities of positive locations are higher than the predict probabilities of negative locations [105, 106]. Assuming the true location labels of n protein sequences is Y which is a $n \times m$ binary matrix and the prediction location probabilities of these sequences is $\hat{Y}$ has the same size as Y. The Ranking Loss between Y and $\hat{Y}$ is

$$RL = \frac{1}{n} \sum_{n}^{i=1} \frac{|S|}{\sum Y_i \sum (1 - Y_i)}, \left(S = \left\{ (k, l) : Y_{ik} = 1, Y_{il} = 0, \hat{Y}_{ik} \leq \hat{Y}_{il} \right\} \right).$$

It counts the ratio of inverse pairs to all possible pairs in the prediction results. The Coverage and Average Precision are defined as

$$Cov = \frac{1}{n}\sum_{i=1}^{n} \max_{\{j:Y_{ij}=1\}} \left|\left\{k : \hat{Y}_{ik} \geq \hat{Y}_{ij}\right\}\right|$$

and

$$AP = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\sum Y_i} \sum_{Y_{ij}=1} \frac{\left|\left\{k : Y_{ik} = 1, \hat{Y}_{ik} \geq \hat{Y}_{ij}\right\}\right|}{\left|k : \hat{Y}_{ik} \geq \hat{Y}_{ij}\right|}.$$

Similar to Ranking loss, Coverage considers the lowest value of positive positions among all the prediction values and Average Precision evaluates how many negative positions are predicted as positive positions of a sequence.

3.5 Experiment Results and Discussion

The four subcellular localization neural network models constructed in Section 3.2 were tested on two benchmark datasets $D3106$ and $D4802$. The evaluation experiments were deployed from the aspect of single-subcellular location prediction and multi-subcellular locations prediction. The experiment results show that the subcellular localization model utilizing evolution information and deep neural network has good performance and could generate more accurate prediction results.

3.5.1 Performance on Single Subcellular Site

Table 3.2 lists out the F_1 score, MCC and AUC of prediction results by four models BLSTM, BLSTM + ConvNet1, ConvNet2 and BLSTM + ConvNet1 + ConvNet2 on benchmark dataset $D3106$. The values in this table is the average value of 14 locations prediction. The model ConvNet2 has the worst performance among these four models while other three models have similar prediction accuracy. The best F_1 score is 0.7843 and the best AUC is 0.9458, both of which are achieved by the model BLSTM + ConvNet1 + ConvNet2. However the MCC value of this model isn't the highest, it is a little bit lower, 0.0009, than the MCC value of BLSTM + ConvNet1.

Table 3.2 Single subcellular prediction performance of four models on benchmark dataset $D3106$

	F_1	MCC	AUC
BLSTM	0.7473	0.6001	0.8841
BLSTM + ConvNet1	0.7775	0.6419	0.9064
ConvNet2	0.6475	0.4819	0.7685
BLSTM + ConvNet1 + ConvNet2	0.7843	0.6410	0.9046

[1] values are the mean accuracy of 14 sites prediction

[2] accuracy results are from five-fold cross validation

Figure 3.3 plots the ROC curves of 14 subcellular sites prediction of models BLSTM, BLSTM + ConvNet1, ConvNet2 and BLSTM + ConvNet1 + ConvNet2 on dataset $D3106$. The highest AUC value achieved by model BLSTM is 0.9242 and the average AUC value is 0.8733. Model ConvNet2 is worse than other three models and the best AUC value of this model is only 0.8785. For average AUC values, the average AUC of model ConvNet2, 0.7641, is less than the average AUC values of models BLSTM + ConvNet1 and BLSTM + ConvNet1 + ConvNet2 whose AUC values are 0.9010

The average accuracy evaluations on benchmark dataset $D4802$ of these four models are shown in table 3.3. Same as the evaluations performed on dataset $D3102$, this experiment also takes five-fold cross validation procedure as evaluation strategy and records the average values of the F_1 score, MCC, AUC values of 33 single subcellular site predictions. According to the average F_1 score, MCC and AUC, model BLSTM + ConvNet1 + ConvNet2 has the best performance on dataset $D4802$. Model ConvNet1 has great contribution to make accurate predictions since ConvNet1 in model BLSTM + ConvNet1 increases the F_1 score by 0.04, MCC value by 0.058 and AUC by 0.02. Figure 3.4 is the plots of ROC curves of these four models on 33 subcellular

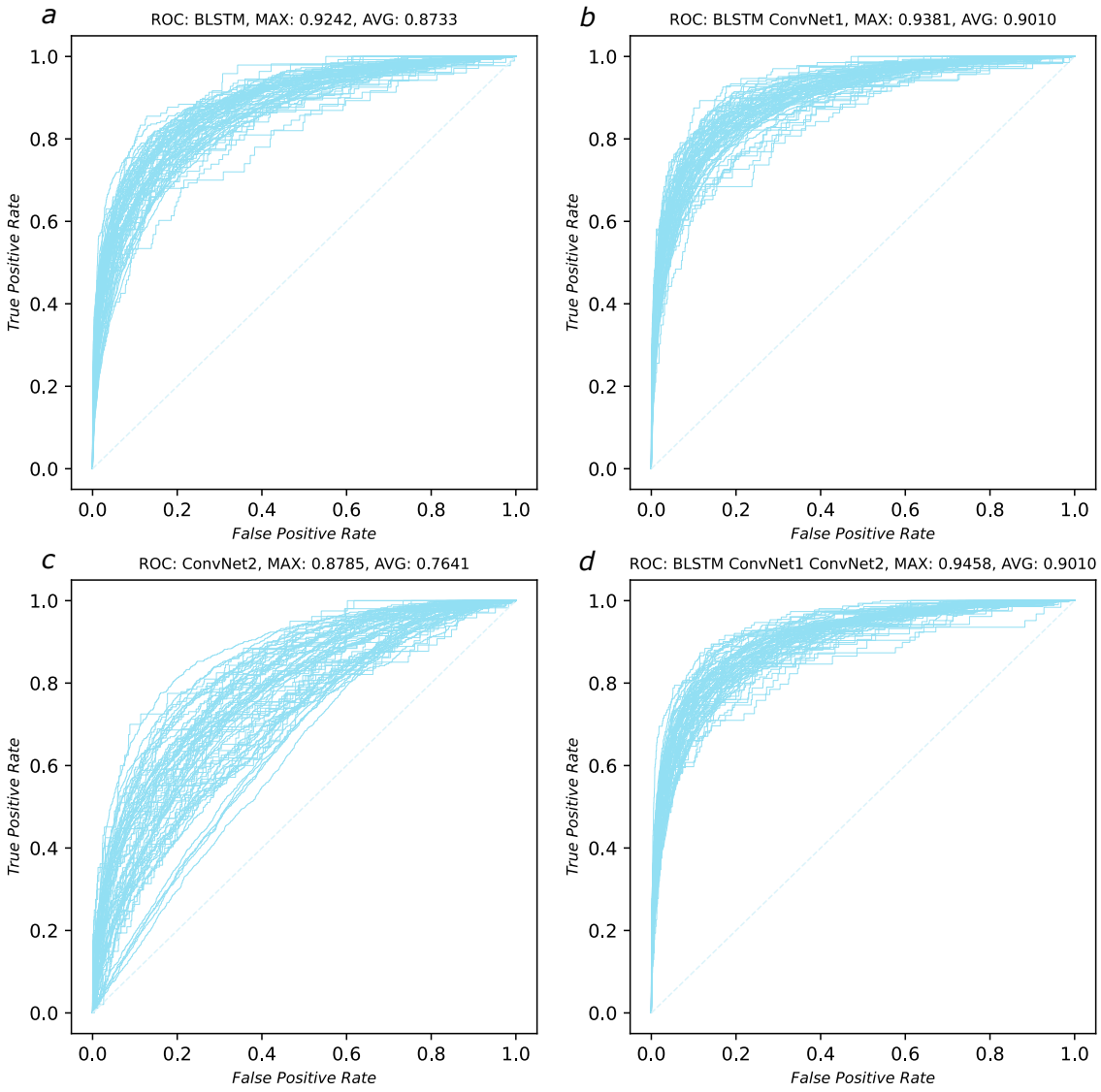

Figure 3.3 ROC plot of four models on benchmark dataset $D3106$

sites prediction. This figure shows that model ConvNet2 has worse performance on dataset $D4802$, especially on the prediction of several subcellular locations as there are many curves that are under the plot figure diagonal. Model BLSTM + ConvNet1 is better than ConvNet2, however there are still two curves that are very close to the diagonal line. Model BLSTM + ConvNet1 + ConvNet2 has a small improvement by merging BLSTM, ConvNet1 and ConvNet2 together. The curve lines of this model

are more concentrated.

Table 3.3 Single subcellular prediction performance of four models on benchmark dataset $D4802$

	F_1	MCC	AUC
BLSTM	0.7419	0.5705	0.7696
BLSTM + ConvNet1	0.7801	0.6284	0.8543
ConvNet2	0.6696	0.4259	0.6806
BLSTM + ConvNet1 + ConvNet2	0.7842	0.6411	0.8594

[1] values are the mean accuracy of 33 sites prediction

[2] models are evaluated with five-fold cross validation

Model BLSTM + ConvNet1 + ConvNet2 has similar performance when tested on dataset $D3106$ and dataset $D4802$. The F_1 scores of this model on these two datasets are very close with a difference of 0.001. The MCC values have the same result. However, the AUC values of this model on dataset $D4802$ is much lower than the value of this model on dataset $D3106$. In the dataset $D4802$, there are several subcellular sites that model BLSTM + ConvNet1 + ConvNet2 has poor performance. The AUC values of this model on these sites are close to 0.5 which significantly reduced the average AUC result. From the plots of ROC curves in figure 3.3 and figure 3.4, it is obvious that the curves on dataset $D3106$ are more concentrate than the curves on dataset $D4802$. This result shows that model BLSTM + ConvNet1 + ConvNet2 is more suitable to make single subcellular location prediction on dataset which has high correlation between each subcellular sites. Dataset $D4802$ has 33 subcellular sites, some of these sites don't have close relation with other sites, which introduce noise data to evolution features and decrease the convolutional feature quality.

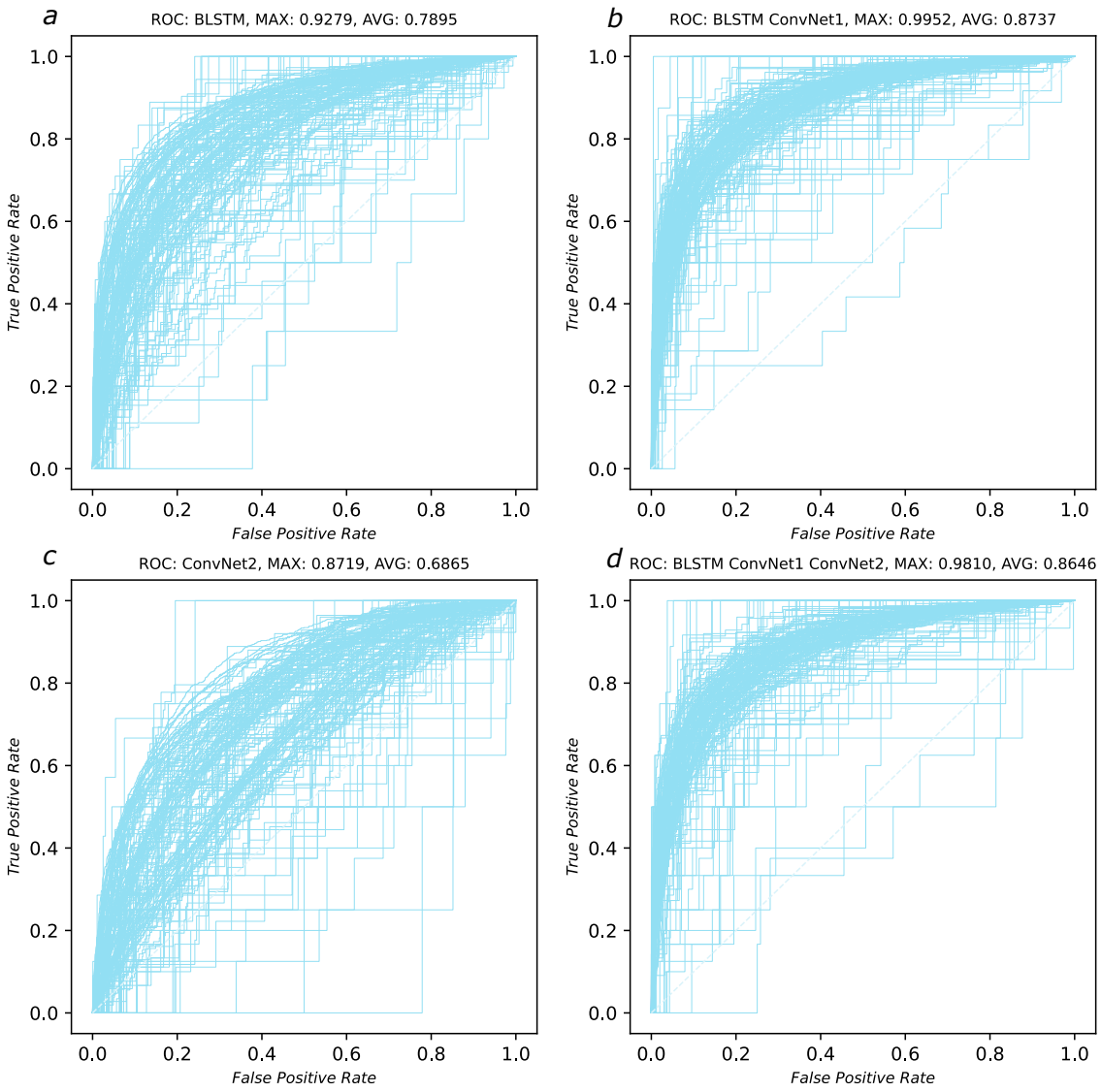

Figure 3.4 ROC plot of four models on benchmark dataset $D4802$

3.5.2 Performance on Multiple Subcellular Sites

Subcellular localization models output multiple possible locations of a sequence in a cell and the prediction ability could be evaluated as multi-label classifications. In this research, the models defined in Section 3.2 were tested on benchmark datasets $D3106$ and $D4802$ and the ranking loss, coverage, average precision values of their prediction results were evaluated. When a model has high average precision, low

ranking loss and low coverage, it is able to predict the subcellular locations of a sequence with accurate results. Table 3.4 is the evaluation result of models BLSTM, BLSTM + ConvNet1, ConvNet2, BLSTM + ConvNet1 + ConvNet2 when they are trained and tested with benchmark dataset $D3106$. The ranking loss of model BLSTM + ConvNet1 + ConvNet1 is 0.0758 which is the best among all these four models. It is very close to the ranking loss of BLSTM + ConvNet1, which is 0.002 higher. Coverage and average precision have similar results, in which model BLSTM + ConvNet1 + ConvNet2 has the best performance on multiple subcellular sites prediction and model BLSTM + ConvNet1 is a little worse than it with a difference less than 0.1. The t-test statistic of ranking loss, coverage and average precision shows that the difference of ranking loss and coverage between model BLSTM + ConvNet1 and model BLSTM + ConvNet1 + ConvNet2 is significant and the difference of average precision between these two models is not significant with $p_value > 0.5$. This result shows that model ConvNet2 improves the model's ability in multiple subcellular sites prediction when it is appended to model BLSTM + ConvNet1, however the effect isn't very large.

Table 3.4 Multi-subcellular localization evaluation results of four models on benchmark dataset $D3106$

	RL	Cov	AP
BLSTM	0.0967	1.5895	0.7523
BLSTM + ConvNet1	0.0778	1.3113	0.7876
ConvNet2	0.1294	2.0113	0.6430
BLSTM + ConvNet1 + ConvNet2	0.0758	1.2848	0.7901

[1] values in this table is the average of 5-fold cross validation

[2] low ranking loss, coverage means accurate prediction

Table 3.5 is the evaluation results of four models, BLSTM, BLSTM + ConvNet1, ConvNet2 and BLSTM + ConvNet1 + ConvNet2, on benchmark dataset $D4802$.

The evaluation process was took out with five-fold cross validation strategy and the values in this table is the average value of five fold evaluation results. On benchmark dataset $D4802$, model BLSTM + ConvNet1 outperforms other three models. The ranking loss of this model is 0.0603 and the coverage is 2.9225, which are the lowest ones among these four models. Its average precision, 0.7453, is the highest and is 0.12 higher than the average precision of model ConvNet2. From the aspect of ranking loss and coverage, model BLSTM has the worst performance on subcellular location prediction, however model ConvNet2 makes the worst prediction when evaluated from the aspect of average precision. The t-test statistic of these results shows that the difference between each model is significant and model ConvNet1 is not good at dataset $D4802$.

Table 3.5 Multi-subcellular localization evaluation results of four models on benchmark dataset $D4802$

	RL	Cov	AP
BLSTM	0.0820	3.3916	0.6901
BLSTM + ConvNet1	0.0603	2.9225	0.7453
ConvNet2	0.0673	3.2868	0.6214
BLSTM + ConvNet1 + ConvNet2	0.0637	3.0528	0.7414

[1] values in this table is the average of 5-fold cross validation

[2] low ranking loss, coverage means accurate prediction

The performance of these four models on benchmark dataset $D3106$ and $D4802$ has great differences. Model BLSTM + ConvNet1 + ConvNet2 has best performance on dataset $D3106$ while model BLSTM + ConvNet1 has best performance on dataset $D4802$. The best ranking loss achieved on dataset $D4802$ is lower than the one achieved on dataset $D3106$, however the coverage and average precision on dataset $D3106$ are better than the corresponding ones on dataset $D4802$. When considering

model ConvNet2 alone, it could learn and classify dataset $D3106$ more accurately. The coverage of this model on $D3106$ is much better than the coverage of it on $D4802$ and the average precision has the same conclusion. These results are consistent with the results of single subcellular site prediction. Model ConvNet2 introduces noise data to the evolution features and reduces prediction accuracy when the correlation between each subcellular site is low in a dataset.

3.5.3 Compare with Other Subcellular Localization Methods

The model BLSTM + ConvNet1 + ConvNet2 was compared with five currently available methods as a multiple subcellular sites prediction model. The evaluations was deployed on dataset $D3106$ and $D4802$ with five-fold cross validation strategy. Table 3.6 and table 3.7 list the evaluation result of model BLSTM + ConvNet1 + ConvNet2 and other five methods, IMMMLGP, Hum-mPloc, mGOF-loc, MKSVM and FSVM-KNR. Table 3.6 is the ranking loss, coverage and average precision when they are trained and tested on dataset $D3106$. The method mGOF-loc doesn't have evaluation values on dataset $D3106$. Model BLSTM + ConvNet1 + ConvNet2 has the best performance on this dataset and it has great improvements than the other four methods. It increases the average precision by 0.08 from 0.7108 of method FSVM-KNR and decreases ranking loss and coverage by 0.032 and 0.42 correspondingly. The t-test statistic on these values shows that the improvements on ranking loss, coverage and average precision by model BLSTM + ConvNet1 + ConvNet2 are significant, that this new model have better performance than other four methods on prediction of dataset $D3106$.

The evaluation results on dataset $D4821$ of model BLSTM + ConvNet1 + ConvNet2 and other five methods, IMMMLGP, Hum-mPloc, mGOF-loc, MKSVM and FSVM-KNR, are recorded in table 3.7. Model BLSTM + ConvNet1 + ConvNet2 doesn't perform well on dataset $D4812$. It has the highest average precision and its

Table 3.6 Multi-subcellular localization evaluation results of proposed model and four currently available methods on benchmark dataset 3106

	RL	Cov	AP
IMMMLGP	0.4190	4.3030	0.5810
Hum-mPloc	0.4906	5.3170	0.5790
MKSVM	0.1085	1.7193	0.7065
FSVM-KNR	0.1071	1.7025	0.7108
BLSTM + ConvNet1 + ConvNet2	0.0758	1.2848	0.7901

value is 0.05 higher than the value of method FSVM-KNR, which is a great improvement when compared with previous methods. However, the ranking loss and coverage of this model isn't the best among these six methods. Its ranking loss is 0.003 greater than the ranking loss of method mGOF-loc which archieves 0.0606 on this evaluation metric and its coverage value, 3.0528 is 0.42 greater than method FSVM-KNR's coverage result. From the aspect of average precision, model BLSTM + ConvNet1 + ConvNet2 has better performance than other five methods, but it doesn't show advantages when evaluated from the aspect of ranking loss and coverage.

Previous discussion shows that model BLSTM + ConvNet1 has better performance than the model with an extra ConvNet2 on dataset $D4802$. The ranking loss of model BLSTM + ConvNet1 on dataset $D4802$ is 0.0603, which is a little better, with only about 0.0003 distance, than ranking loss of method mGOF-loc. The t-test statistics on ranking loss values of this model and method mGOF-loc shows that the difference between these two methods is not significant with a $p_{value} > 0.1$, so model BLSTM + ConvNet1 + ConvNet2 performs as better as method mGOF-loc on dataset $D4812$ when they are evaluated with ranking loss. The coverage of this model is still greater than the value of method FSVM-KNR and the t-test statistic of

these two variables shows the difference is significant, which means it is not as good as method FSVM-KNR on multiple subcellular sites prediction from the aspect of coverage.

Table 3.7 Multi-subcellular localization evaluation results of proposed model and five currently available methods on benchmark dataset 4802

	RL	Cov	AP
IMMMLGP	0.2436	4.9772	0.5725
Hum-mPloc	0.3145	5.6830	0.5644
mGOF-loc	0.0606	3.0227	0.6482
MKSVM	0.0662	2.9753	0.6889
FSVM-KNR	0.0971	2.6339	0.6916
BLSTM + ConvNet1 + ConvNet2	0.0637	3.0528	0.7414

Chapter 4

Solve the Median of Three Problem by Variational Graph Auto-Encoder

4.1 Dataset Generation

The supervised learning model requires dataset which includes data samples with input features and target labels to learn data feature patterns and train the model parameters. In order to train a supervised learning model to learn the median genomes of a group of genomes, a dataset contains groups of three genomes and their corresponding median genomes is constructed according to the median problem rules discussed in previous section.

The dataset is constructed by repeatedly filter out available genome groups and median genomes after a number of random rearrangement events. With the gene set $\mathcal{S}_g = \{g_1, g_2, \cdots, g_n\}$, randomly append $+$ or $-$ to each gene and randomly arrange these n signed genes in a sequence to form a genome G^0. Apply a sequence of random rearrangement events to the genome G^0 to generate a group of genomes $\mathcal{G}^1 = \{G_1^1, G_1^2, \cdots, G_1^m\}$ in which G_1^i is the genome generated by applying rearrangement event on genome G_1^{i-1} at step i. Add the elements in $\mathcal{G}^1$ to genome set $\mathcal{G}$. Repeat these process on G for t times to construct genome set $\mathcal{G}$ as

$$\mathcal{G} = \begin{bmatrix} G_1^1 & G_1^2 & \cdots & \cdots & G_1^m \\ G_2^1 & G_2^2 & \cdots & \cdots & G_2^m \\ \vdots & & \ddots & & \\ G_t^1 & G_t^2 & \cdots & \cdots & G_t^m \end{bmatrix}.$$

Choose three genomes G_i^x, G_j^y and G_k^z from $\mathcal{G}$ as $\mathcal{G}_s^0$, if it satisfies that $i \neq j \neq k$ and $D(G^0|\mathcal{G}_s^0) = \lceil \sum \frac{\Delta}{2} \rceil$, add $(\mathcal{G}_s^0, G^0)$ to training dataset. Repeat the whole process to construct $(\mathcal{G}_s^1, G^1)$, $(\mathcal{G}_s^2, G^2)$ and so on.

The adjacency format of genome represents the order of genes and the directions of genes in the genome. With adjacency format, each gene is represented by two numbers, one for head and another for tail. In the adjacency sequence of a genome, two numbers are adjacent to each other only when these two numbers represent the same gene or their corresponding gene end points adjacent to each other in the genome. Assume a genome G, the number of genes is n and the gene set is $\mathcal{S}_g = \{g_1, g_2, \cdots, g_n\}$. Assign a unique number from 1 to n to each gene of $\mathcal{S}_g$ and translate the genome into a sequence of numbers using the corresponding number of each gene, then for each number i in this sequence, if $sign(i)$ is $+$ then replace it with two number $2|i| - 2$ and $2|i| - 1$, if $sign(i)$ is $-$ then replace it with $2|i| - 1$ and $2|i| - 2$. For example, genome $[+g_3, -g_n, \cdots, -g_1, \cdots, +g_4]$ is translated into signed number form as $[+3, -n, \cdots, -1, \cdots, +4]$, then replace each item with two numbers to get $[3, 4, 2n-1, 2n-2, \cdots, 1, 0, \cdots, 6, 5]$ which is the adjacency format of this genome.

For a data sample $(\mathcal{G}_s^i, G^i)$ with $\mathcal{G}_s^i = (G_a, G_b, G_c)$, transform G_a, G_b, G_c and G^i into adjacency form as A^a, A^b, A^c and A^t. The adjacency tuple set of A^a is

$$\mathcal{A}^a = \{(A_i^a, A_{i+1}^a) | i \in [0, |A^a| - 2]\}.$$

Combine the genome adjacency sequences $\{A^a, A^b, A^c\}$ into one graph $\mathbf{G}_s^i$,

$$\mathbf{G}_s^i = (\mathbf{V}_s, \mathbf{E}_s),$$

in which

$$\mathbf{V}_s = \{0, 1, 2, \cdots, 2n-1\}$$

and

$$\mathbf{E}_s = \{e_{ij} | (i, j) \in \mathcal{A}^a \cup \mathcal{A}^b \cup \mathcal{A}^c\}.$$

Transform adjacency sequence A^t into graph $\mathbf{G}_t^i = (\mathbf{V}_t, \mathbf{E}_t)$, $\mathbf{V}_t = \mathbf{V}_s$ and $\mathbf{E}_t = \{e_{ij}|(i,j) \in \mathcal{A}^t\}$. The three-genome-graph $\mathbf{G}_s^i$ is the input graph of the model and the graph $\mathbf{G}_\mathbf{t}^\mathbf{i}$ is the target graph of the model.

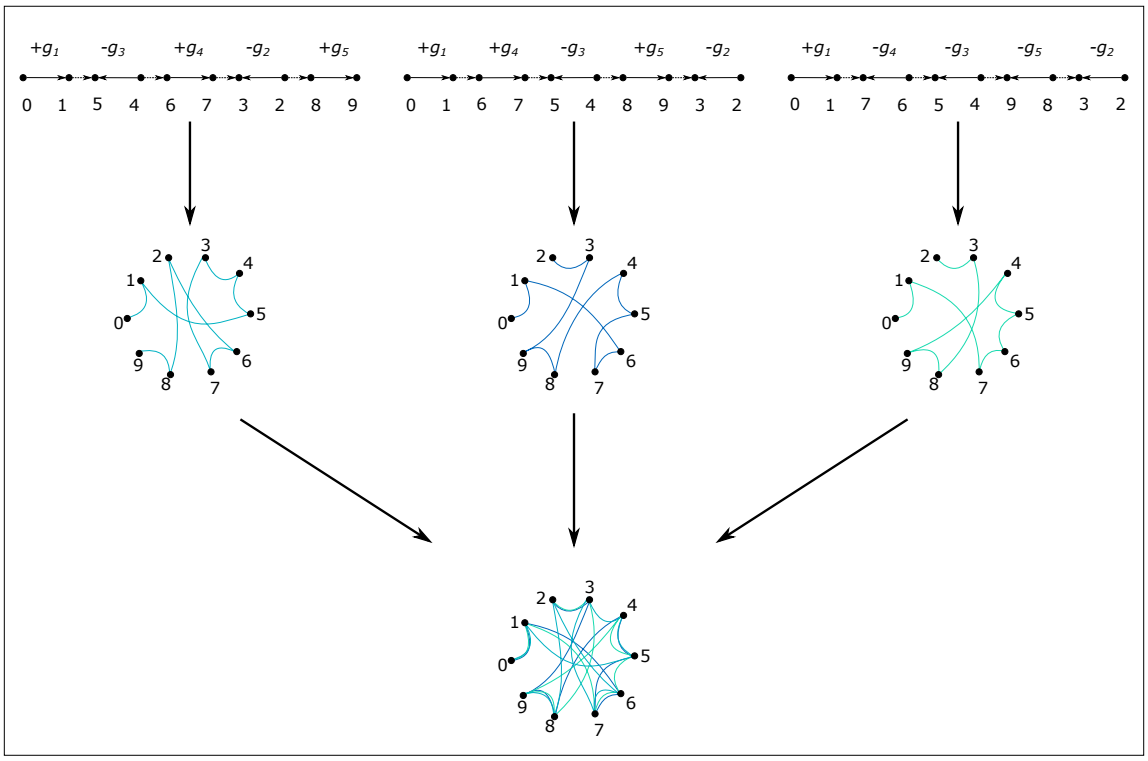

Figure 4.1 Demonstration of how to merge three genomes into one undirected graph

4.2 Graph Encoder and Decoder

4.2.1 Graph Neural Network Model

The graph neural network model to compute the median genome of three genomes contains a graph convolution network model and variational graph encoder model, as shown in figure 4.2. This model takes a three-genome-graph as input. The embedding layer at the top of it encodes the node index into a node feature vector which is fed to the graph convolution layers to generate a higher level node features based on the

edge connections. After 4 times of convolution along the graph edges, the output node features contain the patterns of relations between the nodes.

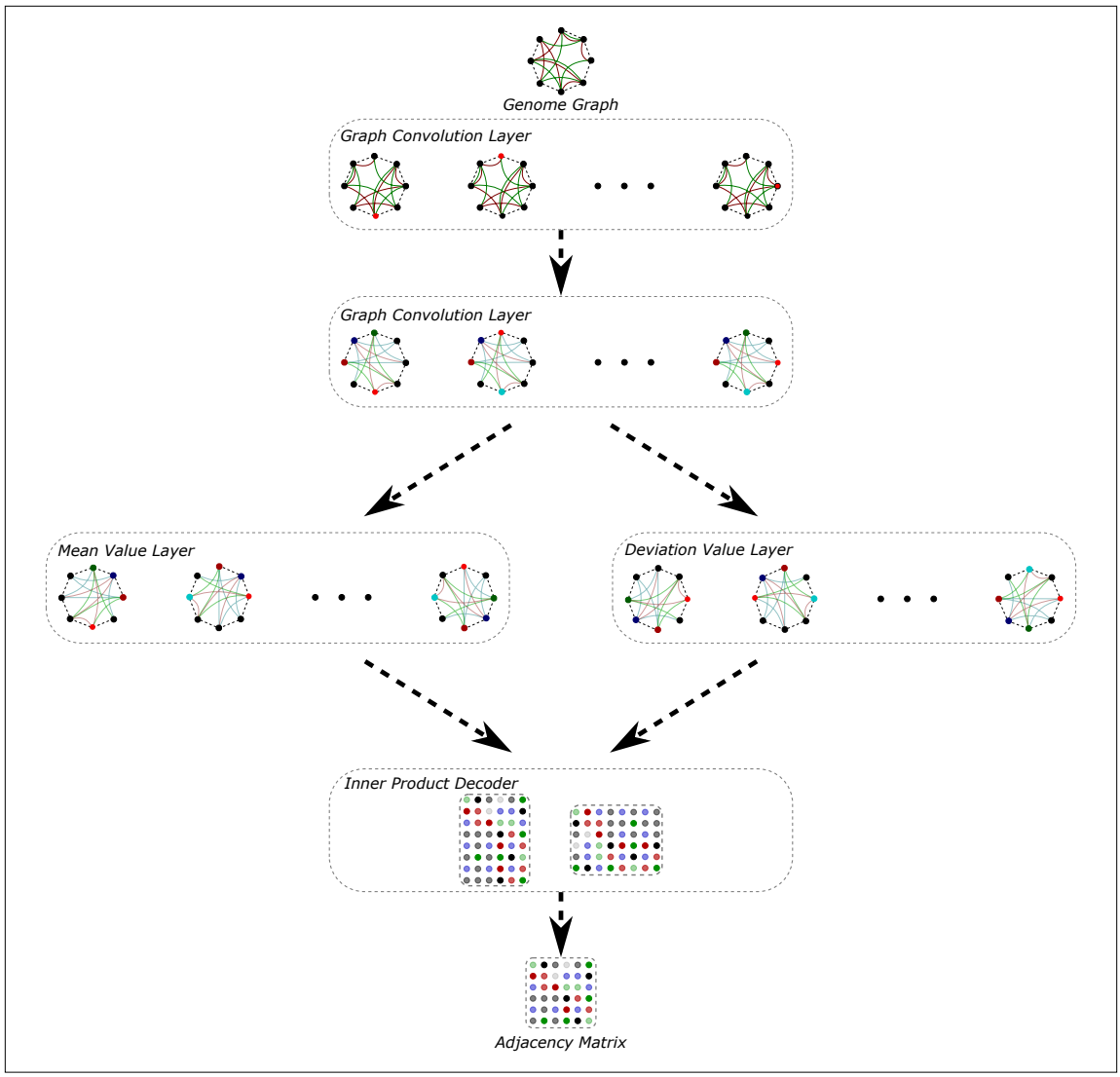

Figure 4.2 Framework of median genome generation model

The variational graph encoder model takes the node features as input and uses two separate graph convolution operations to compute the parameters of the node feature distribution. After these series of convolution operations, the three-genome-graph is encoded as a feature distribution matrix in which each row represents one node in the three-genome-graph. The relations between the graph nodes could be

computed as the multiply of the feature distributions of the corresponding nodes. So the decoder in this model applies the inner product of node feature distribution and translate the distribution matrix into an adjacency matrix in which each element means the probability of the connection between the corresponding two nodes.

4.2.2 Loss Function

Loss function evaluates the difference between prediction values and real values of a learning model and provides support for the backward algorithm to make parameter training. The training process is a series of steps in which an optimization algorithm optimize the model by adjust the parameters in it according to the loss function value.

The output of the graph neural network model in this research is a matrix of probabilities of connections between each graph node. The target of this model is the median graph of the three input genomes, which could be represented in the form of adjacency matrix. So the distance between the predicted connection matrix and the target adjacency matrix could be defined as the binary cross-entropy of the element values of these two matrix.

For an input data sample $(\mathcal{G}_s, G)$, the output matrix of the graph neural network model on the three genome $\mathcal{G}_f$ is named as $\tilde{M}$ and the target matrix which is the adjacency matrix of G is named as M. If the connection between node i and j in the graph G then it's a positive connection and the loss value otherwise it's a negative connection. The loss function could be defined as a weighted sum of binary cross-entropy loss of positive connections and loss of negative connections

$$L_m = -\frac{1}{n^2}\sum_{i,j}^{n}(\alpha M_{ij} log\tilde{M}_{ij} + \beta(1 - M_{ij})log(1 - \tilde{M}_{ij}))$$

in which n is the number of nodes and α, β are the weights of positive connections and negative connections.

In order to prevent model over-fitting, $L2$ regularization could be appended to the loss function to regularize the scale of model parameters. The $L2$ penalty is defined

as $||\omega||_2^2$ in which ω is the kernel of graph neural network model. So the loss function with $L2$ regularization is defined as

$$L = L_m + \lambda||\omega||_2^2$$

with a coefficient λ of the $L2$ regularization penalty $||\omega||_2^2$.

4.2.3 Matrix to Graph

The output of variational graph auto-encoder is the encoded feature vector of each graph node. The decoder in the graph neural network model takes two node feature vectors and decode the features of these two graph node into a probability of connection between these two nodes. With an input graph which is generated from three genomes, the graph neural network model computes feature vectors of all nodes and decode them into an adjacency matrix which represented in the form of probability.

For a three genome set $\mathcal{G}_s^i$ which is composed of n unique genes, the corresponding input graph $\mathbf{G}_s^i$ has $2n$ nodes. After this graph is encoded and decoded by the variational graph auto-encoder and the decoder in the graph neural network, we can get the transformed graph adjacency matrix $\hat{M}$ with size $2n \times 2n$. This matrix could be converted into a gene order graph in following procedures. Firstly set the diagonal of matrix $\hat{M}$ to 0 since the self loops in the gene order graph is not necessary to be considered about. Secondly for each row of $\hat{M}$, change the element adjacent to the diagonal items to 0 based on the row number. For example, row $i(i \in [0, 2n-1])$, if i is even number then set $\hat{M}_{i,i+1} = 0$ and if i is odd number then set $\hat{M}_{i,i-1} = 0$. Because nodes $2i$ and $2i+1$ are connected with each other in the genome graph when $i \in [0, n-1]$, so the connections between these nodes could be ignored.

The next step is to define a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, in which $\mathbf{V} = \{0, 1, \cdots, 2n-1\}$ and $\mathbf{E} = \{e_{ij} | i = [0, 1, \cdots, n-1], j = 2n+1\}$. Find the index (i, j) of the max value in $\hat{M}$, set $\mathbf{E} = \mathbf{E} \cup \{e_{ij}\}$ and set all the elements of row i, j and column i, j in $\hat{M}$ to 0. Repeatedly update edge set $\mathbf{E}$ according to the max value in $\hat{M}$ as shown in

figure 4.3 until all the elements in $\hat{M}$ is 0. The generated graph $\mathbf{G}$ shows the target genome predicted by the graph neural network model based on the three genomes of the input genome set $\mathcal{G}_s^i$.

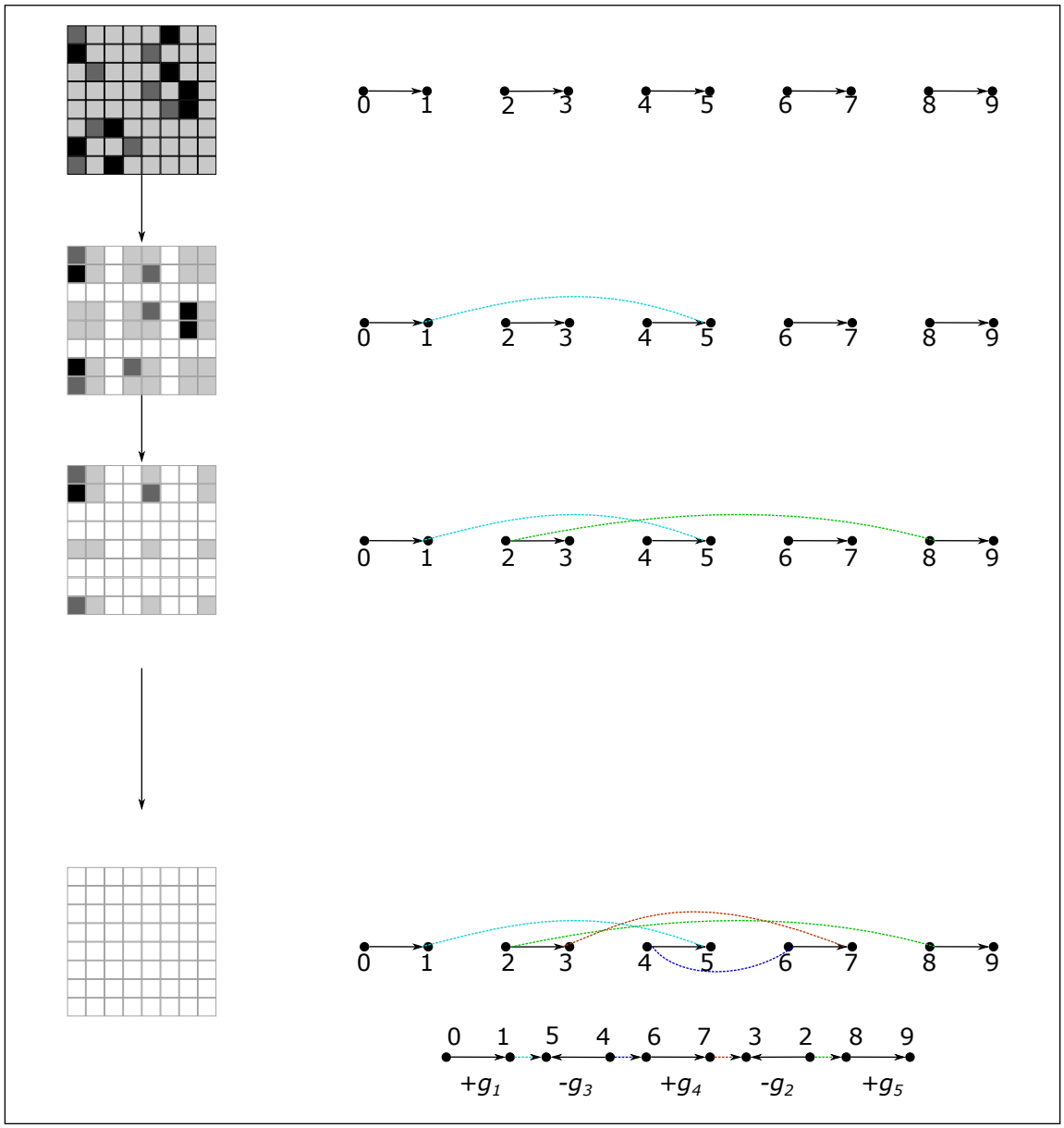

Figure 4.3 Conversion from probability matrix to genome sequence

Graph encoder model was implemented with PyTorch Geometric library. [1] The

[1]https://github.com/pyg-team/pytorch_geometric

python code for this model and related algorithms discussed in previous sections is available online. [2]

4.3 Evaluation Datasets

In order to evaluate the ability to generate the median of three genomes of method discussed in Section 4.2, six training datasets and three validation datasets were constructed. These datasets were generated with random sequences and random evolution events. Detail information about these datasets is discussed in following subsections.

4.3.1 Model Training Datasets

Six datasets were constructed to train the variational graph encoder model. Each dataset corresponds to different genome sequence lengths and different sample numbers. Two datasets were constructed with genomes that contains 10 genes, one of them has 9000 samples and another one has 45000 samples. Similarly, two datasets were constructed for genomes of 50 genes and two datasets were constructed for genomes of 100 genes. These datasets are named under the pattern as g3m + {length}g + {sample number}k, in which 'g3m' means this dataset is used to train model that generates median of three genomes and 'length' is the number of genes in samples and 'sample number' is the number of samples in this dataset. Statistic information of these datasets is listed in table 4.1, 4.2, 4.3.

Evolution rates utilized in dataset construction are from 0.1 to 0.9 with a step of 0.1. For each evolution rate, 1000 samples or 5000 samples are included in the corresponding dataset. For example, when the evolution rate is $r(r \in [0.1, 0.2, \cdots, 0.9])$, genome length is n and sample number is $l(l \in [1000, 5000])$, l genome sequences $S = \{s_1, s_2, \cdots, s_l\}$ are constructed by randomly arrange n genes, then apply $n \times r$

[2] https://github.com/Paeans/phylognn

steps of inversion operation to each genome sequence to get genome sequence set $\{s_1^1, s_2^1, \cdots, s_l^1\}$. Repeat this process for another two times to get sequence sets $\{s_1^2, s_2^2, \cdots, s_l^2\}$ and $\{s_1^3, s_2^3, \cdots, s_l^3\}$. Test the distance between sequence s_i and sequences $\{s_i^1, s_i^2, s_i^3\}$, if the distance equal to the lower bound of genome median then $(s_i, \{s_i^1, s_i^2, s_i^3\})$ could be added into dataset with s_i as the target sequence and $\{s_i^1, s_i^2, s_i^3\}$ as the three source sequence. Repeat these steps until all the sequence in S have found three source genomes.

Table 4.1 Statistic value of training dataset $g3m10g1k$ and $g3m10g5k$

	g3m10g1k				g3m10g5k			
rate	max	average	min	deviation	max	average	min	deviation
0.1	6	6.000	6	0.0	6	6.000	6	0.0
0.2	12	8.729	4	1.6797	12	8.762	2	1.7988
0.3	18	11.026	2	2.6984	18	11.160	4	2.7235
0.4	22	13.264	4	3.4854	24	13.205	4	3.4282
0.5	25	14.489	4	3.8267	27	14.503	4	3.8023
0.6	25	15.289	4	4.0259	27	15.396	4	3.9484
0.7	29	16.152	6	3.9851	28	15.928	4	4.0675
0.8	27	16.147	6	3.9224	27	16.306	4	4.0864
0.9	27	16.553	4	3.9554	27	16.512	4	4.0061

Table 4.1 lists the statistic information about the two datasets generated with 10 genes genome sequences. The dataset g3m10g1k has 9 groups of samples and each group corresponds to a different evolution rate between three source genomes and the median genome of them. The max, average and min list the max value, average value and min value of the data sample sequence distances among each evolution rate group and the deviation value is the standard deviation of these median scores.

When the evolution rate is 0.1, there is 1 inversion operation between the median sequence and each vertex sequence. The distance between each vertex sequence is 2. All data samples in this group have total distance of 6 and the deviation is 0.0. As the evolution rate increases, the max distance sum and average distance sum increase while the min distance sum keeps low. High evolution rate brings more evolution operations to sequences of each data sample and the triangle of three source genomes is enlarged. The max distance sum increased to 29 and deviation becomes 3.9851 when the evolution rate is 0.7. There is no big difference on the statistic values between evolution rate 0.8 and 0.9, which shows the evolution coverage of these two groups is similar. Dataset g3m10g5k contains 5000 samples in each evolution rate group and the corresponding statistic values of each group are similar to the values of dataset g3m10g1k. The deviation of this dataset is a little higher than the deviation of dataset g3m10g1k when the evolution rate is higher than 0.6.

Table 4.2 Statistic value of training dataset $g3m50g1k$ and $g3m50g5k$

	g3m50g1k				g3m50g5k			
rate	max	average	min	deviation	max	average	min	deviation
0.1	30	18.090	6	4.7923	30	17.867	6	4.8829
0.2	60	32.548	6	9.7163	60	32.689	6	10.0317
0.3	90	45.519	6	13.5668	87	43.688	8	13.6971
0.4	104	54.936	10	16.8704	107	49.704	6	15.7209
0.5	113	59.016	10	17.8732	115	53.501	8	16.0204
0.6	108	59.973	10	17.0699	112	60.116	8	17.7572
0.7	107	60.773	12	17.2984	119	61.177	8	17.5545
0.8	99	58.841	8	16.2481	111	60.251	6	17.1459
0.9	108	60.896	6	17.5259	104	58.795	10	16.6235

Table 4.3 Statistic value of training dataset $g3m100g1k$ and $g3m100g5k$

	g3m100g1k				g3m100g5k			
rate	max	average	min	deviation	max	average	min	deviation
0.1	60	33.498	6	9.7507	60	32.326	8	9.9052
0.2	116	61.176	8	19.9525	118	61.783	8	19.5505
0.3	164	88.487	10	28.7602	168	87.969	8	28.0855
0.4	195	98.094	14	30.8143	193	95.589	10	30.0308
0.5	196	110.734	14	32.9829	191	104.657	10	30.9927
0.6	215	105.948	22	32.0009	205	109.197	8	32.3795
0.7	185	109.401	16	31.2119	204	111.348	14	31.8980
0.8	205	108.538	18	32.6364	191	106.424	8	30.7078
0.9	195	112.629	10	32.9768	211	109.360	8	31.9528

The statistic values of datasets g3m50g1k, g3m50g5k, g3m100g1k and g3m100g5k are shown in table 4.2 and 4.3. Since the genome sequence length of these datasets is more than 10, the operation number of evolution events applied on each sequence is more than 1 when the evolution rate is 0.1 and diversity of distance sum appeared in group-0.1. The deviation values in these two tables show that the deviation of distance sum on genomes with 100 genes are close to two times of the deviation values of genomes with 50 genes. By enlarging the sample size of a dataset doesn't have great influence to the operation coverage since the deviation values of 5000 samples have no big difference with the deviation values of 1000 samples in both 50 genes datasets and 100 genes datasets. Figure 4.4 and 4.5 shows the distribution of distance sum of 1000 samples and 5000 samples when the evolution rate is setting as 0.9. These two distribution are very similar to each other. Experiment results also shows that the average precision and AUC of model outputs are very similar no

matter the the size of samples used to train the model is 1000 or 5000.

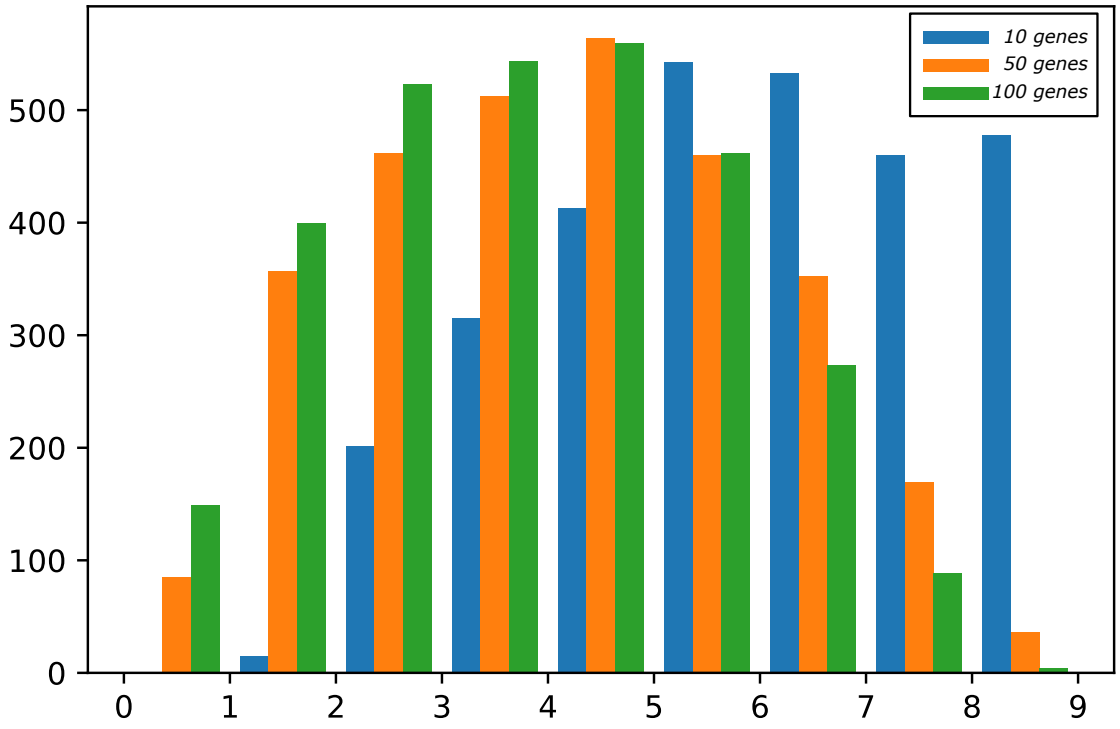

Figure 4.4 Histogram of sample numbers in different evolution rate category of training datasets $g3m10g1k$, $g3m50g1k$, $g3m100g1k$

These statistic values discussed in this section show that the dataset generation method defined in previous section could cover almost all operations on genome sequences with 1000 samples and datasets with 1000 samples are suitable for model training. Each sample is transform into a graph data object with adjacency matrix, edge index, node features and so on. Three source genomes in a sample are combined together and represented as one adjacency matrix. The target median genome is transformed into a group of positive edge indexes which exists in the target median genome and a group of negative edge indexes which are not.

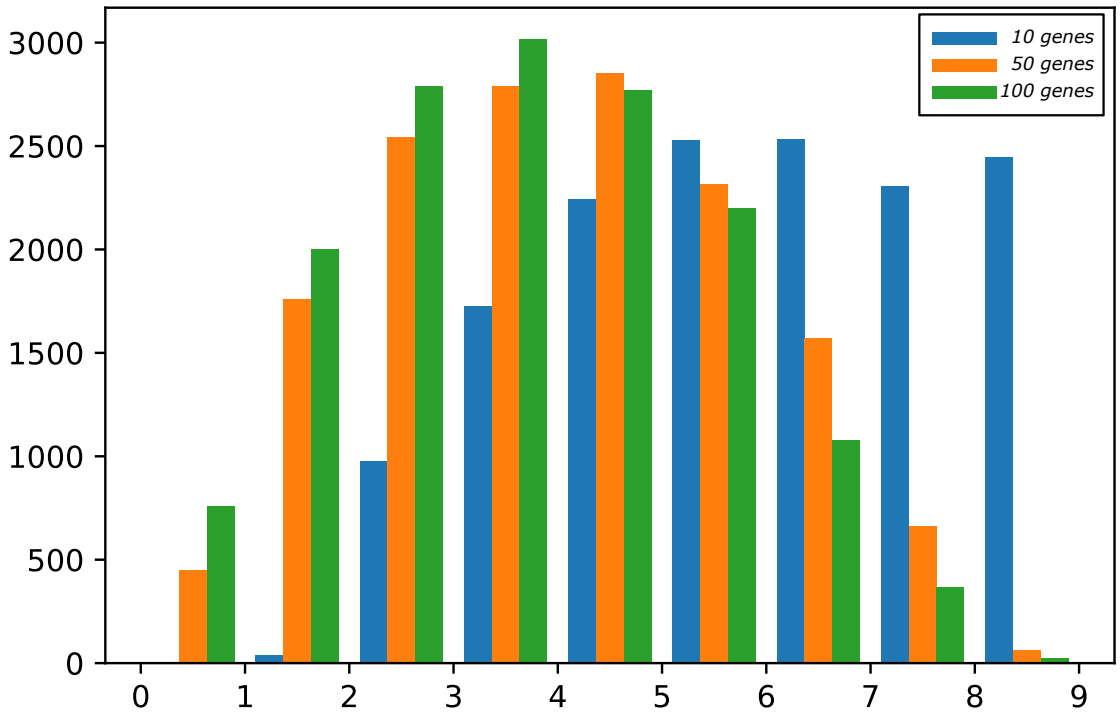

Figure 4.5 Histogram of sample numbers in different evolution rate category of training datasets $g3m10g5k$, $g3m50g5k$, $g3m100g5k$

4.3.2 Validation Datasets

Three validation datasets were constructed to evaluate the performance of median genome model. These three datasets are named as v3m10g, v3m50g and v3m100g. The length of genome sequences in these three datasets are 10, 50 and 100 respectively. Each dataset contains 12 groups with different evolution rate. The evolution rates r utilized in these datasets are ranged from 0.1 to 1.0 with step of 0.1 and two extra rates $[1.5, 2.0]$. Each group in these datasets has 200 samples. The procedure to generate each dataset is same as the procedure used to generate training datasets which are described in previous subsection. One thing that different from training dataset generation is that the source genomes and target genomes don't have to be transformed into graph data objects because these genomes are used to compute the

median score and distance sum which are calculated with original genome sequences.

Table 4.4 Statistic value of validation datasets $v3m10g$, $v3m50g$ and $v3m100g$

	v3m10g			v3m50g			v3m100g		
rate	max	avg	std	max	avg	std	max	avg	std
0.1	6	6.000	0.0000	27	18.250	4.4031	58	33.960	9.8203
0.2	12	8.220	1.8952	58	31.165	10.3411	106	63.700	19.0074
0.3	18	11.605	2.6625	78	45.490	13.9589	159	90.845	30.3130
0.4	24	13.725	3.4175	91	51.350	17.1935	162	91.015	29.0437
0.5	23	13.815	3.4250	95	55.895	17.2440	183	110.270	31.7669
0.6	25	14.910	4.3246	99	59.555	17.1676	169	100.645	30.3667
0.7	25	15.590	4.0227	89	56.475	15.7731	191	113.040	32.4387
0.8	26	17.110	4.1567	105	60.570	17.7886	175	108.775	29.9592
0.9	25	16.915	3.7826	100	57.250	18.0066	197	118.905	34.4956
1.0	26	16.080	4.1199	101	60.410	18.8011	176	112.565	31.1030
1.5	28	17.750	3.6562	96	64.930	15.1903	186	112.105	31.8142
2.0	25	17.750	3.7185	102	60.255	19.7775	189	107.740	35.1911

Table 4.4 lists the statistic values of these three validation datasets. The max distance sum in dataset v3m10g is 28 which appeared in the group with evolution rate 1.5 and the largest deviation of this dataset is 4.3246 when the evolution rate is 0.6. In datasets v3m50g and v3m100g, the max distance sum samples appeared in group-0.8 and group-0.9 respectively. The average distance sum and the standard deviation of each dataset increases as the evolution rate increases. Since the maximum distance between two genomes with n genes is n, so the max distance sum in each dataset reaches two thirds of its possible maximum distance value.

Evaluation process will be deployed as three experiments. In the first one, a varia-

tional graph encoder model will be trained with data samples of group-0.9 in dataset g3m10g1k and evaluate the model performance with evaluation dataset v3m10g. The second one will take datasets g3m50g1k and v3m50g as training dataset and evaluation dataset. The third one will evaluate model ability of median genome generation when the genome sequence length reached to 100.

4.4 Experiment Results and Discussion

There are three major parts in this evaluation experiment. The first one is using data samples with evolution rate 0.9 in dataset g3m10g1k to train a variational graph encoder model then using this model to generate median graphs according to the source sequences in validation dataset v3m10g. The second one tests the variational graph encoder model on longer genome sequences which have 50 genes. The training dataset and validation dataset used in second part is g3m50g1k and v3m50g. The third one evaluates the model with dataset v3m100g after training it with dataset g3m100g1k. The process of model training and performance evaluation of each part is independent and the evaluation results will be computed separately in each part. The structure and hyperparameters of the variational graph encoder model used in this experiment are same across these three parts, the different things are the training dataset and validation dataset.

During each part of this experiment, an encoder model will be trained with the corresponding training dataset. After each epoch of training, use this model to generate the median genome of each data sample in the corresponding validation dataset. Then compute the distance between the generated median genome and the target genome included in the data sample and record the average value of these distance values. For example, a data sample $(s_i, \{s_i^1, s_i^2, s_i^3\})$ with $\{s_i^1, s_i^2, s_i^3\}$ as source genomes and s_i as target genome in validation dataset and the generated median genome from the trained graph encoder model is $\hat{s}_i$, distance of this sample will be $\Delta(s_i, \hat{s}_i)$ and

the mean distance of all these n samples in a validation dataset is

$$md = \frac{1}{n}\sum_{i=1}^{n}\Delta(s_i, \hat{s}_i).$$

This mean distance md shows the difference between model generate median genomes and the real median genomes, so lower md means better model performance. In order to figure out the performance of graph encoder model on different evolution rate samples, the validation samples are separated into 12 groups according to their evolution rate and the evaluation results are discussed in 12 separate groups.

4.4.1 Median of Genomes with 10 Genes

In this part of experiment, a graph encoder model was trained with the training dataset g3m10g1k for 500 epochs. As the training process going, the mean distance of validation dataset v3m10g decreases along with the training epochs. Figure 4.6 plots the mean distances of 12 evolution rate categories in validation dataset v3m10g at each training epoch.

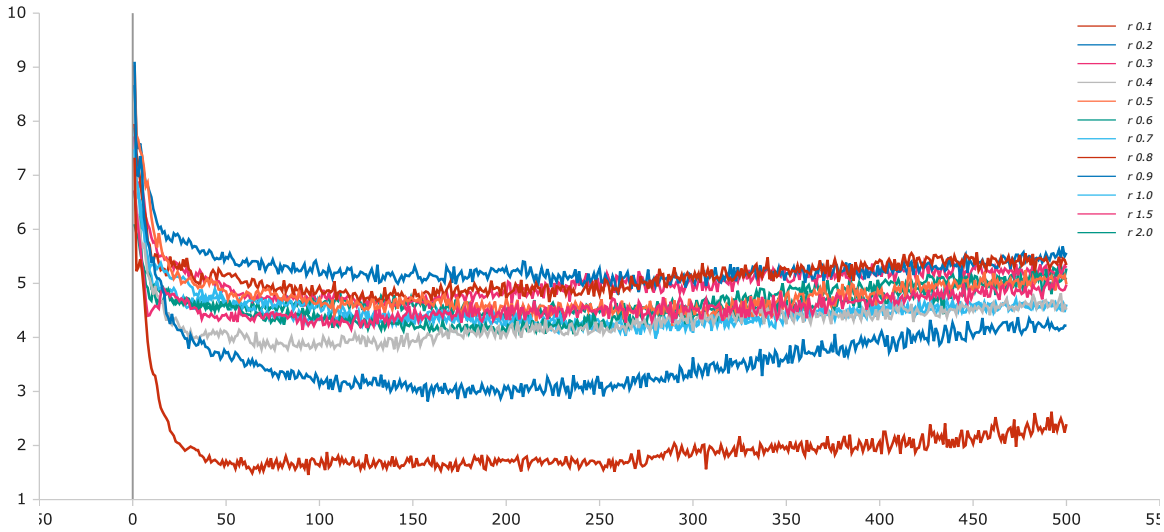

Figure 4.6 The trend of mean distance between generated median genome and lower bound median genome at each training epoch when model is trained with $g3m10g1k$

At the beginning, the mean distances are as high as 9 and these values decrease as epoch number increases. At around 200 epochs, all the 12 categories of validation samples reach to their lowest mean distance. As shown in this figure, difference evolution rate leads to different mean distance at same epoch point. At 200 epoch point, the category with evolution rate of 0.1 has the smallest mean distance value because there are only two evolution operations between each pair of source sequences and it is easy to figure out the median genome of them. Other categories, except for category of evolution rate 0.2, have very close mean distance values which are between 4 and 5.

After 50 epochs training, the mean distance of each evolution rate category fluctuates around a specific value. Even more, the model shows over-fitting after epoch 250 and the mean distances increase as training processing going. By adjusting the learning rate reduction parameter which reduces the learning rate after a specific number of steps could prevent this situation. In this experiment, these kind of parameters are not adjusted specifically, which are using the default parameters of learning model library, in purpose of showing the ability of graph encoder model on solving median of three genomes problem. In real applications, the hyperparameters could be adjusted to most suitable situation to obtain more accurate results.

The plots in figure 4.6 show significant decrease of mean distance as training processes going, which means that the defined variational graph encoder model could effectively learn how to generate median genome, or a genome that closes to median genome, from three homologous genomes. With as little as 50 epochs, the model reaches an acceptable state that could be used to do median genome generation. When using more data samples to train the model, for example all the samples in training dataset g3m10g5k, training epochs could be smaller.

Table 4.5 lists the detail statistic values of 12 categories validation samples. In this table, rate is the evolution rate of validation sequences, lower bound is the minimum

Table 4.5 Statistic results of model trained and evaluated with dataset $g3m10g1k$ and $v3m10g$

rate	lower bound	median distance	ratio	deviation	accurate
0.1	3.0000	1.0200	0.3400	1.0998	87
0.2	4.1100	2.0350	0.4951	1.4470	33
0.3	5.8150	2.7950	0.4807	1.7038	19
0.4	6.9300	2.7600	0.3983	1.7270	22
0.5	7.0200	3.0500	0.4345	1.7051	17
0.6	7.6000	3.3750	0.4441	1.7648	15
0.7	7.9400	3.1250	0.3936	1.7057	14
0.8	8.7500	3.7050	0.4234	1.6637	4
0.9	8.6300	3.9050	0.4525	1.7709	7
1.0	8.2200	3.3600	0.4088	1.7721	8
1.5	9.1100	3.9650	0.4352	1.6321	5
2.0	9.1200	3.6100	0.3958	1.6876	6

possible distance between median genome and source genomes, median distance is the distance between median genome generated by encoder model and target median genome. The ratio column lists ratio of median distance value to lower bound value and deviation column shows the standard deviation of median distances in a evolution rate category. Accurate number counts the number of model generated median genomes which are identical to the target median genomes. All the values in this table are the mean values of 200 validation data samples in each category.

As shown in table 4.5, the lower bound and the median distance increase as the evolution rate increases. However, the ratios between median distance and lower bound don't change very much when the evolution rate changes and the standard deviation of median distance also changes between small range. The smallest ratio

in this validation experiment is 0.34 which is the ratio of median distance 1.02 and lower bound 3.00 of validation category with evolution rate 0.1. When the evolution rate is 0.2, the ratio is the largest one among 12 categories, which is 0.4951. The ratio between median distance and lower bound doesn't show significant correlation with the evolution rate; the performance of graph encoder model on median genome generation is stable on different evolution rate genomes.

When the evolution rate is low, the graph encoder model could generate a genome which is identical to the median genome of source genomes on some situations. The accurate column in table 4.5 is the number of model generations that are identical to the target median genome. The values in accurate column show that the model could get a lot of accurate generations on category with low evolution rate. For example, the accurate generations are 87 when the category has evolution rate 0.1. As the evolution rate increases, the accurate generations become less. Especially when the evolution rate is higher than 0.7, the number of accurate predictions drops significantly, which is lower than 10. It is obvious that the graph encoder model performs better on categories with lower evolution rate.

4.4.2 Median of Genomes with Longer Genes

The variational graph encoder model was also tested with 50 genes genomes and 100 genes genomes. Same as the evaluation processes applied with 10 genes genomes, validation data samples are group into 12 categories based on the evolution rate and each category is evaluated separately. The training procedure and testing procedure are also same as the 10 genes genomes validation process. In both of these two validation experiments, the total training epochs are 500 and the mean distance of each validation category is calculated and compared at each training epoch point. The mean distance trend are plotted in figure 4.7 and 4.8 and the detail statistic values are listed in table 4.6 and 4.7.

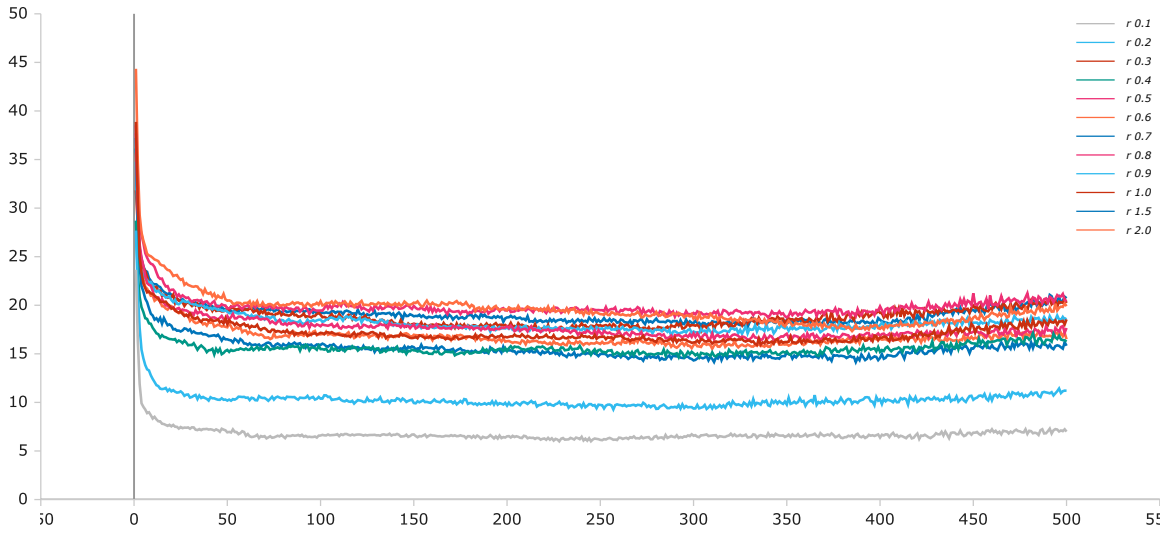

Figure 4.7 The trend of mean distance between generated median genome and lower bound median genome at each training epoch when model is trained with $g3m50g1k$

Figure 4.7 plots the mean distances of validation samples at each training epoch point when training and testing graph encoder model with dataset g3m50g1k and dataset v3m50g. At beginning of training, the parameters of the graph encoder model is randomly initialized and the mean distance between generated genomes and target median genomes is close to 45. The mean distances of 12 categories reduce shapely during the first 10 epochs and they reach to stable state at around epoch point 50. After epoch point 50, these mean distance lines also show a little bit of fluctuation. The over-fitting problem of this training isn't as obvious as the training on 10 genes genomes. During epoch 50 and 400, the model keeps a stable state and the mean distances of 12 categories validation samples have no significant changes.

The trend of mean distances on validation dataset v3m100g have similar phenomenon as the ones of validation dataset v3m50g, which are shown from plots in figure 4.7 and 4.8. However, the over-fitting problem are serious when training process reaches to epoch point 100. The reason of this problem is that there are a lot of unchanged segments in the 100 genes genomes which lean the model parameters to

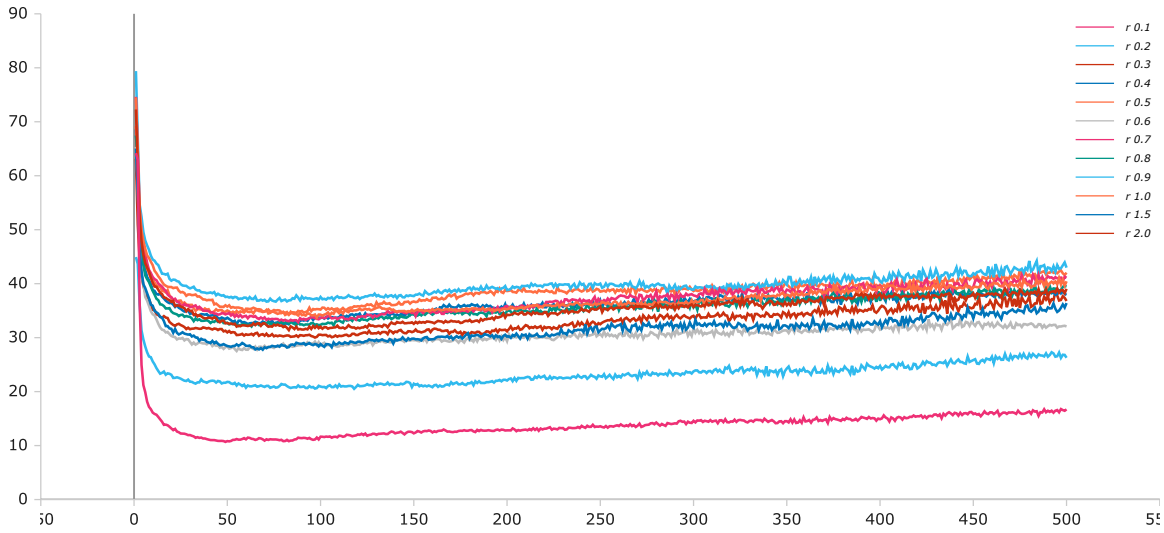

Figure 4.8 The trend of mean distance between generated median genome and lower bound median genome at each training epoch when model is trained with $g3m100g1k$

balance the encoding of unchanged segments after specific number of epochs. This could be prevented by adjusting model parameters and learning rate parameters or by terminating the training processes earlier at epoch point 100.

Table 4.6 and 4.7 have the validation results of validation datasets v3m50g and v3m100g. On validation dataset v3m50g, median distance increases significantly when evolution rate changes from 0.1 to 0.2 and from 0.2 to 0.3. When evolution rate is larger than 0.3, median distance has some changes but the range of changes is not very large. The ratio of median distance to lower bound on this validation dataset is smaller than the ratio on validation dataset v3m10g; the relative distance between the generated median genomes and the target median genomes is closer. Different from dataset v3m10g that has the smallest ratio when the evolution rate is 0.1, the smallest ratio in this dataset is 0.2702 which is the validation result of evolution rate 0.7 category. There is still no significant correlation between the ratio value and the evolution rate. The numbers of accurate generations in each category are very small, especially, there are almost no identical generations but only close generations when

Table 4.6 Statistic results of model trained and evaluated with dataset $g3m50g1k$ and $v3m50g$

rate	lower bound	median distance	ratio	deviation	accurate
0.1	9.1400	2.7400	0.2998	1.7614	17
0.2	15.6100	4.6050	0.2950	2.8194	13
0.3	22.8000	7.9950	0.3507	4.1467	3
0.4	25.7550	7.9550	0.3089	4.5785	7
0.5	28.0600	9.1250	0.3252	4.9909	3
0.6	29.9000	9.7550	0.3263	5.1415	3
0.7	28.3850	7.6700	0.2702	4.2428	8
0.8	30.4450	10.4850	0.3444	5.5884	6
0.9	28.7350	9.1650	0.3189	4.8371	5
1.0	30.3100	9.6300	0.3177	5.4665	2
1.5	32.6100	10.0100	0.3070	4.9416	2
2.0	30.2750	8.6800	0.2867	4.7820	6

the evolution rate is high. The results of dataset v3m100g have similar phenomenon. The ratio values are more smaller with lowest one 0.2662 and largest one 0.3775. However, the number of accurate generations reduces to zero on several categories of this dataset; extra iterations of median genome generation could be applied.

Table 4.7 Statistic results of model trained and evaluated with dataset $g3m100g1k$ and $v3m100g$

rate	lower bound	median distance	ratio	deviation	accurate
0.1	16.9800	4.5200	0.2662	2.6513	9
0.2	31.8550	9.1350	0.2868	4.8401	1
0.3	45.5200	15.1300	0.3324	8.1513	0
0.4	45.6200	14.2300	0.3119	7.4838	2
0.5	55.2900	17.9950	0.3255	8.8207	2
0.6	50.4500	13.9950	0.2774	7.3814	0
0.7	56.6950	17.7600	0.3133	8.1032	1
0.8	54.5050	16.2600	0.2983	8.3632	0
0.9	59.5550	18.7250	0.3144	9.2865	0
1.0	56.3900	21.2900	0.3775	8.2271	0
1.5	56.1900	16.6650	0.2966	7.9040	1
2.0	53.9900	15.6200	0.2893	8.5502	0

Chapter 5

Phylogenetic Tree Construction

5.1 Grappa framework

The task of phylogenetic tree construction is to build up a rooted or unrooted tree to represent the evolution relation between genomes which are located at leaf nodes. These genomes are grouped together and connected to their ancestral genome with tree paths according to their function similarity or evolution distance. Neighbor joining algorithm is a classic bottom up method to cluster genomes based on the evolution distance between each pair of taxa [15]. In this algorithm, a distance matrix is initialized and updated in multiple iterative steps. At each step, two taxa are grouped together and the ancestral is added into the distance matrix. After a bunch of iterations, all taxa are grouped together and joined into a phylogenetic tree. This algorithm assumes the distance between taxa is additive. When the distance between genome a and c is equal to the sum of distance between genome a, b and distance between genomes b and c, this distance property means it is additive. In real situations, the distance between genomes doesn't always obey the rule of additive; the phylogenetic tree constructed with neighbor joining algorithm could not represent the real evolution relation between genomes.

One standard of phylogenetic tree construction is that the total evolution length in the tree, which is the sum of length of all the paths in a tree, lower is better. Both tree topology structure and tree internal ancestral genome could affect the length of a phylogenetic tree. The goal of phylogenetic tree optimization is to find out a

tree topology structure and internal ancestral genome nodes that could minimize the evolution length of a group of genomes. Currently, a great amount of phylogenetic tree construction methods are based on heuristic search of all possible tree topology, in order to find out the best tree topology. For the internal ancestral genomes, some median genomes had been utilized, such as greedy median, break point median, inverse median and so on.

Genome Rearrangements Analysis under Parsimony and other Phylogenetic Algorithms (GRAPPA) defines a framework to generate phylogenetic trees based on evolution distance and median genomes [107, 108]. This algorithm could take break point distance, inverse distance, double cut and join distance, computation of which is linear complexity, as a measure of similarity of genomes when searching the optimal tree topology structure. Many median algorithms could be utilized by GRAPPA to infer the internal ancestral genomes and prune the search branches. [1]

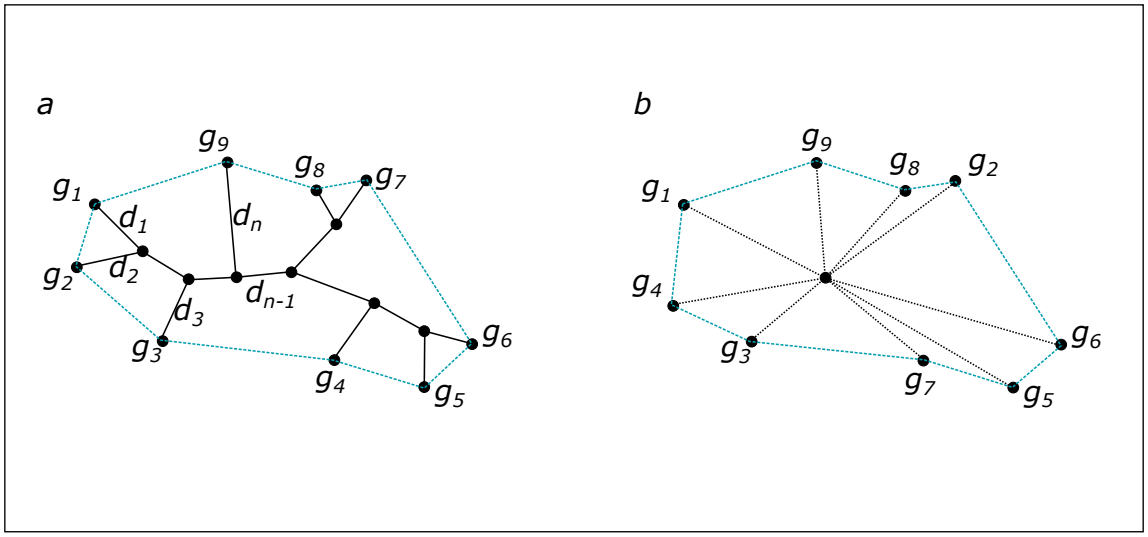

Figure 5.1 Genome circular orders and lower bound phylogenetic tree distance

Algorithm GRAPPA uses lower bound of a candidate tree and upper bound of

[1] https://www.cs.unm.edu/~moret/GRAPPA/

initial tree to prune search branches, which significantly reduces the size of possible tree space [109]. For a genome set S with n genomes $\{g_1, g_2, g_3, \cdots, g_n\}$, assume these genomes formed a phylogenetic tree T and each genome located at one of this tree's nodes, no matter what topology structure this tree takes, the sum of tree paths is greater than half of adjacency distances sum according to the triangle inequality property. As shown in figure 5.1, all the nodes of the phylogenetic tree arranged in a circular style and they are connected by dash lines one by one. The edges in the phylogenetic tree represents the distance between two nodes in the tree, leaf nodes or internal nodes. The circle dash line represent the distances between pair of leaf nodes. Then

$$\sum_{e_{ij} \in T} \Delta(\tilde{g_i}, \tilde{g_j}) \geq \frac{1}{2}(\tilde{d_1} + \tilde{d_2} + \tilde{d_3} + \cdots + \tilde{d_n})$$

in which e_{ij} are the edges in tree T between node i and j, $\tilde{g_i}$ and $\tilde{g_j}$ are the genomes at node i and j. If node i is internal node, then $\tilde{g_i}$ will be the median genome of its three neighbor genomes. And $\tilde{d_i}(1 \leq i \leq n)$ is the distance between a pair of genomes on the dashed circle. So, when the order of genomes in a tree is defined, no matter which topology structure this tree has, its tree path length has a lower bound which is half of the circular distance of all the genomes.

During the search process, algorithm GRAPPA generates a bunch of possible genomes circular orders and considering the circular distance of each circular arrangement. If the circular distance is greater than two times of current tree path length, then new generated trees on this arrangement order must have longer tree path among all genomes; there is no necessary to make further searching on this situation. At the beginning of algorithm, GRAPPA will generate a phylogenetic tree using neighbor joining method as an initial tree, then utilize lower bound properties to filter out a lot of impossible genome arrangements and topology structures. Since the computation of distance between pair of genomes is linear complexity, this strategy could prune a great amount of search branches. As the search processes going

on, the initial tree and tree path upper bound will be updated when there are better phylogenetic trees.

5.2 Median Optimization

Algorithm GRAPPA uses lower bound and upper bound to filter out impossible tree leaf orders. Lower bound of a specific tree leaf order could be computed in linear complexity. Upper bound is the sum of path lengths in a tree and it depends on median algorithms to generate internal ancestral genomes and compute the distances between them and their neighbors. Since the upper bound of a tree could be very helpful on impossible tree filtering, it is better to reduce this value as early as possible in the phylogenetic tree optimization process.

Median algorithms have great effects on the optimization of tree's upper bound. Algorithm GRAPPA takes a median algorithm to generate median genomes as internal nodes and optimize these nodes in multiple iteration updates until all the internal nodes become are stable. Like the phylogenetic tree in figure 5.2(a), node $\tilde{g}_i$ is updated with the median genomes generated by median algorithm based on its neighbor nodes and then nodes $\tilde{g}_{(i+1)}$, $\tilde{g}_{(i+2)}$, $\tilde{g}_{(i+3)}$ $\cdots$ and so on are updated. After multiple iterations of updates, all the ancestral genomes located at each internal nodes become stable and the upper bound of candidate trees could be updated.

As shown in figure 5.2(b), unrooted phylogenetic tree T has an internal node $\tilde{g}_i$ and the three neighbors of this node is $\{\tilde{g_i}^1, \tilde{g_i}^2, \tilde{g_i}^3\}$. The path length between this node and its neighbors will be $l_i = (\tilde{d_i}^1 + \tilde{d_i}^2 + \tilde{d_i}^3)$ and this value is greater or equal to half of the sum of distances between its neighbor nodes, such that

$$l_i \geq \frac{1}{2}(\Delta(\tilde{g_i}^1, \tilde{g_i}^2) + \Delta(\tilde{g_i}^1, \tilde{g_i}^3) + \Delta(\tilde{g_i}^2, \tilde{g_i}^3))$$

and $\Delta(\ .\ )$ is the function to compute the evolution distance between two genomes. The ability of median algorithm to generate median genomes becomes significant to

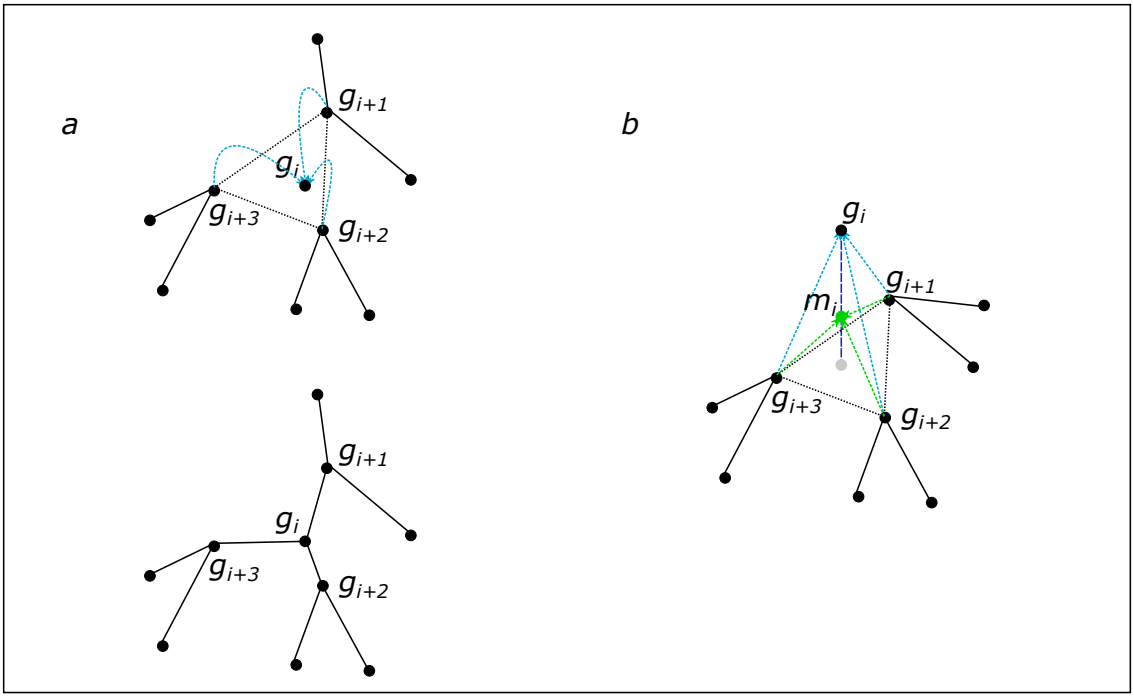

Figure 5.2 Updating the internal ancestral genome based on neighbor genomes in a phylogenetic tree and median ratio between generated median genome and optimal median genome

GRAPPA algorithm. If the median algorithm could generate a genome as close to the ideal median genome as possible then GRAPPA algorithm could finish a candidate tree's optimization in less iterations and decrease the upper bound of candidate trees sooner, both of these two benefits could help reduce tree construction search space.

For a ancestral genome $\tilde{g}_i$ and the ideal median genome, assume there exists one, of its three neighbor genomes is $\tilde{m}_i$, the ratio r of the distance between $\tilde{g}_i$ and $\tilde{m}_i$ to the distance between $\tilde{m}_i$ and three neighbor genomes $\{\tilde{g_i}^1, \tilde{g_i}^2, \tilde{g_i}^3\}$ is

$$r = \frac{2 \times \Delta(\tilde{g}_i, \tilde{m}_i)}{\Delta(\tilde{g_i}^1, \tilde{g_i}^2) + \Delta(\tilde{g_i}^1, \tilde{g_i}^3) + \Delta(\tilde{g_i}^2, \tilde{g_i}^3)}.$$

It shows how close this genome is to the ideal median genome which is the optimal target ancestral genome. Smaller r values mean that the generated genomes are closer to the ideal median genome and the path length is tree will also be smaller. So, this

ratio value could be used to evaluate if a median algorithm is suitable for GRAPPA algorithm.

5.3 Discussion

Median algorithms ASMedian [110], GAMedian [111], SAMedian [112] and IDQPSO-Median [113] utilize heuristic search strategy to find median genome of homologous genomes. They use genetic algorithm or simulated annealing to generate next generation of population from current layer population. After multiple iteration of simulation, the result genomes of these algorithms reach to stable status and the generated genomes are closer to their neighbor genomes than the initial genomes. All these four median algorithms could be utilized in GRAPPA platform as tools to optimize tree's internal nodes.

5.3.1 Algorithm Stability

The median ratios of these four median algorithms on 1000 genes genomes are different from each other. However, the difference between each other at the same evolution rate category isn't very large. As shown in table 5.1, among these four median algorithms, ASMedian, GAMedian and IDQPSO-Median have similar ratios at the same rate category and method GAMedian performs a little worse with a higher ratio value than other three median algorithms. Since they have similar median distance ratio, they will have same effects on GRAPPA platform optimization.

It's obvious from table 5.1 that the ratio value of these four median algorithms increase significantly as the evolution rate increases. When the evolution rate of genomes is small, these algorithms have small ratio values and they performs very well on median genome generation and searching; they become ineffective when evolution rate is large and the evolution patterns in genomes are complex. The reason of this is that these four algorithms utilize heuristic search strategy to optimize the generated

Table 5.1 The average median ratio of proposed model and four currently available methods on 1000 length genomes

		median ratio					
rate	lower bound	ASM^1	GAM^1	SAM^1	IDM^1	$GNNM^2$	
0.1	143.390	0.3629	0.4408	0.3629	0.3629	0.4705	0.3799
0.2	289.185	0.3569	0.5286	0.3576	0.3570	0.4839	0.3351
0.3	405.135	0.4058	0.6265	0.4346	0.4240	0.4829	0.3756
0.4	430.610	0.6167	0.8000	0.6609	0.6269	0.5198	0.3649
0.5	463.405	0.7174	0.8525	0.7572	0.7159	0.4707	0.3684
0.6	424.370	0.9406	1.0529	0.9737	0.9303	0.4868	0.3643
0.7	471.355	0.8825	0.9646	0.9044	0.8643	0.4993	0.3662
0.8	454.540	0.9800	1.0530	0.9950	0.9569	0.5269	0.3564
0.9	429.675	1.0864	1.1548	1.1019	1.0615	0.4378	0.3495
1.0	476.515	0.9651	1.0237	0.9791	0.9401	0.4495	0.3711
1.5	437.295	-	-	-	-	0.4743	0.3820
2.0	454.360	-	-	-	-	0.4420	0.3363

[1] data are from [113], categories 1.5 and 2.0 are missing, four columns correspond to methods ASMedian(ASM), GAMedian(GAM), SAMedian(SAM), IDQPSO-Median(IDM) on 1000 genes genomes

[2] two graph encoder models are evaluated, left column lists the ratio of model trained by 50 genes genomes and right column lists the ratio of model trained by 100 genes genomes

median genomes, considering the time complexity of algorithm running a iteration limitation or target boundary is defined by these algorithms, which reduces the ability of finding optimal median genomes among complex genome rearrangements.

Median algorithms have robust performance on high evolution rate categories is important in success generation of phylogenetic tree, otherwise their ratio values raised quickly when evolution rate increases. High ratio leads to long running time of GRAPPA platform to generate a phylogenetic tree [114]. Since high ratio will impact GRAPPA platform effectiveness from two aspects, search branch pruning and internal nodes updating, both of which could increase the running time. In such a situation, GRAPPA platform will take an exceptionally long time to finish a phylogenetic tree construction.

The graph encoder model discussed in chapter 4 has stable ratio performance on different evolution rate genomes. One advantage of utilizing graph encoder model as median genome generator is that there are no significant difference on ratio values when the evolution rate increases. In order to evaluate the performance of this model on 1000 genes median genome generation, two trained models with 50 genes genome dataset and 100 genes genome dataset are tested on 200 random samples, all of which consist of 1000 genes. Considering the training time complexity and the computing resource requirement, the graph encoder model wasn't trained with 1000 genes datasets. The ratio results of 50 genes model and 100 genes model on 1000 genes genomes are listed in that last two columns of table 5.1. It's obvious that the ratio of these two models on high evolution rate categories has no big difference with the result on low evolution rate categories. On some high evolution rate categories, these two models have lower ratio. For example category of evolution rate 1.0 has ratio results as 0.4495 and 0.3711, which is lower than evolution rate 0.1 category.

Models trained with shorter genomes will cut the long evaluation genome into multiple segments; rearrangement patterns exists in these short segments could be recognized by trained model and sub-optimal median genomes could be generated. However, the inversions or transitions that cover long distances in genomes couldn't be recognized; the trained models will loss some accuracy on median genome gener-

ation when the testing genome is times longer than the training genome. As shown in table 5.1, the ratio results on 1000 genes genomes of 100 genes model the last column and 50 genes model the second last column are significantly higher than the ratio results of these two models on 100 genes genomes and 50 genes genomes. The increasing of ratio value is caused by long range inversions and transitions in long genomes. The ratio will decrease as the training genomes become longer. For example, the ratio of 100 genes model is lower than the ratio of 50 genes model at every evolution rate category. The smallest ratio difference between these two models is 0.07 when the evolution rate is 1.0 and the largest difference is 0.15 when the evolution rate is 0.4 and 0.8. It is obvious that by increasing the genome length of training samples, trained model could generate genomes closer to the ideal median genomes.

5.3.2 Time Consuming

Another advantage of graph encoder model is that it could generate optimal median genome from three genomes in definite time, no matter what is the evolution rate of these three genomes from their ancestral, the computation time only depends on the genome length. Median algorithms ASMedian, GAMedian, SAMedian, IDQPSO-Median need multiple iterations of genome generation and updating to optimize the target genome. When the evolution rate of genomes becomes large, the arrangement operations and genome variation patterns become complex, which requires more and more optimization iterations to achieve acceptable results. Graph encoder model utilizes neural network parameters to optimize the target genome based on genome structure patterns learning from training samples. Different from other median algorithms, it doesn't need iterations to obtain optimal results; even if the genome rearrangement operations are complex, target genome could be generate in the same time as simple ones.

The graph encoder model is suitable to be utilized in GRAPPA platform as me-

dian genome generator because it takes less time on median genome optimization and the time is definite. The running time of four heuristic median algorithms and graph encoder model generating one 1000 genes genome on a 2.2GHz CPU core is very different from each other. Graph encoder model has great advantages on median generation capability when comparing with other four algorithms on time consuming. The graph encoder model could generate one 1000 length median genome in 4 seconds, which outperforms other four median algorithms, even when the genome rearrangement pattern is very simple. Its advantage isn't very obvious compared with IDQPSO-Median which takes 5 seconds at evolution rate 0.1 category. As the evolution rate increases, its excellence shows up because the time of other algorithms increases so fast while its time consuming is remain 4 seconds.

In GRAPPA platform, the most time consuming part is median genome generation and tree nodes update, so the key factor of achieving faster phylogenetic tree construction is reducing the median generation time costed by median algorithms. With the advantages of stable low median ratio and outstanding time consuming on long median genome generation, graph encoder model is the most suitable algorithm to be utilized in GRAPPA platform for phylogenetic tree construction.

Chapter 6

Conclusion

This dissertation talks about subcellular localization and phylogenetic tree construction by utilizing deep neural network and graph neural network. Subcellular localization predicts genome's locations in a cell, which provides helpful information for biology research in identifying of genome function and relation. Phylogenetic tree as a representation of relationships among a group of genomes, could identify the evolution procedure among species. With phylogenetic trees, biologists could research the development of genomes or species and predict their future genotype and phenotype. In order to make accurate predict of subcellular location and generate optimal phylogenetic tree, two neural network learning models are proposed in this dissertation. According to experiment results, these two learning models could outperform other currently available methods on these two tasks.

The deep neural network learning model takes genome sequence information and evolution information, in form of position specific score matrix, to obtain more accurate prediction results on multiple subcellular localization and it has significant improvements on average precision and ranking loss when tested on two widely used benchmark datasets; this learning model is suitable for subcellular localization and could be considered as an alternative to traditional methods.

The graph neural network learning model generates the median of three homologous genomes with the goal of reaching the minimum distance between the median and these three genomes. When compared with other four currently available median algorithms, it is in evidence that this learning model has great advantages than them

on median ratio achievement, especially on genomes with high evolution rate from ancestral genome; its outstanding performance and stability take great improvement to framework GRAPPA on phylogenetic tree construction.

6.0.1 Contribution

The major contribution of this dissertation is it proposes a novel method to generate median genome by utilizing graph neural network and variation graph encoder. Genome median problem is NP-Hard and currently available methods on this problem take heuristic search strategy to find a sub-optimal solution. One significant disadvantage of these methods is their stability and time consuming. The learning model discussed in previous chapters takes a totally different computation framework to generate median genome directly based on gene connections, in which gene rearrangement patterns are abstracted out in forms of kernel parameters and target genome is generated from inner product of source genomes and kernel parameters. These novelty aspects of this method promises its performance and stability on median genome generation.

Other contributions in this dissertation have that it defines a procedure to generate data samples for neural network training. Before this strategy, there are no effective methods to obtain data samples for supervised training on median genome problem. All the median algorithms start from derived genomes and probe the structure of ancestral genome iteratively. In this strategy, data samples are constructed by starting from the assumed ancestral genome and filter out the ones meet the standard. At the same time, the simulation code of genome evolution is implemented out in forms of matrix computation which could be benefit from parallel computing and GPU computing. This makes the generation of large amount of training samples in short time becomes possible.

The algorithm of converting probability matrix into a genome sequence, which

discussed in section 4.2, is also a novelty in this dissertation. This algorithm bridges the gap between probability distribution and genome adjacency connection; utilizing graph neural network model to generate genome sequence becomes available.

6.0.2 Future Work

Besides the advantages and contributions of the graph neural network learning model, many aspects this model are still imperfect, there are many feature works could be attempted to improve model's performance on median genome generation and efficiency on phylogenetic tree construction.

Currently, the graph neural network learning model only takes relations between gene pairs in the form of graph adjacency matrix into consideration. Learning model convolution edge adjacency into node features and then decode the target graph according to these features. Some kinds of information in a 3-genome graph, such as the connection attributes, edge weights, gene features and so on, are not included in the convolution process. For example, connection attributes could mark out one edge comes out from which genome, edge weights could define which connection is more likely to break, gene features could include gene properties beside its position in a genome. If these kinds information are added into the learning model, it could be able to affect the model's performance.

The possible genome length could be processed by the learning model are limited by many aspects, which is an important weak point of this model to be improved in future. Since long genomes couldn't be used to train a learning model because of high requirement of computing resources and exceptionally long training time. Although a median genome could be generated from long genomes with a learning model trained with short genomes, some gene arrangement events that covers long distances in a genome couldn't be figured out and the quality of generated median genome will be cut down. In order to lift the limitation of genome length, graph learning model

that requires low computing resources should be developed. Another solution to this problem could be that generate some short genomes which simulate the long range genome rearrangements with discontinuous gene indexes. Or define a learning model which could prepossess genomes with different length into an identified size data structure to learn gene rearrangement patterns in long genome.

The algorithm used in this dissertation to translate probability matrix into genome sequence could generate a candidate median genome, however the converted genome sequences couldn't be authenticated to be the best one from the probability matrix. Further more, the computation complexity of this algorithm is $O(n^3 logn)$, it will be slow down when genome becomes longer. Future works could be focus on reducing the complexity of this algorithm and figuring out how to guarantee the generated median genome is the best from the probability matrix.

Although the graph neural network learning model could generate accurate median genome in a short time, the computation time of phylogenetic tree construction is still high because there are too many genome circular orders and tree topology structures need to be evaluated by GRAPPA framework. If it is possible to build up a model that could predict the connection strength of genome pair based on genome sequences, a graph clustering model could be defined to dynamically generate phylogenetic tree from genome sequences directly and doesn't need to predefined tree topology structure. Further research works are necessary to fulfil these models.

Bibliography

[1] Carl R Woese. "Interpreting the universal phylogenetic tree". In: *Proceedings of the National Academy of Sciences* 97.15 (2000), pp. 8392–8396.

[2] Paschalia Kapli, Ziheng Yang, and Maximilian J Telford. "Phylogenetic tree building in the genomic age". In: *Nature Reviews Genetics* 21.7 (2020), pp. 428–444.

[3] Haoyang Li et al. "Modern deep learning in bioinformatics". In: *Journal of molecular cell biology* 12.11 (2020), pp. 823–827.

[4] Jennifer L Gardy and Fiona SL Brinkman. "Methods for predicting bacterial protein subcellular localization". In: *Nature Reviews Microbiology* 4.10 (2006), pp. 741–751.

[5] Yinan Shen et al. "Critical evaluation of web-based prediction tools for human protein subcellular localization". In: *Briefings in bioinformatics* 21.5 (2020), pp. 1628–1640.

[6] Shuying Li, Dennis K Pearl, and Hani Doss. "Phylogenetic tree construction using Markov chain Monte Carlo". In: *Journal of the American statistical Association* 95.450 (2000), pp. 493–508.

[7] Hasan H Otu and Khalid Sayood. "A new sequence distance measure for phylogenetic tree construction". In: *Bioinformatics* 19.16 (2003), pp. 2122–2130.

[8] Jeffrey Rizzo and Eric C Rouchka. "Review of phylogenetic tree construction". In: *University of Louisville Bioinformatics Laboratory Technical Report Series* 1 (2007), pp. 1–7.

[9] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *AI Open* 1 (2020), pp. 57–81.

[10] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[11] Alex Fout et al. “Protein interface prediction using graph convolutional networks”. In: *Advances in neural information processing systems* 30 (2017).

[12] Yulei Yang and Dongsheng Li. “Nenn: Incorporate node and edge features in graph neural networks”. In: *Asian conference on machine learning*. PMLR. 2020, pp. 593–608.

[13] Zhitao Ying et al. “Hierarchical graph representation learning with differentiable pooling”. In: *Advances in neural information processing systems* 31 (2018).

[14] Joseph Felsenstein. “The number of evolutionary trees”. In: *Systematic zoology* 27.1 (1978), pp. 27–33.

[15] Naruya Saitou and Masatoshi Nei. “The neighbor-joining method: a new method for reconstructing phylogenetic trees.” In: *Molecular biology and evolution* 4.4 (1987), pp. 406–425.

[16] Robert R Sokal. “A statistical method for evaluating systematic relationships.” In: *Univ. Kansas, Sci. Bull.* 38 (1958), pp. 1409–1438.

[17] Walter M Fitch and Emanuel Margoliash. “Construction of phylogenetic trees: a method based on mutation distances as estimated from cytochrome c sequences is of general applicability.” In: *Science* 155.3760 (1967), pp. 279–284.

[18] Julie D Thompson, Toby J Gibson, and Des G Higgins. “Multiple sequence alignment using ClustalW and ClustalX”. In: *Current protocols in bioinformatics* 1 (2003), pp. 2–3.

[19] Stéphane Guindon and Olivier Gascuel. “A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood”. In: *Systematic biology* 52.5 (2003), pp. 696–704.

[20] Kirill Kryukov and Naruya Saitou. “MISHIMA-a new method for high speed multiple alignment of nucleotide sequences of bacterial genome scale data”. In: *BMC bioinformatics* 11.1 (2010), pp. 1–14.

[21] Manolo Gouy, Stéphane Guindon, and Olivier Gascuel. “SeaView version 4: a multiplatform graphical user interface for sequence alignment and phylogenetic tree building”. In: *Molecular biology and evolution* 27.2 (2010), pp. 221–224.

[22] Ziheng Yang et al. “PAML: a program package for phylogenetic analysis by maximum likelihood”. In: *Computer applications in the biosciences* 13.5 (1997), pp. 555–556.

[23] Stéphane Guindon et al. “New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0”. In: *Systematic biology* 59.3 (2010), pp. 307–321.

[24] Lam-Tung Nguyen et al. “IQ-TREE: a fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies”. In: *Molecular biology and evolution* 32.1 (2015), pp. 268–274.

[25] Morgan N Price, Paramvir S Dehal, and Adam P Arkin. “FastTree 2 – approximately maximum-likelihood trees for large alignments”. In: *PloS one* 5.3 (2010), e9490.

[26] Liang Liu, Lili Yu, and Scott V Edwards. “A maximum pseudo-likelihood approach for estimating species trees under the coalescent model”. In: *BMC evolutionary biology* 10.1 (2010), pp. 1–18.

[27] John P Huelsenbeck and Fredrik Ronquist. “MRBAYES: Bayesian inference of phylogenetic trees”. In: *Bioinformatics* 17.8 (2001), pp. 754–755.

[28] Sebastian Höhna et al. “RevBayes: Bayesian phylogenetic inference using graphical models and an interactive model-specification language”. In: *Systematic biology* 65.4 (2016), pp. 726–736.

[29] Remco Bouckaert et al. “BEAST 2.5: An advanced software platform for Bayesian evolutionary analysis”. In: *PLoS computational biology* 15.4 (2019), e1006650.

[30] Nicolas Lartillot, Thomas Lepage, and Samuel Blanquart. “PhyloBayes 3: a Bayesian software package for phylogenetic reconstruction and molecular dating”. In: *Bioinformatics* 25.17 (2009), pp. 2286–2288.

[31] HU Chen, NI Huang, and Zhirong Sun. “SubLoc: a server/client suite for protein subcellular location based on SOAP”. In: *Bioinformatics* 22.3 (2006), pp. 376–377.

[32] Yinan Shen, Jijun Tang, and Fei Guo. “Identification of protein subcellular localization via integrating evolutionary and physicochemical information into Chou’s general PseAAC”. In: *Journal of Theoretical Biology* 462 (2019), pp. 230–239.

[33] Yijie Ding, Jijun Tang, and Fei Guo. “Human protein subcellular localization identification via fuzzy model on Kernelized Neighborhood Representation”. In: *Applied Soft Computing* 96 (2020), p. 106596.

[34] Jianjun He, Hong Gu, and Wenqi Liu. “Imbalanced multi-modal multi-label learning for subcellular localization prediction of human proteins with both single and multiple sites”. In: *PloS one* 7.6 (2012), e37155.

[35] Leyi Wei et al. “mGOF-loc: A novel ensemble learning method for human protein subcellular localization prediction”. In: *Neurocomputing* 217 (2016), pp. 73–82.

[36] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[37] Yijie Ding, Jijun Tang, and Fei Guo. “Protein crystallization identification via fuzzy model on linear neighborhood representation”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2019).

[38] Nicholas G Polson and Steven L Scott. “Data augmentation for support vector machines”. In: *Bayesian Analysis* 6.1 (2011), pp. 1–23.

[39] Shibiao Wan, Man-Wai Mak, and Sun-Yuan Kung. “mGOASVM: Multi-label protein subcellular localization based on gene ontology and support vector machines”. In: *BMC bioinformatics* 13.1 (2012), pp. 1–16.

[40] Chris Hans. “Bayesian lasso regression”. In: *Biometrika* 96.4 (2009), pp. 835–845.

[41] Hong-Bin Shen and Kuo-Chen Chou. “Hum-mPLoc: an ensemble classifier for large-scale human protein subcellular location prediction by incorporating samples with multiple sites”. In: *Biochemical and biophysical research communications* 355.4 (2007), pp. 1006–1011.

[42] Hong-Bin Shen and Kuo-Chen Chou. “A top-down approach to enhance the power of predicting human protein subcellular localization: Hum-mPLoc 2.0”. In: *Analytical biochemistry* 394.2 (2009), pp. 269–274.

[43] Kuo-Chen Chou, Zhi-Cheng Wu, and Xuan Xiao. “iLoc-Hum: using the accumulation label scale to predict subcellular locations of human proteins with both single and multiple sites”. In: *Molecular Biosystems* 8.2 (2012), pp. 629–641.

[44] D Sundararajan. *Discrete wavelet transform: a signal processing approach*. John Wiley & Sons, 2016.

[45] Kuo-Chen Chou. “Prediction of protein cellular attributes using pseudo-amino acid composition”. In: *Proteins: Structure, Function, and Bioinformatics* 43.3 (2001), pp. 246–255.

[46] Gaofeng Pan et al. “A novel computational method for detecting DNA methylation sites with DNA sequence information and physicochemical properties”. In: *International journal of molecular sciences* 19.2 (2018), p. 511.

[47] Ronan Collobert and Jason Weston. “A unified architecture for natural language processing: Deep neural networks with multitask learning”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 160–167.

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[49] Li Xu et al. “Deep convolutional neural network for image deconvolution”. In: *Advances in neural information processing systems*. 2014, pp. 1790–1798.

[50] Xiaofan Lin, Cong Zhao, and Wei Pan. “Towards accurate binary convolutional neural network”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 345–353.

[51] Jian Zhou and Olga G Troyanskaya. “Predicting effects of noncoding variants with deep learning–based sequence model”. In: *Nature methods* 12.10 (2015), pp. 931–934.

[52] Christof Angermueller et al. “DeepCpG: accurate prediction of single-cell DNA methylation states using deep learning”. In: *Genome biology* 18.1 (2017), pp. 1–13.

[53] Tanlin Sun et al. “Sequence-based prediction of protein protein interaction using a deep-learning algorithm”. In: *BMC bioinformatics* 18.1 (2017), pp. 1–8.

[54] Yan Zhang et al. “Enhancing Hi-C data resolution with deep convolutional neural network HiCPlus”. In: *Nature communications* 9.1 (2018), pp. 1–9.

[55] Kevin P Murphy. *Machine learning: a probabilistic perspective*. Cambridge, MA: MIT press, 2012.

[56] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. California: O’Reilly Media, 2019.

[57] Ilya Sutskever et al. “On the importance of initialization and momentum in deep learning”. In: *International conference on machine learning*. PMLR. 2013, pp. 1139–1147.

[58] Boris T Polyak and Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging". In: *SIAM journal on control and optimization* 30.4 (1992), pp. 838–855.

[59] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[60] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[61] Fangyu Zou et al. "A sufficient condition for convergences of adam and rmsprop". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11127–11135.

[62] Matthew D Zeiler. "Adadelta: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).

[63] José Juan Almagro Armenteros et al. "DeepLoc: prediction of protein subcellular localization using deep learning". In: *Bioinformatics* 33.21 (2017), pp. 3387–3395.

[64] Wei Long, Yang Yang, and Hong-Bin Shen. "ImPLoc: a multi-instance deep learning model for the prediction of protein subcellular localization based on immunohistochemistry images". In: *Bioinformatics* 36.7 (2020), pp. 2244–2250.

[65] Yu-Tao Shao et al. "pLoc_Deep-mHum: Predict Subcellular Localization of Human Proteins by Deep Learning". In: *Natural Science* 12.7 (2020), pp. 526–551.

[66] Yutao Shao, Kuo-Chen Chou, et al. "pLoc_Deep-mEuk: Predict Subcellular Localization of Eukaryotic Proteins by Deep Learning". In: *Natural Science* 12.06 (2020), p. 400.

[67] Yutao Shao and Kuo-Chen Chou. "pLoc_Deep-mVirus: A CNN Model for Predicting Subcellular Localization of Virus Proteins by Deep Learning". In: *Natural Science* 12.6 (2020), pp. 388–399.

[68] Leyi Wei et al. "Prediction of human protein subcellular localization using deep learning". In: *Journal of Parallel and Distributed Computing* 117 (2018), pp. 212–217.

[69] Tanel Pärnamaa and Leopold Parts. "Accurate classification of protein subcellular localization from high-throughput microscopy images using deep learning". In: *G3: Genes, Genomes, Genetics* 7.5 (2017), pp. 1385–1392.

[70] Shraddha Ravindra Masurkar and Priti P Rege. "Human Protein Subcellular Localization using Convolutional Neural Network as Feature Extractor". In: *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE. 2019, pp. 1–7.

[71] Zhen Cao et al. "The lncLocator: a subcellular localization predictor for long non-coding RNAs based on a stacked ensemble classifier". In: *Bioinformatics* 34.13 (2018), pp. 2185–2194.

[72] Zichao Yan, Eric Lécuyer, and Mathieu Blanchette. "Prediction of mRNA subcellular localization using deep recurrent neural networks". In: *Bioinformatics* 35.14 (2019), pp. i333–i342.

[73] Brian L Gudenas and Liangjiang Wang. "Prediction of LncRNA subcellular localization with deep learning from sequence features". In: *Scientific reports* 8.1 (2018), pp. 1–10.

[74] Richard Friedberg, Aaron E Darling, and Sophia Yancopoulos. "Genome rearrangement by the double cut and join operation". In: *Bioinformatics* (2008), pp. 385–416.

[75] Franco Scarselli et al. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[76] Yongji Wu et al. "Graph convolutional networks with markov random field reasoning for social spammer detection". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 01. 2020, pp. 1054–1061.

[77] Alvaro Sanchez-Gonzalez et al. "Graph networks as learnable physics engines for inference and control". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4470–4479.

[78] Takuo Hamaguchi et al. "Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach". In: *arXiv preprint arXiv:1706.05674* (2017).

[79] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[80] Yaguang Li et al. "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting". In: *arXiv preprint arXiv:1707.01926* (2017).

[81] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv: 1710.10903* (2017).

[82] Ruoyu Li et al. “Adaptive graph convolutional neural networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[83] Chenyi Zhuang and Qiang Ma. “Dual graph convolutional networks for graph-based semi-supervised classification”. In: *Proceedings of the 2018 World Wide Web Conference*. 2018, pp. 499–508.

[84] James Atwood and Don Towsley. “Diffusion-convolutional neural networks”. In: *Advances in neural information processing systems* 29 (2016).

[85] Justin Gilmer et al. “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[86] Yujia Li et al. “Gated graph sequence neural networks”. In: *arXiv preprint arXiv:1511.05493* (2015).

[87] Zhijun Liao et al. “Predicting subcellular location of protein with evolution information and sequence-based deep learning”. In: *BMC bioinformatics* 22.10 (2021), pp. 1–23.

[88] Gary D Stormo et al. “Use of the ‘Perceptron’ algorithm to distinguish translational initiation sites in E. coli”. In: *Nucleic acids research* 10.9 (1982), pp. 2997–3011.

[89] Gary D Stormo. “DNA binding sites: representation and discovery”. In: *Bioinformatics* 16.1 (2000), pp. 16–23.

[90] Jorja G Henikoff and Steven Henikoff. “Using substitution probabilities to improve position-specific scoring matrices”. In: *Bioinformatics* 12.2 (1996), pp. 135–143.

[91] Medha Bhagwat and L Aravind. “Psi-blast tutorial”. In: *Comparative genomics*. Springer, 2007, pp. 177–186.

[92] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder - decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[93] Mike Schuster and Kuldip K Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[94] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. "Hybrid speech recognition with deep bidirectional LSTM". In: *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE. 2013, pp. 273–278.

[95] Alan L Hodgkin and Andrew F Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4 (1952), p. 500.

[96] Yann A LeCun et al. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Berlin, Heidelberg: Springer, 2012, pp. 9–48.

[97] Richard HR Hahnloser et al. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". In: *Nature* 405.6789 (2000), pp. 947–951.

[98] Jun Han and Claudio Moraga. "The influence of the sigmoid function parameters on the speed of backpropagation learning". In: *International Workshop on Artificial Neural Networks*. Springer. 1995, pp. 195–201.

[99] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[100] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[101] Hang Zhou, Yang Yang, and Hong-Bin Shen. "Hum-mPLoc 3.0: prediction enhancement of human protein subcellular localization through modeling the hidden correlations of gene ontology and functional domain features". In: *Bioinformatics* 33.6 (2017), pp. 843–853.

[102] David Martin Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: (2011).

[103] Sébastien Rey, Jennifer L Gardy, and Fiona SL Brinkman. "Assessing the precision of high-throughput computational and laboratory approaches for the genome-wide identification of protein subcellular localization in bacteria". In: *BMC genomics* 6.1 (2005), p. 162.

[104] Tom Fawcett. "An introduction to ROC analysis". In: *Pattern recognition letters* 27.8 (2006), pp. 861–874.

[105] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. "Mining multilabel data". In: *Data mining and knowledge discovery handbook*. Berlin: Springer, 2009, pp. 667–685.

[106] Min-Ling Zhang and Zhi-Hua Zhou. "A review on multi-label learning algorithms". In: *IEEE transactions on knowledge and data engineering* 26.8 (2013), pp. 1819–1837.

[107] Bernard ME Moret et al. "A new implementation and detailed study of breakpoint analysis". In: *Biocomputing 2001*. World Scientific, 2000, pp. 583–594.

[108] Bernard ME Moret et al. "Steps toward accurate reconstructions of phylogenies from gene-order data". In: *Journal of Computer and System Sciences* 65.3 (2002), pp. 508–525.

[109] Bernard ME Moret et al. "Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data". In: *International Workshop on Algorithms in Bioinformatics*. Springer. 2002, pp. 521–536.

[110] Andrew Wei Xu. "A fast and exact algorithm for the median of three problem: A graph decomposition approach". In: *Journal of Computational Biology* 16.10 (2009), pp. 1369–1381.

[111] Nan Gao, Ning Yang, and Jijun Tang. "Ancestral genome inference using a genetic algorithm approach". In: *PloS one* 8.5 (2013), e62156.

[112] Ruofan Xia et al. "A median solver and phylogenetic inference based on double cut and join sorting". In: *Journal of Computational Biology* 25.3 (2018), pp. 302–312.

[113] Zhaojuan Zhang et al. "Achieving large and distant ancestral genome inference by using an improved discrete quantum-behaved particle swarm optimization algorithm". In: *BMC bioinformatics* 21.1 (2020), pp. 1–30.

[114] Jijun Tang. "Ancestral Genome Reconstruction". In: *Bioinformatics and Phylogenetics*. Springer, 2019, pp. 193–203.