

Integrating Biological Sequence Analysis and String Processing for Motif Understanding and Neural Network Modeling

Masahiro Ohtomo

Supervisor: Professor Tatsuya Akutsu

Kyoto University

GRADUATE SCHOOL OF INFORMATICS

A dissertation presented for the degree of Doctor of Philosophy

March, 2026

Abstract

Understanding the complex genetic and protein information encoded in living organisms is essential for elucidating the fundamental mechanisms of life. Recent advances in measurement technologies and computational capabilities have resulted in an unprecedented accumulation of biological data, including sequence data that represent genes and proteins as strings and 3D structural data describing the atomic coordinates of biomolecules. Efficient and comprehensive analysis of these biological data is crucial for gaining insights into molecular evolution, biological function, and ultimately for contributing to drug discovery and medical sciences. A variety of computational approaches have been developed for sequence and structural analysis. However, for motifs, which are distinctive sequence patterns closely associated with biological function, the integration of knowledge across gene sequences, protein sequences, and 3D structures remains an important challenge. In addition, methods for expressing fundamental string comparisons that are essential for biological sequence analysis as neural networks have recently been proposed. If such approaches can be realized, it is expected that input sequence estimation, which has been difficult to achieve using conventional methods, could become feasible as a learning task.

In the first part of this research, we constructed a protein–gene motif dictionary that integrates motif-related protein sequence data, corresponding gene sequence data, and 3D structural information. This dictionary enables one-stop retrieval of both sequence and structural information for each motif, allowing biological data to be accessed and compared in a single framework. As a result, by focusing on co-occurring motifs within MMP proteins and comparing their sequence distances with their centroid distances, we obtained new insights into motif distance relationships. In the second part of this research, we proposed a method that identifies motif sites directly from gene sequences by leveraging the genetic information stored in the motif dictionary. To uniquely represent multiple gene sequences, we introduced a codon reduced representation and extended conventional sequence alignment algorithms. This approach revealed that even for the same EF-hand motif, the genomic positions and sequence compositions can differ substantially across species, providing new perspectives on motif evolution and highlighting the importance of comparative sequence analysis at the genomic level. In the third part of this study, we developed a neural network models that use only the ReLU activation function and are capable of computing string distances, including the Hamming distance and the Levenshtein distance. By designing a neural network modules for character matching and minimum-value operations, we constructed architectures that compute these distances with computational complexity comparable to that of conventional algorithms. Our proposed model not only reproduces conventional string computations but also suggests its effectiveness in addressing input estimation problems that have been difficult to solve using traditional methods.

Together, these studies advance computational approaches for integrated motif analysis and neural formulations of string-processing tasks. Collectively, the three research topics illustrate how string-oriented methodologies can bridge biological sequence analysis and neural computation, yielding both biological findings and innovative algorithmic techniques.

Contents

1	Introduction	3
1.1	Background	3
1.2	Challenges	6
1.3	Contribution	7
1.4	Thesis outline	7
2	Preliminaries	9
2.1	Biological basis	9
2.2	Motif	13
2.3	Biological databases	14
2.4	Sequence alignment	17
2.5	Neural networks	18
3	Development of a Protein-Gene Motif Dictionary System for One-Stop Motif Analysis	19
3.1	Background	19
3.2	Materials and methods	20
3.2.1	Overview of constructing the protein-gene motif dictionary	20
3.2.2	PROSITE Motif	22
3.2.3	Mapping of protein sequences and gene sequences	22
3.2.4	Extracting the protein-gene sequence motifs	23
3.2.5	Extraction of 3D structural motif	24
3.2.6	The protein-gene motif dictionary system using the MongoDB	26
3.3	Results and discussion	27
3.3.1	Constructing the protein-gene motif dictionary system for human	27
3.3.2	Constructing the protein-gene motif dictionary system for other species	28
3.3.3	Implementing the management system for the protein-gene motif dictionary system	29
3.3.4	3D structural feature analysis of co-occurrence motifs	29
3.4	Concluding Remarks	33
4	Development of the Dynamic Programming (DP)-based Functional Site Estimation System Using the Motif Codon Reduced Representation	35
4.1	Background	35
4.2	Materials and methods	37
4.2.1	Codon Reduced Representation	37
4.2.2	Motif codon reduced representation	39
4.2.3	DP-based sequence alignment for functional site estimation	40
4.2.4	Accommodating multiple candidate functional sites	43
4.3	Results and discussion	43
4.3.1	Protein-coding region estimation experiments	43
4.3.2	Functional site estimation using codon reduced representation for EF-hand motif	45

4.3.3	Functional site estimation using codon reduced representation for EF-hand motif of the other species	51
4.4	Concluding Remarks	54
5	Computing Hamming Distance and Levenshtein Distance Using ReLU Neural Networks	58
5.1	Background	58
5.2	Method	59
5.2.1	Hamming distance and Levenshtein distance	59
5.2.2	ReLU neural network	60
5.2.3	ReLU neural network for calculating Hamming distance	61
5.2.4	Calculating Levenshtein distance using DP	63
5.2.5	ReLU neural network for calculating Levenshtein distance	63
5.2.6	ReLU neural network for learning	66
5.3	Reducing the number of layers	67
5.3.1	General case	68
5.3.2	Case with a limited number of insertions and deletions	71
5.4	Computational experiments	72
5.4.1	Computational time and construction time of the network	73
5.4.2	Learning of the ReLU neural network	75
5.4.3	Model reuse through zero-padding for Hamming distance learning . .	80
5.4.4	Hyperparameter tuning	81
5.4.5	Comparison with other supervised learning models	84
5.5	Concluding Remarks	86
6	Conclusion and Future Work	90
6.1	Conclusion	90
6.2	Future work	91

1 Introduction

1.1 Background

In the life sciences, genes and proteins constitute the fundamental components that determine the structure and function of living organisms. These molecules interact with one another in diverse and highly precise ways within the cell, and understanding the principles that govern their functions and interactions is essential for elucidating biological phenomena at the molecular level. Advances in molecular-level understanding provide a critical foundation for fields such as medicine and drug discovery, contributing to the elucidation of disease mechanisms and the identification of therapeutic targets [1, 2, 3, 4]. These studies focus on motifs or domains, which are characteristic structure closely related to biological functions. By investigating mutations and activity-related information in such functional sites, these studies have made significant contributions to the fields of medicine and drug discovery.

The completion of the Human Genome Project in 2003 made the full sequence of the human genome publicly available [5]. Furthermore, the ENCODE project was started to achieve the complete functional annotation of the human genome [6]. In addition, the modENCODE project was also started to achieve the complete functional annotation of the model species such as nematoda and fly [7, 8]. Determining biological sequences from various species including human, and comprehensively annotating their biological functions remain an important challenge. In addition to analyzing individual species, comparative analysis across diverse species is also an important research topic for understanding biological evolution and diversity.

Development of next-generation sequencing (NGS) technologies and improvements in computational power have rapidly increased both sequence determination and analysis. Extracting biologically meaningful information efficiently required computational techniques such as string matching and statistical models. For example, traditional string matching approaches include regular expression based matching [9]. Probabilistic methods include hidden Markov models (HMM) and the expectation-maximization (EM) algorithms [10, 11]. An HMM represents state transitions between characters probabilistically. The EM algorithm optimizes model parameters by iterating two optimization steps between the E-step and the M-step. This necessity extends not only to genome sequences but also to gene and protein sequences,

which are essential components of biological information. Various computational methods and models have been proposed to extract motif sites from biological sequences. The PROSITE is a well-known knowledge base that stores protein sequence motif information [12]. The motif information is defined using two types of patterns, regular expression and weight matrix. The weight matrices describe the occurrence probabilities of each amino acid in the motif region in matrix form. By performing string searches using these representations on known or unknown sequences, motifs can be extracted. To enable more flexible motif extraction, statistical methods are used such as HMM or EM algorithms [13, 14]. Furthermore, local alignment based on dynamic programming (DP) has been proposed as a method for estimating motif sites [15]. Given a sequence and a motif sequence, local alignment can identify the position in the sequence where the motif best matches. In contrast, global alignment is used as a method for computing the overall similarity between two sequences [16]. Furthermore, computational methods have also been proposed to identify common regions by analyzing two or more sequences simultaneously [17].

With the growth of large-scale datasets, increasingly complex computational models and machine learning techniques have been actively adopted to extract motifs from extensive sequence collections. In the field of bioinformatics, deep learning has been shown to achieve higher accuracy than previous methods across a wide range of problems. For motif extraction, a convolutional neural network (CNN) models and the Transformer approaches have also been proposed [18, 19, 20, 21]. In particular, the Transformers achieve very high accuracy in string processing tasks such as translation and natural language generation through attention mechanism. AlphaFold has demonstrated that deep learning can predict protein 3D structures using attention mechanisms [22].

Biological data, including biological sequence data and 3D structural data, have increased rapidly in recent years due to advances in experimental instruments and structure-determination technologies. Accordingly, the development of platforms for appropriately organizing and managing the vast amount of experimentally determined or computationally analyzed data has become an important research topic. Without systematic data organization, efficient analysis and the reuse of accumulated knowledge are difficult [23]. Biological data are highly diverse in nature. In addition to biological sequence

data, gene variation data, metabolic pathway information, molecular interaction data, and gene expression data [24]. To handle such heterogeneous data in an integrated manner, it is essential to store them in standardized formats and to organize them with added semantic information. Biological databases are classified into primary and secondary databases. Primary databases archive raw data obtained from experiments or computational analyses; representative examples include GenBank operated by the NCBI [25], the European Nucleotide Archive (ENA) managed by EMBL-EBI [26], and the DNA Data Bank of Japan (DDBJ) [27]. These databases are internationally coordinated and share nucleotide sequence data through collaborative frameworks. In addition, the Protein Data Bank (PDB) serves as a database for storing protein 3D structures [28]. Also, PDB contains structural information for other biomolecules such as nucleic acids and glycans. In contrast, secondary databases provide data enriched with functional annotations as well as structural and evolutionary information derived from primary data. Representative examples include protein functional annotations provided by UniProt [29], motif and domain information of protein curated by the PROSITE database [12] and the Pfam database [30]. Furthermore, the Kyoto Encyclopedia of Gene and Genome (KEGG) [31] is widely recognized as an integrated database that consolidates biological information, such as metabolic and signal transduction pathway data. These databases serve as essential information platforms that enable efficient sequence analysis and functional site analysis. Nevertheless, most existing approaches remain confined to information extraction and analysis at individual molecular levels, such as the genome, gene, or protein sequence level. Therefore, researchers need to collect data from multiple databases to perform more detailed analyses.

As described above, biological data continue to increase in both scale and diversity. Therefore, the development of computational methods is essential. Although sequence analysis relies on general string-processing techniques, it also requires domain-specific biological knowledge to interpret functional sites such as motifs. However, motif-related information is fragmented across individual databases and molecular levels, which limits integrative analysis and systematic reuse of experimentally obtained findings. These limitations motivate the development of knowledge bases that unify motif-related information.

1.2 Challenges

Despite advances in experimental technologies and computational methods, several challenges remain in biological sequence analysis and the development of biological databases. Although numerous biological databases exist, there are relatively few databases that unify functional site information across different molecular levels, such as genes and proteins. The advent of deep learning has also made substantial contributions to research on string processing tasks. However, many deep learning models do not explicitly represent exact string operations, resulting in limited interpretability.

Challenge 1: cross level motif knowledge integration Many studies focus on analyses at the molecular level of either genes or proteins independently. In addition, existing approaches often focus solely on sequence data or 3D structural data. Although motifs have been studied for many years, there have been few attempts to analyze correlations between sequences and structures, or between genes and proteins. This is because information related to motifs is registered across individual databases. As a result, researchers must collect and integrate information by traversing multiple databases for analysis. The development of knowledge bases and analytical methods aimed at such integrative analyses would contribute to a deeper understanding of molecular evolution and pathological mechanisms.

Challenge 2: A neural network model for calculating string distance

The analysis of biological sequences requires domain-specific biological knowledge. However, its foundation lies in string-processing algorithms developed in computer science. Developing such algorithms and considering their application to bioinformatics is therefore highly important from an informatics perspective as well. In addition to classical string-search algorithms, deep learning-based approaches have been proposed in recent years. Motif extraction and feature analysis from sequences, as discussed above, are also forms of sequence and string analysis. Consequently, the development of fundamental algorithms for handling strings is considered indispensable for elucidating the principles of life.

In this research, we focus on the following two challenges:

1. developing and analyzing a motif database that integrates gene and protein information

2. developing a neural network model using only ReLU function that computes traditional string distances.

1.3 Contribution

This thesis addresses the two challenges through three main contributions.

First, we describe our contributions to the first challenge. As the first contribution, we constructed a protein–gene motif dictionary that integrates motif related protein sequence data, corresponding gene sequence data, and 3D structural information. This dictionary system enables one-stop search of both sequence and structural information for each motif, allowing biological data to be accessed and compared in the dictionary system. Using this dictionary to analyze motif co-occurrence relationships in the same protein, we found that the 3D structures are conserved at a certain the centroid distance regardless of the sequence distance between motifs. As the second contribution, we proposed a method that identifies motif sites directly from gene sequences by leveraging the genetic information of protein motif stored in the motif dictionary. To estimate protein motif sites in gene sequence, we introduced a codon reduced representation and extended conventional sequence alignment algorithms. As an application, we estimated EF-hand motif regions in the gene sequences of Troponin C type 1 (TNNC1) and Hippocalcin (HPCA), and found that the intron insertion point within the motif region is conserved across seven species.

Next, we describe our contributions to the second challenge. As the third contribution, we proposed a neural network models that use only the ReLU activation function to calculate string distances, such as the Hamming distance and the Levenshtein distance. For exactly computing these string distances, we proposed a network for calculating character matches and a network for selecting the minimum of three inputs. Furthermore, our proposed model not only reproduces conventional string computations but also suggests its effectiveness in addressing input estimation problems that have been difficult to solve using traditional methods.

1.4 Thesis outline

Chapter 2 summarizes the essential preliminaries required for this study. This chapter includes biological basis information, sequence alignment, and the neural network notation used in later chapters. Chapters 3 and 4 address

Challenge 1 (integrative motif knowledge base and analysis). Chapter 3 discusses the construction of a protein-gene motif dictionary integrating protein sequence motif, corresponding gene sequences, and 3D structural information. Chapter 4 discusses a motif codon reduced representation and extended alignment algorithms. Chapter 5 addresses Challenge 2 (neural modeling of explicit string operations). This chapter discusses developing a neural network using only ReLU activation function that compute string distances, including the Hamming and Levenshtein distances. Finally, Chapter 6 discusses conclusions and future directions.

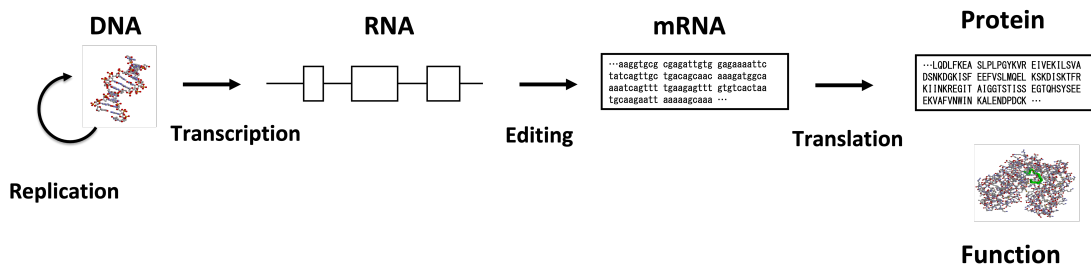


Figure 1: The central dogma of molecular biology

2 Preliminaries

2.1 Biological basis

Nucleic acids play a central role in biological systems, serving as the fundamental molecules responsible for storing and transmitting genetic information. There are two major types of nucleic acids, DNA (deoxyribonucleic acid) and RNA (ribonucleic acid), which have different biological functions. In eukaryotic cells, DNA is located primarily in the nucleus and serves as the long-term repository of hereditary information. In contrast, RNA is produced by transcription and is found in both the nucleus and the cytoplasm, where it contributes to protein synthesis based on the information encoded in DNA. In eukaryotes, DNA is transcribed into pre-mRNA, which is processed into mRNA and then translated into protein. Figure 1 shows the flow of genetic information, which is known as the central dogma of molecular biology.

Nucleobases are classified into two groups according to their ring structures: purines and pyrimidines. Adenine and Guanine are purine bases, and they appear in both DNA and RNA. Cytosine is a pyrimidine base common to both molecules, whereas Thymine appears only in DNA and Uracil only in RNA. Figure 2 shows the structures of each bases. The one-letter representations of these bases are defined by the International Union of Pure and Applied Chemistry (IUPAC). Adenine is represented as “A”, guanine as “G”, cytosine as “C”, thymine as “T”, and uracil as “U”. The backbone structure of DNA and RNA consists of nucleotides linked through phosphodiester bonds between the 5'-phosphate and 3'-hydroxyl groups of deoxyribose or ribose, forming linear polymers with inherent directionality. DNA adopts a double-helical structure composed of two complementary strands. Figure 3 shows image of phosphodiester bonds. The sequences of DNA and RNA can be represented as strings over an alphabet of four bases, commonly referred

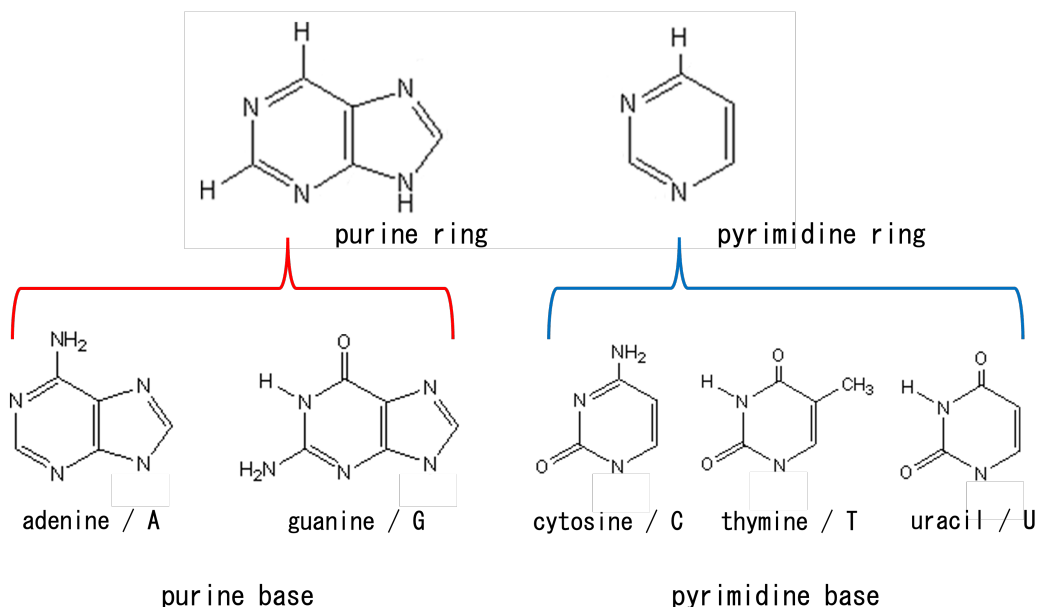


Figure 2: Structure and classification of bases

to as nucleotide sequences. These sequences are written from the 5' end on the left to the 3' end on the right.

Transcription of protein coding regions begins at specific promoter sequences, followed by the removal of non-coding regions (introns) from the transcribed pre-mRNA through splicing. After intron removal, the remaining exon regions constitute the protein coding portions of the gene. Once splicing and other post-transcriptional modifications are completed, the RNA becomes a mature transcript that is competent for translation, which proceeds in the 5'-to-3' direction. The region that is translated into protein is referred to as the coding sequence (CDS), whereas segments that are transcribed but not translated are known as untranslated regions (UTRs). Figure 4 shows the splicing process.

Proteins are polymers composed of chains of amino acids, in which the carboxyl and amino groups of twenty genetically encoded amino acids are linked by peptide bonds. A single chains is called a polypeptide, range in length from a few residues to several hundred residues and possess inherent directionality. Each sequence starts at the N-terminus and ends at the C-terminus. Figure 5 shows image of peptide bonds.

Proteins play highly diverse and essential roles in living organisms. Examples include enzymes that catalyze biochemical reactions and transcription factors that bind to DNA to regulate gene expression. Each of the twenty

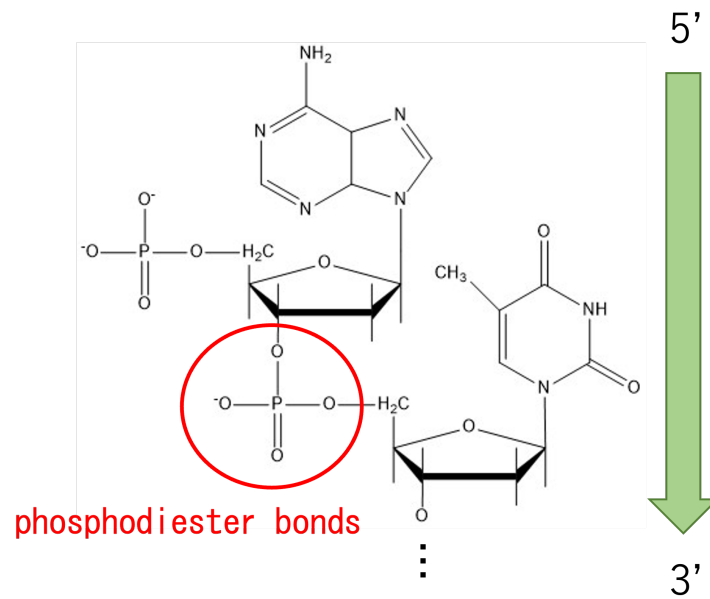


Figure 3: Image of phosphodiester bonds

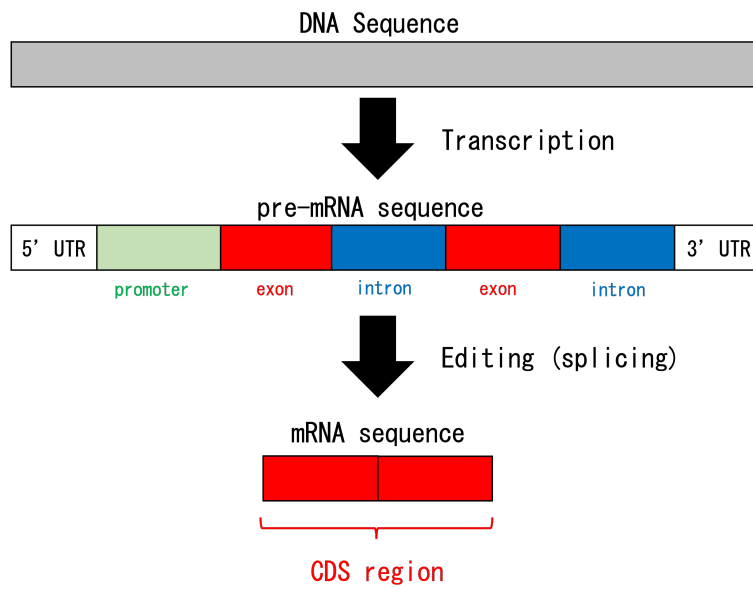


Figure 4: Splicing process for eukaryotes

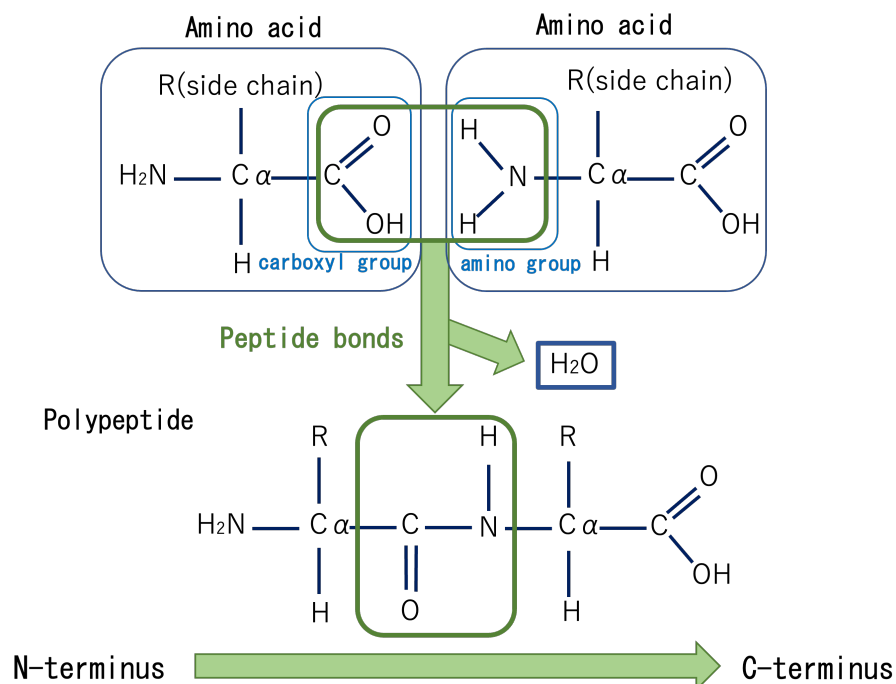


Figure 5: Image of peptide bonds

amino acids is represented by both a one-letter and a three-letter code. Table 1 summarizes the one-letter codes for amino acids. Proteins can be represented as a sequence of amino acid symbols. Protein sequences are written from the N-terminus on the left to the C-terminus on the right.

Table 1: List of Amino Acids

Amino Acid	one-letter codes	Amino Acid	one-letter codes
Alanin	A	Methionine	M
Cysteine	C	Asparagine	N
Aspartic Acid	D	Proline	P
Glutamic Acid	E	Glutamine	Q
Phenylalanine	F	Arginine	R
Glycine	G	Serine	S
Histidine	H	Threonine	T
Isoleucine	I	Valine	V
Lysine	K	Tryptophan	W
Leucine	L	Tyrosine	Y

The 3D structure of a protein is largely determined by its amino-acid sequence and, under physiological conditions, typically adopts a conformation near a free-energy minimum. Protein function is closely associated with its spatial structure. For example, interactions with specific molecules such as calcium or zinc ions, as well as catalytic activities in enzymes, are largely gov-

erned by the shape and chemical composition of surface features or internal pockets. Thus, analyzing structural characteristics is essential for predicting the structure and function of unknown proteins. Protein structure can be described using four hierarchical levels: primary, secondary, tertiary, and quaternary structures. Primary structure corresponds to the amino acid sequence itself. Secondary structure refers to regular local conformations, such as α -helices and β -sheets, stabilized mainly by hydrogen bonds in the peptide backbone. An α -helix is a right-handed helical conformation of the polypeptide backbone. A β -sheet is formed by multiple extended β -strands aligned in parallel or antiparallel orientations. Tertiary structure represents the overall 3D shape formed by the folding of a single polypeptide chain. Quaternary structure refers to complexes composed of multiple folded polypeptide subunits. Each subunit is referred to as a chain. Complexes composed of two or three chains are termed dimers and trimers, respectively. Assemblies consisting of identical chains are called homo-multimers, whereas those composed of different chains are referred to as hetero-multimers. Many proteins in living organisms function as such complexes, and their biological activity often depends on these higher-order assemblies.

2.2 Motif

It is well known that an amino acid sequence of a protein is closely related to its structure and function. This is especially true for particular structural features called motifs, and they are considered to be well-conserved sites in the genome sequences. Also, multiple motifs may be conserved within a sequence. 3D sequence motifs are referred to as structural motifs or 3D motifs. Sequence motifs are typically well conserved through evolutionary processes, making them recognizable as regions with higher conservation compared with surrounding amino acids. Structural motifs refer to conserved 3D features within protein structures that correspond to functionally important regions.

For example, the EF-hand motif is a well-known motif found in calcium-binding sites of proteins [32]. The EF-hand motif is composed of a combination of two α -helices and a single loop site, and its function is exhibited through calcium binding in the loop region. Figure 6 shows an image of the EF-hand motif.

Another example, the zinc finger C2H2 motif plays a central role in DNA binding. This motif consists of two antiparallel β -sheets and one α -helix [33].

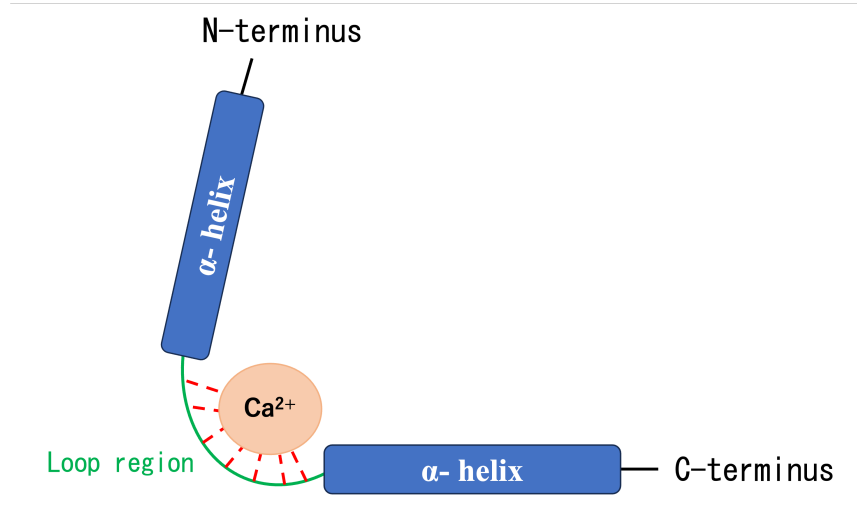


Figure 6: Image of the EF-hand motif

In this motif, two cysteine residues located within the β -sheet region and two histidine residues within the α -helix coordinate a zinc ion, enabling the motif to interact with DNA. These motifs are often composed of combinations of secondary structural elements and are therefore classified as supersecondary structures, positioned between secondary and tertiary structures in the hierarchical organization of protein architecture.

At the gene level, several conserved sequence motifs have been identified. In eukaryotes, a well-known example is the TATA box, a motif that determines the transcription start site and the direction of transcription [34]. The TATA box is recognized by TATA-binding proteins, which subsequently recruit other transcription factors to initiate transcription. Other examples include the CAAT box and GC box, which influence the frequency of transcription. These conserved sequence features are all located near promoter regions [34]. In contrast, sequence motifs within coding sequences (CDS regions) remain far less well understood, and many aspects of their functional roles are still unresolved.

2.3 Biological databases

As described in Section 1.1, biological data are highly diverse, and a wide variety of databases are maintained. In this subsection, we describe in detail the two databases that manage the data used in this study. Furthermore, we describe the regular expression patterns of the PROSITE that are used to define motifs in this study.

```
>gi|4507615|ref|NP_003271.1|TroponinC
MDDIYKAAVEQLTEEQNEFKAAFDIFVLGAEDGCISTKELGKVMRMLGQNPTPEELQEMIDEVDEDGSGTVDFDEFLVMMVRCMKDDSKGKSEEELSDLFR
MFDKNADGYIDLDELKIMLQATGETITEDDIEELMKDGDKNNDGRIDYDEFLEFMKGVE
```

Figure 7: Example of FASTA format in RefSeq

DDBJ, ENA, and GenBank database contains a large number of experimentally determined biological sequence data. However, despite their extensive coverage, these databases contain a substantial number of redundant, biased, or low-quality entries. To address these issues, the NCBI Reference Sequence (RefSeq) database was established as a secondary database that reconstructs curated, non-redundant reference sequences by manually selecting [35]. Protein and DNA sequence data are standardized using a representation format known as the FASTA format [36]. The FASTA format consists of a header line beginning with the character “>”, followed by one or more lines of sequence data. The header line typically contains information such as the protein name corresponding to the sequence, the database identifier, and the organism of origin. The sequence is represented using one-letter codes for nucleotides or amino acids. For instance, Figure 7 shows a protein sequence of the refseq. The header includes information such as the protein name and the identifier assigned in the corresponding database. RefSeq identifiers (RefSeq IDs) include an accession prefix: two leading characters that indicate the sequence category. For instance, prefixes such as “NP_” for proteins, “NM_” for mRNA, “NG_” for incomplete genome sequences, and “NC_” for complete genome sequence are used. The data are categorized by organisms, such as mammalian, protozoa, and others. Also, the FASTA format allows multiple sequences to be included within a single file, a representation commonly referred to as the multi-FASTA format.

PDB archive is managed by the Worldwide Protein Data Bank (wwPDB), an international consortium whose core members include RCSB PDB (USA), PDBe (Europe), EMDB (Europe), PDBj (Japan), and BMRB (USA). As of December 2025, more than 240,000 experimentally determined protein structures have been deposited in PDB [37], obtained through various techniques such as X-ray crystallography and nuclear magnetic resonance (NMR) spectroscopy. As an example, a part of a PDB file is shown in Figure 8. Each PDB entry contains extensive information, including the protein name, identifier (PDB ID), experimental method, sequence data, and 3D atomic coordinates. Each PDB ID is a unique four-character identifier. The first character is a

Record name (1~6)	Index (7~11)	Type (13~16)	Amino acid (18~20)	chainID (22)	Residue number (23~26)	X-coordinate (31~38)	Y-coordinate (39~46)	Z-coordinate (47~54)	occupancy (55~60)	tempFactor (61~66)	Charge (78~79)
ATOM	368	N	PRO	A	90	49.705	-24.087	65.474	1.00	31.59	N
ATOM	369	CA	PRO	A	90	48.997	-25.336	65.191	1.00	30.14	C
ATOM	370	C	PRO	A	90	48.984	-25.579	63.690	1.00	28.23	C
ATOM	371	O	PRO	A	90	48.831	-24.658	62.929	1.00	25.81	O
ATOM	372	CB	PRO	A	90	47.590	-25.048	65.694	1.00	30.72	C
ATOM	373	CG	PRO	A	90	47.499	-23.531	65.614	1.00	32.29	C
ATOM	374	CD	PRO	A	90	48.830	-23.111	66.152	1.00	32.28	C
ATOM	375	N	ARG	A	91	49.172	-26.807	63.269	1.00	26.77	N
ATOM	376	CA	ARG	A	91	49.378	-27.053	61.860	1.00	28.14	C
ATOM	377	C	ARG	A	91	49.104	-28.497	61.521	1.00	28.10	C
ATOM	378	O	ARG	A	91	48.759	-29.284	62.388	1.00	28.34	O
ATOM	379	CB	ARG	A	91	50.806	-26.643	61.469	1.00	27.69	C
ATOM	380	CG	ARG	A	91	51.900	-27.341	62.265	1.00	30.97	C
ATOM	381	CD	ARG	A	91	52.732	-28.290	61.390	1.00	34.77	C
ATOM	382	NE	ARG	A	91	53.733	-27.547	60.706	1.00	35.36	N
ATOM	383	CZ	ARG	A	91	54.232	-27.794	59.496	1.00	31.73	C
ATOM	384	NH1	ARG	A	91	55.111	-26.943	59.051	1.00	28.65	N
ATOM	385	NH2	ARG	A	91	53.870	-28.816	58.733	1.00	29.41	N

Figure 8: Example of PDB format (excerpt from the ATOM record.)

numeral from 1 to 9, followed by three alphanumeric characters. This excerpt corresponds to part of an ATOM record, which lists atomic coordinate information. Each line includes not only the 3D coordinates of an atom but also additional details such as the atom type and chain identifier. In addition to coordinate data, PDB entries also contain various other types of information, including the experimental method used to determine the structure, classification data, and annotations of secondary structural elements.

In PROSITE, motifs are represented using both qualitative pattern descriptors and quantitative matrix based models. The qualitative patterns, expressed in a form analogous to regular expressions, are the most widely used and describe sequence motifs in a rule based manner. In contrast, the quantitative matrix representation characterizes motifs by specifying the frequency of each amino acid residue at every position within the motif. In this study, we focus on motif representations defined by regular expression pattern descriptors. As of December 2025, 1,311 such pattern descriptors had been registered in the PROSITE [38]. A regular expression of the EF-hand motif as follows:

D-{W}-[DNS]-{ILVFYW}-[DENSTG]-[DNQGHRK]-{GP}-[LIVMC]-
[DENQSTAGC]-x(2)-[DE]-[LIVMFYW].

A regular expression of PROSITE has several rules. Each amino acid is expressed as one character. “-” represents the delimiter between each rep-

resentation. “[]” pattern allows for an amino acid in a bracket. “[DNS]” pattern allows for three types of amino acids such as D, N, and S. “{ }” pattern does not allow an amino acid in a bracket. “{W}” pattern does not allow one types of amino acids W. If a particular regular expression is repeated, the number of times it is repeated is described in “()”. “()” contains one or two numbers that specify the number of repetitions. When two numbers are provided, they indicate the minimum and maximum repetition counts. “x(2)” pattern repeats the “x” twice, and “x(2,3)” pattern repeats “x” between 2 and 3 times. For example, A sequence “DKNNDGRIDFDEF” matches the regular expression pattern of the EF-hand motif.

2.4 Sequence alignment

Sequence alignment is a method used to evaluate the similarity between different sequences by identifying regions in which characters appear in the same order. It allows for the comparison of two or more sequences. Many classical sequence comparison methods are formulated as DP-based optimization problems. The DP method is an algorithmic framework for solving optimization problems. the DP solves problems with optimal substructure by computing and storing solutions of overlapping subproblems.

As an example, we introduce the Needleman–Wunsch algorithm for aligning two given sequences [16]. This algorithm is known as global alignment. Given two sequences, it searches for the alignment pattern that maximizes sequence similarity by applying three types of operations: substitution, deletion, and insertion. The optimal alignment pattern can be obtained by computing a score matrix according to a given formulation. For two input strings, x and y of lengths is m and n , respectively, the size of the score matrix, denoted as S , is $(m + 1)(n + 1)$. The computation of each element in S is carried out as follows:

$$S(i, j) = \max \begin{cases} S(i - 1, j - 1) + \text{cost}(x_i, y_j), \\ S(i - 1, j) - d, \\ S(i, j - 1) - d, \end{cases}$$

where $i(0 \leq i \leq m)$ and $j(0 \leq j \leq n)$ represent positions of x and y , respectively. Initially, $S(i, 0) = -id$ and $S(0, j) = -jd$ is calculated. d represents the insertion or deletion cost as a constant value, $\text{cost}(x_i, y_j)$ represents substitution cost that defined as the matching score between characters. For

example, the character matching score can be defined as +1 for a match and −1 for a mismatch. The element $S(0,0)$ is initialized to zero. Finally, an optimal alignment pattern can be obtained by performing a traceback on the computed score matrix starting from $S(m,n)$.

In addition, the alignment with the minimum operation cost is defined as the edit distance. In particular, the Levenshtein distance is defined as the minimum total cost of insertions, deletions, and substitutions that transform x into y [39]. This can be considered as an optimization problem that minimizes the total cost under the similar equation.

2.5 Neural networks

Neural networks are computational models inspired by neurons and are composed of an input layer, one or more hidden layers, and an output layer. Each layer consists of multiple nodes, and the nodes are connected by weighted links. At each node, a nonlinear transformation is performed by applying an activation function to the weighted sum of the inputs. A feed-forward neural network represents a function as a composition of affine transformations and activation functions. Let $\mathbf{h}^{(0)} = \mathbf{x}$ represent input layer, and $\mathbf{x}$ is input vector. For $l = 1, \dots, L$, a layer is defined as:

$$\begin{aligned}\mathbf{z}^{(l)} &= \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}, \\ \mathbf{h}^{(l)} &= \mathbf{f}(\mathbf{z}^{(l)})\end{aligned}$$

where $\mathbf{W}^{(l)}$ and $\mathbf{h}^{(l-1)}$ are the weight matrix and the output vector of $(l-1)$ layer, respectively. $\mathbf{b}^{(l)}$ is the bias vector. $\mathbf{f}()$ is an activation function that is applied element-wise. Various activation functions have been proposed [40], among which the Rectified Linear Unit (ReLU) function [41] is widely used in deep learning. The ReLU function is defined as follows:

$$\text{ReLU}(z) = \max(0, z)$$

Neural networks can be realized by composing smaller neural network modules. Although neural networks are often trained from data, our focus is on constructed ReLU architectures that reproduce classical string processing computations.

3 Development of a Protein-Gene Motif Dictionary System for One-Stop Motif Analysis

In this chapter, we discuss the protein–gene motif dictionary system, a database that integrates protein sequence motifs with their associated gene sequence information and corresponding 3D protein structures. This chapter is based on the content of my first published papers. Please refer to the first item in the List of Publications at the end of this thesis (Copyright 2020 Society of Computer Chemistry, Japan, DOI: 10.2477/jccjie.2020-0008).

3.1 Background

Discovering common structural features of proteins and gene motifs is important for understanding protein and gene functions. As biological data are rapidly increasing, the development of an information platform for efficient analysis is highly desirable.

Motif information for proteins and genes is publicly available in various databases, each maintained independently for different purposes. PROSITE and Pfam are databases of protein sequence motif [12, 30]. Furthermore, 3D structural information of protein can be obtained from MegaMotifBase database [42]. In recent years, systems have also been proposed that enable the search for 3D structural motifs from structure data in PDB [43]. For gene sequence motifs, TRANSFAC and JASPER database are available [44, 45]. These motifs are located in non-coding regions and are often conserved. In contrast, gene sequence motifs in coding regions have not been well explored. Some databases integrate information from distributed biological databases and provide it through a web portal. NCBI, GenomeNet and ExpASY are examples of integrated database that provides cross-search for the collaborating biological databases [46, 47, 48]. These databases are useful for cross-database searches across distributed data resources. However, there is currently no system that enables the retrieval of gene information and 3D structural information associated with protein sequence motifs. If motifs could be searched and analyzed in a one-stop manner across protein and gene sequences, as well as between protein sequences and 3D structural information, we could uncover motif features that cannot be identified from any single representation.

In previous work, a protein–gene motif dictionary for human was proposed

to store protein sequence motifs and their corresponding gene sequences, and the sequence characteristics of protein motif sequences were analyzed with a focus on codon usage in the gene sequences [49]. In addition, as described in Chapter 4, we discuss the motif codon reduced representation of gene motifs and develop a system to estimate conserved motif positions in gene sequences, including introns. This system successfully estimated EF-hand motif sites that were interrupted by introns using protein-gene motif. These results suggest that protein-gene motifs enable more detailed analysis of sequence motif features. Protein function is performed through the formation of specific 3D structures. Accordingly, we consider both structural and sequence information essential for more detailed analysis of motif features. However, data preprocessing tasks such as mapping sequence data to structural data and extracting motif information from structural data require substantial effort. In addition, we also consider that extending the analysis beyond humans to other species can provide insights into molecular diversity and evolution.

Here, we construct a protein-motif dictionary system for multiple species that stores protein-gene motifs with 3D structural information for the corresponding protein sequence motifs. We also implement a management system that enables one-stop search and analysis of protein-gene motif information.

3.2 Materials and methods

3.2.1 Overview of constructing the protein-gene motif dictionary

We constructed a protein-gene motif dictionary system that stores protein sequence motifs with their corresponding gene sequence motifs and 3D structural motifs. Using this dictionary system, we can easily obtain genetic and 3D structural information for protein sequence motifs. Therefore, we developed the protein-gene motif dictionary system to support feature analysis of motif structures.

The protein-gene motif dictionary system was developed in three steps. First, we mapped protein and mRNA sequences and extracted protein sequence motifs and their corresponding gene sequence motifs. Second, we mapped protein sequences to structural information and extracted 3D structural information for the protein sequence motifs. Finally, we stored these three types of motif information in the protein-gene motif dictionary system. Figure 9 provides an overview of the construction workflow of the protein-gene motif dictionary system.

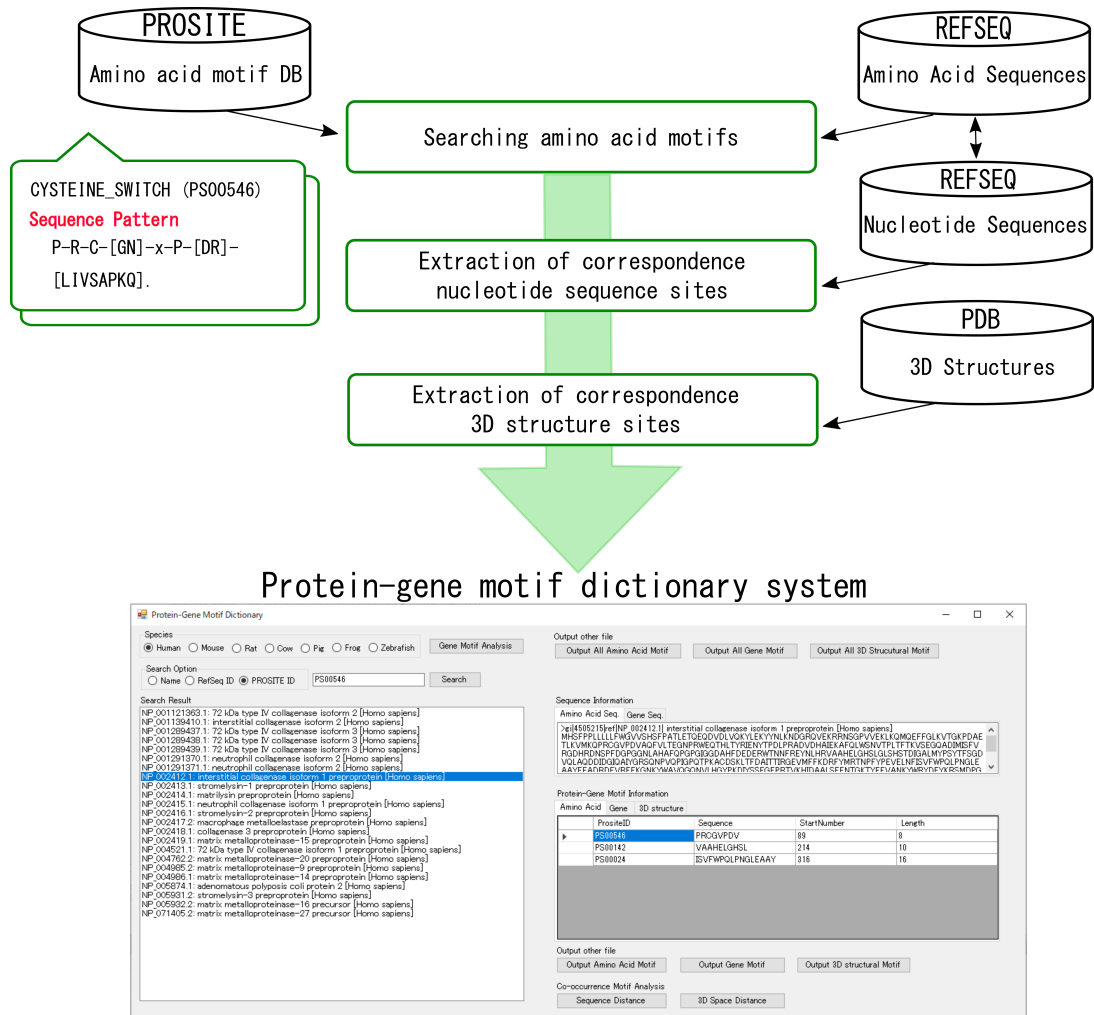


Figure 9: Overview of the construction of the protein-gene motif dictionary system

3.2.2 PROSITE Motif

PROSITE contains various types of amino acid sequence motif information, such as binding sites and enzyme active sites for calcium and zinc. PROSITE motif representations are defined in two forms: regular expression patterns of amino acid sequences, weight matrices and scores derived from sequence alignments. Among these, regular expression patterns are the most abundant in PROSITE. Here, we used 1,309 regular expression patterns registered in PROSITE as of December 6, 2016. For example, a regular expression pattern of zinc protease motif (PROSITE ID: PS00142) is defined as follow:

[GSTALIVN]-{PCHR}-{KND}-H-E-[LIVMFYW]-{DEHRKP}-H-{EKPC}-
[LIVMFYWGSPQ].

In addition, a regular expression pattern of cysteine switch motif (PROSITE ID: PS00546) is defined as follow:

P-R-C-[GN]-x-P-[DR]-[LIVSAPKQ].

Some PROSITE regular expression patterns are not intended to represent biological functions, yet they match many sequences as strings. Such patterns are marked as “SKIP-FLAG = TRUE” in the “PROSITE.DAT” file. In this study, we excluded these motifs from our dictionary system. “PROSITE.DAT” provides an entry for each motif, including its definition. A regular expression pattern can be identified by “PATTERN” in “ID” record. The file also contains various additional fields and descriptions. When performing batch motif searches, scanning the full definition file to extract regular expression patterns one by one is computationally costly. To reduce this overhead, we prepared a simplified file that contains only “ID”, “AC”, and “PA” records for each entry. “ID” record specifies the entry name and type, “AC” record specifies the PROSITE accession, and “PA” record specifies the motif’s regular expression pattern.

3.2.3 Mapping of protein sequences and gene sequences

To construct the dictionary system, we collected protein sequences and their corresponding mRNA sequences. These sequences were obtained from NCBI RefSeq database on December 6, 2016 [50] and are provided in FASTA format. We collected sequence data for human and six model species: mouse, rat, cow, pig, frog, and zebrafish. Model species are commonly used in biological

experiments to understand biological phenomena. They are selected based on factors such as ease of maintenance and breeding, as well as the extent to which their genetic information has been characterized. These species were selected because their data are arranged in the top-level directory of the FTP site of RefSeq and were therefore considered to be in high demand in species. Using the cross-reference data in RefSeq, we mapped protein sequences to their corresponding mRNA sequences. To extract cross-reference data for each target species, we used the NCBI Taxonomy ID (Tax-id). Table 2 lists the Tax-id of the seven model species.

Table 2: List of Taxon identifiers

Taxonomy	Tax-id
Homo sapiens (human)	9606
Mus musculus (mouse)	10090
Rattus norvegicus (rat)	10116
Bos taurus (cow)	9913
Sus scrofa (pig)	9823
Xenopus tropicalis (frog)	8364
Danio rerio (zebrafish)	7955

RefSeq mRNA sequences include introns and non-coding regions. To extract protein–gene motifs using the Shoji system, we prepared protein sequences and their corresponding mRNA sequences such that the translated coding region matches the protein sequence. We then extracted the coding sequence (CDS) region from each mRNA sequence. The CDS region was extracted by global pairwise alignment between the mRNA sequence and a reverse-translated gene sequence. Here, the reverse-translated gene sequence was generated from the protein sequence using a representative codon expression. We performed CDS region extraction in batch using protein sequences, mRNA sequences, and RefSeq cross-reference information. For CDS region estimation, we improved and used a previously developed system [51]. For example, Figure 10 shows the estimated CDS region in a matrix metalloprotease 1 (MMP1) mRNA sequence based on the MMP1 protein sequence.

3.2.4 Extracting the protein-gene sequence motifs

We extracted protein–gene sequence motifs using the Shoji system. Protein and CDS sequences were provided to the system as multi-FASTA files containing multiple FASTA entries. Note that, in each multi-FASTA file, the

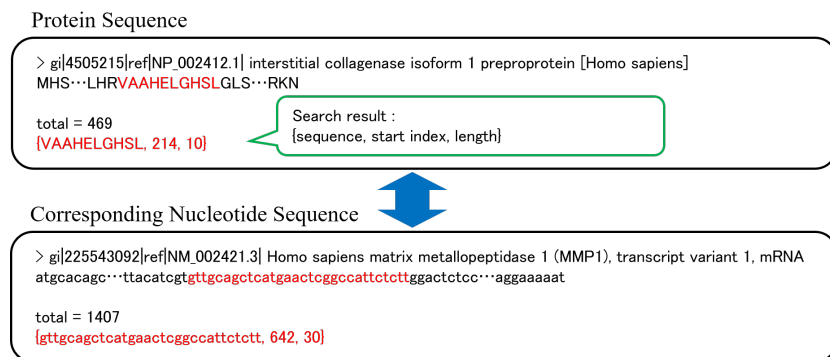


Figure 11: Result of extracting zinc protease protein-gene sequence motif in human MMP1 protein sequence and its corresponding CDS sequence. “total” represents the sequence length, and the extracted results are described under the “total” information.

for a given protein sequence are not available in PDB. The other issue arises when the structure is registered as a quaternary structure, in which case it becomes necessary to determine how multiple chains should be associated with the sequence. To extract structural information corresponding to protein sequence motifs, we check for the presence of motif sequences in each chain by means of string matching. For efficiency, a per-chain amino acid sequence database was preconstructed. Using cross-reference data, we extracted 3D structural data corresponding to the motifs. The structural data comprised all ATOM records in PDB files that correspond to the motifs. The extracted ATOM information was saved to files with the header information, following the naming convention described below.

(PDB ID)+(chain ID)[(RefSeq ID)](PROSITE ID)_(order).pdb

For example, the human MMP1 protein (RefSeq ID: NP_002412.1) contains three conserved sequence motifs. The corresponding PDB structure is 1SU3 chain A (PDB ID + chain ID: 1SU3A). In the following, a five-character representation of a PDB ID indicates that the chain ID is appended as the final character. The file containing the 3D structural information for the second motif is named “1SU3A[NP_002412.1]PS00142_2.pdb”. Furthermore, by assigning indices according to the order of motif occurrences within the sequence, we ensured that repeated occurrences of the same motif could be uniquely identified. Also, some PDB entries are structures determined by nuclear magnetic resonance (NMR). Such NMR structures are often deposited as multi-model ensembles. In this study, we used the first model as the representative structure. This is annotated as “MODEL 1” in the PDB file.

3.2.6 The protein-gene motif dictionary system using the MongoDB

We constructed a protein–gene motif dictionary system that stores protein sequence motifs together with their corresponding gene sequence motifs and 3D structural motifs. The system was implemented using MongoDB, a document-oriented Not Only SQL (NoSQL) database [52]. In MongoDB, data are stored as JSON schema, providing a flexible and human-readable representation. In contrast, relational databases such as MySQL and Oracle typically require a predefined schema, and schema changes often involve explicit modifications when the data model evolves. To address the need for more flexible data models, NoSQL databases have been developed to support schema-flexible storage. Traditional relational database management systems can be insufficient when handling rapidly growing and highly heterogeneous biological data. Accordingly, several studies have adopted flexible NoSQL databases for biological database construction [53, 54].

Table 3 shows the JSON schema used in the protein–gene motif dictionary system. The NAME column stores the FASTA header information. The SEQUENCE and LENGTH columns store the sequence and its length, respectively. The MOTIF column stores protein–gene motif information, including the sequence, length, and start position for each motif instance. The 3D MOTIF column stores reference information for the corresponding 3D structural motif. By managing the NAME and MOTIF fields separately, this system is possible to narrow the search and specify the protein family, and protein with a specific conserved motif.

Table 3: JSON schema of the protein-gene motif dictionary system by the MongoDB

Name	Description
NAME	ID, entry name, species
SEQUENCE	Information of each sequence
LENGTH	Length of each sequence
MOTIF	Protein-gene motif information
3D MOTIF	Reference for 3D motif information

3.3 Results and discussion

3.3.1 Constructing the protein-gene motif dictionary system for human

First, we extracted protein-gene motifs for human. The zinc protease motif was extracted from 215 sites across 213 entries. The corresponding 3D structural motif was linked to 459 sites across 23 entries. In the human MMP1 protein (NP_002412.1), the zinc protease pattern matched the sequence “VAA-HELGHSL” at residues 214–223. The corresponding gene sequence motif matched “gtt gca gct cat gaa ctc ggc cat tct ctt” at bases 642–671 within the CDS region of the MMP1 mRNA sequence (NM_002421.3). This result was confirmed to be accurate because the motif positions were consistent between the protein sequence motif and the gene sequence motif. The MMP1 protein also contains the cysteine switch and hemopexin motifs. These motifs were correctly extracted and linked. The human MMP1 protein was associated with 24 PDB structures. For example, the 3D structural motif correspondence was mapped to residues 215–224 in the 1SU3A and 1SU3B structures, and to residues 115–124 in the 1AYKA structure. These results indicate that protein-gene motifs can be extracted and linked even when the residue positions differ between the protein sequence motif and the 3D structural motif, and when the motif occurs in multi-chain structures. In addition, the 3D structural motifs for the cysteine switch and hemopexin motifs were extracted and linked to the 1SU3A and 1SU3B structures.

The protein-gene motif dictionary system for human contains 15,761 entries and 907 motifs, comprising a total of 45,273 sites. These motifs cover approximately 69% of the PROSITE regular expression patterns. The 3D structural motif dataset contains 407 motifs, comprising 44,242 sites in total. Table 4 summarizes the results of the constructed human protein-gene motif dictionary system.

Table 4: Statistical data of the protein-gene motif dictionary system

	human	mouse	rat	cow	pig	frog	zebrafish
Entries	44,567	30,499	17,569	13,235	4,075	8,566	15,508
Residues	23,271,034	16,134,417	8,780,680	6,443,710	1,696,481	3,792,530	7,162,373
Hit entries	15,761	11,279	7,036	4,793	1,823	2,911	5,662
Motifs	907	899	880	874	632	719	797
Motif sites	45,273	27,573	14,585	10,603	3,402	5,843	12,768
3D motifs	475	196	139	149	77	1	17
3D motif sites	44,242	3,976	3,138	4,808	1,191	4	93

3.3.2 Constructing the protein-gene motif dictionary system for other species

We constructed the protein–gene motif dictionary system for other model species, including mouse, rat, cow, pig, frog, and zebrafish. In total, across the seven model species, 956 sequence motif types comprising 120,047 sites and 569 3D structural motif types comprising 57,452 sites were stored. The zinc protease motif was extracted from 164 sites in mouse, 86 in rat, 67 in cow, 27 in pig, 40 in frog, and 71 in zebrafish. The corresponding 3D structural motifs were linked to 14 sites in mouse, 4 in rat, 9 in pig, and 1 in zebrafish; no 3D structural motifs were linked for cow or frog. In addition, no sequences containing multiple conserved zinc protease motifs were observed in frog or zebrafish. A zinc protease motif sequence, “VTAHELGHSL”, in rat MMP1 differs from the other species at the second residue (alanine to threonine). No MMP1 protein entry was stored in the zebrafish database. Among the non-human species, only the pig database contained 3D structural information linked to the MMP1 protein. In the human MMP1 protein, the cysteine switch and hemopexin motifs co-occur with the zinc protease motif, whereas this co-occurrence was not observed in frog MMP1. These cross-species protein–gene motifs enable analysis of motif differences among species, even within the same motif or protein family. Table 4 provides detailed statistics for the protein–gene motif dictionary systems of the other model species. Among these species, the mouse database contained the largest number of entries after human, whereas the pig database contained the fewest. Because the pig database contains the fewest entries, it is considered to have the smallest number of motifs represented. In particular, although the frog database contained approximately twice as many entries as the pig database, the number of motif types was not substantially different. Compared with the motif types of human database, the pig databases contained 632 motif types, covering approximately 70% of the total entries. The pig database also contained more 3D structural motif types and sites than the frog and zebrafish databases. However, these 3D structural motifs covered only about 20% of those in the human database, and a similar trend was observed for the mouse database. As a result, relatively few 3D structural data for motifs could be registered for model organisms other than human. This difference likely reflects the limited protein structural data available for model species other than humans.

As a result, we constructed a protein–gene motif dictionary system for seven model species that contains protein sequence motifs with their corresponding gene sequence motifs and 3D structural motifs. By classifying information for each protein, the system enables one-stop search and analysis of protein–gene motifs and their co-occurrence patterns conserved within proteins. Furthermore, the system supports comparative analyses of motif features across species, which were not possible with the previous protein–gene motif dictionary system.

3.3.3 Implementing the management system for the protein–gene motif dictionary system

For one-stop motif analysis, we implemented a management system with a graphical user interface (GUI) for the protein–gene motif dictionary system. This management system enables intuitive search and analysis of protein–gene motifs. It was developed in Visual C# using the .NET Framework 4.5 on Windows 10.

To search for protein–gene motifs for a given protein, the user first selects a species and enters a protein name. The search results are displayed as a list from which the target protein can be selected. After selecting a protein, the system displays the corresponding protein–gene motif information, including the PROSITE ID, motif sequences, and the length and start position of each motif. The 3D structural information includes the PDB ID and chain ID. In addition, the protein–gene motif dictionary management system provides visualization of 3D structural information using Jmol [55]. The system also supports searches by RefSeq ID and PROSITE ID. Using a PROSITE ID, users can retrieve proteins that contain a specific conserved motif. Furthermore, we implemented export functions for protein–gene sequence motif information in CSV format and 3D structural information in PDB format. These data can be exported for a selected protein or for an entire species database. Figure 12 shows the main window of the management system and the results of a filtered search in the human database using the PROSITE ID of the zinc protease motif.

3.3.4 3D structural feature analysis of co-occurrence motifs

Because some proteins perform multiple biological functions, they may contain multiple conserved motifs. Such motifs can co-occur within the same

Protein-Gene Motif Dictionary

Species: ☒ Human ☐ Mouse ☐ Rat ☐ Cow ☐ Pig ☐ Frog ☐ Zebrafish

Search Option: ☐ Name ☐ RefSeq ID ☒ PROSITE ID

Search: PS00546

Search Result:

- NP_001121363.1: 72 kDa type IV collagenase isoform 2 [Homo sapiens]
- NP_001139410.1: interstitial collagenase isoform 2 [Homo sapiens]
- NP_001289437.1: 72 kDa type IV collagenase isoform 3 [Homo sapiens]
- NP_001289438.1: 72 kDa type IV collagenase isoform 3 [Homo sapiens]
- NP_001289439.1: 72 kDa type IV collagenase isoform 3 [Homo sapiens]
- NP_001291370.1: neutrophil collagenase isoform 2 [Homo sapiens]
- NP_001291371.1: neutrophil collagenase isoform 2 [Homo sapiens]
- NP_002412.1: interstitial collagenase isoform 1 preproprotein (Homo sapiens)**
- NP_002413.1: stromelysin-1 preproprotein [Homo sapiens]
- NP_002414.1: matrilysin preproprotein [Homo sapiens]
- NP_002415.1: neutrophil collagenase isoform 1 preproprotein [Homo sapiens]
- NP_002417.2: macrophage metalloelastase preproprotein [Homo sapiens]
- NP_002418.1: collagenase 3 preproprotein [Homo sapiens]
- NP_002419.1: matrix metalloproteinase-15 preproprotein [Homo sapiens]
- NP_004521.1: 72 kDa type IV collagenase isoform 1 preproprotein [Homo sapiens]
- NP_004762.2: matrix metalloproteinase-20 preproprotein [Homo sapiens]
- NP_004965.2: matrix metalloproteinase-9 preproprotein [Homo sapiens]
- NP_004966.1: matrix metalloproteinase-14 preproprotein [Homo sapiens]
- NP_005874.1: adenomatous polyposis coli protein 2 [Homo sapiens]
- NP_005931.2: stromelysin-3 preproprotein [Homo sapiens]
- NP_005932.2: matrix metalloproteinase-16 precursor [Homo sapiens]
- NP_071405.2: matrix metalloproteinase-27 precursor [Homo sapiens]

Protein-Gene Motif Information

Amino Acid	Gene	3D structure
PS00546	PRCGVPDV	89
PS00142	VAAHELGHSL	214
PS00024	ISVFWPLPGLGLEAAY	316

Output other file:

Co-occurrence Motif Analysis:

Protein-Gene Motif Information

Amino Acid	Gene	3D structure
PS00142	1OGL	A
PS00142	1OGL	B
PS00142	1HFG	A
PS00546	1SLU	A
PS00142	1SLU	A
PS00024	1SLU	A
PS00546	1SLU	B
PS00142	1SLU	B
PS00024	1SLU	B

View Jmol

Figure 12: A snapshot of the management system and results from searching for a protein-gene motif. ① Human species is selected, ② Then, a protein with a zinc protease motif (PROSITE ID: PS00546) is searched. ③ The search results are shown as a list, and interstitial collagenase isoform 1 preproprotein (RefSeqID: NP_002412.1) is selected from the list. ④ Protein-gene motif, and protein information such as sequence and co-occurrence of each motif are obtained. ⑤ By selecting the tab button, motif information of the protein sequence, gene sequence, or structural information can be changed. ⑥ In the “3D structure” tab, ID information of the motif can be obtained. ⑦ By clicking the “View Jmol” button, the 3D model can be viewed in the Jmol software

protein. As a representative example, S100 proteins contain two distinct EF-hand type calcium binding motifs. These two motifs are known to differ in the strength of their calcium-binding [56]. The conformations of secondary structures conserved even when their sequential order in the protein sequence is not conserved [57, 58]. Therefore, both sequence distance and spatial distance are important for protein structure and function. We consider that analyzing sequence sequence and spatial distances between co-occurring motifs can provide new insights into protein function and structure. To enable such analyses, we implemented a module for calculating these distances between motifs in the management system.

We calculated the sequence distance and spatial distance between motifs within a protein. The sequence distance is defined as the number of residues between the residue immediately following the end of the N-terminal motif and the start of the C-terminal motif. The spatial distance is defined as the Euclidean distance between the centroids of the two corresponding 3D structural motifs. This spatial distance is also referred to as the 3D space distance. Each centroid is defined as the mean coordinate of all α carbons within the 3D structural motif.

We explored co-occurrence motif patterns involving the zinc protease motif. This analysis was performed using the management system’s filtered search by PROSITE ID. As a result, we found that cysteine switch motifs are conserved in a specific protein type of zinc proteases, such as matrix metalloproteinases (MMPs). These motifs co-occurred in three proteins comprising four chains. Although the sequence distances ranged from 117 to 293 residues, the 3D space distance was consistently conserved at 12 Å. Figure 13 shows the 3D structures of MMP1 and MMP9 and their co-occurring motifs. Co-occurrence motif analysis for other MMP proteins could not be performed because the 3D structural information for the relevant motifs has not been stored in the dictionary system.

MMPs is a family of proteolytic enzymes. This protein family degrade extracellular matrix components and play important roles in physiological processes such as wound healing and angiogenesis [59, 60]. MMPs are secreted as zymogens and are activated by proteolytic removal of the propeptide region. In the zymogen state, the catalytic zinc ion at the active site is coordinated by a cysteine residue conserved in the propeptide region [61]. This activation mechanism is known as the cysteine switch and is registered in the PROSITE

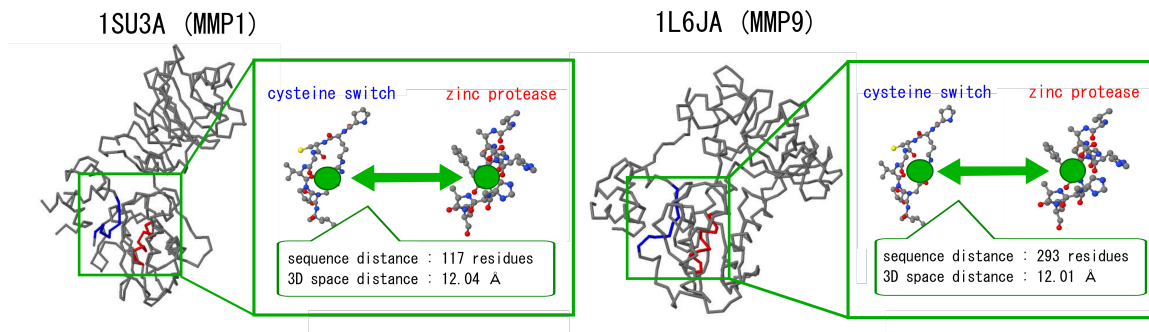


Figure 13: 3D view of the MMP protein (1SU3A and 1L6JA) and colored co-occurrence motif. A cysteine switch motif and zinc protease motif are represented in blue and red. A green point of center of each motif represents the centroid of a motif

as the cysteine switch motif [62]. In addition, a region containing two histidine residues that coordinate the zinc ion is registered in the PROSITE as the zinc protease motif [63]. For MMPs, the cysteine switch motif and the zinc protease motif co-occur in the propeptide region, and their 3D space distance is conserved even when their sequence distance varies. Therefore, it is reasonable that these two motifs are conserved within the same proteins and maintain a characteristic Euclidean distance. Using our management system, users can perform one-stop analyses of co-occurring motifs.

There is no existing system that provides a direct, end-to-end comparison with the results produced by our system. Therefore, verification typically requires manually consulting entries in RefSeq, UniProt, and PDB. Using the human MMP1 protein (NP_002412.1) as an example, we verify our results as follows. First, we confirm the extract translated region and the corresponding protein sequence using the “CDS” feature section in RefSeq. This section specifies the CDS coordinates and links to the related RefSeq protein. For NM_002421.3, the CDS is annotated as bases 144–1553 (including the stop codon). Second, we verify the extracted motifs using the “Region” feature section in RefSeq. The “Region” annotations indicate that a cysteine switch motif is conserved at residues 90–97. The zinc protease motif is included within the metalloprotease region. Third, we verify the mapping between the RefSeq protein and PDB structures via UniProt. The UniProt entry P03956 lists 24 associated structures in the “Structure” section. Finally, we confirmed the extracted 3D structural motifs using PDB. The PDB structure 1SU3 contains three motifs, and our system correctly extracted the corresponding 3D structural information.

The sequence distance is also obtained manually as the number of residues between the co-occurring motifs. The 3D space distance is defined as the Euclidean distance between the motif centroids, where each centroid is the mean coordinate calculated from the 3D structural information in PDB. Based on the verification procedures described above, we confirm that the co-occurrence motif analysis was performed correctly.

3.4 Concluding Remarks

In this chapter, we discussed the protein-gene motif dictionary system for the seven species that stored protein sequence motifs and their corresponding gene sequence and 3D structural information. The constructed motif dictionary covers all seven species and stored 956 sequence motif types (120,047 sites) and 569 structural motif types (57,452 sites). As a result, for human, which had the largest number of entries, we were able to store 907 sequence motif types, corresponding to a total of 45,273 sites. In addition, 475 3D structural motif types (44,242 sites) were stored. Among the other six model species, mouse had the largest number of stored entries after human. Specifically, 899 sequence motif types (27,573 sites) and 196 3D structural motif types (3,976 sites) were stored. For the remaining species, substantially fewer motifs were stored than in the human database, largely because fewer sequence entries were available. Moreover, the difference was even greater for structural motifs. This is likely due to limited cross-reference data and the lack of experimentally determined 3D structures for non-human species.

In addition, we implemented the GUI management system for one-stop search and analysis based on the protein-gene motif dictionary. Using this analysis module in the management system, we analyzed the distances between the zinc protease motif and the cysteine switch motif that co-occur in MMP proteins. As the result, we found that the Euclidian distance between the centroids of the cysteine switch motif and the zinc protease motif was maintained at approximately 12 Å, although their sequence distance was not conserved. This finding provides new insights into motifs that were enabled by the construction of an integrative motif database. Unfortunately, comparative experiments with other species could not be conducted because the corresponding 3D structural data had not yet been stored in the motif dictionary. However, this problem is expected to be resolved once the relevant structural data are registered in PDB.

In this research, the sequence and structural data were obtained as snapshots from specific time points. Therefore, the dictionary system does not currently accommodate ongoing updates to RefSeq and PDB. In future work, an automated update workflow will be required to periodically query each database via APIs and to add or update entries in the motif dictionary. In addition, improving motif search speed is considered an important task for periodically updating the motif dictionary. At present, all PROSITE regular expression patterns are searched against each protein sequence. However, a single protein sequence is expected to contain only a limited number of motif types, making exhaustive regular expression searches over all patterns inefficient. One possible approach to accelerating the search is to annotate protein sequences with functional tags and apply filtering of candidate motif types based on sequence similarity. BLAST [17] is a well known method for searching for similar sequences. In recent years, several libraries that enable faster sequence searching have been proposed [64, 65].

4 Development of the Dynamic Programming (DP)-based Functional Site Estimation System Using the Motif Codon Reduced Representation

In this chapter, we discuss a genome functional site estimation system that employs dynamic programming in combination with the motif codon reduced representation. This chapter is based on the content of my second published papers. Please refer to the second item in the List of Publications at the end of this thesis (Copyright 2022 Society of Computer Chemistry, Japan, DOI: 10.2477/jccj.2022-0005).

4.1 Background

Genetic information stored in the genome underlies the structure and function of living organisms. In eukaryotic genomes, protein-coding regions are composed of exons, while introns are removed during RNA processing. Analysis of genome sequences provides fundamental insights into biological function and evolution.

With the progress of post-genome projects and advances in structural determination techniques, the biological sequence data increasing rapidly, such as nucleic acids and proteins. Biological databases include nucleotide sequence databases such as GenBank and EMBL, and amino acid sequence databases such as UniProt and Swiss-Prot [25, 26, 29]. In addition, NCBI RefSeq provides high-quality and non-redundant sequences that have been checked by researchers [35]. From the sequence data in these databases, it is desired to extract meaningful molecular biological information such as relationship between sequence and function. Although projects are currently underway to determine the genome sequences of more than 60 eukaryotic. However, biological interpretation has not kept pace with the growth of sequence data [66]. Accordingly, efficient tools have been developed to predict genes and protein coding regions directly from genome sequences. If the sequence features are known, the simplest approach is to search using regular expressions. Several tools have been proposed for searching gene sequences with regular expressions [67, 68]. In these researches, regular expression search was implemented to support nucleotide wildcard representations. However, these methods have difficulty in accounting for long gap insertions. Krogh et al. proposed an HMM for detecting protein coding gene in *Escherichia coli* (E.

coli) [69]. The parameters of the HMM are estimated using approximately one million nucleotides, the HMM successfully estimated location of about 90% of the known *E. coli* genes. Uberbacher et al. estimated protein coding regions in eukaryotic genomes by calculating the frequencies of characteristic sequence patterns and applying values obtained using artificial neural networks [70]. To date, many gene prediction methods have been proposed by researchers including Claverie, Guigó, Haussler, Burge, and Karlin [71, 72, 73, 74]. Because higher organisms have complex genomic structures, accurate prediction of coding regions remains a challenging task [75].

Amino acid sequences of protein including characteristic pattern named motif that is closely associated with their biological functions. Motif is considered to be conserved at the gene sequence level. Motifs can be estimated by sequence alignment. Sequence alignment for motif extraction can be computed using DP. Among the tools capable of rapidly computing pairwise alignments, BLAST is widely used, while ClustalW is a well-known method for performing multiple sequence alignment [17, 76]. In addition, PROSITE is a widely recognized biological database that stored information on amino acid sequence motifs [12].

We proposed the protein–gene motif dictionary system, which integrates PROSITE amino acid sequence motifs and their corresponding gene sequences, and protein 3D structural information of seven model species (See Chapter 3). This system enables the identification of motif characteristics at the nucleotide sequence level that cannot be found from amino acid sequences alone. Kato et al. proposed the codon reduced representation in which codon and amino acids are mapped in a one-to-one correspondence [77]. They also developed a system for estimating motif sites within genome sequences based on simple pattern matching between genome sequences and the codon reduced representation. However, when applying this method to estimate EF-hand motifs in the Troponin C type 1 (TNNC1) genome sequence, the system failed to estimate the motif despite its known conservation in the TNNC1 protein sequence. Subsequent investigation based on genome annotations revealed that the EF-hand motif is interrupted by an intron. Consequently, the system could not handle cases in which introns are inserted within motifs.

We considered DP–based sequence alignment to be an effective approach for estimating protein motif sequence sites from genome sequences. By applying the codon reduced representations to protein sequence motifs and using

them in sequence alignment, the corresponding motif sites can be directly and correctly estimated even when they are conserved with intron insertions. However, the codon reduced representation does not maintain a strict one to one correspondence for all codons and amino acids, making direct application of DP methods difficult. If it becomes possible to estimate protein-coding regions, protein sequence motif sites, and functional genomic regions considering intron insertions, we would obtain knowledge of biological function and conserved position for an unknown genome sequence. Moreover, comparing intron interrupted functional sites across species may provide insights into their evolutionary processes.

In this chapter, we modify the codon reduced representation using its approximate representation. We also develop a DP-based sequence alignment system using these representations. Furthermore, we propose motif codon reduced representation based on a codon weight matrix that quantifies nucleotide usage within gene sequences corresponding to protein sequence motifs. Using the motif codon reduced representation, we attempt to estimate functional sites in genome sequences and analyze them.

4.2 Materials and methods

4.2.1 Codon Reduced Representation

Nucleic acid sequences are composed of four bases and a triplet of three consecutive bases that specifies an amino acid is referred to as a codon. Because the combinations of four bases yield $4^3 = 64$ possible codons, several codons correspond to the same amino acid. In addition to the four bases, the IUPAC defines wildcard symbols. For example, adenine or guanine (the purine bases) are represented by p (purine), cytosine or thymine (the pyrimidine bases) by y (pyrimidine), and any base by n (any). Table 5 shows the wildcard representation. Applying these wildcard representations to the genetic code table yields the codon reduced representation [77]. In this chapter, nucleotide bases and wildcard in genome sequences are represented in lowercase letters, whereas amino acids are denoted using uppercase one-letter codes.

In the codon reduced representation, phenylalanine, which is encoded by two codons, can be represented by “tty”. Similarly, arginine is encoded by six codons, can be represented by “cgn” and “agr”. Using the representation, 17 out of the 20 amino acids can be mapped to codons in a one-to-one manner. However, leucine, serine, or arginine cannot be expressed in a single pattern.

Table 5: Wildcard representation of nucleotide code

code	nucleotide	code	nucleotide	code	nucleotide
a	a (Adenine)	r	a/g (puRine)	h	a/c/t
g	g (Guanine)	y	c/y (pYrimidine)	b	g/c/t
c	c (Cytosine)	m	a/c (aMino)	v	a/g/c
t	t (Tymine)	k	g/t (Keto)	d	a/g/t
		s	c/g (Strong)	n	a/g/c/t
		w	a/t (Weak)	-	(gap)

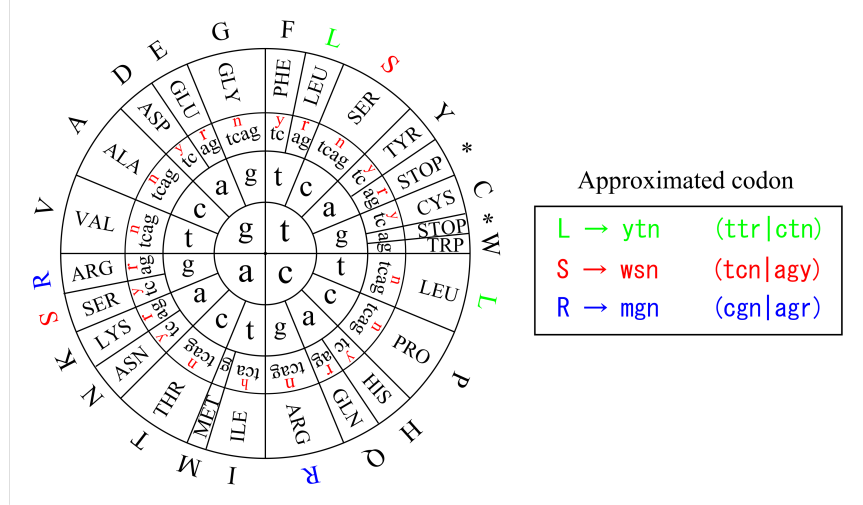


Figure 14: Genetic code table of codon reduced representation and its approximation

To uniquely specify these amino acids, we propose approximate codon reduced representations. For instance, arginine is represented “cgn” and “agr”. By approximating the first base “c” or “a” as “m”, and the third base “n” or “r” as “n”, arginine can be expressed as a single pattern “mgn”. Similarly, approximate representations can be defined for leucine and serine, which can be expressed as “ytn” and “wsn”, respectively. Figure 14 shows the genetic code table after applying both the codon reduced representation and its approximations. It should be noted that approximate representation matches additional amino acids. For instance, the arginine pattern “mgn” also corresponds to serine patterns such as “agr”. Therefore, caution is required when using these approximations. Although not used in this research, the stop codons “taa”, “tag”, and “tga” can also be approximated and represented by the pattern “trh”.

By applying the approximate representation of codon reduced representation, reverse translation of an amino acid sequence can be expressed as a gene sequence uniquely. As an example, we focus on the EF-hand motif, which is

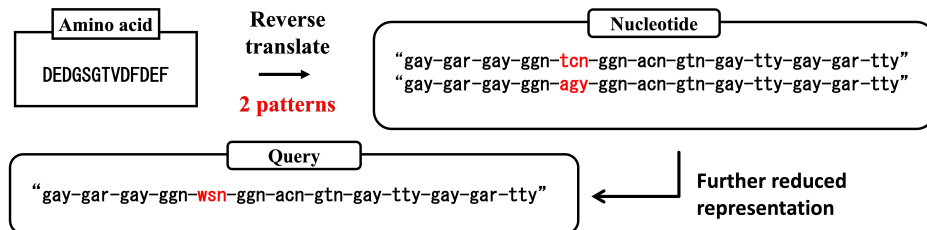


Figure 15: Reverse translation of “DEDGSGTVD FDEF”

involved in calcium binding [32]. The EF-hand motif sequence “DEDGSGTVD FDEF” conserved in the human TNNC1 protein yields two possible gene sequences when reverse-translating the fifth residue serine because serine can be encoded by “tcn” or “agy”. Using the approximate representation, serine is represented as “wsn” allowing the gene sequence to be uniquely determined. Figure 15 illustrates the process of reverse translation from an amino acid sequence to a gene sequence.

4.2.2 Motif codon reduced representation

If the regular expressions of amino acid motifs registered in PROSITE are directly converted into codon reduced representations, most codons match any pattern, resulting in the extraction of numerous non-functional site. To address this issue, we represent gene motif for protein motif pattern in PROSITE using a weight matrix and wildcard symbols. We define this representation as the motif codon reduced representation. This approach enables motif patterns that are well conserved at the nucleotide level to be expressed as query sequences. Therefore, the motif codon reduced representation allows efficient estimation of conserved motif regions within genome sequences.

A weight matrix is constructed by calculating, for each codon position, the nucleotide frequencies observed in the nucleotide sequences corresponding to protein motif sites defined by the protein–gene motif. This matrix quantitatively captures nucleotide biases that are not found from regular expressions of protein motifs. For instance, D is encoded by two codons, “gat” and “gac”. If “gat” appears eight times and “gac” appears two times in the nucleotide sequences, the resulting weight matrix is shown in Figure 16.

To associate the weight matrix with motif patterns, we first apply a threshold to the nucleotide frequencies. Each codon position in the motif pattern is then compared with the corresponding values in the weight matrix. If a nucleotide frequency exceeds the threshold, it is considered a matching nu-

Codons of Aspartic Acid		
g	a	t
g	a	t
g	a	c
g	a	t
g	a	t
g	a	t
g	a	t
g	a	t
g	a	t
g	a	c



a	0	1	0
c	0	0	0.2
g	1	0	0
t	0	0	0.8

Figure 16: Codon weight matrix for the aspartic acid pattern

[DENSTG]			
a	0.256	0.804	0.063
c	0	0.108	0.322
g	0.655	0.088	0.026
t	0.079	0	0.589



a	1	1	0
c	0	0	1
g	1	0	0
t	0	0	1

$\underbrace{\hspace{1.5cm}}$
r
(a | g)

$\underbrace{\hspace{1.5cm}}$
a

$\underbrace{\hspace{1.5cm}}$
y
(c | t)

Figure 17: Codon reduced representation based on weight matrix with threshold

cleotide for that motif position. Otherwise, it is treated as non-matching. If the nucleotide usage at a given position exceeds the threshold, the corresponding nucleotide is accepted. If multiple nucleotides are accepted, the position is represented using the wildcard symbol. For instance, if the weight matrix for “[DENSTG]” pattern is given by the matrix in left matrix of Figure 17, then, with a threshold of 0.25, the corresponding codon reduced pattern can be represented as “ray” (See right table of Figure 17).

4.2.3 DP-based sequence alignment for functional site estimation

Sequence alignment is a method used to assess similarity between different sequences by identifying regions in which the same ordering of characters occurs. By inserting gaps so that a larger number of characters match, two or more sequences can be compared and an alignment with the maximum possible alignment score can be obtained. To found locally similar regions, local alignment using DP is commonly used. However, functional regions in genome sequences are not always conserved to a single exon. Therefore, it is necessary to insert gaps so that alignment does not match intron regions. The Smith–Waterman algorithm is a well-known DP-based local alignment method that allows inserting consecutive gaps [15]. Gotoh extended this method by introducing an affine gap penalty, in which gap opening and gap extension have different penalties, and adapted it to the Smith–Waterman

algorithm [78]. This approach is widely used and considering long gaps insertion by assigning very small penalties to gap extensions. In this research, we extended a sequence alignment method for estimating functional sites in genome sequences using gap penalty. Note that Gotoh's algorithm computes alignments using three types of tables to reduce computational complexity. In contrast, our approach adopts an affine gap penalty in the Smith-Waterman algorithm to enable a simpler implementation. Specifically, to enable sequence alignment using gene sequences represented by the codon reduced representation, we extended a substitution matrix for wildcard character matching. We first present the recurrence equations for the Smith-Waterman algorithm with affine gap penalties.

$$S(i, j) = \max \begin{cases} S(i-1, j-1) + \text{cost}(M_i, Q_j), \\ \max_{1 \leq k \leq i} \{S(i-k, j) - w(k)\} \\ \max_{1 \leq k \leq j} \{S(i, j-k) - w(k)\} \\ 0 \end{cases}$$

$$w(k) = \alpha + \beta(k-1)$$

Here, M_i denotes the i -th nucleotide of the comparison sequence M , and Q_j denotes the j -th nucleotide of the query sequence Q . $S(i, j)$ represents the alignment score at position (i, j) , and $\text{cost}(M_i, Q_j)$ denotes the similarity between M_i and Q_j . $w(k)$ is the total affine gap penalty, where α and β represent the gap open and gap extension penalties, respectively, and k is the number of continued gaps. Similarity between nucleotides is defined by a substitution matrix, where the matrix entries are assigned based on the degree of nucleotide matches. When an amino acid sequence is reverse-translated, the resulting gene sequence may contain wildcard symbols. To perform alignment using such reverse-translated sequences, we extended the substitution matrix to correspond to use of wildcard symbols. When extending the substitution matrix, it is reasonable to weight the scoring such that exact matches assign higher scores than matches involving wildcard symbols. Therefore, we modified the scoring rule for wildcard symbols. The newly proposed substitution matrix is shown in Figure 18. Specifically, if a wildcard matches two of the four nucleotides, a score of 5 is assigned. If it matches three nucleotides, a score of 2 is assigned. For the wildcard "n" a score of 0 is assigned.

Based on the recurrence equations and the substitution matrix described

	a	g	c	t	r	y	m	k	s	w	h	b	v	d	n
a	10	-10	-10	-10	5	-10	5	-10	-10	5	2	-10	2	2	0
g	-10	10	-10	-10	5	-10	-10	5	5	-10	-10	2	2	2	0
c	-10	-10	10	-10	-10	5	5	-10	5	-10	2	2	2	-10	0
t	-10	-10	-10	10	-10	5	-10	5	-10	5	2	2	-10	2	0

Figure 18: Extended substitute matrix

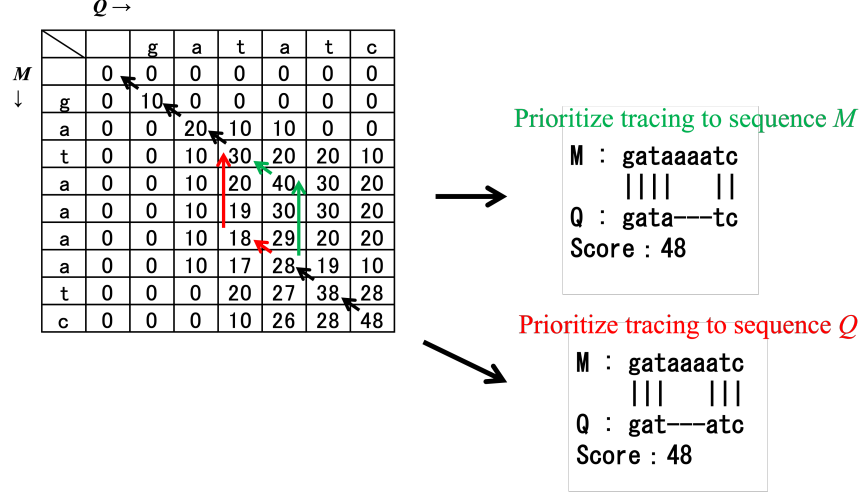


Figure 19: An example of calculating sequence alignment

above, an alignment score matrix is constructed, and the alignment scores are computed accordingly. During the construction of the score matrix, the direction of the cell used to obtain the maximum score at each position is recorded. After the score matrix is completed, the alignment result is obtained by tracing back from the cell with the highest score, following the recorded directions. When the traceback enters a region corresponding to an affine gap, gaps are continuously inserted until the end of the affine gap region is reached.

As an illustrative example, we perform an alignment using the a sequence M = "gataaaatc" and the query sequence Q = "gatatc". The substitution matrix is defined as in Figure 18, with a opening gap penalty of $\alpha = 10$ and a extension gap penalty of $\beta = 1$. During traceback from the cell with the maximum score, there may be positions at which the traceback direction branches. Depending on the scoring matrix, multiple traceback paths may exist. In this study, we performed both comparison sequence prioritized and query sequence prioritized traceback procedures to detect multiple alignment patterns. Figure 19 shows the score matrix constructed from the two sequences and the corresponding traceback results.

In this example, two alignment patterns are considered for sequences M and Q , and depending on the priority of traceback directions. As a result, different alignment results may be obtained even when the alignment score is the same.

4.2.4 Accommodating multiple candidate functional sites

In local alignment based on the DP method, an optimal alignment can be obtained by performing traceback from the cell with the maximum score. However, initiating traceback only from the maximum score cell may overlook functional regions when multiple functional regions exist in a sequence. Assuming that functional sites are conserved in the alignment starting from the maximum score, it is considered that alignments beginning from scores close to the maximum may also contain functional sites. Therefore, we define the starting score for traceback as follows.

$$\text{Starting score of trace} = \text{maximum score} \times \text{search range (\%)}$$

Traceback is performed from any cell whose score is equal to or greater than this threshold, and the resulting alignments are extracted as candidate functional regions. This approach enables the detection of multiple functional sites that may exist within a sequence.

4.3 Results and discussion

4.3.1 Protein-coding region estimation experiments

To evaluate whether alignment with affine gaps and the substitution matrix based on the codon reduced representation is effective for estimating functional regions in genome sequences, we first conducted experiments to estimate CDS regions. Specifically, the gene sequence obtained by reverse translating the entire amino acid sequence into the codon reduced representation was used as a query, and CDS regions were estimated by alignment with the corresponding genome sequence. For the experiments, we used data from RefSeq: the human TNNC1 genome sequence (NG_008963.1), whose CDS spans multiple exons, and the corresponding amino acid sequence (NP_003271.1). All input sequences were formatted in FASTA. Note that, the nucleotides in the genome sequence of the dataset used in our experiments are represented in uppercase, reflecting the original RefSeq data. The reverse translated and query sequences are represented in lowercase. Figure 20 shows the reverse

```

>NP_003271_g30_p100_d30
atggaygayathtayaargcngcngtngarcarytnacngargarcaraaaygartttaaargcngcnttygayathttygtntnyngngongargaygg
ntgyathwsnacnaargarytnggnaargtnatghgnatgytnggncaraayccnaccngargarytnccargaratgathgaygartngaygaraygg
gnwsngnacngtngayttygaygartttytngtnatgatggtnhgtgyatgaargaygaywsnaarggnaarwsngargargarytnwsngayytnnty
hgnatgtygayaaraaygcngayggntayathgaytngaygarytnaarathatgytnccargcncngngargacnathacngargaygayathgarga
rytnatgaargayggngayaaraayaaygayggnhgnathgaytaygaygartttytngarttyatgaarggngtngar

```

Figure 20: A reverse translated nucleotide sequence of NP_003271.1.

translated gene sequence represented by the codon reduced representation. Also, protein sequence data of human TNNC1 is shown in Figure 7. An additional annotation was included in the header line of the reverse translated gene sequence to indicate that the sequence was generated by reverse translation.

The number of introns contained in a single gene varies widely, ranging from as little as one to more than forty. Intron lengths also differ substantially, from approximately 50 nucleotides to over 10,000 nucleotides. In this experiment, the opening gap penalty was set to $\alpha = 20$, the extension gap penalty to $\beta = 0$, and the search range to 80%. These settings expected high-quality alignment for genome sequence include very long intron region.

Figure 21 presents the estimated alignment results. In the output, the line beginning with “M:” shows the alignment of the dataset sequence, whereas the line beginning with “Q:” shows the alignment of the query sequence. The line between them indicates the matching symbols: a vertical bar (“—”) denotes an exact match, “+” indicates a match with a wildcard, “.” indicates a match with “any,” and a blank space indicates a mismatch. The alignment score and the matched positional information are shown at the end of the output. The estimation results indicate that the aligned region is divided into multiple segments. Each “segment” in Figure 21 represents one such region. For example, “(5027 $\cdots$ 5049)” denotes a contentious nucleotide segment 5,027 to 5,049 in the sequence. To verify the correctness of the estimated regions, we compared the results with the CDS annotation of the genome sequence. The annotation information is shown in Figure 22. The annotation file begins with metadata such as the ID, registration date, and definition line, followed by various annotation entries. The CDS annotation includes the CDS region, the corresponding gene name, and the translated protein sequence. Although the start and end positions of some estimated segments differ by one nucleotide from the CDS annotation, the estimated regions matched almost with the CDS annotation. The final annotated segment is three nucleotides longer than

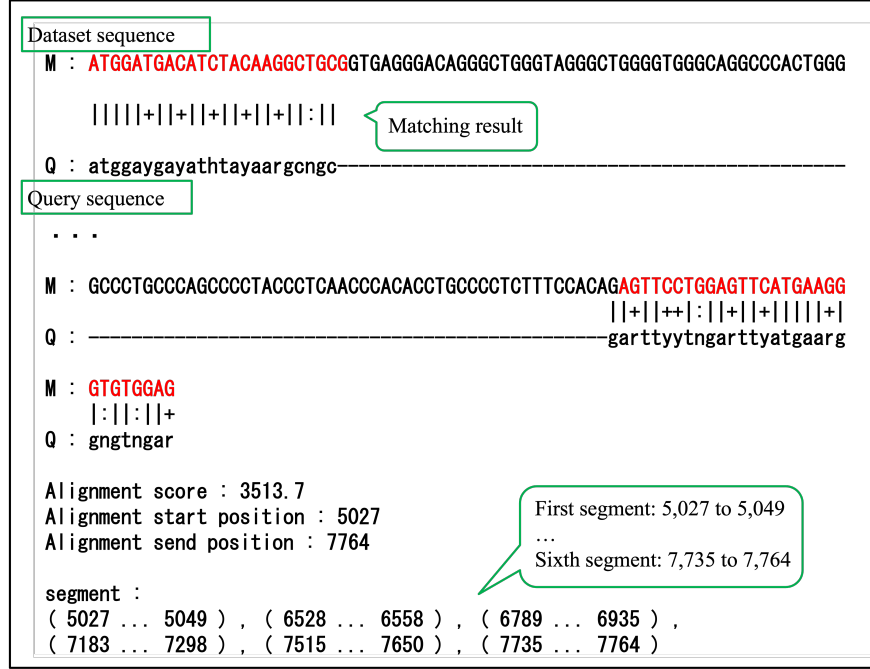


Figure 21: Estimated result of CDS region in TNNC1 genome sequence (excerpt). 'M:' lines show alignment result of dataset sequence. 'Q:' lines show alignment result of query sequence. Matching result between dataset and query sequences is shown matching symbols, '—': exact match, '+': wildcard match, and ':': any match. Correct CDS region is colored red in dataset sequence. Lines after 'segment:' show estimated CDS region. In this result, it is shown that CDS regions have conserved six regions separately.

the estimated one. This difference corresponds to the stop codon. Overall, these results demonstrate that alignment method using affine gaps and the extended substitution matrix is effective for estimating functional regions in genome sequences.

4.3.2 Functional site estimation using codon reduced representation for EF-hand motif

Previous studies have shown that three EF-hand motifs in the human TNNC1 genome sequence, and a EF-hand motif conserved inserting intron [77]. Therefore, when the CDS regions estimation for genome sequence using the motif codon reduced representation as a query, it is expected that protein motif sites has intron can be detected. Based on this assumption, we performed functional site estimation on the genome sequence using the motif codon reduced representation. The motif codon reduced representation was constructed from the codon weight matrix using 796 EF-hand motif sites in human from the protein–gene motif dictionary system. The regular expression pattern for the

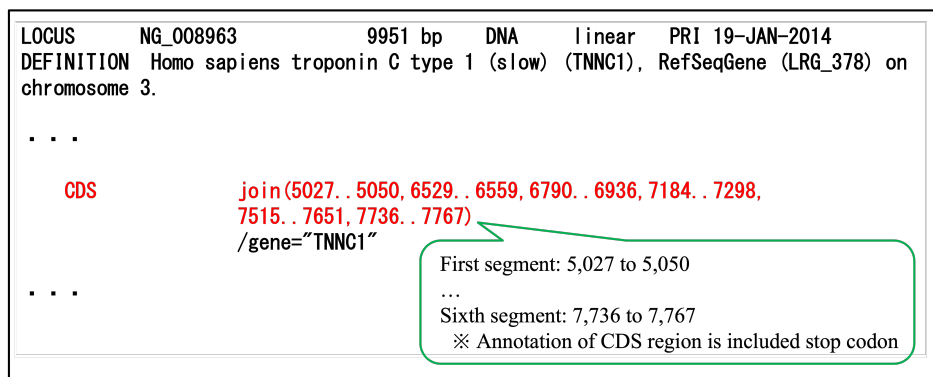


Figure 22: Annotation of NG_008963.1 (excerpt). 'CDS' annotation shows gene name and position of CDS region. 'join' annotation shows position of the CDS region. '/gene' annotation shows gene name. In the sequence, CDS region of TNNC1 gene is preserved six segments between 5027 and 7767.

EF-hand motif defined in PROSITE (See Section 2.3). The weight matrix is shown in Figures 23. The resulting motif codon-reduced representation as following:

gay vhn ray vrn ray grv nhv vtb dvv nwn vmd gar htb

The weight matrix is provided for each pattern in the PROSITE regular expression. A threshold of 0.15 was used for generating the motif codon-reduced representation. The estimation hyperparameters were set as follows: opening gap penalty $\alpha = 20$, extension gap penalty $\beta = 0.001$, and traceback search range of 80%. Unless otherwise noted, all estimation experiments use the same hyperparameter settings. The extension gap penalty was set to a very small value in order to reduce false positive regions resulting from chance matches.

Applying the functional site estimation to the TNNC1 genome sequence resulted in the detection of two motif regions. The estimated motif sites are shown in Figure 24. In the dataset sequence, the characters highlighted in red and blue indicate the CDS regions based on the annotation. The annotation information of the genome sequence (NG_008963.1) is shown in Figure 25. The red and blue regions labeled "CDS" in Figure 25 correspond to the respective regions highlighted in the dataset sequence in Figure 24. A comparison between the estimation results and the RefSeq annotation revealed that the detected motif sites are conserved within two CDS segments separated by an intron.

For the first motif site, the estimation indicated that it is conserved in two

D				{GP}			
a	0	1	0	a	0.381	0.427	0.207
c	0	0	0.617	c	0.178	0.231	0.408
g	1	0	0	g	0.163	0.095	0.242
t	0	0	0.383	t	0.278	0.246	0.142
{W}				{LIVMC}			
a	0.448	0.351	0.249	a	0.477	0	0.119
c	0.224	0.220	0.191	c	0.246	0	0.334
g	0.186	0.116	0.344	g	0.229	0.004	0.303
t	0.142	0.314	0.216	t	0.048	0.996	0.244
{DNS}				{DENGSTAGC}			
a	0.398	0.959	0.005	a	0.403	0.445	0.112
c	0	0.025	0.511	c	0.006	0.332	0.494
g	0.577	0.016	0.003	g	0.408	0.224	0.065
t	0.025	0	0.481	t	0.182	0	0.329
{ILVFW}				x(2)			
a	0.389	0.445	0.245	a	0.230	0.294	0.187
c	0.167	0.067	0.185	c	0.227	0.132	0.280
g	0.425	0.469	0.314	g	0.246	0.124	0.303
t	0.019	0.020	0.256	t	0.296	0.450	0.230
{DENSTG}				{DE}			
a	0.256	0.804	0.063	a	0	1	0.335
c	0	0.108	0.322	c	0	0	0.092
g	0.665	0.088	0.026	g	1	0	0.536
t	0.079	0	0.589	t	0	0	0.036
{DNQGHK}				{LIVMFYW}			
a	0.114	0.162	0.219	a	0.163	0.062	0.050
c	0.049	0	0.425	c	0.241	0	0.351
g	0.837	0.838	0.232	g	0.033	0.036	0.339
t	0	0	0.124	t	0.563	0.902	0.260

Figure 23: A codon weight matrix of EF-hand motif. A codon weight shows each pattern of the regular expression of EF-hand motif.

Human TNNC1 EF-hand motif①	
M : GACGAGGACG GTGAGCCCCCTCTCCCGAGGCTCCAGAAGAACCCAGCTGGCTGGGGCTGGAATGCTG	+++:+ +++:+
Q : gayvhnrayvrnra-----	
. . .	
M : GCAGTACCTACCCTGGCCTCTGTAACGGCCATGCTGCCACCCCGAG GCAGCGGCACGGTGGACTTTGAT	+ ++:+++ +++++:+:+
Q : -----ygrvnhvvtbdvynwnvmd	
M : GAGTTC	++ +
Q : garhtb	
Alignment score : 152.7	
Alignment start position : 6927	
Alignment end position : 7212	
segment : (6927 ... 6940) , (7188 ... 7212)	
Human TNNC1 EF-hand motif②	
M : GACAAGAACAACGACGGCCGATCGACTATGATG GTAAGCGGGTGGGTGGGCTGATCTCCTGCCTCCATG	+++:+ +++:+ ++: ++ +++++:+:+
Q : gayvhnrayvrnraygrvnhvvtbdvynwnvmdg-----	
M : CCCTGCCAGCCCTACCCTCAACCCACACCTGCCCTCTTCCACAG AGTTC	++ +
Q : -----arhtb	
Alignment score : 140.9	
Alignment start position : 7618	
Alignment end position : 7740	
segment : (7618 ... 7651) , (7736 ... 7740)	

Figure 24: Estimated result of EF-hand motif in the TNNC1 genome sequence (excerpt). Correct CDS region is colored red and blue in the dataset sequences.

LOCUS	NG_008963	9951 bp	DNA	linear	PRI 19-JAN-2014
DEFINITION	Homo sapiens troponin C type 1 (slow) (TNNC1), RefSeqGene (LRG_378) on chromosome 3.				
. . .					
CDS	join(5027..5050,6529..6559,6790..6936,7184..7298,7515..7651,7736..7767) /gene="TNNC1"				
. . .					
/translation="MDDIYKAAVEQLTEEQNEFKAAFDIFVLGAEDGCISTKELGKV MRMLGQNPTPEELQEMIDEVDEDEGS TVDFDEFL VMMVRCMKDDSGKSEEELSDLFR MF DKNADGYIDLDEL KIMLQATGETITEDDIEELMKDGD KNNDGRIDYDEFL EFMKGV E"					

Figure 25: Annotation of NG_008963.1 (excerpt). The '/translation' annotation shows a protein sequence. The motif ① information is colored red, and the motif ② information is colored blue. A green string in the protein sequence shows the EF-hand motif that failed to be estimated.

segments, positions (6927 $\cdots$ 6940) and (7188 $\cdots$ 7212). A comparison with the annotation revealed that the estimated region partially matched the beginning of the intron by four nucleotides. Thus, the motif and the intron region was not perfectly estimated. Nevertheless, the estimated start and end positions of the motif correctly estimated, even when the motif region conserved intron. For the second motif site, the estimation indicated two conserved segments at positions (7618 $\cdots$ 7651) and (7736 $\cdots$ 7740). This result was consistent with the annotation, including the intron region, demonstrating that the motif site was accurately estimated. In contrast, one additional EF-hand motif located at (7294 $\cdots$ 7298) and (7515 $\cdots$ 7548) in two segments. This site was not estimated under the parameter settings. In Figure 25, the corresponding motif site in the protein sequence is highlighted in green. This failure is likely due to the rarity of the residue pattern of EF-hand motif. The four residues in this motif are alanine, but the second nucleotide of the alanine codons is cytosine. As shown in the weight matrix of Figure 23. Specifically, at the fourth motif position, the frequency of cytosine at the second codon position is 0.067, which is below the contraction threshold. Consequently, the alignment score was low, preventing successful detection of this motif site.

Next, we examined hippocalcin (HPCA), another type of calcium-binding protein in which EF-hand motifs are conserved as a mixture of intron containing or not-containing. Figure 26 shows the three estimated motif sites, and Figure 27 shows the annotation information for the genome sequence (NG_042176.1).

The estimation results indicated that some motif sites are conserved entirely within a single exon, whereas others separate two exons. Comparison with the annotation information confirmed that each estimated site corresponded to the annotated EF-hand motif site.

These results demonstrate that the motif codon reduced representation can estimate motif sites conserved at the nucleotide level, even when intron is inserted within the motif. In contrast, motifs containing codons with very low occurrence frequencies could not be estimated. Although lowering the threshold of the motif codon reduced representation may estimate of such motifs, increasing the number of wildcard symbols may lead to the estimation of many sites that are likely non-related to the functional site.

Human HPCA EF-hand motif①
M : GACACCAACAGCGATGGCACCATAGACTTTCGGGAGTTC +++:+ +++:+ + ++:+++ +++:+:~+ + ++ + Q : gayvhnrayvrnraygrvnhvvtbdvynwnvmdgarhtb Alignment score : 146.0 segment : (8029 ... 8067)
Human HPCA EF-hand motif②
M : GACCTGGACGGGCAACGGGTACATGACCCGGGAGGAGATG +++:+ +++:+ + ++:+++ ++++: :+++ ++ + Q : gayvhnrayvrnraygrvnhvvtbdvynwnvmdgarhtb Alignment score : 158.0 segment : (8137 ... 8175)
Human HPCA EF-hand motif③
M : GACACAACAACGACGG GTAGGGGCGAGGGCGGGACGGGGTGGACGGGGCGGGCGCCTTCCCTCCCTCC +++:+ +++:+ +~ Q : gayvhnrayvrnray----- . . . M : TCGCCCTGCAG GCAAGCTGTCCTTGGAGGAGTTC ++:+++ ++++:~:+++ ++ + Q : -----grvnhvvtbdvynwnvmdgarhtb Alignment score : 152.8 segment : (12528 ... 12542) , (12678 ... 12701)

Figure 26: Estimated result of EF-hand motif in HPCA genome sequence (excerpt). The correct CDS region of a motif ③ is colored blue in the dataset sequence.

LOCUS	NG_042176	15560 bp	DNA	linear	PRI 19-JUN-2015
DEFINITION	Homo sapiens hippocalcin (HPCA), RefSeqGene on chromosome 1.				
. . .					
CDS	join(7813..8190, 12438..12543, 12679..12776)				
. . .					
/translation="MGKQNSKLRPEMLQDLRENTFSELELQEWYKGLKDCPTGILN VDEFKKIYANFFPYGDASKFAEHVFRFTD TNSDGTIDFREF I IALSVTSRGRLEQKLM WAFSMY DL DGN GYISREEM LEIVQAIYKMVSSVMKMPDESTPEKRTEKIFRQ MDTNN DGKLSLEEF IRGAKSDPSIVRLQLQCDPSSASQF"					

Figure 27: Annotation of NG_042176.1 (excerpt). The motif ① and motif ② information is colored red, and the motif ③ information is colored blue.

4.3.3 Functional site estimation using codon reduced representation for EF-hand motif of the other species

Motifs are well conserved not only at the amino acid level but also at the nucleotide level, even across different species. Therefore, it is considered that functional site estimation using motif codon reduced representations based on other species information. We attempted motif site estimation by applying the EF-hand motif codon reduced representation of human to several model species. From the RefSeq, we extracted the genome regions annotated as TNNC1 from the following species: mouse, rat, cow, pig, frog, and zebrafish (accessions: NC_000080.7, NC_051351.1, NC_037349.1, NC_010455.5, NC_030680.2, and NC_007134.7). Estimation experiments were conducted using only these extracted annotated regions. Because the pig genome sequence is registered on the complementary strand, it was converted into the original sequence.

For mouse, rat, cow, and pig, we estimated that two EF-hand motifs are conserved across intron regions, as observed in case of the human experiments. The first motif was estimated in all four species, consistent with the results of the human experiments, in which a partial match to the first four nucleotides of the intron was estimated. For the second motif, a deviation from the human experiments was found in mouse, rat, and pig. In these species, the 33rd nucleotide of the motif site changed from thymine to cytosine. Because of this substitution, the wildcard symbol d failed to match, resulting in discrepancies between the estimation and the annotation. As an example, the estimated result of the second motif in mouse is shown in Figure 28. For comparison, the Human estimation is shown, and the differing positions are highlighted with green boxes. According to the annotation, the start and end positions of the motif were correctly estimated. Figure 29 shows the annotation information for mouse.

In zebrafish, only the second motif was estimated. For frog, none of the motifs could be estimated. One possible reason for the failure to estimate certain motifs, despite their conservation in the sequences, is that the algorithm could not handle the very long introns. We conducted estimation experiments for frog and zebrafish with the extension gap penalty set to zero, while keeping all other hyperparameters unchanged. As a result, although many non-functional sites were estimated, the second EF-hand motif was successfully estimated in both species. Figure 30 shows a result of frog in which

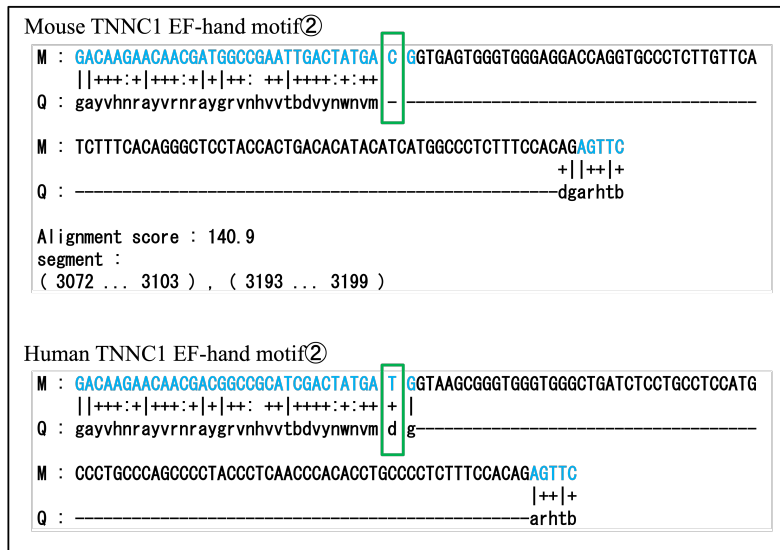


Figure 28: Comparison of estimated result of the EF-hand motif ② in TNNC1 genome sequence between mouse and human. In mouse, unlike in case of Human, 33rd base is changing from thymine to cytosine (see green boxes in the Figure).

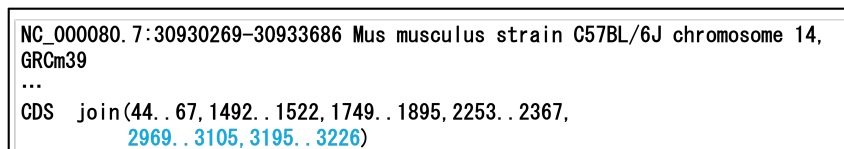


Figure 29: Annotation of TNNC1 genome sequence for mouse (excerpt). The regions are colored blue shows that are conserved the mouse TNNC1 EF-hand motif ②.

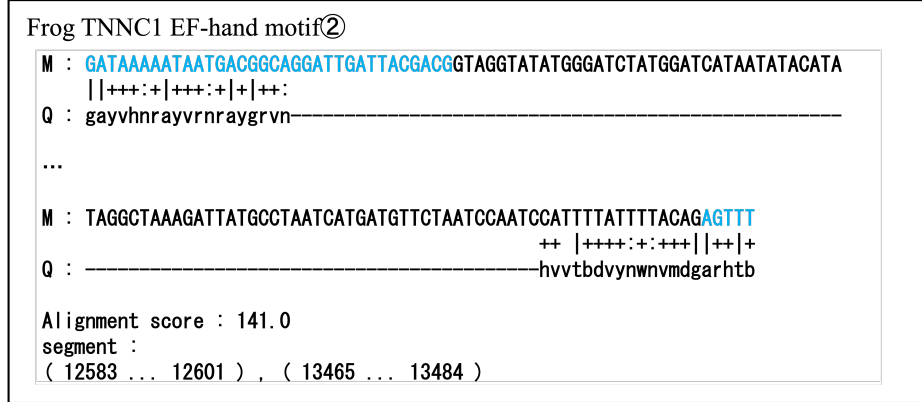


Figure 30: Successful estimated result by changing to the extension gap is zero in the TNNC1 genome sequence for frog (excerpt). Correct CDS region is colored blue in the dataset sequence.

estimation succeeded when the extension gap penalty was set to zero. The EF-hand motif that could not be estimated in the human experiments also failed estimation in all other species. In every species, the fourth residue of this motif is alanine.

Similarly, for HPCA, we conducted estimation experiments using only the extracted genome regions corresponding to the HPCA gene from the six model species (NC_000070.7, NC_051340.1, NC_037329.1, NC_010448.4, NC_030678.2, and NC_007130.7). Because the genome sequences for all species except pig were registered on the complementary strand, they were converted to original sequence.

For mouse and rat, all three motifs were successfully estimated, as in the human experiments. For cow, pig, frog, and zebrafish, the first motif was correctly estimated. However, the second motif could not be estimated in cow or pig, the third motif could not be estimated in frog, and neither the second nor the third motif could be estimated in zebrafish. For the third motif in mouse, rat, cow, and pig, the estimations indicated an intron insertion between the 15th and 16th nucleotides, consistent with the human experiments. Figure 31 shows the estimated results for these third motifs, each containing an intron insertion. Comparison with the corresponding annotation information confirmed that the estimated regions correctly correspond to the conserved motif sites. In frog and zebrafish, the annotations indicate that the third motif contains a very long intron. Although we repeated the experiments with the extension gap penalty set to zero, the motif could not be correctly estimated. the estimation results revealed that, although the region before the intron

matched the motif, the alignment inserted gaps indefinitely until reaching a distant high-scoring region after the intron. Setting the extension gap penalty to zero enables handling of very long introns, but also allows endlessly long gap extensions.

The functional site estimation experiments confirmed that the length of introns inserted within EF-hand motifs varies depending on the protein and the specific conserved position within the gene. These introns range from about 10 to about 1000 nucleotides different across species. But, the positions at which they are inserted were found to be conserved among species. For TNNC1, the first EF-hand motif contains an intron inserted between the 10th and 11th nucleotides, and the second motif contains an intron between the 34th and 35th nucleotides. For HPCA, the third motif contains an intron inserted between the 16th and 17th nucleotides. The vertebrate troponin C gene is known to contain five introns, and except for the first and fourth introns, the insertion positions are conserved even across different species [79]. Furthermore, previous studies have shown that, in the genome sequences of certain calcium-binding proteins, introns are inserted within the loop region of the helix-loop-helix structure, corresponding to the EF-hand motif defined in PROSITE [80, 81]. In both the TNNC1 and HPCA genome sequences, EF-hand motifs were conserved regardless of species, and the intron insertion positions were fixed. These estimation results can be considered valid in correct of previous studies. While the presence of intron inserted EF-hand motifs in the TNNC1 genome sequence has been previously documented, our research provides new information on the conserved positions of intron inserted EF-hand motifs in the TNNC1 and HPCA genome sequences across the seven species. In addition, the intron insertion regions were shared across all species. This information regarding intron inserted EF-hand motifs is not available in existing databases and represents a novel contribution of this study.

4.4 Concluding Remarks

In this chapter, we discussed a sequence alignment system based on dynamic programming (DP) for functional site estimation in genome sequence. To implement this estimation system, we proposed an approximate representation of the codon reduced representation and extended the substitution matrix to enable its use in sequence alignment. Furthermore, using codon reduced

Mouse HPCA EF-hand motif③ M : GACACAAACAATGACGGTGAGTGGCGGGGGGGGGGGGGAGATTGGGGGGGAGTGATAGGGGTGGAG +++:+ +++:+ + Q : gayvhnrayvrnray----- ... M : TTACAGCCCTCCCTTTGCCCTGCAGGCAAGCTGTCTGCTGGAGGAGTTC ++:+++ +++ :+:+++ ++ + Q : -----grvnhvvtbdvynwnvmdgarhtb Alignment score : 137.8 segment : (13230 ... 13244) , (13394 ... 13417)
Rat HPCA EF-hand motif③ M : GACACAAACAATGACGGTGAGGGGCGGGGGATGGGGGCGGAGGAGTGGGGGCGGGGCGGGGCGGAA +++:+ +++:+ + Q : gayvhnrayvrnray----- ... M : TCTCAGCCCTCCCTTCTCTCTGCAGGCAAGCTGTCACTGGAGGAGTTC ++:+++ +++ :+:+++ ++ + Q : -----grvnhvvtbdvynwnvmdgarhtb Alignment score : 137.8 segment : (9510 ... 9524) , (9674 ... 9697)
Cow HPCA EF-hand motif③ M : GACACAAACAACGACGGTGAGGAGGCAGGGGCGGGGCGGGCGGGGCGCCACTCCCACTTCCTATGCCGCAG +++:+ +++:+ + Q : gayvhnrayvrnray----- M : CTCTGCTCGGCCACCCCTAGCTGCTGTTGCCTTCTCCCCCTTCATTGCTCCCCGCCTCGCCCCGAGGC ++ Q : -----grv M : AAGCTGTCCCTGGAGGAGTTC :+++ +++++:+++ ++ + Q : nhvvtbdvynwnvmdgarhtb Alignment score : 152.8 segment : (7634 ... 7648) , (7771 ... 7794)
Pig HPCA EF-hand motif③ M : GATACAAACAACGACGGTGAGGAGGCCGGGGCGGGGCGGGCGCCACTCCCCCTTCTCTTCTGCGGTCT +++:+ +++:+ + Q : gayvhnrayvrnray----- M : GCTCTGCGTCCCCTCTCTTTGCTGCCGCTTCTCCCCATCATTGCGCCCTTGTCTGCGAGGCAAGCTG ++:+++ + Q : -----grvnhvvtb M : TCCCTGGAGGAGTTC +++:+:+++ ++ + Q : dvynwnvmdgarhtb Alignment score : 152.8 segment : (8375 ... 8389) , (8506 ... 8529)

Figure 31: Successful estimated results of the motif ③ in HPCA genome sequence for mouse, rat, cow and pig. Correct CDS region is colored blue in the dataset sequences of each species.

representations and its approximation, we proposed a motif codon reduced representation that quantitatively transforms the gene sequence information of protein sequence motifs.

Using the motif dictionary system proposed in Chapter 3, we transformed the gene sequences of EF-hand motifs into the motif codon reduced representation and conducted estimation experiments on the TNNC1 and HPCA genome sequence of seven species. In estimation experiments on the human TNNC1 genome sequence, two of the three EF-hand motif sites were correctly estimated even when introns were inserted. In contrast, the remaining motif could not be estimated using the proposed representation, as its sequence pattern was rare. In the estimation results on the human HPCA genome sequence, all EF-hand motif sites were correctly estimated. In estimation experiments using the TNNC1 genome sequences of other model species, the results for mouse, rat, and pig were consistent with those for the human estimation experiments: two of the three motif sites were correctly estimated. In estimation experiments on frog and zebrafish, some motif sites were correctly estimated by adjusting the estimation parameters. Finally, in estimation experiments on the HPCA genome sequences of other model species, all motif sites were successfully estimated in both mouse and rat. However, for the other model species, several motif sites could not be estimated even after adjusting the estimation parameters. These results demonstrate that the proposed system and the motif codon reduced representation are effective for estimating protein sequence motif sites within genome sequences.

However, accurate motif site estimation requires careful tuning of the motif codon reduced representation and estimation parameters. In the estimation experiments in this research, the gap penalty and the thresholds for the motif codon reduced representation were determined empirically. Since this tuning is currently performed manually, there is a need to design a system that can automatically optimize these parameters. For example, as the simplest approach, grid search can be used to exhaustively explore combinations of parameters. In recent years, the use of libraries for efficient hyperparameter tuning of large black box models such as deep learning has also been proposed. Optuna [82] and Tune [83] are widely used for this purpose. In addition, focusing on codon or nucleotide usage frequencies is considered effective for optimizing codon reduced representations and substitution matrices. A comparative analysis of the genomes of 33 mammalian species revealed that

several mammals (i.e. microbat and rabbit) exhibit a markedly higher proportion of GC usage compared with other species [84]. Recently, a codon usage database covering more than 15,000 species has been developed [85]. If such statistical information can be incorporated as weights into the thresholds of motif codon reduced representations and the scores of substitution matrices, more accurate estimations are expected to be achieved.

5 Computing Hamming Distance and Levenshtein Distance Using ReLU Neural Networks

In this chapter, we discuss a neural network model using only the ReLU activation function to calculate the Hamming distance and the Levenshtein distance. This chapter is based on the content of my third published papers. Please refer to the third item in the List of Publications at the end of this thesis.

5.1 Background

Knowing how similar two given strings are is a major area of interest within informatics. Hamming distance is a simple string similarity score defined by counting unmatched characters between two input strings of the same length [86]. Levenshtein distance is a string similarity score that measures the minimum of substitutions, insertions and deletions required to transform one input string into another [39]. It is known that this distance can be calculated using Dynamic Programming (DP) by constructing a cost (score) matrix [87], as well as alignment score in bioinformatics [15, 16].

Due to the non-differentiability of DP, applying neural networks with the backpropagation algorithm becomes challenging. Recently, several methods have been proposed to develop a differentiable DP algorithm for effectively training neural networks. Mensch and Blondel proposed a differentiable DP approach for time series data [88]. They replaced the max function, used in calculating the score matrix, with a logsumexp function. Libovický and Fraser proposed a word translation model utilizing the differentiable DP, where the max function is replaced with the sum of the probabilities of edit operations [89]. Morton et al. and López et al. proposed a differentiable DP approach for the alignment of protein sequences, which was then applied to a deep learning model, respectively [90, 91]. These models learn the cost of edit operations from sequences using pre-trained models, such as Bi-Directional Long Short Term Memory (Bi-LSTM) and Transformer [92, 93]. Additionally, Petti et al. developed a differentiable DP method for multiple sequence alignment, achieving comparative analysis of multiple sequences against a query sequence [94]. In this method, a differential DP for pairwise sequences was combined with the Markov random field to learn multiple sequence alignments. A convolutional neural network (CNN) model was proposed for learning string dis-

tance [95]. This model learns a one-hot encoding matrix of sentences, without utilizing the differentiable DP approach.

If character matching, addition, and maximum or minimum functions can be achieved using a neural network with sigmoid or similar functions, the calculation of Hamming distance and Levenshtein distance would be replicated with a neural network. It might be possible to implement the neural network by appropriately combining them, but this would make the resultant network considerably deep, and learning would be quite difficult. Therefore, we consider using the Rectified Linear Unit (ReLU) activation function, which is less prone to gradient vanishing even in deep layers, making it a viable option for learning. In previous studies, various formulations have been attempted to make dynamic programming differentiable, aiming to adapt it to neural networks. However, a network model using only ReLU activation has not been proposed. If the network model can compute distances exactly, then inferring input sequences becomes feasible. This task is difficult for conventional methods but can be formulated as a simple model learning problem using the network model.

We propose neural network models using only ReLU activation for calculating Hamming distance and Levenshtein distance. Furthermore, we extend the models for learning a string from a set of strings and their respective distances to the target string. Additionally, we implement the models using Python and perform computational experiments.

5.2 Method

5.2.1 Hamming distance and Levenshtein distance

Hamming distance is a metric that counts the number of different characters between two strings of length n . The similarity between the strings is higher when this count is lower. Due to the comparison of each character, the time complexity of calculating Hamming distance is $O(n)$. For instance, considering two strings $x = 101$ and $y = 010$, the Hamming distance is three since all characters are different, where the distance between two different characters is one. Although Hamming distance is straightforward to compute, it is limited to comparing strings of the same length and only allows for comparisons of characters in corresponding positions.

Levenshtein distance, also known as edit distance (under the unit cost model), is defined as the minimum cost to convert one input string to another

through three editing operations. This is applicable even if the input strings, x and y , have different lengths m and n , respectively. The three operations are defined as follows: substitution, insertion, and deletion of a single character. The substitution operation involves converting a character in one string to the corresponding character in the other string. For instance, let us consider two strings, $x = 101$ and $y = 010$. In this case, the Levenshtein distance between these strings is two, as we can obtain string y from x deleting the first character '1' and inserting '0' at the end of x . It is important to note that the cost associated with insertion and deletion operations is defined as one in this particular example. It is worth mentioning that each operation cost can be assigned individual values, allowing for flexibility in the Levenshtein distance calculation based on the specific requirements of the application.

In order to calculate a score matrix using DP for Levenshtein distance, it is necessary to compute the edit cost for all combinations of characters in the strings x and y . As a result, the time complexity of the Levenshtein distance algorithm is $\Omega(nm)$. Furthermore, it is known that even for a fixed-size alphabet, computation of the Levenshtein distance cannot be done in $O(n^{2-\epsilon})$ time for any constant ϵ under a reasonable assumption on the computational complexity theory [96], where the size of input strings is around n .

5.2.2 ReLU neural network

A ReLU neural network is a feed-forward network that employs the ReLU function in each unit. A feed-forward network computes from input to output without any feedback loops. Neural networks typically comprise an input layer, multiple hidden layers, and an output layer. The ReLU function is a nonlinear activation function that outputs zero for negative values and retains positive values as they are (See Section 2.5) [41]. During training, the weights in a neural network are learned by calculating gradients through the backpropagation algorithm. However, when a neural network contains many hidden layers, the gradients tend to converge to zero, particularly based on the activation function used. The ReLU function is less likely to suffer from gradient vanishing in networks with multiple hidden layers since its derivative is equal to one for positive values. Hence, the ReLU function is often favored in deep learning architectures that involve many hidden layers, providing a practical solution to mitigate the vanishing gradient problem [40].

The ReLU neural network is a unidirectional computational model that

processes information from input to output. Conceptually, it can be visualized as a directed graph structure without any loop edges, resembling a feedforward architecture. In this graph representation, each edge is assigned a weight value that influences the computation within the network. Hence, the ReLU neural network can be viewed as a weighted directed acyclic graph (DAG) due to its directed nature and the presence of weight values on the edges.

5.2.3 ReLU neural network for calculating Hamming distance

In this section, we present a ReLU neural network for calculating Hamming distance. Throughout this chapter, we assume without loss of generality that each character is represented as a non-negative integer number.

To calculate Hamming distance, a ReLU neural network can be implemented by sequentially adding the outputs of each subnetwork designed for character matching. Alternatively, the network can be realized by summing the outputs of each character-matching subnetwork. The network's final output corresponds to the Hamming distance between the two given input strings.

First, we define a ReLU neural network for character matching, which is based on [97]. The network equations are written as:

$$\begin{aligned} u_{i,j}^1 &= \max(x_i - y_j, 0), \\ u_{i,j}^2 &= \max(y_j - x_i, 0), \\ v_{i,j}^1 &= \max(u_{i,j}^1 + u_{i,j}^2 + \epsilon, 0), \\ v_{i,j}^2 &= \max(u_{i,j}^1 + u_{i,j}^2 - \epsilon, 0), \\ \delta &= 2 - \frac{1}{\epsilon}(v_{i,j}^1 - v_{i,j}^2). \end{aligned}$$

where x and y are input strings, whereas i and j indicate positions of x, y . $\max(a, 0)$ is a function that outputs the larger value of a or zero. δ represents the result of character matching, that is, $\delta = 1$ if $x_i = y_j$, otherwise $\delta = 0$. This property is guaranteed if x and y are non-negative integers and ϵ is a positive constant such that $\epsilon \leq 1$.

Next, we consider the implementation of the aforementioned equations using a ReLU neural network. The ReLU function $\text{ReLU}(a)$ is employed, as its output is equivalent to that of the max function $\max(a, 0)$. The ReLU subnetwork designed to determine character matches or mismatches is defined as

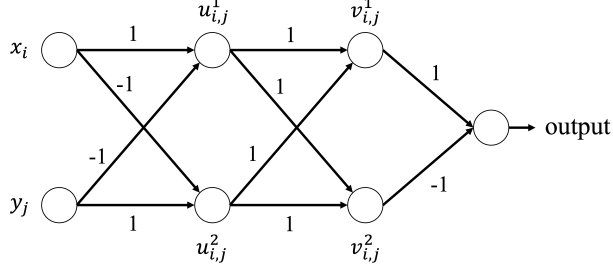


Figure 32: The ReLU subnetwork for character matching

follows:

$$\begin{aligned}
u_{i,j}^1 &= \text{ReLU}(x_i - y_j), \\
u_{i,j}^2 &= \text{ReLU}(y_j - x_i), \\
v_{i,j}^1 &= \text{ReLU}(u_{i,j}^1 + u_{i,j}^2 + \epsilon), \\
v_{i,j}^2 &= \text{ReLU}(u_{i,j}^1 + u_{i,j}^2 - \epsilon), \\
\text{output} &= \text{ReLU}\left(\frac{1}{\epsilon}(v_{i,j}^1 - v_{i,j}^2) - 1\right).
\end{aligned}$$

The output is zero if input characters match (i.e., $x_i = y_j$), otherwise it is one. We use $\epsilon = 1$ in our implementation. Figure 32 illustrates the ReLU subnetwork for character matching. This module is referred to as the *matching module*.

If the length of the inputs is one, the Hamming distance is equal to the output of the ReLU neural network designed to determine whether character matches or mismatches. Additionally, if the length of the inputs is greater than or equal to two, the Hamming distance is the sum of the outputs from the ReLU neural network for character matching, computed for each character in the input strings. Figure 33 illustrates the ReLU neural network for calculating Hamming distance.

It is obvious that the above ReLU neural network can compute the Hamming distance between x and y of length n . Furthermore, the number of required ReLU units is $O(n)$ because the size of each matching module is $O(1)$ and we need only one output unit that computes the sum of the outputs of matching modules. Therefore, we have the following.

Proposition 1. *The Hamming distance between two strings of length n can be computed by a ReLU neural network using $O(n)$ units.*

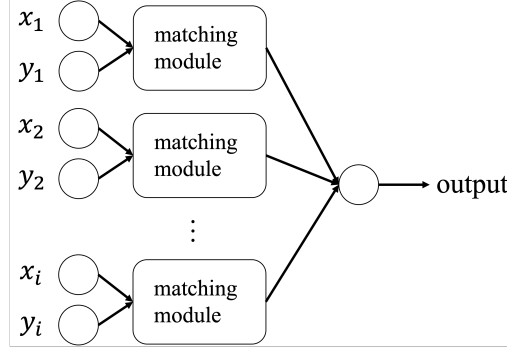


Figure 33: The ReLU neural network for calculating the Hamming distance. The matching module indicates the ReLU subnetwork for character matching

5.2.4 Calculating Levenshtein distance using DP

Before presenting our proposed ReLU neural network for Levenshtein distance, we briefly review the DP equations for calculating the Levenshtein distance. For two input strings, x and y whose length is m and n , respectively, the size of the score matrix, denoted as S , is $(m + 1)(n + 1)$. The computation of each element in S is carried out as follows:

$$S(i, j) = \min \begin{cases} S(i - 1, j - 1) + \text{cost}(x_i, y_j) & \text{(substitution),} \\ S(i - 1, j) + 1 & \text{(insertion),} \\ S(i, j - 1) + 1 & \text{(deletion),} \end{cases}$$

$$\text{cost}(x_i, y_j) = \begin{cases} 0 & (x_i = y_j), \\ 1 & (x_i \neq y_j). \end{cases}$$

where $i(1 \leq i \leq m)$ and $j(1 \leq j \leq n)$ represent positions of x and y , respectively. The element $S(0, 0)$ is initialized to zero. In addition, $S(i, 0)$ and $S(0, j)$ are set to i and j , respectively. Then, $S(m, n)$ gives the Levenshtein distance between x and y . It is worthy to note that the Levenshtein distance corresponds to the edit distance under the unit cost model, that is, the cost of each editing operation is 1. Therefore, throughout this chapter, we assume the unit cost model.

5.2.5 ReLU neural network for calculating Levenshtein distance

We consider the implementation of a ReLU neural network for calculating the Levenshtein distance. When computing the score matrix S using DP, the

Levenshtein distance is calculated sequentially in a one-way manner, starting from the initial element $S(0, 0)$ and progressing to the terminal element $S(m, n)$. The sequential calculation process can be transformed into a ReLU neural network. However, it is important to note that the calculation of each element in the score matrix requires the utilization of the ReLU neural network both for determining the minimum value and for character matching operations. In the character matching process, we calculate the cost associated with the substitution operation, while the minimum value determination corresponds to the computation of elements in the score matrix, specifically, $S(i, j)$. If a ReLU neural network capable of determining the minimum value can be successfully implemented, the entire calculation of the Levenshtein distance can be expressed in the form of ReLU neural networks.

We define a ReLU neural network for determining the minimum value among two or three inputs. For the case of three inputs, the minimum value is either the minimum value of two inputs or the remaining input. The equations for determining the minimum value among two and three inputs in the neural network are defined as follows:

$$\begin{aligned}\max(x, y) &= \frac{1}{2}(\max(x - y, 0) + \max(y - x, 0) + x + y), \\ \min(x, y) &= -\max(-x, -y), \\ \min(x, y, z) &= \min(\min(x, y), z).\end{aligned}$$

where x, y, z are inputs. Similar to the approach used for character matching, we replace $\max(a, 0)$ with the ReLU function $\text{ReLU}(a)$. The equations for the ReLU neural network to determine the minimum value are as follows:

$$\begin{aligned}\max(x, y) &= \text{ReLU}\left(\frac{1}{2}(\text{ReLU}(x - y) + \text{ReLU}(y - x) \right. \\ &\quad \left. + x + y)\right), \\ \min(x, y) &= \text{ReLU}\left(-\frac{1}{2}(\text{ReLU}(y - x) \right. \\ &\quad \left. + \text{ReLU}(x - y)) + \frac{1}{2}(x + y)\right), \\ \min(x, y, z) &= \text{ReLU}\left(-\frac{1}{2}(\text{ReLU}(\min(x, y) - z) \right. \\ &\quad \left. + \text{ReLU}(z - \min(x, y)) \right. \\ &\quad \left. + \frac{1}{2}(\min(x, y) + z))\right).\end{aligned}$$

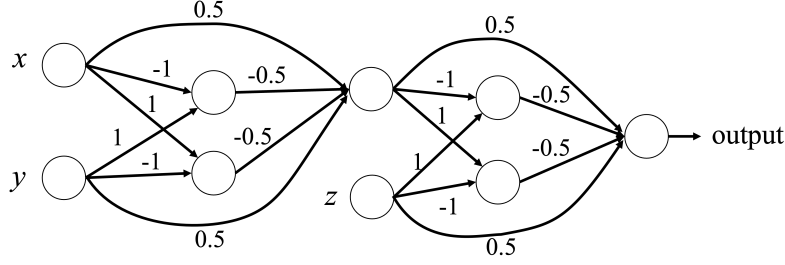


Figure 34: The ReLU neural network for determining the minimum value of three inputs

Figure 34 illustrates the ReLU subnetwork for determining the minimum value of the three inputs. This module is referred to as the *minimum module*.

By appropriately connecting the ReLU neural network for character matching and the ReLU neural network for determining the minimum value, we can construct a ReLU neural network for calculating Levenshtein distance calculation. For each $S(i, j)$ with $i > 0$ and $j > 0$, we construct a node (i.e., a unit) $v_{i,j}$ to retain the value of $S(i, j)$. The cost for x_i and y_j can be computed by a matching module explained in Section 5.2.3. We construct a node $v'_{i-1,j-1}$ to compute $S(i-1, j-1) + \text{cost}(x_i, y_j)$. Then, we use the minimum module to compute the minimum value of $v'_{i-1,j-1}$, $v_{i-1,j} + 1$, and $v_{i,j-1} + 1$, where the output of this minimum module corresponds to a node $v_{i,j}$. For computation of each of $S(i, 0)$ and $S(0, j)$, we simply construct a subnetwork to compute $v_{i,0} = v_{i-1,0} + 1$ and $v_{0,j} = v_{0,j-1} + 1$, respectively, where some special treatment is done for computing $S(1, 1)$.

Finally, the network structure is arranged so that $S(i, j)$ s are computed along the reverse diagonal, i.e., computing $S(i, j)$ s from $h = 2$ to $h = m + n$ where $h = i + j$. When dealing with input strings of different lengths, situations may arise where i or j exceeds the string length. In such cases, this issue can be resolved by skipping the construction of a part of the network. Figure 35 illustrates the ReLU neural network for calculating Levenshtein distance.

It is obvious that the ReLU neural network constructed above can correctly compute the Levenshtein distance for given x and y . Furthermore, since each of the matching module and the minimum module uses $O(1)$ units, the total number of the units is $O(mn)$. Therefore, we have the following.

Theorem 1. *The Levenshtein distance between two strings with lengths m and n can be computed by a ReLU neural network using $O(mn)$ units.*

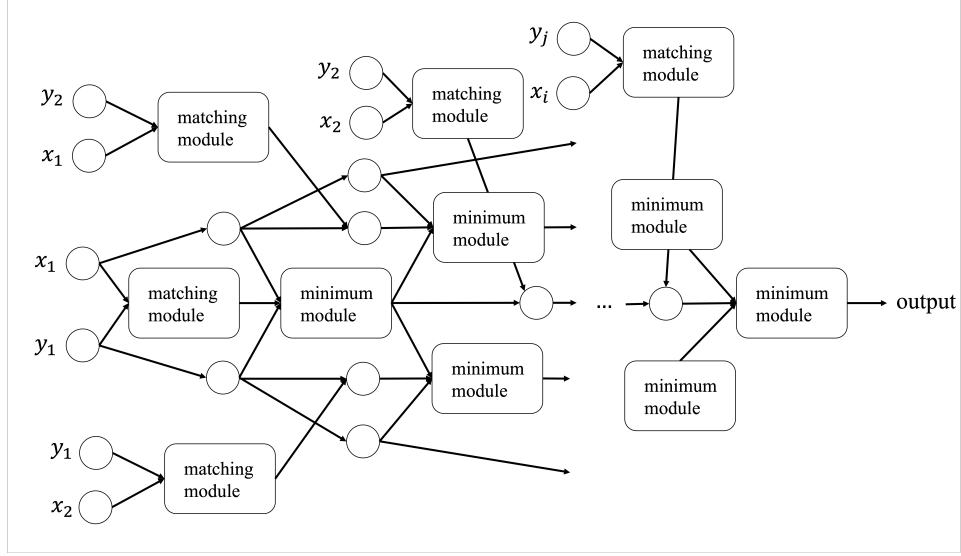


Figure 35: The ReLU neural network for calculating Levenshtein distance. The matching module indicates the ReLU subnetwork for character matching or mismatch detection, and the minimum module indicates the ReLU subnetwork for determining the minimum value of three inputs

5.2.6 ReLU neural network for learning

One of the advantages of using neural networks is that once a network for computing some value is constructed, it can be used for learning some parameters using well established optimization techniques and tools like the stochastic gradient decent algorithm.

We modify the ReLU networks for computing Hamming and Levenshtein distances for leaning of a string from given strings and their distances. This learning problem is formalized as below.

Input: strings $x^1, \dots, x^N$ and distances $d^1, \dots, d^N$.

Output: string y such that $dist(x^i, y) = d^i$ for all $i = 1, \dots, N$.

where $dist(x, y)$ denotes either Hamming distance or Levenshtein distance. It seems that this problem is computationally difficult because a similar problem is known to be NP-hard [98].

In this chapter, we only consider the case of binary strings (i.e., strings over an alphabet $\{0, 1\}$) because unless otherwise the ReLU network architecture would be much more complicated (e.g., we may need one-hot encoding to represent strings). To learn a string y , we represent y using weights. We use two units u_i and v_i to represent each character y_i , where u_i is a constant node

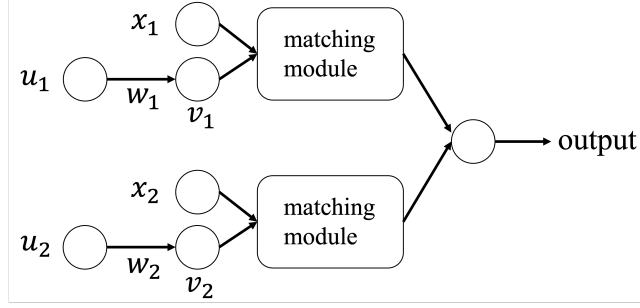


Figure 36: The ReLU neural network for learning Hamming distance (input length is two)

always outputting 1, and the weight w_i from u_i to v_i corresponds to y_i , that is, w_i should be 1 or close to 1 if $y_i = 1$, otherwise w_i should be 0 or close to 0. For instance, when the Hamming distance computation ReLU neural network is provided with data $x = 0101$ and a distance of two, one of the potential solutions for the learned weights would yield $y = 1100$ as an output. In other words, the learned weights will be $w_1, w_2 = 1$ and $w_3, w_4 = 0$. This approach enables the ReLU neural network to effectively learn and adapt its weights, generating appropriate strings that satisfy the desired distance constraints.

We utilize the following as the loss function:

$$L = \frac{1}{n} \sum_{i=1}^n (d_i^{true} - d_i^{pred})^2$$

where n is number of data, d_i^{true} is a desired distance between y and x^i , and d_i^{pred} is a predicted distance between y and x^i . To ensure that each ReLU neural network outputs the correct distance, we freeze all parameters, except weights w_i s corresponding to y_i s. Figures 36 and 37 illustrate ReLU neural networks for learning a string y under Hamming distance and Levenshtein distance, respectively, when the length of each string is two. It should be noted that these networks can also be used to compute Hamming distance and Levenshtein distance if we fix w_i to be y_i .

5.3 Reducing the number of layers

We have shown that Levenshtein distance can be computed using a ReLU neural network using $O(mn)$ nodes. However, the number of layers would be large ($O(m + n)$) if we rearrange the network structure to be a layered one. Therefore, one simple question arises: can the number of layers be substantially reduced? In this section, we present two theoretical results to positively

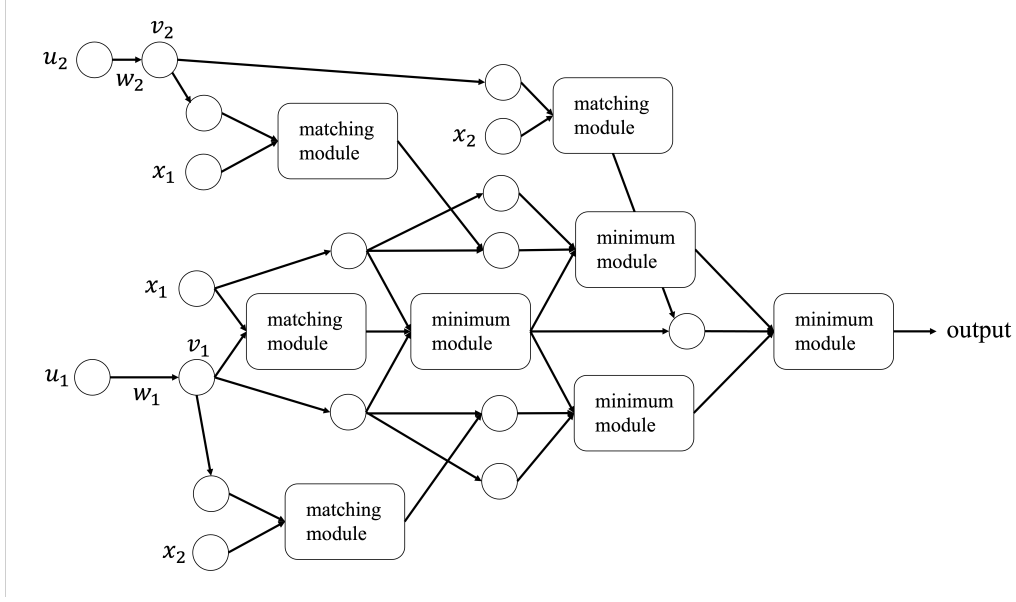


Figure 37: The ReLU neural network for learning Levenshtein distance (input length is two)

answer this question while the number of nodes increases substantially.

5.3.1 General case

In this subsection, we consider the general case and show that the number of layers can be reduced to $O(\log(m+n) \log(mn))$ while increasing the number of nodes to $O(n^3 m^3 \log(m+n))$. Recall that two input strings are given as $x = x_1 \dots x_m$ and $y = y_1 \dots y_n$.

It is well-known that the Levenshtein distance problem can be transformed into the shortest path problem, which can be done as follows. Let $I_0 = \{0, 1, \dots, m-1\}$, $I_1 = \{0, 1, \dots, m\}$, $J_0 = \{0, 1, \dots, n-1\}$, and $J_1 = \{0, 1, \dots, n\}$. We define a directed graph $G(V, E)$ by

$$\begin{aligned}
 V &= \{v_{i,j} \mid i \in I_1, j \in J_1\}, \\
 E &= \{(v_{i,j}, v_{i+1,j}) \mid i \in I_0, j \in J_1\} \\
 &\quad \cup \{(v_{i,j}, v_{i,j+1}) \mid i \in I_1, j \in J_0\} \\
 &\quad \cup \{(v_{i,j}, v_{i+1,j+1}) \mid i \in I_0, j \in J_0\}.
 \end{aligned}$$

Furthermore, the length of each edge $w(\dots)$ is given as follows.

$$\begin{aligned} w(v_{i,j}, v_{i+1,j}) &= 1, & \text{for } i \in I_0, j \in J_1, \\ w(v_{i,j}, v_{i,j+1}) &= 1, & \text{for } i \in I_1, j \in J_0, \\ w(v_{i,j}, v_{i+1,j+1}) &= \begin{cases} 0 & \text{if } x_i = y_j, \\ 1 & \text{if } x_i \neq y_j, \end{cases} & \text{for } i \in I_0, j \in J_0. \end{aligned}$$

Then, the Levenshtein distance between x and y is equal to the length of the shortest path from $v_{0,0}$ to $v_{m,n}$. Although we assume the unit cost model, i.e., Levenshtein distance, for the simplicity of explanation, it is straightforward to modify the method for more general edit distance models in which a positive cost is assigned to each edit operation.

Here, we consider a parallel version of the Floyd-Warshall algorithm [99, 100] that computes the lengths of shortest paths for all pairs of nodes. The pseudo-code of this algorithm is given below.

Procedure *ParallelFloydWarshall*($G(V, E)$)
for all $(u, v) \in V \times V$ **do** in parallel $D^0(u, v) \leftarrow \infty$ **od**
for all $v \in V$ **do** in parallel $D^0(v, v) \leftarrow 0$ **od**
for all $(u, v) \in E$ **do** in parallel $D^0(u, v) \leftarrow w(u, v)$ **od**
for $h = 0$ **to** $\lceil \log(m+n) \rceil$ **do** in parallel
 for all $(u, v) \in V \times V$ **do**
 $D^{h+1}(u, v) \leftarrow \min_{w \in V} (D^h(u, w) + D^h(w, v))$ ($\#$)
 od
od

Note that $D^h(u, v)$ finally stores the shortest path length between u and v if it is at most 2^h , otherwise it stores $+\infty$. Therefore, $\lceil \log(m+n) \rceil$ repetitions are enough in our case because the maximum path length is $m+n$.

We can implement this algorithm using a layered ReLU network as follows.

- The input layer consists of x_i ($i \in \{1, \dots, m\}$) and y_j ($j \in \{1, \dots, n\}$).
- For each h , we construct a layer consisting of nodes $D_{u,v}^h$, each of which will have the value of $D^h(u, v)$.
- To compute $D^h(u, w) + D^h(w, v)$, we construct a node $D_{u,w,v}^h$.
- The output (edit distance) is given by the node $D_{v_{0,0}, v_{m,n}}^{\lceil \log(m+n) \rceil}$.

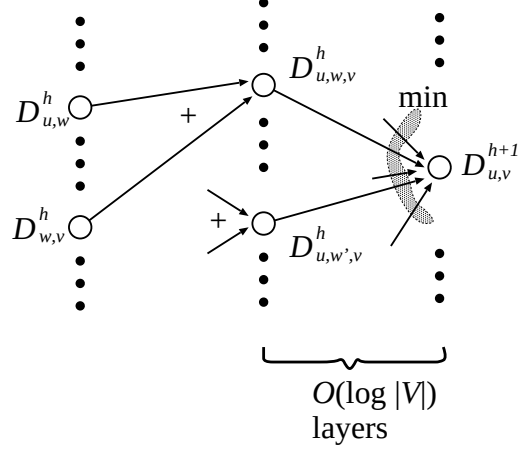


Figure 38: ReLU network architecture for computing (#) part in $ParallelFloydWarshall(G(V, E))$.

Computation of each $D_{u,v}^0$ can be done as in Figure 32 using $O(1)$ layers and $O(1)$ nodes. Thus, computation of all $D_{u,v}^0$ s can be done using $O(1)$ layers and $O(|V|^2)$ nodes. Accordingly, the non-trivial part is how to compute the part of (#) using a ReLU neural network (see Figure 38). Since we only consider positive lengths, $D_{u,w}^h + D_{w,v}^h$ can be obviously computed using one ReLU unit per (u, w, v) . As mentioned in Section 5.2.5, $\min(p, q)$ can be computed by using a constant number of ReLU units. Thus, $\min(p_1, \dots, p_{|V|})$ can be computed using $O(\log |V|)$ layers and $O(|V|)$ nodes by repeatedly applying the following construction:

$$\min(p_1, \dots, p_N) = \min(\min(p_1, \dots, p_{\lceil N/2 \rceil}), \min(p_{\lceil N/2 \rceil+1}, \dots, p_N)).$$

Therefore, we can compute (#) for all (u, v) s using $O(\log |V|) = O(\log(mn))$ layers and $O(|V|^3) = O(m^3n^3)$ nodes. Since we repeat this construction for $O(\log(m+n))$ layers, we have the following theorem.

Theorem 2. *The Levenshtein distance between two strings x and y with $|x| = m$ and $|y| = n$ can be computed by a ReLU neural network using $O(\log(m+n) \log(mn))$ layers and $O(m^3n^3 \log(m+n))$ units.*

Although we have significantly reduced the number of layers by this theorem, the number of units is much larger than that of the ReLU neural network in Theorem 1. Therefore, this result is only of theoretical interest.

5.3.2 Case with a limited number of insertions and deletions

Since the number of nodes of the network constructed in Section 5.3.1 is too large, we consider a special case in which the sum of insertion and deletion operations is limited. Although sophisticated algorithms have been developed for such cases [101], they are too complex to be implemented as layered neural networks. Therefore, we present a simple dynamic programming-based implementation.

The purpose of this section is to compute the Levenshtein distance between x and y under the constraint that the number of insertion and deletion operations is at most K . For the simplicity, we assume without loss of generality that $m \leq n$.

First, for $1 \leq d \leq \min(m, n)$, $1 \leq i \leq m - d + 1$, $1 \leq j \leq n - d + 1$, and $|i - j| \leq K$, we define $E(i, j, d)$ by

$$E(i, j, d) = d_H(x_i \dots x_{i+d-1}, y_j \dots y_{j+d-1}),$$

where $d_H(x, y)$ denotes the Hamming distance between x and y , which also means the Levenshtein distance between x and y when only substitution operations are allowed. Since it is shown in Section 5.2.3 that $d_H(x_i, y_j)$ can be computed using $O(1)$ nodes and the sum of positive integer values can be computed using one ReLU unit, computation of all $E(i, j, d)$ s can be done by a ReLU neural network using $O(1)$ layers and $O(Kn^2)$ nodes.

Next, we define the main table $D^h(i, j)$, which stores the edit distance between $x_1 x_2 \dots x_i$ and $y_1 y_2 \dots y_j$ under the condition that the number of insertion and deletion operations is exactly h . It is straightforward to see that $D^h(i, j)$ s can be computed by the dynamic programming procedure shown below (see also Figure 39).

For $0 \leq i \leq m$ and $0 \leq j \leq n$, we compute $D^0(i, j)$ as follows.

$$D^0(i, j) = \begin{cases} 0, & \text{if } i = j = 0, \\ E(1, 1, i), & \text{if } i = j > 0, \\ +\infty, & \text{otherwise.} \end{cases}$$

For $0 \leq i \leq m$, $0 \leq h \leq K - 1$, and $-K \leq k \leq K$, we compute $D^{h+1}(i, i + k)$ s as follows, where irrelevant entries (e.g., $D^h(-1, j)$) are ig-

nored (or, regarded as $+\infty$).

$$D^{h+1}(i, i+k) = \min \begin{cases} D^h(i-1, i+k), \\ D^h(i, i+k-1), \\ \min_r \{ D^h(r-1, r+k) \\ + E(r+1, r+1+k, i-r) \}, \\ \min_r \{ D^h(r, r+k-1) \\ + E(r+1, r+1+k, i-r) \} \end{cases} + 1. \quad (1)$$

Finally, we compute the following as the required Levenshtein distance (i.e., output):

$$\min_{0 \leq h \leq K} D^h(m, n).$$

The above dynamic programming procedure can be implemented in a ReLU neural network by creating nodes storing $D^h(i, j)$ s that are layered according to h . The number of such nodes is clearly $O(K^2n)$. To compute each $D^h(i, j)$, we need $O(n)$ nodes and $O(\log n)$ layers because it is required to compute the minimum value of $O(n)$ nodes. Therefore, computation of all $D^h(i, j)$ s needs $O(K^2n^2)$ nodes and $O(K \log n)$ layers. Since the other operations can be implemented within these orders, we have the following.

Theorem 3. *The Levenshtein distance between two strings x and y with $|x| = m$ and $|y| = n$ ($m \leq n$) can be computed by a ReLU neural network using $O(K \log n)$ layers and $O(K^2n^2)$ units if the number of deletions and insertions is at most K .*

Compared with Theorem 2, this theorem uses a much smaller number of units (for small K). However, the number of units is still large. Therefore, this result is only of theoretical interest, too.

5.4 Computational experiments

In this section, we present the results of computational experiments. First, we present the results on computing Hamming and Levenshtein distances. Then, we present the results on learning a string using Hamming and Levenshtein distances. All experiments presented in this chapter were executed on a computer equipped with an Intel Xeon Gold 5222, NVIDIA A100 GPU,

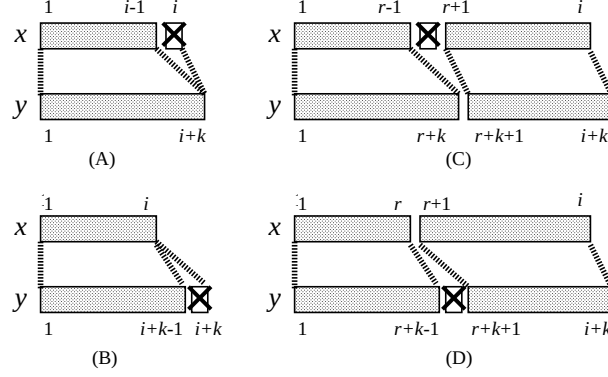


Figure 39: Dynamic programming procedure for computing $D^h(i, i+k)$, where (A)-(D) correspond to the four terms in the outer min of equation (1)

and 128GB RAM. The implementation of the networks was carried out using Python 3.8.9 and Tensorflow 2.4.0. In all experiments, we used the ReLU neural network for the learning purposes because these could also be used for the computation purposes.

5.4.1 Computational time and construction time of the network

We measured the computational time and construction time for the ReLU neural network. Note that we need time for constructing a ReLU neural network from the code. For the measurement of computational time, we utilized 500 strings to obtain the mean value. The construction time is measured as the mean of five repetitions.

First, we examined the ReLU neural network for Hamming distance. Table 6 shows the computational time and construction time for string lengths ranging from 1 to 50. The mean of computation time is fast, less than 0.1 second for $n = 50$. Additionally, the mean construction time is approximately 2.2 seconds for $n = 50$. This tendency was also observed with longer strings. For instance, it was confirmed that the computation and construction times for a length of 100 were approximately twice those of the case with a length of 50. We observe a consistent trend of increasing time with string length, exhibiting an approximately linear rise. This observation aligns with the time complexity of $O(n)$ for the Hamming distance calculation. Next, we measured the ReLU neural network for Levenshtein distance. Table 7 shows the computational time and construction time. In contrast to the Hamming distance experiment, these times show a significant increase as the string length

grows. When the string length is $n = 50$, the computational time is approximately 7.7 seconds, and the construction time is around 270 seconds. Even when observing the trend up to a length of 100, it was confirmed that the increasing tendency continues. The observed trend of increasing time is approximated to $O(n^2)$, which reflects the time complexity of the Levenshtein distance. However, for very long sequences, the construction time imposes a significant limitation on practical use.

To further explore the impact of different input string lengths, additional measurements were made. Table 8 shows the computation and construction times for string lengths of 5, 10, and 20, while varying the other string length from 2 to 5. The computation time remains similar across different string lengths when the size of the score matrix is the same. For instance, when comparing the computational time between $m = 10, n = 2$ and $m = 5, n = 4$, the times are equivalent because the size of the score matrix is the same. Similarly, for other combinations, the computation and construction times are comparable when the size of the score matrix is approximately the same.

In both models, a common issue is that the initial computation time is slightly longer compared to subsequent computations. We speculate that this behavior is a result of the Tensorflow library’s specification.

Table 6: Computation and construction times of the ReLU neural network for Hamming distance

length	computation time	construction time
1	0.003	0.466
2	0.005	0.474
3	0.006	0.508
4	0.008	0.545
5	0.008	0.571
6	0.010	0.608
7	0.011	0.640
8	0.012	0.679
9	0.013	0.728
10	0.014	0.764
15	0.020	0.932
20	0.029	1.084
25	0.031	1.258
30	0.036	1.443
40	0.049	1.773
50	0.061	2.139
60	0.071	2.277
70	0.084	2.623
80	0.097	2.992
90	0.108	3.330
100	0.117	3.663

Table 7: Computation and construction times of the ReLU neural network for Levenshtein distance (inputs are of the same length)

length	computation time	construction time
1	0.004	0.493
2	0.014	0.790
3	0.028	1.321
4	0.050	1.943
5	0.073	2.813
6	0.105	3.829
7	0.146	5.088
8	0.186	6.633
9	0.233	8.338
10	0.298	10.121
15	0.662	22.343
20	1.171	39.485
25	1.900	62.301
30	2.735	88.987
40	4.856	165.688
50	7.747	266.900
60	11.481	374.210
70	15.066	534.653
80	20.001	695.258
90	25.414	896.931
100	31.656	1146.616

5.4.2 Learning of the ReLU neural network

We experimented to assess the learnability of the proposed network. As mentioned in Section 5.2.6, the purpose is to infer a string y from a set of strings and their distances to y . For example, when the desired string is $y = 01$, the training data consist of four strings 00, 01, 10, and 11 and their distances 1, 0, 2, and 1.

First, we experimented on learning the ReLU neural network for Hamming distance. In all cases, the initial weights of w_i s were randomly set within the range of zero to one, and the training process ran for 20 epochs. Figure 40 shows the transition of weights per epoch. The weights were initialized to approximately 0.9 for w_1 and 0.6 for w_2 . In the first epoch, the input to the model is $u_1 \times w_1 = 0.9, u_2 \times w_2 = 0.6$. Because the model is trained to minimize the loss to zero, the weights are updated such that u_1 approaches 0 and u_2 approaches 1. It is seen that as the training progress, w_1 converges to zero, and w_2 converges to one. It is worth noting that w_1 converges to zero despite having an initial weight close to one. This result indicates that the network successfully learns the input to be $y_1 = 0$ and $y_2 = 1$. As a result, the

Table 8: Computation and construction times of the ReLU neural network for Levenshtein distance (inputs are of different lengths)

length m	length n	computation time	construction time
5	2	0.031	1.365
5	3	0.046	1.807
5	4	0.061	2.329
10	2	0.068	2.330
10	3	0.088	3.315
10	4	0.117	4.212
10	5	0.168	5.559
20	2	0.134	4.270
20	3	0.177	6.329
20	4	0.234	8.304
20	5	0.290	10.829

proposed network demonstrates the ability to learn and produce the desired output of $y = 01$.

Next, the proposed network was trained for $y = 01011$. Figure 41 shows the transition of weights per epoch. For example, it is seen that the initial value of w_2 is approximately 0.25, but it gradually approaches to 1 during the training process. It is also seen that the network successfully learns the other weights to attain the correct values.

In addition, we examined whether the proposed network can learn the following long input sequences.

$$y = 101001001110100100111010010011$$

Figure 42 shows the transition of weights per epoch. To facilitate the learning process, 500 strings were selected in advance because dealing with all the data, i.e., 2^{30} strings, would make learning difficult. Remarkably, the proposed network demonstrated the ability to learn even with a somewhat long string length. Unlike the learning experiments with string lengths of two and five, in this case, there might be multiple strings for y that satisfy the distance condition because the data are randomly selected. In such instances, each weight might converge as close as possible to its target value. Furthermore, if additional data with distances close to the desired distance are included, the candidate solutions would decrease, and the likelihood of uniquely determining the desired y is expected to increase.

Second, we experimented on learning the ReLU neural network for Levenshtein distance. Unfortunately, the network did not learn as effectively as in the case of Hamming distance. Figures 43 and 44 show the transition of

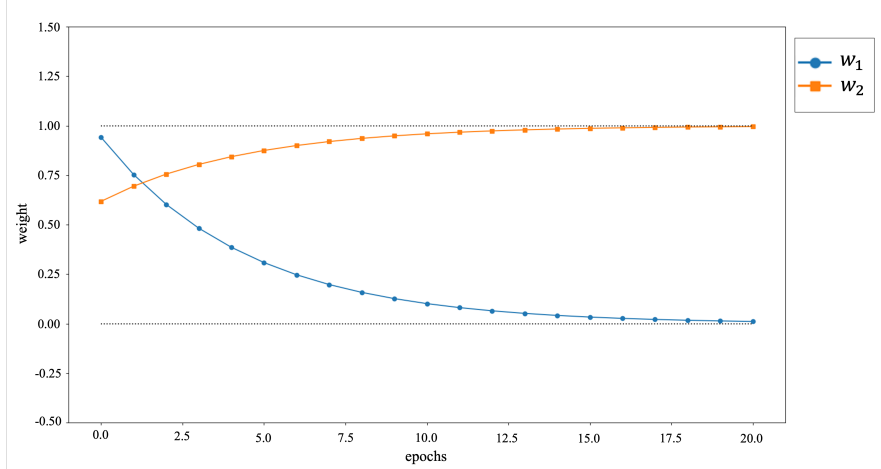


Figure 40: Transition of weights of the ReLU neural network for Hamming distance when desired output is $y = 01$

weights per epoch for $y = 01$ and $y = 01011$, respectively. In the case of $y = 01$, it appears that both weights converged to around 0.4, indicating that the network could not fully capture the desired output. Furthermore, for $y = 01011$, the weights seem to have stalled after a few epochs, signifying that the learning process encountered difficulties and could not effectively converge to the desired output.

One possible cause of the difficulty in learning the ReLU neural network for Levenshtein distance is that the network might get stuck in a local solution. To address this issue, we explored the use of the Adam optimizer, a popular choice in deep learning models [102]. The Adam optimizer incorporates the concept of inertia in the movement of gradients, making it less likely to fall into local solutions. Figure 45 shows the transition of weights during the learning process of $y = 01$ using the Adam optimizer. The learning rate for the Adam optimizer was set to 0.1, and the other hyperparameters were kept at the default values provided by the Tensorflow library. However, both weights converged to around 0.4. Despite using the Adam optimizer, the problem of falling into local solutions persisted. It suggests that simply changing the optimizer based on gradients may not be sufficient to overcome this difficulty. Further investigations and enhancements are required to improve the network's ability to learn Levenshtein distance effectively.

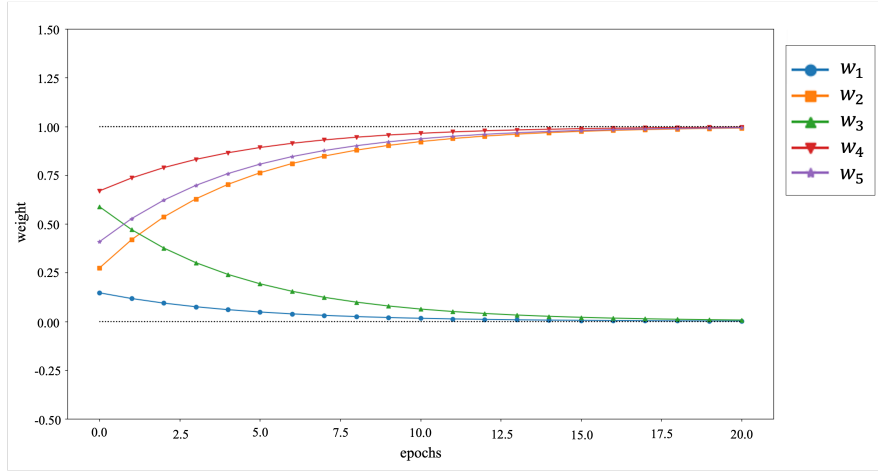


Figure 41: Transition of weights of the ReLU neural network for Hamming distance when desired output is $y = 01011$

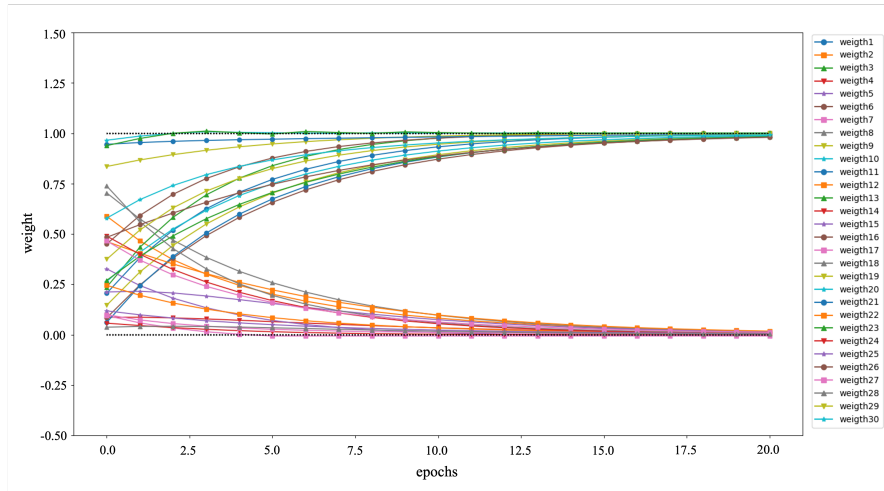


Figure 42: Transition of weights of the ReLU neural network for Hamming distance when input length is 30

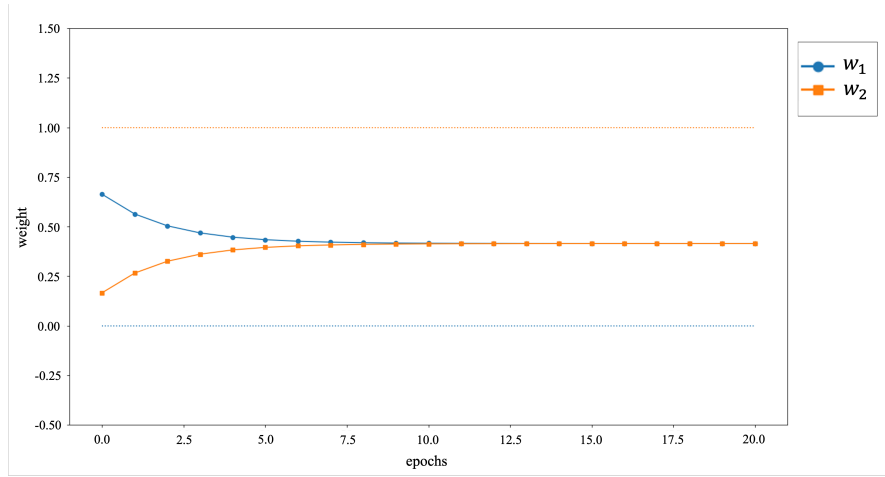


Figure 43: Transition of weights of the ReLU neural network for Levenshtein distance when desired output is $y = 01$

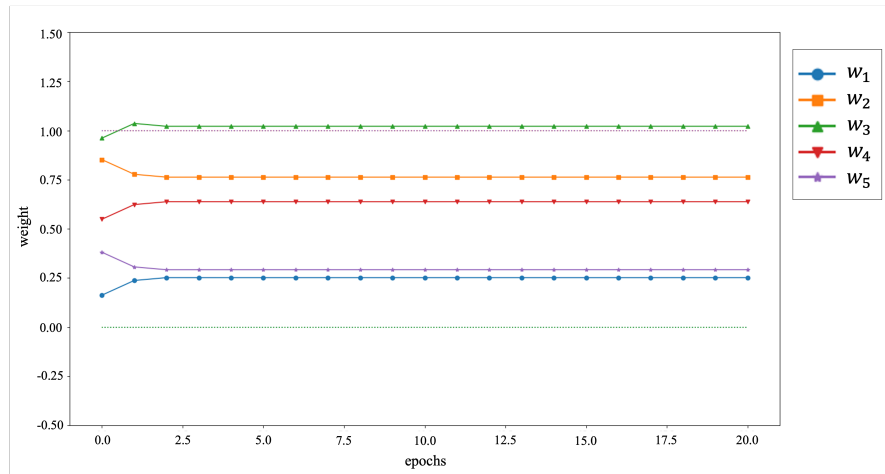


Figure 44: Transition of weights of the ReLU neural network for Levenshtein distance when desired output is $y = 01011$

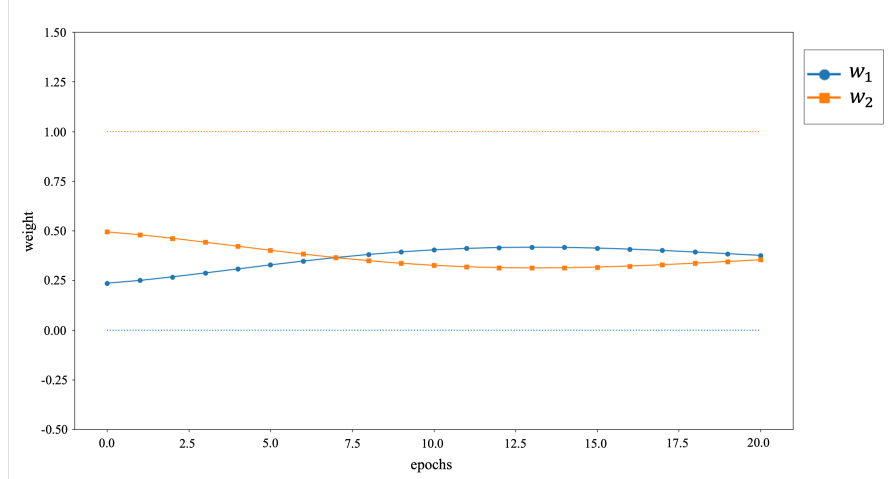


Figure 45: Transition of weights of the ReLU neural network for Levenshtein distance using the Adam optimizer when desired output is $y = 01$

5.4.3 Model reuse through zero-padding for Hamming distance learning

If a model independent of the input length can be constructed, there would be no need to rebuild the model for every calculation. For the Hamming distance, constructing a model capable of processing longer sequences as input allows for model reuse by addressing input length deficiencies through zero-padding. The Hamming distance is based on counting mismatched characters. To examine the impact of zero-padding on input sequence learning experiments, training was conducted using sequences of two characters for a 5-character model and sequences of 5 characters for a 10-character model. Figures 46 and 47 show the progression of the weights during training. From these figures, it can be observed that, in both experiments, the weights converge to the desired values around epochs 16 and 17. This behavior is consistent with the model trained using sequences where the input length matches the model's intended sequence length. As a result, the effect of zero-padding on learning is considered minimal. This study demonstrates that constructing a model capable of accepting larger inputs and supplementing input deficiencies with zero-padding allows for effective model reuse.

Also, regarding the reuse of models for calculating Levenshtein distance, it is suggested that input deficiencies could be addressed by padding with meaningless characters, such as 0 or 1. However, when the two inputs have different lengths, they cannot simply be padded with 0 or 1. Specifically,

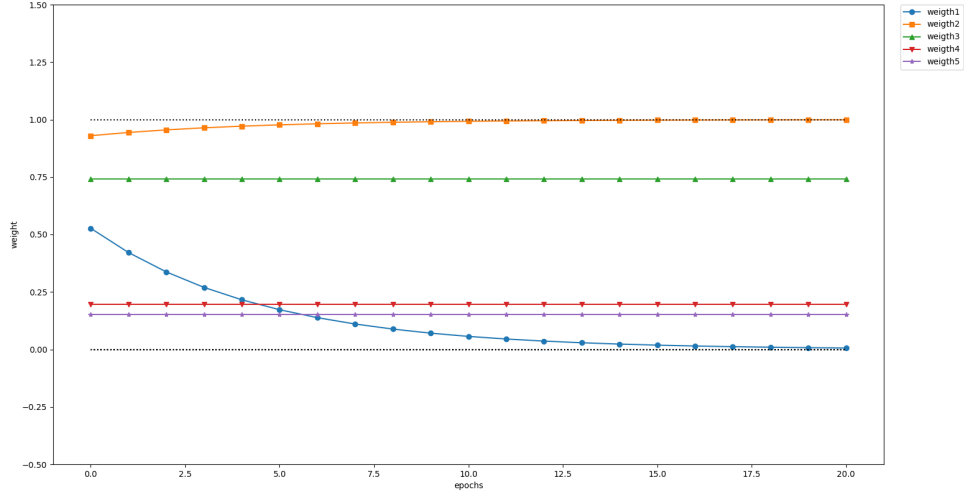


Figure 46: Transition of weights of the ReLU neural network for Hamming distance with zero-padding when using 2 characters for the 5-character model

consider a model that accepts four characters and input strings $x = 010$ and $y = 01$. If we apply zero-padding, we obtain $x = 0100$ and $y = 0100$, which alters the Levenshtein distance. To address this issue, one approach is to fill the short input with the bitwise complement of the longer string at the missing positions and then pad the remainder with 0 or 1. In this example, since the third character of x is 0, we set the third character of y to 1 and pad the rest with 0, yielding $x = 0100$ and $y = 0110$. This preserves the original distance. Note that this property holds only when the insertion and substitution costs are equal.

5.4.4 Hyperparameter tuning

In the optimization of neural networks, the choice of methods and the determination of hyperparameters play a crucial role in the accuracy and stability of the model. Because the determination of hyperparameters has a significant impact on the performance of deep learning methods, a number of optimization approaches have been proposed in recent years [103, 104, 82]. In the input sequence learning experiments, we investigated how the choice of optimization methods and their hyperparameters affects the performance of the proposed model. The experiments used three types of sequences: 01, 01011, and 0101101011. Optimization was performed using SGD and Adam optimiz-

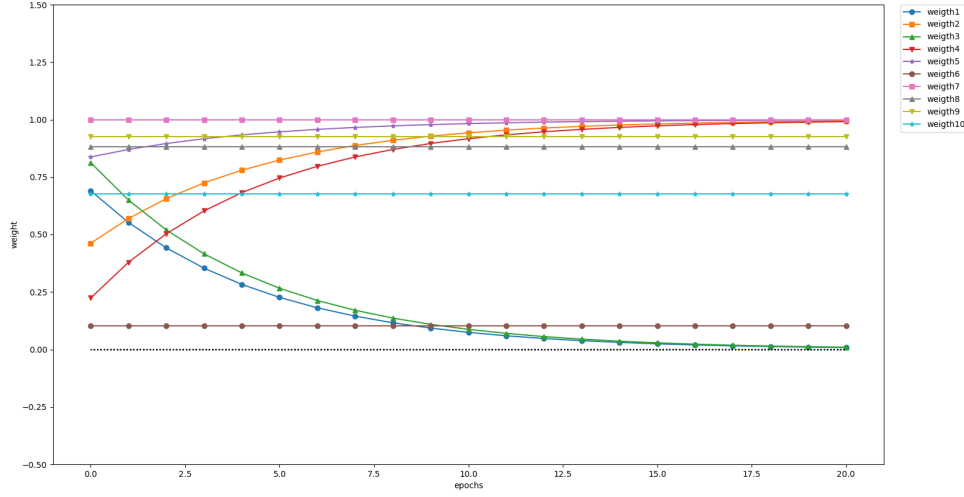


Figure 47: Transition of weights of the ReLU neural network for Hamming distance with zero-padding when using 5 characters for the 10-character model

ers, with a learning rate ranging from 0.1 to 1.0. The optimization process was repeated 100 times, with 20 epochs in each iteration, utilizing the Optuna library [82].

The scatter plot of the final loss for the experiments is shown in Figure 48. For SGD, the final loss is generally quite low. However, for the sequence 01011, it slightly increases around a learning rate of 1.0. In contrast, Adam exhibits a more scattered distribution overall, but the final loss remains sufficiently small, staying below 0.14 even for a sequence length of 10. The difference between SGD and Adam becomes more pronounced with the sequence 0101101011. The final loss of SGD increases significantly at a learning rate of approximately 0.8, whereas Adam maintains stability with low loss values. Adam’s additional adaptive parameters, such as those influencing the learning rate, are believed to enhance the stability of the learning process.

As a result, it was demonstrated that Adam is more likely to achieve stable learning for longer sequences compared to SGD. Even with SGD, a sufficiently small learning rate, such as the 0.1 used in this study, can effectively reduce the loss for Hamming distance learning, particularly for input sequences of at least 10 characters.

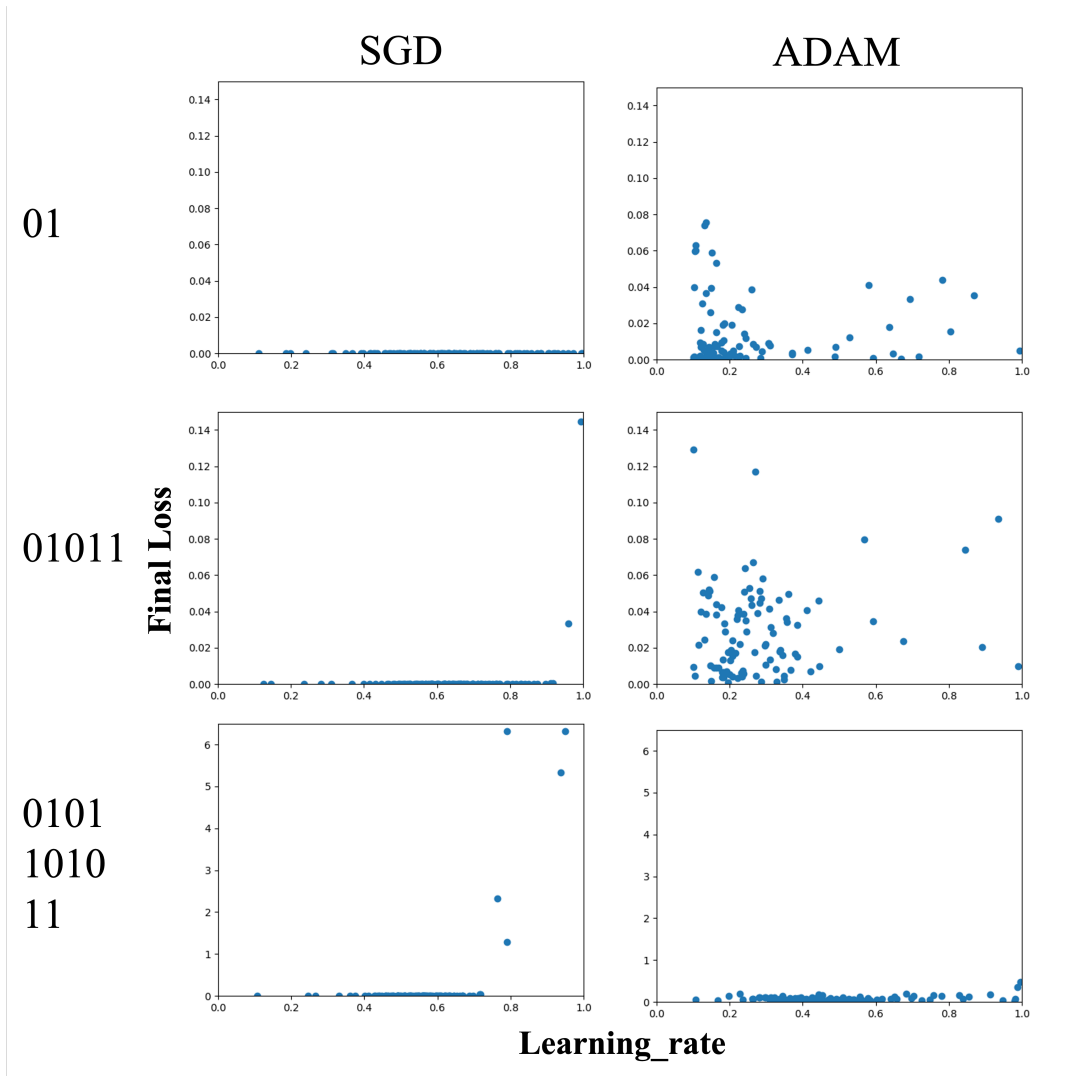


Figure 48: Results of hyperparameter tuning.

5.4.5 Comparison with other supervised learning models

Supervised learning encompasses various proposed models. Regression models are widely recognized as a fundamental method within supervised learning, whose examples include Support Vector Regression (SVR) and Lasso Regression (Lasso) [105, 106]. Furthermore, neural network learning utilizing backpropagation is also categorized as supervised learning. Notable examples include multi-layer perceptron (MLP) and deep learning models such as convolutional neural network (CNN) has been proposed [107, 108]. To evaluate the accuracy and stability of the proposed model, we experimented input learning with four models: SVR, Lasso, MLP, and CNN, where the task was to predict one input sequence based on a known sequence and the correct distance between them.

However, the experiments cannot be performed using SVR or Lasso, as these models are unable to predict the other input sequence based on a known sequence and its corresponding correct distance. Therefore, a different experimental approach was designed for SVR and Lasso. The procedure of the learning experiments for the regression model is shown below.

1. A model was trained with the input string x and the distance d , where d represents the distance between string x and the target input string y .
2. The input string x is reapplied to the trained model and calculates the respective error.
3. The input yielding the smallest error is considered as string y and taken as the ground truth.

Comparable experiments were additionally performed using neural network models such as MLP and CNN.

Next, input learning experiments based on the weighting of two inputs were carried out, in a manner consistent with the experiments described in Section 5.4.2. However, untrained CNN and MLP models cannot compute the correct distance. Accordingly, a neural network model was developed to produce the correct distance, after which training experiments consistent with those in Section 5.4.2 were performed. Figure 49 shows the architectures of the CNN and MLP models for the learning experiments. The procedure of the learning experiments is shown below.

1. Construct a distance computation model by inputting strings x and y ,

with the output defined as their distance, where all the weights of string y are fixed to one.

2. Freeze the trained model and assign a random value between zero and one to the weight of string y .
3. Re-train the distance computation model by inputting a string x and a string u consisting entirely of 1s, with the output defined as the distance between string x and the target string y .

The MLP model consisted of four dense layers, with each layer having a total number of units equal to the sum of the lengths of the two inputs. The CNN model consisted of three convolutional layers and a single dense layer. These convolution layers were configured with $n \times 2$, $n \times 4$ and $n \times 8$, where n denotes the input length. These models were constructed with four layers to maintain a size similar to the proposed model.

Each model was implemented using scikit-learn for SVR and Lasso, and TensorFlow 2.0 for MLP and CNN. SGD was used as the optimization method. Additionally, hyperparameter tuning for each model was performed using the Optuna library [82]. The SVR and Lasso regression models were trained until convergence, while the neural network models were trained for up to 20 epochs. Since the proposed method converged in about 20 epochs in the experiments described in Section 5.4.2, the neural network model was likewise trained for 20 epochs. The model was trained five times, and the mean values of loss and weights were calculated.

First, we conducted the learning experiments for regression, targeting $y = 01$ and $y = 01011$. The training results are shown in Table 9. The loss values in the table are rounded to the fourth decimal place. For comparison, the experimental results conducted in Section 5.4.4 were included in Table 9 as the proposed model (see the upper and middle scatter plots in the left column of Figure 48). As a result, CNN only successfully learned the correct input string $y = 01$ in all five trials. Then, we conducted the learning experiments with $y = 01011$ for each model. This learning experiment revealed that none of the models were able to learn the correct input y . From these results, it is inferred that learning the input string y is difficult for regression models. In contrast, the proposed model exhibited an average final loss that was nearly zero across 100 trials. Based on the weight transitions illustrated in Figures 40 and 41, it can be inferred that the proposed model achieves correct predictions. These

results suggest that the proposed model is more accurate and stable than the CNN.

Next, we conducted the learning experiments with MLP and CNN models for predicting the input y . The training results are shown in Table 10. The loss and weight values in the table are rounded to the fourth decimal place. For comparison, the experimental results conducted in Section 5.4.4 were included in Table 10 as the proposed model (see the upper and middle scatter plots in the left column of Figure 48). As a result, the loss was small, but the input y could not be learned even with hyperparameter optimization. Specifically, in the learning experiment of the CNN for input $y = 01$, the loss was rounded to zero. However, the average of the learned weights was $w_1 = 0.272$ and $w_2 = 0.538$, respectively. Similarly, in the case of learning experiments with five characters for the CNN, average of the learned weights was $w_1 = 0.446$, $w_2 = 0.350$, $w_3 = 0.520$, $w_4 = 0.232$ and $w_5 = 0.553$, respectively. This indicates that while the models are capable of learning from the provided data, none of them are suitable for the input estimation problem, as demonstrated by the proposed method. The final loss values shown in Table 10 suggest that MLP and CNN failed to infer the correct string in many cases whereas the proposed model could infer the correct string in most cases for both $y = 01$ and $y = 01011$. This indicates that each weight was consistently updated to yield the correct string y , as demonstrated in Figures 40 and 41. Conventional learning models are designed to learn the relationship between correct inputs and outputs, making it extremely challenging to perform tasks that involve input estimation. Conventional learning models such as MLP and CNN adjust their many parameters to fit the training data. When the primary objective is to minimize a loss function, the resulting internal parameters are not necessarily unique. Therefore, even if the loss becomes small, it is difficult for conventional models to make the input related weights converge to 0 or 1. However, the proposed model enables such learning by allowing the model itself to accurately compute string distances. This suggests that the proposed model not only achieves high stability in string distance calculations but also offers potential for inference of input strings from distance data.

5.5 Concluding Remarks

In this chapter, we discussed the ReLU neural network models for computing Hamming distance and Levenshtein distance. To construct neural networks

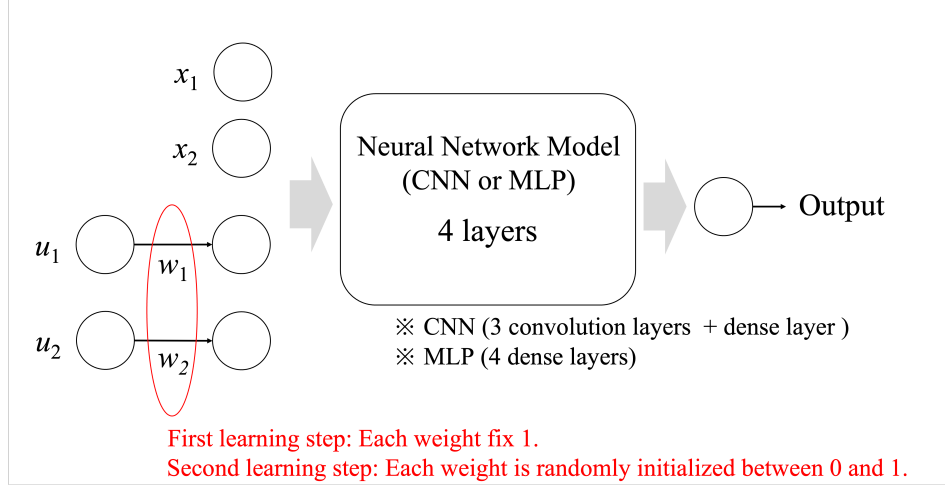


Figure 49: Overview of the architectures of the CNN and MLP models

Table 9: Average final loss and accuracy rate for the other input prediction using the regression model. For reference, the results of the proposed model in Section 5.4.4 are also included at the end of the table (see the upper and middle scatter plots in the left column of Figure 48). Loss values were rounded to four decimal places.

correct	model	final loss (MSE)	accuracy rate
y = 01	Lasso	0.081 (0.001)	0
	SVR	0.005 (0.000)	0
	MLP	0.226 (0.228)	0
	CNN	0.003 (0.002)	100
y = 01011	Lasso	0.202 (0.002)	0
	SVR	0.0061 (0.000)	0
	MLP	0.7810 (0.575)	0
	CNN	0.2635 (0.081)	0
y = 01	proposed model	0.000 (0.000)	-
y = 01011	proposed model	0.002 (0.015)	-

Table 10: Average final loss and weights the other input prediction using neural network model. For reference, the results of the proposed model in Section 5.4.4 are also included at the end of the table (see the upper and middle scatter plots in the left column of Figure 48). Loss and weights Values were rounded to four decimal places.

correct	model	final loss (MSE)	weights (MSE)
y = 01	MLP	0.144 (0.180)	[0.650, 0.382] (0.289, 0.279)
	CNN	0.000 (0.000)	[0.272, 0.538] (0.109, 0.338)
y = 01011	MLP	0.862 (0.471)	[0.218, 0.493, 0.639, 0.592, 0.355] (0.111, 0.279, 0.323, 0.365, 0.263)
	CNN	0.822 (0.382)	[0.446, 0.350, 0.520, 0.232, 0.553] (0.335, 0.150, 0.233, 0.148, 0.221)
y = 01	proposed model	0.000 (0.000)	-
y = 01011	proposed model	0.002 (0.015)	-

for distance computation, we proposed two module networks that determine character matches and minimum value selections. By composing these modules appropriately, we implemented neural network models that exactly compute both the Hamming and Levenshtein distances. In addition, we theoretically showed that the number of layers can be significantly reduced at the expense of employing a larger number of units. We measured the computation time and construction time of the model using Python and confirmed that they have the same computational order as conventional algorithms. Furthermore, we conducted learning experiments to predict the other input from a given input and the corresponding distance. As a result, the learning experiments were successful for the Hamming distance. By comparing with conventional machine learning and deep learning models, we demonstrated that this learning task is difficult for the conventional models.

In contrast, the learning experiments for the Levenshtein distance did not succeed. One possible reason for the failure of the learning experiments for the Levenshtein distance is that the optimization may have become trapped in a local minimum. In addition, the dataset used in the learning experiments for the Levenshtein distance may have led to training failure because the dynamic programming paths are not uniquely determined. In other words, multiple edit operation sequences can achieve the same minimum cost. Constructing

and evaluating datasets that avoid this problem is also an important topic for future work. For learning the Levenshtein distance, it is necessary to investigate the training behavior in more depth by adjusting the dataset and the hyperparameters.

6 Conclusion and Future Work

6.1 Conclusion

In this study, we presented three research approaches to address the two challenges outlined in Section 1.2.

First, we developed an information platform aimed at enabling integrative analysis of motifs. In particular, Chapter 3 discussed the development of the protein-gene motif dictionary system. This motif dictionary stored protein sequence motifs and their corresponding gene sequence motif and 3D structural motifs. To analyze motif characteristics across species, we constructed a motif dictionary covering seven species, including human. Across these motif dictionaries, we stored 956 sequence motif types (120,047 sites) and 569 structural motif types (57,452 sites). We also have implemented a management system where users are able to do one-stop search and analysis using the protein-gene motif dictionary system. This system has a distance analysis method for co-occurrence motifs. As a result, we found that the Euclidean distance between the centroids of the cysteine switch motif and the zinc protease motif was conserved at 12 Å, although the sequence distance was not constant. This is novel knowledge discovered by our system because the distance of co-occurrence motif has not shown quantitatively in any databases. However, we could not conduct comparative co-occurrence motif analysis between species due to insufficient 3D structural information for the motifs.

Second, we conducted sequence analyses leveraging the data in the constructed motif dictionary. In particular, Chapter 4 discussed a sequence alignment system based on DP that incorporates the codon reduced representation for predicting functional regions in genome sequences. In addition, we proposed the motif codon reduced representation, which is translated to a codon weight matrix that quantitatively captures nucleotide usage within gene sequences corresponding to protein sequence motifs. Functional site estimation was performed on the TNNC1 and HPCA genome sequences from multiple species, including human, using query sequences based on the EF-hand motif codon reduced representation. These experiments successfully estimated motif sites inserting intron regions. Estimating protein sequence motif sites from genome sequences containing introns has been difficult for conventional systems. Collectively, our findings demonstrate the effectiveness of the motif codon reduced representation for functional site estimation as a foundation

for genome sequence analysis. In contrast, motif site estimation is highly dependent on parameters such as gap penalties and search ranges. Specifically, simply lowering the gap penalties increases the number of candidate estimates that are not biologically relevant.

Third, we focused on string distance computation, which is a fundamental technique in sequence analysis, and proposed a new neural network model. In particular, Chapter 5 discussed a ReLU neural networks for computing Hamming distance and Levenshtein distance. By combining neural networks for character matching and minimum value selection, we proposed a neural network that exactly computes the two string distances. The networks were implemented in Python, and we measured computation time and construction time. The results indicate that the networks exhibit reasonable CPU time, whose order matches that of traditional algorithms. We also theoretically showed that the number of layers can be significantly reduced at the expense of employing a larger number of units. Furthermore, we extended the proposed neural networks for learning a desired string when provided with a set of strings and their desired distances. As a result, we demonstrated that input sequence estimation is achievable through learning for the Hamming distance. While the network successfully learned the Hamming distance, it encountered difficulties in correctly learning the Levenshtein distance, possibly due to being trapped in local solutions.

As a result, the first and second contributions of our study advanced the understanding of molecular mechanisms through the focus on protein and gene motif. In addition, the third contribution involves the development of a neural network model for calculating string distance. This approach offers a new insight on string similarity search.

6.2 Future work

Here, we discuss future directions and remaining challenges related to the issues examined in this work.

For the motif dictionary, a future work is the characterization of sequence and structural features of motifs in species other than humans. At the time of constructing the database, many motif instances lacked experimentally determined structures, and only sequence information could be collected. In recent years, the availability of experimental structural data has increased. In addition, 3D structures predicted by AlphaFold have been released as a

publicly available database [109]. Leveraging these resources would enable a more comprehensive analysis of motif characteristics across diverse species. In addition, we are going to consider co-occurrence motif analysis using combination of our dictionary system and the motif searching system based on sequence binary decision diagram (seqBDD) [110]. By combining two systems, a comprehensive search of co-occurrence motifs will be possible, and the search results can be efficiently analyzed and registered in the dictionary system. In addition, developing an automated update mechanism for the dictionary system is another important task. Specifically, this includes automatic download of data resources via APIs and acceleration of motif information extraction.

For the estimation of protein motif regions on gene sequences, a more detailed investigation is required to understand how prediction accuracy depends on various hyperparameters. The hyperparameters include the gap penalties and traceback ranges used during alignment, as well as the threshold values for the motif codon reduced representations. Developing an automated pipeline for evaluating and exploring these hyperparameters could help identify parameter configurations that generalize across the various proposed alignment methods. For example, optimization frameworks such as the Optuna [82] and Tune [83] libraries used in the third research may be effective for this purpose.

For the ReLU neural network, a future work is to further investigate whether the Levenshtein distance can be learned effectively. As a potential improvement to hyperparameter optimization, exploration strategies based on swarm intelligence may offer a promising alternative to conventional search methods [111, 112, 113]. Moreover, adapting the proposed model to biological sequences represents another important direction, as doing so would enable its application to practical tasks in computational biology. Expanding the range of target characters to include 4 or 20 may enable its application in learning gene and protein sequences, opening up new possibilities for the network’s practical use. For example, gene sequences may be handled in the same way as our method if we define A as 0, T as 1, G as 2, C as 3. Evaluating the model’s generalization performance by inputting actual biological sequences and previously unseen patterns remains an important direction for future work. In addition, we aim to evaluate the usefulness of the proposed model by comparing it with recently proposed approaches such as differen-

tiable DP and differentiable edit distance [89, 94]. We consider the proposed method to be useful for ancestral sequence inference and sequence generation under distance constraints in proteins and genes.

Addressing these future work is expected to further advance our understanding of motifs that play key roles in molecular mechanisms. Moreover, applying the proposed ReLU-based neural network to biological sequences represents a promising direction. As the results, we believe that this study will contribute to understanding of motifs and string computation from a string processing technique.

References

- [1] J. Kliche, L. Simonetti, I. Krystkowiak, H. Kuss, M. Diallo, E. Rask, J. Nilsson, et al., “Proteome-scale characterisation of motif-based interactome rewiring by disease mutations,” *Mol. Syst. Biol.*, vol. 20, no. 9, pp. 1025-1048, Sep. 2024.
- [2] C. Savojardo, G. Babbi, P. L. Martelli, and R. Casadio, “Mapping OMIM disease-related variations on protein domains reveals an association among variation type, Pfam models, and disease classes,” *Front. Mol. Biosci.*, vol. 8, pp. 617016, May 2021.
- [3] A. Yoshimori, and J. Bajorath, “Motif2Mol: Prediction of new active compounds based on sequence motifs of ligand binding sites in proteins using a biochemical language model,” *Biomolecules.*, vol. 13, 5, pp. 833, May 2023.
- [4] H. Nada, Y. Choi, S. Kim, K. S. Jeong, N. A. Meanwell, and K. Lee, “New insights into protein-protein interaction modulators in drug discovery and therapeutic advance,” *Signal Transduct. Target. Ther.*, vol. 9, pp. 341, Dec. 2024.
- [5] International Human Genome Sequencing Consortium, “Finishing the euchromatic sequence of the human genome,” *Nature*, vol. 431, pp. 931-945, Oct. 2004.
- [6] ENCODE Project Consortium, “An integrated encyclopedia of DNA elements in the human genome,” *Nature*, vol. 489, pp. 57-74, Sep. 2012.
- [7] M. B. Gerstein, Z. J. Lu, E. L. V. Nostrand, C. Cheng, B. I. Arshinoff, T. Liu, K. Y. Yip, et al., “Integrative analysis of the *caenorhabditis elegans* genome by the modENCODE project,” *Science*, vol. 330, pp. 1775-1787, Dec. 2010.
- [8] The modENCODE Consortium, S. Roy, J. Ernst, P. V. Kharchenko, P. Kheradpour, N. Negre, M. L. Eaton, et al., “Identification of functional elements and regulatory circuits by *Drosophila* modENCODE,” *Science*, vol. 330, pp. 1787-1797, Dec. 2010.
- [9] K. Tompson, “Regular expression search algorithm,” *Commun. ACM*, vol. 11, no. 6, pp. 419-422, Jun. 1968.

- [10] L. Rabiner, and B. Juang, “An introduction to hidden Markov models,” *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4-16, Jan. 1986.
- [11] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *J. R. Stat. Soc. Ser. B Stat. Methodol.*, vol. 39, no. 1, pp. 1-38, 1977.
- [12] C. J. A. Sigrist, B. A. Cucho, E. d. Castro, E. Coudert, N. Redaschi, and A. Bridge, “The PROSITE database for protein families, domains, and sites,” *Nucl. Acids Res.*, pp. gkaf1188, Nov. 2025.
- [13] T. Yada, Y. Totoki, M. Ishikawa, K. Asai, and K. Nakai, “Automatic extraction of motifs represented in the hidden Markov model from a number of DNA sequences,” *Bioinformatics*, vol. 14, no. 4, pp. 317-25, 1998.
- [14] C. E. Lawrence, and A. A. Reilly, “An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences,” *Proteins*, vol. 7, no. 1, pp. 41-51, 1990.
- [15] T. F. Smith, and M. S. Waterman, “Identification of common molecular subsequences,” *J. Mol. Biol.*, vol. 147, no. 1, pp. 195-197, Mar. 1981.
- [16] S. B. Needleman, and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *J. Mol. Biol.*, vol. 48, no. 3, pp. 443-453, Mar. 1970.
- [17] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” *J. Mol. Biol.*, vol. 215, no. 3, pp. 403-410, Oct. 1990.
- [18] P. K. Koo, and S. R. Eddy, “Representation learning of genomic sequence motifs with convolutional neural networks,” *PLoS Comput. Biol.*, vol.15, no. 12, pp. e1007560, Dec. 2019.
- [19] A. M. Tseng, G. Eraslan, T. Biancalani, and G. Scalia, “A mechanistically interpretable neural network for regulatory genomics,” Oct. 2024, *arXiv:2410.06211*.
- [20] H. Shen, M. Yang, J. Liu, K. Chen, and X. Li, “Development of a deep learning model for cancer diagnosis by inspecting cell-free DNA end-motifs,” *npj Precis. Oncol.*, vol. 8, no. 160, Jul. 2024.

- [21] C. Savojardo, P. L. Martelli, and R. Casadio, “Finding functional motifs in protein sequences with deep learning and natural language models,” *Curr. Opin. Struct. Biol.*, vol. 81, no. 102641, Aug. 2023.
- [22] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, et al., “Highly accurate protein structure prediction with AlphaFold,” *Nature*, vol. 596, pp. 583–589, Jul. 2021.
- [23] J. Kumuthini, M. Chimenti, S. Nahnsen, A. Peltzer, R. Meraba, R. McFadyen, G. Wells, et al., “Ten simple rules for providing effective bioinformatics research support,” *PLoS Comput. Biol.*, vol. 16, no. 3, pp. e1007531, Mar. 2020.
- [24] C. Brooksbank, M. T. Bergman, R. Apweiler, E. Birney, and J. Thornton, “The european bioinformatics institute’s data resources 2014,” *Nucl. Acids Res.*, vol. 42, pp. D18-D25, Jan. 2014.
- [25] E. W. Sayers, J. Beck, E. E. Bolton, J. R. Brister, J. Chan, R. Connor, M. Feldgarden, et al., “Database resources of the national center for biotechnology information in 2025,” *Nucl. Acids Res.*, vol. 53, no. D1, pp. D20-D29, Nov. 2024.
- [26] C. O’Cathail, A. Ahamed, J. Burgin, C. Cummins, R. Devaraj, K. Gueye, D. Gupta, et al., “The european nucleotide archive in 2024,” *Nucl. Acids Res.*, vol. 53, no. D1, pp. D49-D55, Jan. 2025.
- [27] A. Fukuda, Y. Kodama, J. Mashima, T. Fujisawa, and O. Ogasawara, “DDBJ update: Streamlining submission and access of human data,” *Nucl. Acids Res.*, vol. 49, no. D1, pp. D71-D75, Jan. 2021.
- [28] S. K. Burley, R. Bhatt, C. Bhikadiya, C. Bi, A. Biester, P. Biswas, S. Bittrich, et al., “Updated resources for exploring experimentally-determined PDB structures and computed structure models at the RCSB protein data bank,” *Nucl. Acids Res.*, vol. 53, no. D1, pp. D564-D574, Jan. 2025.
- [29] The UniProt Consortium, “UniProt: The universal protein knowledgebase in 2025,” *Nucl. Acids Res.*, vol. 53, no. D1, pp. D609-D617, Jan. 2025.
- [30] T. Paysan-Lafosse, A. Andreeva, M. Blum, S. R. Chuguransky, T. Grego, B. L. Pinto, G. A. Salazar, et al., “The Pfam protein families database:

- Embracing AI/ML,” *Nucl. Acids Res.*, vol. 53, no. D1, pp. D523-D534, Jan. 2025.
- [31] M. Kanehisa, and S. Goto, “KEGG: kyoto encyclopedia of genes and genomes,” *Nucl. Acids Res.*, vol. 28, no. 1, pp. 27-30, Jan. 2000.
 - [32] C. W. Heizmann, and W. Hunziker, “Intracellular calcium-binding proteins: More sites than insights,” *Trends Biochem. Sci.*, vol. 16, pp. 98-103, 1991.
 - [33] F. Payre, and A. Vincent, “Finger proteins and DNA-specific recognition: Distinct patterns of conserved amino acids suggest different evolutionary modes,” *FEBS. Lett.*, vol. 234, no. 2, pp. 245-250, Jul. 1988.
 - [34] P. Bucher, “Weight matrix descriptions of four eukaryotic RNA polymerase II promoter elements derived from 502 unrelated promoter sequences,” *J. Mol. Biol.*, vol. 212, no. 4, pp. 563-578, Apr. 1990.
 - [35] T. Goldfarb, V. K. Kodali, S. Pujar, V. Brover, B. Robbertse, C. M. Farrell, D. Oh, et al., “NCBI RefSeq: Reference sequence standards through 25 years of curation and annotation,” *Nucl. Acids Res.*, vol. 53, no. D1, pp. D243-D257, Jan. 2025.
 - [36] D. J. Lipman, and W. R. Pearson, “Rapid and sensitive protein similarity searches,” *Science*, vol. 227, no. 4693, pp. 1435-1441, Mar. 1985.
 - [37] RCSB PDB, RCSB PDB [online], [Available]:<https://www.rcsb.org/>.
 - [38] Expasy, Prosite [online], [Available]:<https://prosite.expasy.org/>.
 - [39] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” *Sov. phys. dokl.*, vol. 10, no. 8, pp. 707-710, Feb. 1966.
 - [40] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, “Activation functions in deep learning: A comprehensive survey and benchmark,” *Neurocomputing*, vol. 503, pp. 92-108, Sep. 2022.
 - [41] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” *AISTATS*, vol. 15, pp. 315-323, 2011.
 - [42] G. Pugalenth, P. N. Suganthan, R. Sowdhamini, and S. Chakrabarti, “MegaMotifBase: A database of structural motifs in protein families and superfamilies,” *Nucl. Acids Res.*, vol. 36, pp. D218-21, Jan. 2008.

- [43] S. Bittrich, S. K. Burley, and A. S. Rose, “Real-time structural motif searching in proteins using an inverted index strategy,” *PLoS Comput. Biol.*, vol. 16, no. 12, pp. e1008502, Dec. 2020.
- [44] V. Matys, O. V. Kel-Margoulis, E. Fricke, I. Liebich, S. Land, A. Barre-Dirrie, I. Reuter, et al., “TRANSFAC and its module TRANSCompel: Transcriptional gene regulation in eukaryotes,” *Nucl. Acids Res.*, vol. 34, pp. D108-D110, Jan. 2006.
- [45] A. Sandelin, W. Alkema, P. Engström, W. W. Wasserman, and B. Lenhard, “JASPAR: An open-access database for eukaryotic transcription factor binding profiles,” *Nucl. Acids Res.*, vol. 32, pp. D91-D94, Jan. 2004.
- [46] N. A. O ’ Leary, E. Cox, J. B. Holmes, W. R. Anderson, R. Falk, V. Hem, M. T. N. Tsuchiya, et al., “Exploring and retrieving sequence and metadata for species across the tree of life with NCBI Datasets,” *Sci. Data*, vol. 11, pp. 732, Jul. 2024.
- [47] M. Kanehisa, “Linking databases and organisms: GenomeNet resources in Japan,” *Trends Biochem. Sci.*, vol. 22, no. 11, pp. 442-444, Nov. 1997.
- [48] P. Artimo, M. Jonnalagedda, K. Arnold, D. Baratin, G. Csardi, E. d. Castro, S. Duvaud, et al., “ExPASy: SIB bioinformatics resource portal,” *Nucl. Acids Res.*, vol. 40, no. W1, pp. W597-W603, Jul. 2012.
- [49] J. Shoji, and H. Kato, “Development of the protein-gene sequence motif dictionary system using the corresponding codon,” *Joint Conference on Informatics in Biology, Medicine and Pharmacology*, pp. 35, 2013.
- [50] NCBI RefSeq, RefSeq FTP [online],
[Available]:<ftp://ftp.ncbi.nlm.nih.gov/ref-seq/>.
- [51] T. Kobayashi, and H. Kato, “Development of the DP-based genome sequence analysis system using the motif codon reduced representation,” *The 43th Symposium on Structural Activity Relationships 2015 and the 10th Japan-China Joint Symposium on Drug Discovery and Development*, pp. KP09, Sep. 2015.
- [52] MongoDB [online], [Available]:<https://www.mongodb.com/>.

- [53] N. K. Gundla, and Z. Chen, “Creating NoSQL biological databases with ontologies for query relaxation,” *Procedia Comput. Sci.*, vol. 91, pp. 460-469, Aug. 2016.
- [54] S. Wang, I. Pandis, C. Wu, S. He, D. Johnson, I. Emam, F. Guitton and Y. Guo, “High dimensional biological data retrieval optimization with NoSQL technology,” *BMC Genomics*, vol. 15, suppl. 8, pp. S3, Nov. 2014.
- [55] R. M. Hanson, “Jmol – a paradigm shift in crystallographic visualization,” *J. Appl. Cryst.*, vol. 43, no. 5, pp. 1250-1260, Jul. 2010.
- [56] D. Kligman, and D. C. Hilt, “The S100 protein family,” *Trends Biochem. Sci.*, vol. 13, no. 11, pp. 437-443, Nov. 1988.
- [57] A. Abyzov, and V. A. Ilyin, “A comprehensive analysis of non-sequential alignments between all protein structures,” *BMC Struct. Biol.*, vol. 7, no. 78, Nov. 2007.
- [58] N. V. Grishin, “Fold change in evolution of protein structures,” *J. Struct. Biol.*, vol. 134, no. 2-3, pp. 167-185, May. 2001.
- [59] W. G. Stetler-Stevenson, “Matrix metalloproteinases in angiogenesis: A moving target for therapeutic intervention,” *J. Clin. Invest.*, vol. 103, no. 9, pp. 1237-1241, May 1999.
- [60] M. P. Caley, V. L. C. Martins, and E. A. O ’ Toole, “Metalloproteinases and wound healing,” *Adv. Wound Care*, vol. 4, no. 4, pp. 225-234, Apr. 2015.
- [61] G. Bozzuto, P. Ruggieri, and A. Molinari, “Molecular aspects of tumor cell migration and invasion,” *Ann. Ist. Super. Sanita*, vol. 46, no. 1, pp. 66-80, 2010.
- [62] J. F. Woessner, Jr., “Matrix metalloproteinases and their inhibitors in connective tissue remodeling,” *FASEB J.*, vol. 5, no. 8, pp. 2145-2154, 1991.
- [63] C. V. Jongeneel, J. Bouvier, and A. Bairoch, “A unique signature identifies a family of zinc-dependent metalloproteinases,” *FEBS Lett.*, vol. 242, no. 2, pp. 211-214, 1989.

- [64] B. Buchfink, C. Xie, and D. H. Huson, “Fast and sensitive protein alignment using DIAMOND,” *Nat. Methods*, vol. 12, pp. 59-60, Nov. 2014.
- [65] F. Kallenborn, A. Chacon, C. Hundt, H. Sirelkhatim, K. Didi, S. Cha, C. Dallago, et al., “GPU-accelerated homology search with MMseqs2,” *Nat. Methods*, vol. 22, pp. 2024-2027, Sep. 2025.
- [66] C. Mathé, M. F. Sagot, T. Schiex, and P. Rouzé, “Current methods of gene prediction, their strengths and weaknesses,” *Nucl. Acids Res.*, vol. 30, no. 19, pp. 4103-4117, Oct. 2002.
- [67] H. J. Mangalam, “tacg – a grep for DNA,” *BMC Bioinform.*, vol. 3, pp. 8, Mar. 2002.
- [68] T. Yan, D. Yoo, T. Z. Berardini, L. A. Mueller, D. C. Weems, S. Weng, J. M. Cherry, and S. Y. Rhee, “PatMatch: a program for finding patterns in peptide and nucleotide sequences,” *Nucl. Acids Res.*, vol. 33, Suppl. 2, pp. W262-W266, Jul. 2005.
- [69] A. Krogh, I. S. Mian, and D. Haussler, “A hidden Markov model that finds genes in E. coli DNA,” *Nucl. Acids Res.*, vol. 22, no. 22, pp. 4768-4778, Nov. 1994.
- [70] E. C. Uberbacher, Y. Xu, and R. J. Mural, “Discovering and understanding genes in human DNA sequence using GRAIL,” *Methods Enzymology*, vol. 266, pp. 259-281, 1996.
- [71] J. Claverie, “Computational methods for the identification of genes in vertebrate genomic sequences,” *Hum. Mol. Genet.*, vol. 6, no. 10, pp. 1735–1744, Sep. 1997.
- [72] R. Guigó, “Computational gene identification: An open problem,” *Comput. Chem.*, vol. 21, no. 4, pp. 215-222, 1997.
- [73] D. Haussler, “Computational genefinding,” *Trends Biotechnol.*, vol. 16, no. 1, pp. 12-15, Nov. 1998.
- [74] C. B. Burge, and S. Karlin, “Finding the genes in genomic DNA,” *Curr. Opin. Struct. Biol.*, vol. 8, no. 3, pp. 346-354, Jun. 1998.
- [75] S. Maji, and D. Garg, “Progress in gene prediction: Principles and challenges,” *Curr. Bioinform.*, vol. 8, no. 2, pp. 226-243, 2013.

- [76] J. D. Thompson, D. G. Higgins, and T. J. Gibson, “CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice,” *Nucl. Acids Res.*, vol. 22, no. 22, pp. 4673–4680, Nov. 1994.
- [77] H. Kato, and J. Yamamoto, “Development of the protein-gene sequence motif analysis system based on the codon reduced representation,” *2015 International Chemical Congress of Pacific Basin Societies, De Novo Drug Design*, pp. 476, 2015.
- [78] O. Gotoh, “An improved algorithm for matching biological sequences,” *J. Mol. Biol.*, vol. 162, no. 3, pp. 705–708, Dec. 1982.
- [79] H. J. Yuasa, J. A. Cox, and T. Takagi, “Diversity of the troponin C genes during chordate evolution,” *J. Biochem.*, vol. 123, no. 6, pp. 1180–1190, Jun. 1998.
- [80] H. Nojima, “Molecular evolution of the calmodulin gene,” *FEBS Lett.*, vol. 217, no. 2, pp. 187–190, Jun. 1987.
- [81] P. Christine, N. Lomri, and M. Thomasset, “Evolution of the EF-hand calcium-binding protein family: Evidence for exon shuffling and intron insertion,” *J. Mol. Evol.*, vol. 27, pp. 351–364, Aug. 1988.
- [82] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” 2019, *arXiv:1907.10902v1*, [online]. Available:<https://optuna.readthedocs.io/en/stable/index.html>
- [83] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, “Tune: A research platform for distributed model selection and training,” 2018, *arXiv:1807.05118*
- [84] J. Romiguier, V. Ranwez, E. JP. Douzery, and N. Galtier, “Contrasting GC-content dynamics across 33 mammalian genomes: Relationship with life-history traits and chromosome sizes,” *Genome Res.*, vol. 20, no. 8, pp. 1001–1009, Aug. 2010.
- [85] K. Subramanian, B. Payne, F. Feyertag, and D. Alvarez-Ponce, “The codon statistics database: A database of codon usage bias,” *Mol. Biol. Evol.*, vol. 39, no. 8, pp. msac157, Aug. 2022.

- [86] R. W. Hamming, “Error detecting and error correcting codes,” *Bell Syst. tech. j.*, vol. 29, no. 2, pp. 147-160, Apr. 1950.
- [87] A. W. Robert and M. J. Fischer, “The string-to-string correction problem,” *JACM*, vol. 21, no. 1, pp. 168-173, Jan. 1973.
- [88] A. Mensch, and M. Blondel, “Differentiable dynamic programming for structured prediction and attention,” Feb. 2018, *arXiv:1802.03676*.
- [89] J. Libovický, and A. Fraser, “Neural string edit distance,” Apr. 2022, *arXiv:2104.08388*.
- [90] J. T. Morton, C. E. M. Strauss, R. Blackwell, D. Berenberg, V. Gligorijevic, and R. Bonneau, “Protein structural alignment from sequence,” Nov. 2020, *bioRxiv 2020.11.03.365932*.
- [91] F. L.-López, Q. Berthet, M. Blondel, O. Teboul, and J.-P. Vert, “Deep embedding and alignment of protein sequence,” *Nat. Methods*, vol. 20, pp. 104-111, Dec. 2022.
- [92] T. Bepler, and B. Berger, “Learning protein sequence embeddings using information from structure,” Oct. 2019, *arXiv:1902.08661*.
- [93] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *NeuroIPS*, pp. 6000-6010, Dec. 2017.
- [94] S. Petti, N. Bhattacharya, R. Rao, J. Dauparas, N. Thomas, J. Zhou, A. M. Rush, et al., “End-to-end learning multiple sequence alignments with differentiable Smith-Waterman,” *Bioinformatics*, vol. 39, no. 1, pp. btac724, Nov. 2022.
- [95] X. Dai, X. Yan, K. Zhou, Y. Wang, H. Yang, and J. Cheng, “Convolutional embedding for edit distance,” May 2020, *arXiv:2001.11692*.
- [96] A. Backurs, and P. Indyk, “Edit distance cannot be computed in strongly subquadratic time (unless SETH is false),” *SIAM J. Comput.*, vol. 47, no. 3, pp. 1087-1097, 2018.
- [97] S. Kumano, and T. Akutsu, “Comparison of the representational power of random forests, binary decision diagrams, and neural networks,” *Neural Comput.*, vol. 34, no. 4, pp. 1019-1044, Mar. 2022.

- [98] J. Gramm, R. Niedermeier, and P. Rossmanith, “Fixed-parameter algorithms for CLOSEST STRING and related problems,” *Algorithmica*, vol. 37, no. 1, pp. 25-42, Sep. 2003.
- [99] R. W. Floyd, “Algorithm 97: Shortest path,” *Commun. ACM*, vol. 5, no. 6, pp. 345, June 1962.
- [100] S. Warshall. “A theorem on Boolean matrices,” *J. ACM*, vol. 9, no. 1, pp. 11-12, Jan. 1962.
- [101] A. Cassis, T. Kociumaka, and R. Wellnitz, “Optimal algorithms for bounded weighted edit distance,” May 2023, *arXiv:2305.06659*.
- [102] D. P. Kingma, and J. L. Ba, “Adam: A method for stochastic optimization,” Dec. 2014, *arXiv:1412.6980*.
- [103] N. Calik, F. Günes, S. Koziel, A. P. Dabrowska, M. A. Belen and P. Mahouti, “Deep-learning-based precise characterization of microwave transistors using fully-automated regression surrogates,” *Sci. Rep.*, vol. 13, pp. 1445, Jan. 2023.
- [104] N. Bacanin, C. Stoean, M. Zivkovic, M. Rakic, R. S. Wójcikiewicz and R. Stoean, “On the benefits of using metaheuristics in the hyperparameter tuning of deep learning models for energy load forecastings,” *Energies*, vol. 16, no. 3, pp. 1434, Feb. 2023.
- [105] D. Harris, J. C. B. Chris, K. Linda, S. Alex, and V. Vladimir, “Support vector regression machine,” *NIPS96*, pp. 155-161, 1996.
- [106] T. Rovert, “Regression shrinkage and selection via the lasso,” *J. R. Stat. Soc., B*, vol. 58, no. 1, pp. 267-288, 1996.
- [107] E. R. David, E. H. Geoffrey, and J. W. Ronald, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533-536, Jul. 1986.
- [108] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. of the IEEE*, pp. 2278-2324, 1998.
- [109] J. Fleming, P. Magana, S. Nair, M. Tsenkov, D. Bertoni, I. Pidruchna, M. Q. L. Afonso, et al., “AlphaFold protein structure database and 3D-

- beacons: New data and capabilities,” *J. Mol. Biol.*, vol. 437, no. 15, pp. 168967, Aug. 2025.
- [110] K.Yamato, H. Kato, T. Katsuragi, and Y. Takahashi, “The multiple representation of protein sequence motifs using sequence binary decision diagrams,” *J. Comput. Chem. Jpn.*, vol. 19, no. 1, pp. 8-17, Jun. 2020.
- [111] N. Kayarbizhy, S. Kanmani, and V. Uthariaraj, “ANN models optimized using swarm intelligence algorithms,” *WSEAS Trans. Comput.*, vol. 13, pp. 501-519, Jan. 2014.
- [112] S. S. Band, S. Janizadeh, S. C. Pal, A. Saha, R. Chakraborty, M. Shokri, and A. Mosavi, “Novel ensemble approach of deep learning neural network (DLNN) model and particle swarm optimization (PSO) algorithm for prediction of gully erosion susceptibility,” *Sensors*, vol. 20, no. 19, pp. 5609, Sep. 2020.
- [113] F. Ghashami, K. Kamyar, and S. A. Riazi, “Prediction of stock market index using a hybrid technique of artificial neural networks and particle swarm optimization,” *Appl. Econ. Finance*, vol. 8, no. 3, pp. 1-8, Dec. 2021.

List of publications

- Chapter 3 is based on a paper in *Journal of Computer Chemistry, Japan - International Edition*

Masahiro Ohtomo, Hiroaki Kato, “Development of a Protein-Gene Motif Dictionary System for One-Stop Motif Analysis”, *Journal of Computer Chemistry, Japan - International Edition*, vol. 7, pp. 1-9, Oct. 2021, DOI: 10.2477/jccjie.2020-0008.

- Chapter 4 is based on a paper in *Journal of Computer Chemistry, Japan*

大友将宏, 小林貴史, 加藤博明, “モチーフコドン縮約表現に基づいた動的計画 (DP) 法によるゲノム機能部位推定システムの開発”, *Journal of Computer Chemistry, Japan*, vol. 21, no. 1, pp. 20-32, Aug. 2022, DOI: 10.2477/jccj.2022-0005.

- Chapter 5 is based on a paper in *IEEE Access*

Masahiro Ohtomo, Atsuhiro Takasu, Tatsuya Akutsu, “Computing Hamming Distance and Levenshtein Distance Using ReLU Neural Networks”, *IEEE Access*, vol. 13, pp. 210089-210101, Dec. 2025, DOI: 10.1109/ACCESS.2025.3639297.