Journal of Education for Pure Science- University of Thi-Qar
Vol.13, No.1 (March., 2023)
Website: jceps.utq.edu.iq Email: jceps@eps.utq.edu.iq

DOI: http://doi.org/10.32792/utq.jceps.10.01.01

Motif Discovery in DNA Sequences Using Scaled Conjugate Gradient Neural Networks

Ali Basim Yousif[1], Thekra Abbas[2], Hussein Keitan Al-Khafaji

Thekra-abbas@gmail.com hussain-ketau-elc@ruc.edu.iq

[1,2]Department of Computer Science, College of Science, Mustansiriyah University, Baghdad, Iraq
[3]Department of Computer Communication Engineering, Al-Rafidain University College, Baghdad, Iraq

Received 19/12/2022,Accepted 15/1/2023,Published /March/2023

Abstract:

Finding motifs in DNA sequences is a current challenge and an essential step in bioinformatics. Processing these issues needs considerable data analysis due to technical advancements in the industry. Artificial Neural Networks (ANNs) are increasingly used, particularly for motif identification and genomic analysis. In order to find motifs in DNA sequences, this work proposed a supervised learning algorithm for feed-forward neural networks called Scaled Conjugate Gradient (SCG) algorithm. The SCG algorithm utilizes a step-size scaling mechanism that is fully automated to minimize time-consuming row searches during each training iteration. This algorithm was used in this work for motif discovery to train code patterns and to reduce a multivariate global error function dependent on the network weights. It trains many code patterns of lengths between 4 to 509 bases to find them in a database with 2,227,382 bases; many experiments were done with different numbers of hidden layers; our finding ten hidden layers provide the best results, with training percentage is 100%. Compared to the other supervised learning neural network algorithms, One Step Secant, Gradient Descent, Bayesian Regularization, and BFGS Quasi-Newton; our find SCG algorithm produced higher accuracy (100%) and less time during the training and testing phases.

Key word: Bioinformatics, Data Mining, Deoxyribonucleic Acid (DNA), Motif Discovery, Artificial Neural Networks (ANNs), SCG.

1. Introduction:

Bioinformatics was first used by Paulien Hogeweg and Ben Hesper in 1970 to describe the study of informatics procedures in biological systems. It analyzes biological data (genes, genomes, proteins, cells, ecosystems, medical information, robotics, artificial intelligence, etc.) [1]. It blends biology, computer science, mathematics, statistics, physics, and engineering principles to analyze and understand biological data, including DNA, RNA, protein sequences, genomic analysis, and gene expression [2,3]. It combines principles from several disciplines as shown in Fig. 1.

Deoxyribose Nucleic Acid (DNA) is a molecule that contains the genetic data necessary for an organism's growth and operation. Nucleotides, also known as base pairs, are the four nucleic acid units that makeup DNA. Although they have a similar chemical structure, the nitrogenous bases in each nucleotide allow for differentiation: Cytosine (C), Thymine (T), Guanine (G), or Adenine (A) [4].

Cytosine and Thymine are pyrimidines, while Adenine and Guanine belong to the purines group. Typically, their first letter stands for nucleotides, such as Adenine (A), Guanine (G), Thymine (T), or Cytosine (C) [5]. Knowledge discovery and data mining refer to the methods, processes, and equipment used to extract informational value from vast volumes of data. The entire information extraction process is referred to as the KDD process. Data mining is merely one phase of the whole KDD process. Data mining is typically used in business and marketing to view the complete KDD process [6]. Traditional statistical methods are combined with computer science algorithms in data mining to extract knowledge from massive amounts of data for application in science, computation, or industry. With a wealth of unstructured data and a lack of theoretical backing, data mining is one of the core topics of bioinformatics. The biological field makes extensive use of data mining to help extract hidden knowledge from sizable datasets gathered in the biological and medical fields [7].

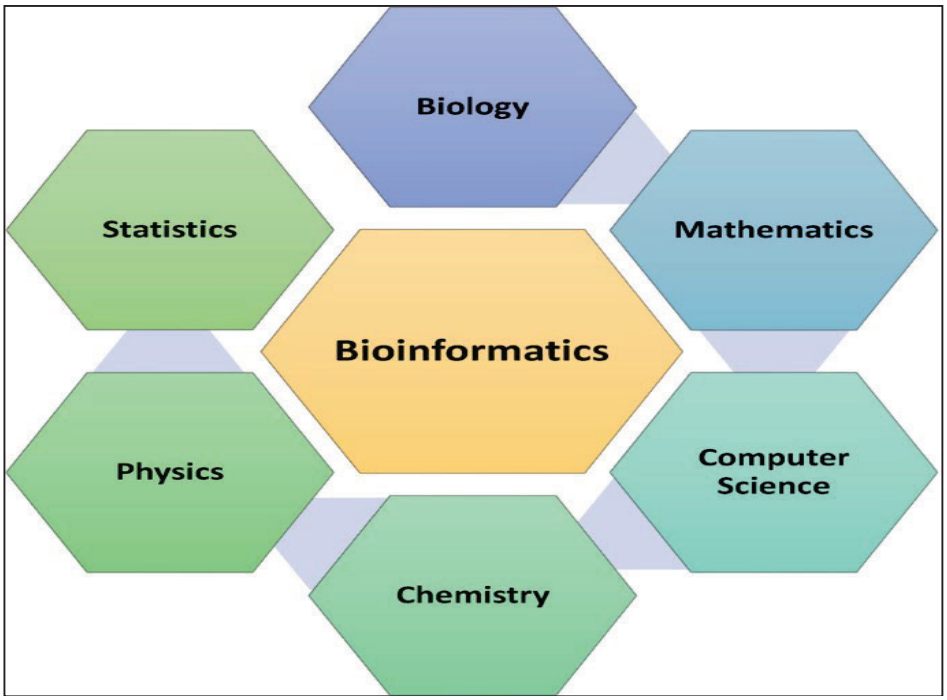

Figure (1) : Shows how bioinformatics incorporates the fundamental ideas of various fields [2]

A motif is a regularly occurring pattern of amino acids or nucleotides that are thought to have biological significance, such as serving as DNA binding sites for a transcription factor or other regulatory protein [8]. The main focus is on the two types of motifs are simple and structured. They are generated based on how the motif templates are arranged. Simple motifs are simpler to discover in biological sequences because they don't have gaps. Contrarily, structured motifs have gaps and more complex than simple motifs. The SCG algorithm has several main advantages. Finding the motif takes very little time, doesn't take up much memory, and can be done accurately in finding all the motifs occurrences even while searching through massive amounts of data. Compared to the rest of the supervised learning neural network algorithms, such as One Step Secant, Gradient Descent, Bayesian Regularization, and BFGS Quasi-Newton, It produced unsatisfactory results due to high time consumption, high memory

consumption, and low accuracy in finding all occurrences of the motifs. The problem in this paper is how to find all the occurrences of bio motifs in bio-databases in fast and low memory consumption with keeping the scalability feature [9].

The main contribution of this paper is the suggestion of a set of algorithms (read motifs templates, motifs templates analysis, generate pattern, code pattern, training, read dataset, segmentation, testing, and verify distance) to discover biological motifs in biodata. In the training phase, a proposed algorithm for motif mining will be designed and implemented, characterized by inter and intra-scalability, fast, low memory consumption, variety, and accuracy. It depends on the partitioning of motif template MT and dataset segmentation.

2. Literature Review

Finding DNA motifs is crucial for studying gene control. Nowadays, neural networks are the most well-liked technology because of their outstanding pattern recognition performance. Although neural networks used for motif finding usually have numerous hidden layers in their architecture, these topologies might differ [10]. This section reviewed some research on DNA motif mining algorithms and techniques.

Cao et al. [11], in this study, the authors do develop DeepARC. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are used in this precise and interpretable attention-based hybrid technique to predict transcription factor binding sites (TFBSs). DeepARC was employed positional embedding techniques to extract hidden embeddings from DNA sequences. In this study distributed embedding from DNA2Vec and positional information from one-hot coding. They used 50 datasets from ENCODE, chosen randomly, to execute the model training in their studies.

Shen et al. [12] developed a self-supervised Motif Learning Graph Neural Network (MoLGNN). Utilizing unlabeled chemicals on model proteins can considerably enhance the efficiency of computational drug screening. MoLGNN does self-learning by Utilizing network motifs and graph node attributes as self-generated labels. Additionally, it reconstructs edges using Graph Isomorphism Network Variational Auto-Encoder (GINVAE). According to experimental findings, MoLGNN performs at the cutting edge on several benchmarks.

Mohanty et al. [13] studied and analyzed the 54 most popular motif discovery procedures and algorithms from different techniques and also included a summary of their advantages and disadvantages.

Wang et al. [14] used a convolutional neural network-based architecture (CNN) that incorporates embedding layers and GloVe. In this study, every sequence in the ChIP-seq dataset was divided into several subsequences called k-mers by CNN using sliding windows and then encoded k-mers into relatively low-dimensional vectors by GloVe. Experiments demonstrate that this architecture could successfully discover motifs.

Lee et al. [15] suggested a method an improved solution for a three-step DNA motif prediction method. Only a portion of the input sequences was used in the three-step method's initial motif prediction. The remaining non-overlapping input subsets are used for position detection using the resulting starting motifs. This method, known as DeepFinder, constructs motif models using deep learning neural networks containing binding site-linked characteristics. Compared to current solutions, the results demonstrate a significant improvement.

Lanchantin et al. [16] suggested Deep Motif Dashboard. With the help of this toolset, you can classify transcription factor binding (TFBS) by extracting motifs or sequence patterns from deep neural network

models. Convolutional, recurrent, and convolutional-recurrent networks, three significant DNN models, are visualized and explained using examples. According to experimental findings, the convolutional-recurrent architecture performs the best among his three architectures. Utilizing visualization tools, it can be seen that CNN-RNN models both motivations and the relationships between them to create predictions.

G.S.pugalendhi [17] used self-organizing maps (SOM) and neural networks to find new and old regulatory motifs in DNA sequences. Based on a novel intra-node soft competition approach, this system maximizes the separation of motif signals from background signals in the data set. To more accurately represent these two classes of signals, intra-node competition is based on an adaptive weighting technique for two alternative signal models. Researchers can uncover motif sequences for various natural and synthetic datasets using the system, which is being developed as a motif analysis tool. The planned project determines the positions and weights of adenine, guanine, thymine, and cytosine. As a result, these positions can be used to determine transcription factor binding sites and transcription factors. Compared to ongoing initiatives, turnaround time is quick.

3. Theoretical Basis

The motif template consists of a bases sequence and lower and upper numbers of gaps that don't care about bases that could include any characters. A simple motif composed of nucleotides in the DNA sequence. The general structure of the motif template is represented by the Backus-Naur form (BNF) in Eq. (1) as follows [18]:

$$S1\{[l1, u1]2\{[l2, u2]3\}\{. [ln, un]\}Sn + 1\} \qquad (1)$$

Where:

S1: first simple motif.

l1: low limit of gap.

u1: upper limit of gap.

Consider the DNA motif in the example below:

AATATA[3, 5]CTG[2, 4]AATGCCG

A complex motif includes simple triple motifs of DNA bases that match the following pattern:

$S1\ \{[l1, u1]\ S2\ \{[l2, u2]\ S3$ such that:

S1 is AATATA, a simple motif containing 6 bases.

S2 is CTG, a simple motif containing 3 bases.

S3 is AATGCCG, a simple motif containing 7 bases.

[3,5] is a first gap, the distance between two simple motifs, with unspecified lower bases equal to 3 and unspecified upper bases equal to 5.

[2,4] is a second gap, the distance between two simple motifs, with unspecified lower bases equal to 2 and unspecified upper bases equal to 4.

The lower limit must have an integer value that is less than or equal to the higher limit. Motif discovery systems typically use motif templates to find the motifs that match the input template.

Motif mining in biological sequences can be defined as the problem of finding sets of short similar conserved sequence elements ('motifs') that are often short similar in nucleotide sequences with a standard biological function [19, 20].

The research problem is finding an algorithm that discovers all the occurrences of bio motifs in bio-databases in fast and low memory consumption while keeping the scalability feature. Structure or compound mining is more difficult because it contains gaps, and the gap gives results of varying lengths depending on the lower and upper limits of the gap. To show the complexity of compound motif mining, suppose that the DNA is a nibble numeric system with four digits A, C, G, and T, then the number of base four is calculated as follows:

$$[4^0\ 4^1\ 4^2\ 4^3\ 4^4 \cdots 4^n]$$

Accordingly, the possible numbers that a gap can represent in Eq. (2) as follows:

$$Number\ of\ possible\ motif = 4^{\wedge L} + 4^{\wedge L+1} + \cdots 4^{upper} \quad (2)$$

For example, the possible motifs that can be represented by the [2, 4] are:

$$(4^{\wedge 2} + 4^{\wedge 3} + 4^{\wedge 4})\ motifs$$

Therefore, the possible DNA motifs according to a submitted motif template are defined in Eq. (3) as follows:

$$no\ of\ possible\ mptifs = \sum_{i=1}^{n} \|SM_i\| + \sum_{j=1}^{M} (\sum_{k=IJ}^{H_i} (4^k)) \quad (3)$$

Where
n: the number of simple in the Motif template (MT).
M: the number of gaps in MT.
L: the lower limit in the gap.
H: the upper limit in the gap.

The most recent challenging and exciting trend in bioinformatics is motif mining in biological databases because of its importance in many applications, including detecting the predisposition to diseases, diagnostics, forensic medicine, prosthesis laboratories, criminal laboratories, corpses identification, medicines manufacturing, uncovering chemical and nuclear environmental pollution causing genetic mutations. Despite significant progress to date, discovering motif in biological data continues to be a difficult task for biologists and computer scientists. Researchers have developed motif discovery algorithms using various methods, and the progress made in this area of research is very encouraging. In order to mine simple and gaped DNA patterns, was developed a scalable, reliable, and effective algorithm. Numerous suggested algorithms for motif template parser and type checker, motif template analyzer, as well as algorithms to generate patterns, generate variance patterns, code patterns, training patterns, parsing dataset, dataset segmentation, testing of coded segmented data, and verify distance of complex motifs, will be used to complete the motif discovery process.

3.1 Artificial Neural Networks (ANNs)

The study of biological brain processing served as the basis for the computing architecture known as the artificial neural network. Numerous neural networks vary in complexity from somewhat simple to highly sophisticated [21]. We start by describing a layered feed-forward neural network as shown in Fig. 2. The processing elements in a layered feed-forward neural network are organized into layers or subgroups. A layer of processing elements independently computes the data that has been received and

forwards the results to another layer [22]. The following layer can perform its calculations and communicate the results to the layer beyond. Finally, a subset of one or more processing elements determines the network's output. Based on the weighted sum of its inputs, each processing component computes. The input layer is the first layer, and the output layer is the last [23]. Hidden layers come after the first layer and before the last layer. The processing element is called an artificial neuron because it is thought to be the same unit as a neuron in the human brain [24].

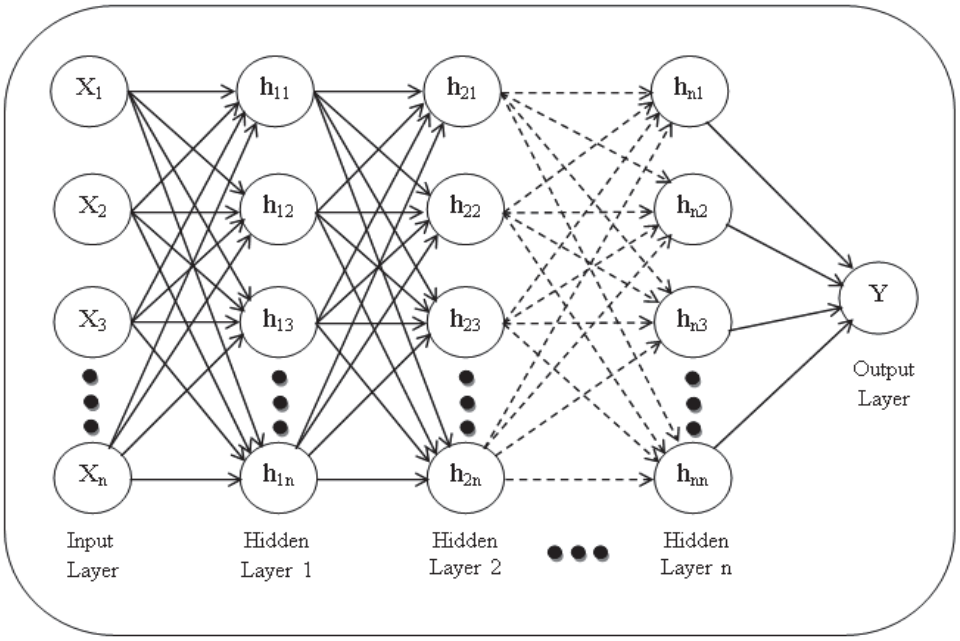

Figure (2) : Feed-Forward Neural Network

4. The Proposed System

In the structured motif mining process, the objects to be mined are undetermined, but they are described in a template covering several motifs. The number of motifs depends on many factors, such as the number of simple motifs, the sizes of simple motifs, the number of gaps, and the sizes of the gaps. In this research, we suggest training ANNs on the motif template patterns and not the contents of the DNA databases. We outline the architecture of the suggested system in this section.

The proposed approach was divided into five phases (motif preprocessing, training, dataset preprocessing, testing, and verifying distance), as presented in Fig. 3. The process's input during the motif preprocessing phase is a motif template. As a result, we require a computational model that generates patterns that are concealed within templates. Our suggested method generates a neural network for each simple motif based on a Scaled Conjugate Gradient (SCG) algorithm for training the extracted patterns from the motif template. As a result, we will receive N machines, each dedicated to a simple motif. We will go over numerous processes in the motif preprocessing phase in more detail later, such as reading and parsing the motif templates to check the template's correct writing and finding any syntactic and semantic errors. Also, it is required to determine the type of motif template, such as whether it contains only a simple motif or is structured.

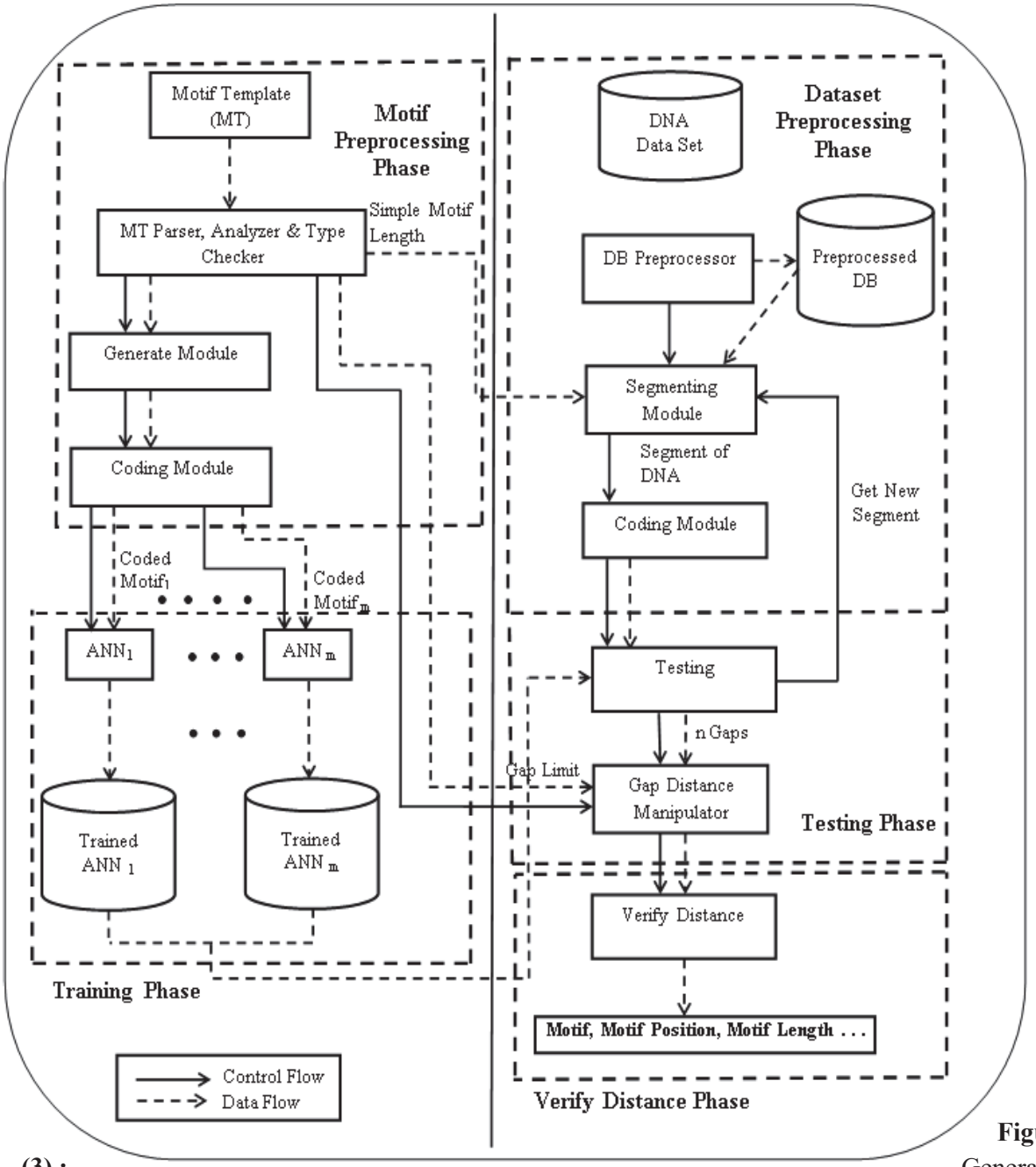

Figure (3) : General Architecture of the Proposed System

In the proposed approach, several patterns were learned; they are generated according to the simple motifs available in the motif template. The number of simple motifs and their length will affect the number of machines and the size of the input vectors for each machine. Suppose that there is one machine for each simple motif SM, and the length of SM is i bases; then the size of the training space equals 4^i because 4 bases represent the DNA sequences: A, C, G, and T. In the case of lengthened simple motifs, a biased sample of patterns from the training space is used. The patterns should be binary-coded before the training and testing phases. The DNA database is used only during

the testing phase. The dataset preprocessing phase includes several different processes, which we will review in more depth later, including reading and parsing the dataset from the fasta file. The database is queried for segments based on the lengths of simple motifs in the motif template. These segments are submitted to the testing phase.

Let's trace an illustrative example. Suppose that the submitted motif template, MT, is:
"AAACCTCCCCGCTCTGTGGCGCGC[1,3]ACGGGCCCAA"

Accordingly, the mined motifs of length 35-37 bases. This MT is in the correct form. It contains two simple motifs and one gap. The simple motif AAACCTCCCCGCTCTGTGGCGCGC contains 24 bases; therefore, it requires three dynamic ANNs, the first two machines of which will be trained to recognize AAACCTCCCC and GCTCTGTGGC. It will be fed a negative sample drawn from 1,048,576 patterns. The number of inputs to this ANN is a vector of 20 binary bits because A, C, G, and T will be coded as 00, 01, 10, and 11, respectively. The third ANN related to the first simple motif will be built to recognize GCGC. It is eight bits vector and will be trained using 44 patterns, while ACGGGCCCAA contains 10 bases. The coding process will produce the sequences presented in Table 1.

The gap [1,3] can be substituted by 1, 2, or 3 DNA bases. These possibilities will frequently be checked when a simple motif or fragment of a simple motif is recognized. After the training process of the ANNs, they are ready for the mining process of a DNA database. Suppose that the database to be mined is:
"GGAAACCTCCCCGCTCTGTGGCGCGCCACGGGCCCAAAAAGGAAACCTCCCCGCTCTG
TGGCGCGCCCCACGGGCCCAAACCTCCCGCTCTGTGGCGCGCTTACGGGCCCAACCCC".

Table 1. Motif Binary Coding

Sequence	Motifs	Motif's Binary Coding	ANNs
1	AAACCTCCCC 1st simple motif	00000001011101010101	1st ANN
2	GCTCTGTGGC 1st simple motif	10011101111011101001	2nd ANN
3	GCGC 1st simple motif	10011001	3rd ANN
4	ACGGGCCCAA 2nd simple motif	00011010100101010000	4rd ANN

After the suggested motif mining process explained in Section 3, there are three discovered motifs, as shown in Table 2.

Table 2. Illustrative Results

Sequence	Motif Sequence	Start Position	End Position	Motif Length
1	AAACCTCCCCGCTCTGTGGCGCGCCACGGGCCCAA	3	37	35
2	AAACCTCCCCGCTCTGTGGCGCGCCCCACGGGCCCAA	43	79	37
3	AAACCTCCCCGCTCTGTGGCGCGCTTTACGGGCCCAA	75	111	37

The suggested method divides the process of discovering DNA motifs into nine parts, each with its proposed algorithm. They are as follows:

First step: Read Motif Template, Parsing, and Type Checker

This step's input is a motif template, which can be taken from a text file or the keyboard. The motif template should be written using the syntax specified in the motif template. The next step is to parse the motif template to check for actual writing errors and catch both syntax and semantic errors. Additionally, the DNA bases are changed to uppercase letters. For more explanation, suppose that the input motif template is AAG!@(#$%)^7[1,40]*=AATCG[2,20] aaccGGAC [25, 50] ttAC357, the parsing of the motif template removes special characters (!@(#$%)7*=), numerals (357), and changes the lower-case letters (aacctt) within DNA bases to upper-case letters (AACCTT). This module performs its duties according to the algorithm presented in Algorithm 1.

Algorithm (1) Read Motifs Templates, Parsing, and Type Checker

Input: Motifs Templates, MT;
Output: Correct Motifs Templates, CMT;
{
Read MT;
CMT = upper case (MT);
Remove all characters not in ['A,' 'C', 'G', 'T', '[', ']', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] from CMT;
}

Second step: Motifs Templates Analysis

This step involves analyzing the motif templates to determine if they are simple or complex (compound). Determine the simple motif included in the complex motif template in the first instance, the number of gaps, and each gap's lower and upper limits. In the second scenario, where the motif template is simple, we count the bases for this motif before sending it to generate a pattern. This step is essential since the outcomes impact the following steps. For example, in the training step, a learning model will be generated and learned for each simple motif in parallel. For more explanation, consider that the input motif template is CAT[2,4]TATG[1,50]AGC.

The motif template analysis classified CAT, TATG, and AGC as simple motifs. Additionally, it will identify [2, 4] and [1, 50] as gaps, with 2 and 1 as the lower limits of the first and second gaps, while 4 and 50 are the upper limits of the first and second gaps. Remember that the gap [2,4] can match $4^2 + 4^3 + 4^4 = 336$ patterns with any bases, whereas the gap [1,50] can match [$4^1 + 4^2 + 4^3 +... + 4^{49} + 4^{50}$] patterns with any bases. This module performs its duty according to the algorithm presented in Algorithm (2).

Algorithm (2) Motifs Templates Analysis

Input: Correct Motifs Templates, CMT;
Output: Simple Motif, SM; Lower Gap, LG; Upper Gap, UG;
{
open bracket-index = find the index of ('[');
close bracket-index = find the index of (']');
comma index = find the index of (',');
calculate the number of gaps from the open
bracket number;

```
        if number of gaps = zero then
        {
           SM {1} = CMT;
           LG = [ ];
           UG = [ ];
        }
       else
           SM {1} = CMT (1 to open bracket (1) – 1);
         for each gap i
          {
             SM {i+1} = motif (close bracket (i) + 1 to open bracket (i+1)-1);
             LG (i) = eval (motif (open bracket (i) + 1 to comma (i) -1));
             UG (i) = eval (motif (comma (i) + 1 to close bracket (i) -1));
          }
      return SM, LG, UG;
}
```

Third step: Generate Pattern

After the acquisition of simple motifs from the previous step, all patterns are generated according to the number of characters in each simple motif, as seen below:

$$\underset{0}{[}\ \underset{1}{4}\ \underset{2}{4}\ \underset{3}{4}\ \underset{4}{4}\ \underset{\ldots.\ n}{4}\ ]$$

For example: if ACT is the simple motif. There are (4^3)=64 total probabilities. Then, we substitute characters for the numbers.

Algorithm (3) Generate Pattern

```
Input: Simple Motif, SM;
Output: All Patterns, AP;
 {
    if the number of characters in SM  > 10
    {
            //generate sample of random number array //
       sampleindx= randi ([1,4 exponent number of characters in SM], 1000000, 1);
            //get the probability for each random  number //
     AP= string (permn (['A' 'C' 'G' 'T'], number of characters  in SM, sampleindx));
            // generate variance pattern //
     varpat=varpattern SM, call algorithm (3.1) ;
           // union the random pattern with the variance pattern //
     AP= merge (AP, varpat);
     }
      else
       AP=string (permn(['A' 'C' 'G' 'T'], number of characters in SM));
       falsepat=  setdiff (AP, SM);
      AP=[SM, falsepat];
      output=[1, zeros ([number of characters in falsepat, 1])];
   return Ap;
}
```

Generate Variance Pattern

With this algorithm, we generate the variance patterns by exchanging each letter with all of the probabilities that are still there when the other characters are left unchanged. This module performs its duty according to the algorithm presented in Algorithm (3.1).

Algorithm (3.1) Generate Variance Pattern

Input: Simple Motif, SM;
Output: Variance Patterns, VP;

```
{
   k= 0;
   for each i character in SM
    {
      a = SM (i);
      b = remove a from ['A', 'C', 'G', 'T'];
    }
      for each element j in b
       {
          k = k + 1;
          newpattern=replace character a with the jth charachtar in b;
          add the new pattern to VP;
       }

      return VP;
}
```

Fourth step: Code Pattern

The input for this module comes from the simple motifs produced from the motif template analysis. This module will change the bases of a simple motif to its binary representation according to the following code:

Table 3. Base binary representation

No.	Base	Binary Code
1	A	00
2	C	01
3	G	10
4	T	11

Accordingly, the representation of simple motifs will be:

Table 4. The binary representation of some simple motifs

No.	Simple Motifs	Binary Code
1	CAT	010011
2	TATG	11001110
3	AGC	001001

This module performs its duty according to the algorithm presented in Algorithm (4).

Algorithm (4) Code Pattern

```
Input: All Patterns, AP;
Output: Code Patterns, CP;
  {
     for each i^th pattern in the AP
      {
         pat = AP {i};
          pat = replace 'A' with '00' in pat;
         pat = replace 'C' with '01' in pat;
         pat = replace 'G' with '10' in pat;
         pat = replace 'T' with '11' in pat;
        CP {i,1} = pat;
      }
  return CP;
 }
```

Fifth step: Training

The scaled conjugate gradient (SCG) is chosen to train the ANNs. The ANNs are trained in parallel according to the patterns generated in the previous step. Many experiments are done with different numbers of hidden layers, but ten hidden layers provide the best results; therefore, they will be adopted in this paper. The ANNs' parameters are set in Algorithm 5, such as training percentage, number of hidden layers, input vector size, etc. This module performs its duty according to the algorithms presented in (5) and (5.1).

Algorithm (5) Training

```
Input: Code Patterns, CP; Exact Output, EO;
Output: Training Function Net, TFN;
  {
     set the number of hidden layers to 10.
     set the training method as SCG call algorithm (5.1).
     initialize the neural network by the method and hidden layer.
     set all patterns as training data (net.divideparam.trainRatio = 100/100).
     use the SCG method to find the TFN.
  return TFN.
   }
```

Scaled Conjugate Gradient (SCG) Algorithm

Despite being based on conjugate directions, SCG does not execute a line search with each iteration, in contrast to other Conjugate Gradient (CG) algorithms that do. SCG is faster than other previously proposed quadratic algorithms since it is fully automated and uses a step-size scaling method to prevent time-consuming line searches at each learning iteration. The SCG algorithm is more reliable and less reliant on user-defined parameters because the step size is a function of the error function's second-order approximation. There are several methods used to estimate the step size. The SCG algorithm is proposed

as a new solution to the problem of failures in the Conjugate Gradient (CG) algorithm to minimize the global error function $E(\tilde{w})$. One call to the global error function E $(\tilde{w})$ and two calls to the gradient to the global error function E' $(\tilde{w})$are made for each iteration, resulting in calculation complexity O $(7N^2)$. Because the calculations of E $(\tilde{w})$ may be incorporated into one of the calculations of E'$(\tilde{w})$, this complexity can be decreased to O $(6N^2)$ when the algorithm is implemented. We employ this algorithm in motif discovery to train code patterns to reduce a multivariate global error function dependent on the network weights. This algorithm's inputs are code patterns and exact output, and its output is weight. The SCG algorithm is displayed as follows in [25]:

Algorithm (5.1) Scaled Conjugate Gradient (SCG)

Input: Code Patterns, CP; Exact Output, EO;
Output: weights, w;

{

Choose weight vector $\tilde{w}_1$and scalars $0 < \sigma \leq 10^{-4}$, $0 < \lambda_1 \leq 10^{-6}$, $\bar{\lambda}_1=0$;
Set $\tilde{p}_1 = \tilde{r}_1 = -E'(\tilde{w}_1)$, k =1 and success = true ;

// $\tilde{w}_1$is a weight vector in the N-dimensional euclidean space R^N, where N is the number of weights and biases in the network. σ , λ_1 Are the factors not crucial to SCG performance? Due to this, SCG does not appear to include any user-dependent parameters whose values are essential to the algorithm's performance. A clear advantage exist in comparing this to line search-based algorithms that use those parameters. $\bar{\lambda}_1$is new value of λ_1 . when the value of σ is small ($\sigma \leq 10^{-4}$) we observe that the average performance of SCG is not significantly affected. For $\sigma \leq 10^{-4}$, the number of failures (local minima) was in the range 0-2 and the standard deviation 330. When σ was less than 10^{-12}, round off errors began to have an effect. $\tilde{p}_1$, $\tilde{r}_1$are set of non zero weight vectors in R^N. $-E'(\tilde{w}_1)$is the negative gradient to global error function. k is counter //

$E(\tilde{w}_1)$ = CP * w_1 – EO ;
// $E(\tilde{w}_1)$is global error function depending on all the weights and biases is attached to the neural network //

If success = true, then calculate second order information:

$$\sigma_k = \frac{\sigma}{|\tilde{p}_k|} \; ; \; // \; \tilde{p}_k \text{set of non zero weight vector in } R^N //$$

$$\tilde{s}_k = \frac{\left(E'^{(\tilde{w}_k + \sigma_k \tilde{p}_k)} - E'^{(\tilde{w}_k)}\right)}{\sigma_k} \; ;$$

// $\tilde{s}_k$which is referred to as the scaling factor; is not strictly necessary//

$\delta_k = \tilde{p}_k^T \tilde{s}_k$; //δ_k is iteration//

Scale δ_k: $\delta_k = \delta_k + (\lambda_k - \tilde{\lambda}_k)\,|\tilde{p}_k|^2$;

// λ_k is scale the step size, the bigger of the $\lambda_k$value, the smaller the step size//

If $\delta_k \leq 0$ then make the Hessian matrix positive definite:

$$\tilde{\lambda}_k = 2\left(\frac{\lambda_k - \delta_k}{|\tilde{p}_k|^2}\right) ;$$

$$\delta_k = -\delta_k + \lambda_k |\tilde{p}_k|^2 ;$$
$$\lambda_k = \tilde{\lambda}_k ;$$

Calculate step size:

$$\mu_k = \tilde{p}_k^T \tilde{r}_k ; \qquad //\tilde{r}_k \text{ is steepest descent direction}//$$

$$\alpha_k = \frac{\mu_k}{\delta_k} ; \qquad //\alpha_k \text{ is step size}//$$

Calculate the comparison parameter:

$$\Delta_k = \frac{2\delta_k[E(\tilde{w}_k) - E(\tilde{w}_k + \alpha_k \tilde{p}_k)]}{\mu_k^2} ;$$

// Δ_k is a measure of how well $E_{qw}(\alpha_k \tilde{p}_k)$ approximates $E(\tilde{w}_k + \alpha_k \tilde{p}_k)$ in the sense that the closer Δ_k is to 1 , the better is the approximation //

If $\Delta_k \geq 0$ then a successful reduction in error can be made:

$$\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k \tilde{p}_k ;$$
$$\tilde{r}_{k+1} = -E'(\tilde{w}_{k+1}) ; \quad //\tilde{r}_{k+1} \text{ the current steepest descent direction}//$$
$$\tilde{\lambda}_k = 0, \text{success} = \text{true}.$$

If $k \mod N = 0$ then restart algorithm:

$$\tilde{p}_{k+1} = \tilde{r}_{k+1} ;$$

else:

$$\beta_k = \frac{(|\tilde{r}_{k+1}|^2 - \tilde{r}_{k+1}^T \tilde{r}_k)}{\mu_k} ; \; // \beta_k \text{ new conjugate direction}//$$

$$\tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k \tilde{p}_k ;$$

If $\Delta_k \geq 0.75$, then reduce the scale parameter:

$$\lambda_k = \frac{1}{4}\lambda_k ;$$

else:

$$\tilde{\lambda}_k = \lambda_k ;$$

Success = false.

If $\Delta_k < 0.25$, then increase the scale parameter:

$$\lambda_k = \lambda_k + \left(\frac{\delta_k(1 - \Delta_k)}{|\tilde{p}_k|^2}\right) ;$$

If the steepest descent direction $\tilde{r}_k \neq \tilde{0}, then\ set\ k = k + 1\ and\ go\ to\ If\ success = true, then$ calculate second order information else terminate and return $\tilde{w}_{k+1}$ as the desired minimum.

return w

}

In consideration of machine precision, the value σ should be as small as feasible. Experiments show that if σ is kept small ($\leq 10^{-4}$) , the value of σ does not significantly affect how well SCG performs; as a result, SCG does not appear to include user-dependent parameters, which are crucial to the algorithm's effectiveness. This is a significant advantage compared to line-search-based algorithms involving those parameters.

Sixth step: Read, Parsing and Coding Dataset

Read the dataset from the fasta file in this step. The next step is to parse a dataset to look for actual writing, remove any special characters, and identify any syntax or semantic errors. Furthermore, change every lowercase letter of a DNA base to an uppercase letter. Finally, we use a type checker to determine that the dataset's data type is DNA. This module performs its duty according to the algorithms presented in Algorithm (6). Usually, the DNA sequences are stored in FASTA format. Preprocessing includes removing special characters and identifying syntax or semantic errors. In addition, another operation on the dataset is needed, such as converting the bases to their uppercase and binary codes, which can be accomplished as a preprocessing step or during the testing phase. This module performs its duty according to the algorithms presented in Algorithm (6).

Algorithm (6) Read, Parsing, and Coding Dataset

Input: fasta file (path and file);
Output: coded data set (coded data);

```
{
  fastadata = fastaread (fullfile (path, file)) ;
  for each i^th sequence in a fasta file
   {
     data {i} = fasta data.sequence ;
     dat = upper (data {i}) ;
     s= remove all characters not in ['A' 'C' 'G' 'T'] ;
     for each character in dat
      {
        dat = replace 'A' with '00' in pat;
        dat = replace 'C' with '01' in pat;
        dat = replace 'G' with '10' in pat;
        dat = replace 'T' with '11' in pat;
      }
        coded data{i,1} = dat;
   }
}
```

Seventh step: Segmentation

In this step, we use the sliding window approach to cut the dataset at the same length as the simple motif; the simple motif's length equals the window's length. The sliding window method computes the statistic over the data in the window as a window of a defined length (Len) slides over the data character by character. This method considers the overlapping in the dataset until you check all the occurrences of the simple motif. For example, suppose that the simple motif ACCGTA consists of six characters or bases according to the sliding window method. We cut the first six characters from the dataset and moved on to characters in line window character after another to reach the last character in the dataset. The result is a binary representation of bits according to the algorithm (4), with each pair of 2 bits standing for a character and all bits representing a data segment. This module performs its duty according to the algorithm presented in Algorithm (7).

Algorithm (7) Segmentation

Input: coded Data set, coded data; simple motif, SM;
Output: Segmented Data, SD;

```
{
  for each ith SM
   {
    m = number of characters of the ith SM ;
    n = number of characters of coded data;
    get the first odd sequence from D (1:2:n- 2*m+1) ;
  for j = 2 to 2*m
   {
     add coded data (j:2:n-2*m+j) to the previous sequences;
   }
  SD = reshape the repeated sequences matrix as [n/2-m+1, 2*m];
   }
return SD;
}
```

Eighth step: Testing

In this step, the results of multiplying the values of the weights by the values of the code-segmented data equal 0 (which means the simple motif does not exist in the dataset). Alternatively, the outcome is 1 (indicating that the dataset contains the simple motif). This module performs its duty according to the algorithm presented in Algorithm (8).

Algorithm (8) Testing of coded segmented data

Input: Coded Segmented Data (CSG);
Training function net (TFN), which is a function of Y_1 and wi, i =1 to k
Output: predicted output (PY);

```
{
  Y= Y1 + w1* CSG1 + w2* CSG2 +…+ wkCSGk ;
   PY= round of Y;
return PY;
}
```

Ninth step: Verify Distance

In this step, we find the complex motif by verifying the condition of the gaps' limits. Added to the end of simple motif upper and lower limits of a gap for testing the next simple motif that falls within these limits. This module performs its duty according to the algorithm presented in Algorithm (9).

Algorithm (9) Verify Distance

```
Input: Dataset, D; Predicted Output, PO;
       Predicted Output of Simple Motif, POSM;
Output: Result, R;
 {
   if number of gaps = zero then
    {
      Calculate the number of characters in SM ;
      Index = index of (PO equal to one) ;
    }
  else
    for i =1 to the number of SM
    Calculate the number of characters in the ith SM ;
     if i =1 then
     Index = index of (PO of first SM equal to one) ;
     else
     for j =1 to the number of index
     Indx = Read PO of the ith SM ;
     Index = union (index, indx ((indx>=ind1 (j))  & (indx<=ind2 (j)))) ;
     ind = indg ;
    if ii < number of characters in SM
    {
      ind1= indx+numel (SM{ii}) + lower gap (ii) ;
      ind2= indx+numel(SM {ii}) + upper gap (ii) ;
      motifstart = ind-sum (charno (1: end-1))-sum (uppergap) ;
      motifend = ind + strlength (SM {end})-1 ;
      indx = find (PO {1}==1) ;
     }
     if the number of (lower gap) >=1
     indx2 = find (po{2}==1) ;
     else
     indx2 = ind ;
     k=0 ;
     for ii =1 to number of (motif start)
     motif begin = intersect ([motif start (ii): motif end (ii)], indx) ;
     for j = 1 to numel (motif begin)
   if ~ is empty (intersect ([motifbegin (j): motif end (ii)], indx2))
   k = k+1 ;
   R (k).segment = extract segment from location motif begin (j) to motif end (ii)) ;
return R;
 }
```

5. Results and Discussion

The experimental findings from comparing the Scaled Conjugate Gradient (SCG), One Step Secant, Gradient Descent, Bayesian Regularization, and BFGS Quasi-Newton algorithms are presented in this section. In actual data collection, these algorithms are evaluated for the results obtained using Homo sapiens dystrophin (DMD); RefSeqGene (LRG 199) on chromosome X, this dataset included 2,227,382 DNA bases, and they can be downloaded at https://www.ncbi.nlm.nih.gov/nuccore/256355061. Compared to the other supervised learning neural network algorithms, such as One Step Secant, Gradient Descent, Bayesian Regularization, and BFGS Quasi-Newton, the proposed SCG algorithm produced higher accuracy (100%) and less time during the training and testing phases. For more explanation, Table 6 shows ten motif templates and one result for each motif template, which contains the start position, end position, and motif length.

Table 6. The Experiments Results of Motif Templates

No. of Templates	Motif Templates	Mined Motif	Start Position	End Position	Motif Length
Template1	CAT[2,4]TATG[1,50]AGC	CATAATATGAAAGC	1	13	13
Template2	TAC[1,30]GTAT[2,10]CAG	TACCCGTATTCCAG	1359	1372	14
Template3	AAG[1,40]AATCG[2,20]GGAC[25,50]AC	AAGAAATCGCGGGACAAAAACCCCCGG…	471515	471583	69
Template4	AAG[1,60]AATCG[2,20]GGAC[25,70]AC	AAGCAGGAAAGGGTCGGTGATGAA…	471422	471583	162
Template5	AAAGG[1,100]CTCA	AAAGGCCATTTATGAAAAACTCA	373	395	23
Template6	ATTCCGGA	ATTCCGGA	176436	176443	8
Template7	AAAGG[1,150]CTCA	AAAGGAAAACCAATTCTAACTCA	373	395	23
Template8	ACGC	ACGC	1080	1083	4
Template9	ACACT[1,10]CCAGGC[1,20]ACACTAC[1,30]AAAAATCC[1,40]AAAGG[1,50]T	ACACTGGCACCAAAGCCAGGCAAAGATACCACAATAAAAGAA…	21	151	131
Template10	AAAGG[1,500]CTCA	AAAGGATTATATACCATGATCAGGT…	145	395	251

Table 7 compares each template's training and testing times for the following algorithms: SCG, One Step Secant, Gradient Descent, Bayesian Regularization, and BFGS Quasi-Newton. This reveals that the SCG algorithm takes the least time overall.

Table 7. The Comparison between training and testing times in seconds for five algorithms

No. of Templates	SCG		One Step Secant		Gradient Descent		Bayesian Regularization		BFGS Quasi-Newton	
	Training Time	Testing Time	Training Time	Testing Time	Training Time	Training Time	Training Time	Testing Time	Training Time	Testing Time
Template1	2.773	2.102	12.609	2.677	6.385	6.385	3.402	2.54	6.385	2.491
Template2	2.707	1.945	11.361	2.675	4.562	4.562	4.655	2.484	4.562	2.585
Template3	3.025	2.539	15.897	3.494	6.917	6.917	5.369	3.221	6.917	3.546
Template4	2.949	2.608	13.53	3.281	5.498	5.498	5.646	3.159	5.498	3.383
Template5	2.471	1.525	12.831	1.811	3.114	3.114	4.82	2.041	3.114	1.543
Template6	3.580	1.063	138.557	1.431	5.931	5.931	13.927	1.396	5.931	1.057
Template7	2.479	1.429	11.993	1.853	4.897	4.897	3.274	2.015	4.897	1.521
Template8	1.928	0.793	3.378	0.960	2.202	2.202	3.147	0.974	2.202	0.830
Template9	5.323	4.681	197.689	6.149	21.075	21.075	40.287	5.79	21.075	4.957
Template10	2.478	1.411	13.1	1.842	3.497	3.497	4.881	1.721	3.497	1.484

Table 8 shows the results for the SCG algorithm in terms (No. of Simple Motifs, No. of Gaps, Motif Length, No. of Results, and Verify Distance Time).

Table 8. Illustrative results

No. of Templates	No. of Simple Motifs	No. of Gaps	Motif Length	No. of Results	Verify Distance Time (second)
Template1	3	2	13	369	6.194
Template2	3	2	14	452	5.207
Template3	4	3	69	7	4.102
Template4	4	3	162	11	4.190
Template5	2	1	23	1337	2.453
Template6	1	0	8	3	0.015
Template7	2	1	23	1950	3.497
Template8	1	0	4	971	1.419
Template9	6	5	131	17	3.924
Template10	2	1	251	5121	7.405

6. Conclusion

In this study, we used five algorithms (SCG, One Step Secant, Gradient Descent, Bayesian Regularization, and BFGS Quasi-Newton) to search in DNA database with 2,227,382 bases for patterns that ranged in length from 4 to 509 bases. After extensive testing and experimentation with a variety of patterns, we came to the conclusion that Scaled Conjugate Gradient (SCG) is the best algorithm when compared to the rest of supervised learning neural network algorithms because it includes the following major benefits: Finding the motif takes very little time, doesn't take up much memory, and can be done

high accurately (100%) in finding all the motifs occurrences even while searching through massive amounts of data. The rest of algorithms produced unsatisfactory results due to high time consumption, high memory consumption, and low accuracy in finding all occurrences of the motifs.

References

[1] P. Singh and N. Singh, "Role of Data Mining Techniques in Bioinformatics", *International Journal of Applied Research in Bioinformatics*, Vol. 11, No. 1, pp. 51–60, 2021, DOI: 10.4018/ijarb.2021010106.

[2] Y. Wani *et al.*, "Advances and applications of Bioinformatics in various fields of life", *International Journal of Fauna and Biological Studies*, vol. 5, no. 2, pp. 3–10, 2018, [Online]. Available: http://www.ncbi.nlm.nih.gov/BLAST/ed.

[3] P. Thareja and R. S. Chhillar, "A review of data mining optimization techniques for bioinformatics applications", *International Journal of Engineering Trends and Technology*, Vol. 68, No. 10, pp. 58–62, 2020, doi:10.14445/22315381/IJETT-V68I10P210.

[4] M. Rocha and P. Ferreira, Bioinformatics Algorithms, *Elsevier*, Braga, Portugal, 2018.

[5] S. Choudhuri, BIOINFORMATICS FOR BEGINNERS, *Elsevier*, Maryland, U.S., 2014.

[6] G. Mariscal, Ó. Marbán, and C. Fernández, "A survey of data mining and knowledge discovery process models and methodologies", *The Knowledge Engineering Review Cambridge University*, Vol. 25, No. 2, pp. 137–166, 2010, DOI: 10.1017/S0269888910000032.

[7] A. Yang, W. Zhang and J. Wang, "Review on the Application of Machine Learning Algorithms in the Sequence Data Mining of DNA", *Frontiers in Bioengineering and Biotechnology*, Vol. 8, No. 2, September, pp. 1– 13, 2020, DOI: 10.3389/fbioe.2020.01032.

[8] R. Hasan and J. Uddin, "Data Mining Techniques for Informative Motif Discovery", *International Journal of Computer Applications*, Vol. 88, No. 12, pp. 21–24, February 2014, DOI:10.5120/15405-3901.

[9] Y. He, Z. Shen and Q. Zhang, "A survey on deep learning in DNA/RNA motif mining", *Briefings in Bioinformatics*, Vol. 22, No. 4, pp. 1–10, November 2021, DOI: 10.1093/bib/bbaa229.

[10] M. Mišić, A. Stanimirović, and L. Stoimenov, "Evaluation of Neural Networks Based Systems for DNA Motif Discovery", *Information Society of Serbia - ISOS, Serbia | Creative Commons License: CC BY-NC-ND*, pp. 232–236, 2018.

[11] L. Cao, P. Liu, J. Chen, and L. Deng, "Prediction of Transcription Factor Binding Sites Using a Combined Deep Learning Approach," *the journal frontiers in Oncology*, Vol. 12, No.1 June, pp. 1–10, 2022, DOI:10.3389/fonc.2022.893520.

[12] X. Shen, Y. Liu, and Y. Wu, "MoLGNN : Self- Supervised Motif Learning Graph Neural Network for Drug Discovery", *Machine Learning for Molecules Workshop at NeurIPS*, pp.1–8, 2020, [Online]. Available: https://ml4molecules.github.io.

[13] S. Mohanty, P. Kumar and A. Abdulhakim, "A Review on Planted (*l*, d) Motif Discovery Algorithms for medical Diagnose", *Multidisciplinary Digital Publishing Institute (MDPI),* Vol. 22, No. 3, pp. 1– 27, 2022, https://doi.org/10.3390/s22031204.

[14] D. Wang, Q. Zhang, C. A. Yuan, X. Qin, Z. K. Huang, and L. Shang, "*Motif Discovery via Convolutional Networks with K-mer Embedding*", *Springer International Publishing*, Vol. 11644 LNCS., pp. 374–382, 2019, https://doi.org/10.1007/978-3-030-26969-2_36.

[15] N. K. Lee, F. L. Azizan, Y. S. Wong, and N. Omar, "DeepFinder: An integration of feature-

based and deep learning approach for DNA motif discovery", *BIOTECHNOLOGY & BIOTECHNOLOGICAL EQUIPMENT*, Vol. 32, No. 3, pp. 759–768, 2018, DOI: 10.1080/13102818.2018.1438209.

[16] J. Lanchantin, R. Singh, B. Wang, and Y. Qi, "Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks", *Pacific Symposium on Biocomputing*, vol. 0, no. 212679, pp. 254–265, 2017, DOI: 10.1142/9789813207813_0025.

[17] G. S. Pugalendhi, "Detection of Regulatory Motif in Eukaryotes by Self Organizing Map Neural Networks", *International Journal of Advanced Research in Computer Science*, Vol. 4, No. 10, pp. 92–96, 2013. Available Online at www.ijarcs.info, ISSN No. 0976-5697.

[18] A. B. Yousif, H. K. Al-Khafaji, and T. Abbas, "A survey of exact motif finding algorithms", *Indones. J. Electr. Eng. Comput. Sci.*, Vol. 27, No. 2, pp. 1109–1118, 2022, DOI: 10.11591/IJEECS.v27.i2.pp1109-1118.

[19] F. Zambelli, G. Pesole, and G. Pavesi, "Motif discovery and transcription factor binding sites before and after the next-generation sequencing era", *Briefings in Bioinformatics*, Vol. 14, No. 2, pp. 225–237, April 2012, DOI:10.1093/bib/bbs016.

[20] G. Pavesi, G. Mauri, and G. Pesole, "In silico representation and discovery of transcription factor binding sites", *Briefings in bioinformatics*, Vol. 5, No. 3, pp. 217–236, September 2004, DOI: 10.1093/bib/5.3.217.

[21] V. Rao, "C++ neural networks and fuzzy logic", Vol. 3, No. 8*, IDG Books Worldwide*, 1995.

[22] C. Aggarwal, "Neural Networks and Deep Learning", USA, Springer, 2018.

[23] X. Wu, F. Lü, B. Wang, and J. Cheng, "Analysis of DNA sequence pattern using probabilistic neural network model", Journal of Research and Practice in Information Technology, Vol. 37, No. 4, pp. 353–362, 2005, Online ISSN: 1443-458X.

[24] U. S. Reddy, M. Arock, and A. V. Reddy, "Planted (l, d) - Motif Finding using Particle Swarm Optimization", *International Journal Computation Applied*, Vol. ecot, No. 2, pp. 51–56, 2010, DOI: 10.5120/1541-144.

[25] M. Moller, "A scaled conjugate gradient algorithm for fast supervised learning", *the official journal of the International Neural Network Society*, Vol. 6, No. 4, pp. 525–533, November 1993, DOI:10.1016/S0893-6080(05)80056-5