Pravinkumar M. Sonsare, Chellamuthu Gunavathi*

A novel approach for protein secondary structure prediction using encoder–decoder with attention mechanism model

https://doi.org/10.1515/bmc-2022-0043
received October 18, 2023; accepted February 9, 2024

Abstract: Computational biology faces many challenges like protein secondary structure prediction (PSS), prediction of solvent accessibility, etc. In this work, we addressed PSS prediction. PSS is based on sequence-structure mapping and interaction among amino acid residues. We proposed an encoder–decoder with an attention mechanism model, which considers the mapping of sequence structure and interaction among residues. The attention mechanism is used to select prominent features from amino acid residues. The proposed model is trained on CB513 and CullPDB open datasets using the Nvidia DGX system. We have tested our proposed method for $Q_3$ and $Q_8$ accuracy, segment of overlap, and Mathew correlation coefficient. We achieved 70.63 and 78.93% $Q_3$ and $Q_8$ accuracy, respectively, on the CullPDB dataset whereas 79.8 and 77.13% $Q_3$ and $Q_8$ accuracy on the CB513 dataset. We observed improvement in SOV up to 80.29 and 91.3% on CullPDB and CB513 datasets. We achieved the results using our proposed model in very few epochs, which is better than the state-of-the-art methods.

Keywords: bioinformatics, machine learning, deep learning, protein secondary structure prediction

Introduction

Scientists are working together with computer scientists to analyze the structure of a protein using techniques such as computational analysis. Proteins are composed of 20 amino acids. There are four types of structures that are commonly used in protein production. The primary, secondary, tertiary, and quaternary structures are the main types of protein structures. The primary structure of a protein refers to the linear sequence of amino acids in a polypeptide chain. It is determined by the order and type of amino acids linked together by peptide bonds. The primary structure is the most fundamental level of protein structure and is crucial for understanding the subsequent levels. Secondary structure refers to the local folding patterns within a polypeptide chain. Tertiary structure is the three-dimensional arrangement of the entire polypeptide chain including its secondary structures. It results from the interactions between amino acid side chains including hydrogen bonds, disulfide bonds, hydrophobic interactions, and van der Waals forces. The tertiary structure is critical for a protein’s overall shape and function [1,2]. Quaternary structure refers to the arrangement of multiple polypeptide chains (subunits) into a functional protein complex. Proteins with quaternary structure consist of two or more interacting polypeptide chains called subunits. The interactions between subunits involve the same types of forces that stabilize the tertiary structure.

Unfortunately, current methods for making predictions are time-consuming. Unfortunately, the current method could not address the real-world issue of measuring the percentage of amino acids in different forms or shapes of a protein [3–5]. An accurate prediction of the protein’s tertiary structure is also essential to determine its function. In this study, we developed a deep learning algorithm that can analyze the secondary structure of protein sequences. It can also predict the structure of the protein’s tertiary structures.

Deep learning techniques, especially neural networks, have become the dominant approach in protein secondary structure prediction. Convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformer-based models have shown significant improvements in accuracy. Transfer learning where models are pre-trained on large datasets and fine-tuned for specific tasks has gained popularity. Pre-training on related tasks such as language modeling has shown to enhance the performance of protein
structure prediction models. Methods that leverage evolutionary information such as position-specific scoring matrices (PSSMs) derived from multiple sequence alignments have been integrated into prediction models. This allows models to capture evolutionary conserved features. Hybrid models combining different types of neural network layers such as combining CNNs with RNNs have been explored to capture both local and global features in protein sequences. The availability of larger and more diverse datasets has facilitated the training of more accurate models. Efforts have been made to develop real-time prediction tools that can rapidly predict protein secondary structures. This is important for applications such as drug discovery and structural bioinformatics.

Scientists use different computer methods to figure out how amino acids are arranged in a protein. It’s like solving a puzzle. They use techniques such as clustering, support vector machines (SVMs), random forests, and RNNs. Deep learning, which is a type of advanced computer learning, is also used to check how well we can predict the structure of proteins. Scientists use a special computer method called the K-nearest neighbor to describe the sequence of amino acids in a protein. It helps them understand the patterns and relationships in the sequence [6].

The SVM classifier is developed using PSSMs. SVM is combined with the jury decision system to generate a classification [7]. Machine learning techniques separate a globular protein into membrane protein [8]. Various probabilistic approaches are also used to predict its structure [9]. The folding shape is identified by proper PSS prediction. A Fourier technique is also used to predict the percentage of residues in a protein [10]. The random forest method is also used to generate a classification for an unknown protein [11]. This method is combined with the classification-based association algorithm to predict its structure [12].

Position-specific iterative basic local alignment search tool is a computational tool that allows developers to create a position-specific scoring matrix. It is used for input in a deep neural network architecture [13].

Convolutional layers are employed to capture the local patterns and dependencies in the protein sequence. Filters (kernels) slide across the input sequence, learning hierarchical features at different levels of abstraction. These features may represent the relevant motifs or patterns associated with secondary structure elements. Stacking multiple convolutional layers enables the network to learn increasingly complex and abstract representations of the input sequence. This hierarchical learning is essential for capturing the hierarchical structure of proteins. After convolutional layers, the feature maps are typically flattened and fed into fully connected layers. These layers integrate the information from different parts of the input sequence and provide the final prediction. Proteins are composed of amino acids arranged in a specific sequence. The RNN with their ability to capture sequential dependencies is well-suited for learning patterns and relationships within these sequences.

Neural network-based five models (feed forward neural network, learning vector quantization, probabilistic neural network, CNN, and CNN fine tuning) are used to predict PSS after it has been composed of residues components [14]. RNN is used for sequence data, while the long-term memory cell is used for the recurrent neural system. The bidirectional method in which input is passed both ways predicts the PSS [3]. A supervised deep model is used to perform the sampling and scaling of the conditional mark distribution [15]. Various machine-learning components are then used to classify the data collected from sequential data [16]. The deep neural network architecture is composed of feature extractors, which are used to predict the secondary structure of a protein. The complex and straightforward relationship between the structure and sequence of the protein is maintained [17].

A CNN is a method to capture the residues. It uses a bidirectional recurrent unit to capture the global sequence feature [18]. Multi-hyperplane classification is a method to predict the structure of the protein. It considers sample points near the hyperplane for training [19]. The RNN is an algorithm used for structure prediction. It combines the deep learning capabilities of an encoder–decoder (Enc–Dec) with the recurrent architecture of a network [5]. Reciprocal and bidirectional multiple layers RNN can extract information from the residues [20]. A cascaded network of CNN and a bidirectional RNN is used to predict the PSS [21–23].

Protein secondary structures have been identified as the links in the physical processes of primary sequences, typically random coils, folding into functional tertiary structures. An efficient protein secondary structure predictor was important especially when high-resolution experiments do not solve the structure of an amino acid sequence fragment. A reductive deep learning model called multi-layer perceptron RNN has been proposed to predict either 3-state or 8-state protein secondary structures. Two sorts of characteristics are generated using the query protein's PSSM. The first feature includes fuzzy class memberships for each residue in the three secondary structure classes. The normalized PSSM feature set was the second feature set. The features are converted into vectors appropriate for neural network training using a sliding-window method with a window length of W. In convolutional neural network with a highway (CNN-H), any two neighboring convolutional layers share a highway that transports information from the current layer to the output of the next one which allows
local contexts to be maintained. Lower layers extract local context and higher layers extract long-range interdependencies which allows for efficient protein secondary structure prediction. CNN-H not only uses fewer computer resources but also performs better in terms of prediction.

We have used the attention mechanism with Enc–Dec, which allows models to focus on specific parts of the input sequence. This has been integrated into protein secondary structure prediction models. This has improved the ability of models to capture long-range dependencies in amino acid sequences. We have designed a model with a COCOB optimizer to solve the sequence learning problem. We focus on the dependency between a variable number of inputs for dealing with amino acid residues.

This article is organized into seven sections. Preliminary Section presents a glimpse of deep learning architecture. In Proposed methodology Section, the proposed models are explained. Proposed methodology and Data preparation and experimental setup Sections present data preparation and experiment details. Results and Discussion Sections present the result of the proposed model and a comparison with the existing model. Conclusion and future work Section draws conclusions and future work.

Preliminary

Sequence learning problem

The length of amino acid sequence from which the protein secondary structure is predicted is not fixed. Protein secondary structure depends on the order of amino acids that appeared in the chain. The input size is not fixed in protein secondary structure prediction where residues are dependent on each other. Each input is dependent on previous and future inputs. This sequence learning problem cannot be solved by a feed-forward neural network and CNN efficiently.

Recurrent neural networks

The number of residues from which we predict a particular protein structure is unknown. The output at each time step depends on the input given at that time step. The traditional approach may require an exact number of output functions as input length. The solution is to use RNN for handling sequence problems which are shown in Figure 1. Recurrent connections between each time steps address the problem of dependent input.

State of the network $s_t$ at timestamp $t$ is given by equations (1) and (2).

$$s_t = \sigma(Ua + Ws_{t-1} + b), \quad (1)$$

$$Z = c(Vs_t + c). \quad (2)$$

The parameters $W, U, V, c,$ and $b$ are shared across each timestep. We have used $n$-dimensional $(a_t \in \mathbb{R}^n)$ input and $d$-dimensional state $(s_t \in \mathbb{R}^d)$ with eight classes ($Z' \in \mathbb{R}^8$). The dimensions of $U, V,$ and $W$ are $U \in \mathbb{R}^{n \times d}$, $V \in \mathbb{R}^{d \times k}$, and $W \in \mathbb{R}^{d \times d}$ respectively. We used softmax as the output function and cross-entropy as the loss function. The total loss is simply the sum of the loss over all time steps shown in the equation:

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \mathcal{L}_t(\theta). \quad (3)$$

Backpropagation with a COCOB optimizer is used to minimize the loss. It is simply the derivative of loss concerning the previous layer’s weights, as shown in Figure 2 and the following equation:

$$\frac{\delta \mathcal{L}_t(\theta)}{\delta W} = \sum_{t=1}^{T} \frac{\delta \mathcal{L}_t(\theta)}{\delta S_t} \frac{\delta S_t}{\delta W}. \quad (4)$$

$\frac{\delta S_t}{\delta W}$ cannot be calculated by simply treating $S_3$ as a constant. It can be calculated as shown in the following equation:
In such network, $L_c(v)$ depends on $S_t$. $S_t$ depends on $S_{t-1}$ and $W, S_{t-1}$ in turn, it depends on $S_{t-2}$ and so on. This is called backpropagation through time. During backpropagation through time, the magnitude of $\frac{\delta L_c}{\delta S_t}$ will be very small or very large. So, $\frac{\delta L_c}{\delta W}$ will vanish or explode. This problem is known as vanishing or exploding gradients. This can be avoided by truncating backpropagation.

**Gated recurrent unit**

In traditional RNN, the old information gets morphed by current information at each new timestamp ($t$). It would be tough to get old information from previous time steps. Some problem occurs during backpropagation. So, there is a need to selectively read, write, and forget information to identify the responsible gradient, which causes the error. Numerous variations like LSTM and gated recurrent unit (GRU) are used to avoid the above-said problem. We have used GRU to deal with vanishing or exploding gradient problems.

In GRU, New input $x_t$ is overloaded with $S_{t-1}$ at timestamp $t-1$ to compute the new state $S_t$. Here, we returned only necessary information in $S_t$ from $S_{t-1}$. We introduced a vector $O_{t-1}$ (output gate) which decides what fraction of each element of $S_{t-1}$ should be passed to the next state. Learning with another parameter $(W, U, V)$ as shown in the following equation is computed as $O_{t-1}$.

$$O_{t-1} = \sigma(W_0S_{t-1} + U_0x_{t-1} + b_0). \quad (6)$$

Parameters $W_0, U_0, b_0$ are learned along with existing $(W, U, V)$.

$S_t$ capture all the information from the previous state and current input $x_t$. However, we want to use only selective information to construct a state $S_t$. This is done by the input gate, as shown in equation (7).

$$i_t = \sigma(W_iS_{t-1} + U_i x_t + b_i). \quad (7)$$

$i_t \odot \hat{S}_t$ is used for selective read state information. We combine the previous state $S_{t-1}$ and $\hat{S}_t$ to get a new state.

$$\hat{S}_t = \sigma(Wi_t \odot S_{t-1}) + Ux_t + b, \quad (8)$$

$$S_t = (1 - i_t) \odot S_{t-1} + i_t \odot \hat{S}_t. \quad (9)$$

Here, forget gate is not used. Both the gates are combined, which depends directly on $S_{t-1}$.

During forward propagation, the gates control the flow of information. They prevent any irrelevant information from being written to the state. Similarly, they control the flow of gradients during backward propagation. During the backward pass, the gradient will get be multiplied by the gate.

**Encoder–Decoder**

In Enc–Dec architecture, both are neural networks. The decoder takes encoder output as input in every step. We design Enc–Dec architecture by selecting the appropriate encoder, decoder, parameter, and loss function. We used GRU in the encoder as well as the decoder. Further, we used the attention mechanism to make Enc–Dec architecture more expressive.

All information from the encoder is not essential to decode at each timestamp. Only relevant information is fed up to the decoder at a given timestep. The decoder is fed with relevant information from the data by using the following equations:

$$e_t = f_{ATT}(S_{t-1}, c_t), \quad (10)$$

$$e_t = v^T_{att} \tanh(U_{att}S_{t-1} + W_{att}c_t). \quad (11)$$

This is useful to capture the importance of $j$th input from the encoder to get $j$th output of the decoder. The softmax function normalizes three weights.

$$a_{jt} = \frac{\exp(e_{jt})}{\sum_{j=1}^{T} \exp(e_{jt})}. \quad (12)$$

It shows the probability of getting $j$th input to produce the $t$th output. $a$ is learned based on the same parameter used in the Enc–Dec model with the COCOB optimizer.

**COCOB optimizer**

Most of the optimization algorithms in deep learning and machine learning are based on the parameters like $k$ in $k$-nearest neighbor and various hyperparameters in deep learning architecture. We are not sure about the computational price and complexity of such an algorithm. So, we thought to use a parameter-free optimization algorithm. In deep learning architecture, optimization is essential. Selection of learning rate for faster convergence is a black art. A small learning rate takes more time to reach an optimal solution whereas a significant learning rate overshoots the optimal solution. Adaptive learning rates adapt to the seal of each coordinate but do not adapt to the
distance from the optimal solution. Learning rate is difficult to tune because the distance from the initial learning rate to the optimal solution is unknown and hard to predict. So, it is difficult to achieve a convergence rate \( O\left(\frac{\|G(x^*) - x_t\|}{\sqrt{t}}\right) \sqrt{\sum \log(G(x^*) - x_t)} / \sqrt{T} = 1 + \frac{1}{T} \). The only solution is to try different learning rates.

COCOB avoids the use of a learning rate. It started optimization from stochastic optimization which was converted from online optimization where we tried to minimize loss based on knowledge of gradient. This is done by the following algorithm:

\[
\begin{align*}
&\text{for } t = 1 \text{ to } T \text{ do} \\
&\quad \text{Get } x_t \text{ from the online learning algorithm} \\
&\quad \text{Receive stochastic gradient } g_t \text{ such that} \\
&\quad E[g_t] = \nabla f(x_t) \\
&\quad \text{pass loss } l_t(x) = \langle g_t, x \rangle \text{ to the online learning algorithm} \\
&\quad \text{end for} \\
&\text{return } y_t = \frac{1}{T} \sum_{t=1}^{T} x_t \\
&\text{Some famous online algorithms are designed to work in adversarial settings and have a } O(\sqrt{T}) \text{ regret bound. They can also be used as stochastic optimization algorithms with } O(\frac{1}{\sqrt{T}}) \text{ convergence rate.}
\end{align*}
\]

Nemirovsky finds a way of getting an optimal learning rate, but it does not work in a stochastic setting [24]. Learning rate defines stepwise approach to reach optimal solution by \( \eta_t = \frac{a}{\sqrt{T} \\| \nabla f(x_t) \|} \) where \( a = 1, 2, 4, \ldots, 2^n \). Streeter proposed a method of getting a learning rate where the learning rate got doubled for every unsuccessful acceleration [25].

COCOB gives a new interpretation to optimization as gambling, compression, and prediction with log loss. The wealth after the \( T \) round will be at least \( H[S_{T+1}] \) for any random coin flip. A better convergence rate depends on a better betting strategy. COCOB tries to reach an optimal solution as fast as possible. COCOB guarantees second-order bound as follows:

\[
\beta_t = 0, \text{ Wealth}_t = 1 \\
\text{for } t = 1 \text{ to } T \text{ do} \\
\quad \text{play } x_t = \beta_t, \text{ Wealth}_{t-1} \\
\quad \text{Get gradient } g_t, \text{ define } l_t(\beta) = \log(1 - \beta g_t) \\
\quad \text{Compute } Z_t = l_t(\beta_t) = \frac{1}{1 - \beta_t g_t} \\
\quad \text{Set } \beta_{t+1} = \text{clip}(\beta_t - \frac{x_t}{1 + 2 \beta_t g_t}, \frac{1}{2}, 1) \\
\quad \text{Set Wealth}_t = \text{Wealth}_{t-1} - g_t x_t \\
\text{end for}
\]

Mathew’s correlation coefficients (MCC) and segment of overlap (SOV)

The MCC is given for particular state(s) which belongs to (H,E,C) and (G,H,I,E,B,T,S,L).

\[
\text{MCC}_t = \frac{p_m \times i_m - u_m \times o_m}{\sqrt{(p_m + u_m) \times (p_m + o_m) \times (n_m + u_m) \times (n_m + o_m)}},
\]

where

\[
p_m = R_{mn} \times \sum_{j=1}^{3/8} \sum_{k=1}^{3/8} R_{jk}, \quad o_m = \sum_{j=1}^{3/8} \sum_{k=1}^{3/8} R_{mj}.
\]

The SOV measures the average overlap between the observed and predicted states. The SOV of all states is given by the following equation:

\[
\text{SOV} = \frac{1}{\text{Size}(m)} \sum_{m=0}^{\text{Size}(m)} \left[ \frac{\text{LOV}(O, P) \times \text{Del}(O, P)}{\text{HOV}(O, P)} \times \text{Size}(states(m)) \right]
\]

where \( O \) and \( P \) are observed and predicted structure in the given state, \( O \text{ and } P \) is the number of overlapping states between \( O \text{ and } P \), \( \text{LOV}(O, P) \) is the actual overlap, \( \text{HOV}(O, P) \) is the length of the common residues for \( O \text{ and } P \) in state \( m \), and \( i_m \) is a number of residues observed in a given state. Del is the minimum difference between \( \text{LOV} \) and \( \text{HOV} \), \( \text{LOV} \), \( 0.5 \times \text{size}(O) \) and \( 0.5 \times \text{size}(P) \). Size (states) is the number of residues in given states.

Proposed methodology

This article presents a model which consists of an RNN, GRU, Enc–Dec, and attention mechanism. Training of this architecture with a modified COCOB optimizer is also presented. The proposed architecture is based on a GRU and a focused attention mechanism. It aims to provide a simple and efficient way to extract information from a source. This means that it needs to handle long sequences. The attention mechanism is used with the Enc–Dec network to handle long sequences. Here, instead of trying to encode a whole sentence, it selects a subset of vectors that can be used to translate the input. This method eliminates the need to store all of the information from the input sequence.
In the context of protein secondary structure prediction using neural networks, padding is often introduced to standardize the length of protein sequences, allowing for efficient batch processing during training. When representing protein sequences and their secondary structures in a format suitable for neural network training, encoding is essential. Amino acids are typically encoded using a numerical or one-hot encoding scheme. In a one-hot encoding, each amino acid is represented as a binary vector with all zeros except for the position corresponding to the index of the amino acid, which is set to 1. The padding is also represented using the same encoding scheme. However, a special token or vector is assigned to represent the padding. This token could be an additional binary vector of the same length as the amino acid vectors, with all zeros except for a special index set to 1.

All sequences, including the padded ones, are standardized to a fixed length of 700 residues. The amino acid vectors for the actual amino acids and the padding vector are concatenated to create the final input sequence for the neural network. Similarly, the secondary structure information is encoded for each residue in the sequence. Commonly, secondary structures like helix, sheet, and coil are represented using numerical values or one-hot encoding. For example, a three-class classification could use [1, 0, 0] for helix, [0, 1, 0] for sheet, and [0, 0, 1] for coil. The predicted output for the padding residues will also have a corresponding representation. The output layer of the neural network, which performs the classification, will generate a vector indicating the predicted secondary structure for each residue, including the padding residues.

Encoder

The GRU encoder has an input amino acid sequence like S, K, etc. Encoder states are indicated by $c_1, c_2, c_3, \ldots, c_{n-1}$. The encoder yields a solitary result vector $c$ passed as an input to the decoder. The input of the model is an amino acid sequence which is stored in vector $A$. $A = (A_1, \ldots, A_T\alpha)$, $A \in \mathbb{R}^{K_a}$ and output is given by $B = (B_1, \ldots, B_T\beta)$, $B \in \mathbb{R}^{K_b}$ where the size of input and output is given by $K_a$ and $K_b$ and the length of input and output is given by $T\alpha$ and $T\beta$.

Decoder

The decoder has additional solitary layered GRU. Decoder states are addressed by $s_1, s_2, s_3, s_{n-1}$. There are eight secondary structures named G, H, I, E, B, T, S, and L in the model as output. The issue with this design is that it uses a single vector $c$ to represent the whole sequence. The hidden state $S_i$ of the decoder from the encoder is calculated by $S_i = (1 - Z_i) \odot Z_i \odot \tilde{S}_i$.

Where

$$S_i = \tanh(W_E B_{i-1} + U_f S_{i-1} + C_i C_i),$$  \hspace{1cm} (15)

$$Z_i = \sigma(W_F E_{B_{i-1}} + U_b S_{i-1} + C_i C_i),$$  \hspace{1cm} (16)

$$\eta_i = \sigma(W_F E_{B_{i-1}} + U_b S_{i-1} + C_i C_i),$$  \hspace{1cm} (17)

where $E$ is the embedding matrix of the target protein structure. $W_F, W_e \in \mathbb{R}^{n \times m}$, $U_F, U_e \in \mathbb{R}^{n \times n}$, $C_1, C_2, C_r \in \mathbb{R}^{n \times 1}$ are weights. The initial hidden states $S_0 = \tanh(W_F \tilde{h}_0)$, where $W_e \in \mathbb{R}^{n \times n}$. Context vectors $C_i = \sum_{j=1}^{n} a_i h_j$ where $h_j$ is the $j$th annotation in the input sequence.

Attention

We used attention mechanism which allows the model to focus on different parts of the input sequence when generating each element of the output sequence. In sequence-to-sequence tasks, the goal is to map an input sequence to an output sequence. The Bahdanau Attention Mechanism introduces the concept of “context vectors.” These are dynamically calculated weighted sums of the encoder’s hidden states, and they provide the decoder with relevant information for generating each element of the output sequence [26]. Instead of using a fixed context vector for the entire decoding process, Bahdanau attention computes the alignment scores for each pair of encoder and decoder hidden states. These scores represent how well the current decoder state aligns with each encoder state.

The alignment scores are transformed into attention weights using the softmax function. These weights determine the importance of each encoder state in calculating the context vector. The context vector is then calculated as the weighted sum of the encoder hidden states, where the weights are given by the attention weights. This dynamic calculation allows the model to selectively focus on different parts of the input sequence during the generation of each output element. The context vector is concatenated with the input to the decoder, providing additional information about the relevant parts of the input sequence for generating the next output element. During training, the attention mechanism is trained alongside the rest of the model parameters. The attention weights are learned through backpropagation, allowing the model to learn which parts of the input are important at different decoding steps.
The Bahdanau attention mechanism allows the model to handle cases where different parts of the input sequence contribute differently to different parts of the output sequence. This is especially useful for tasks where the alignment between input and output sequences is not strictly monotonic. The attention mechanism provides flexibility by allowing the model to attend to different parts of the input sequence at each decoding step.

The attention system combined with the GRU enables it to focus on specific pieces in the input sequence to predict the state. This makes it an essential part of the network’s algorithm for prediction tasks. The attention model is composed of a single-layer GRU encoder and a decoder. Its inputs consist of the encoder’s result $h_t$ and the decoder’s state. The output of the attention is referred to as context vectors $c_i$.

Our attention model has a single-layer GRU encoder, again with $n$-time steps. We denote the encoder’s input vectors by $S$, $K$, etc. and the result vectors by $h_1$, $h_2$, $h_3$, $h_4$, $h_n$. The attention mechanism is situated between the encoder and the decoder; its input is made out of the encoder’s result vectors $h_1$, $h_2$, $h_3$, $h_4$, $h_n$ and the states of the decoder are $S_0$, $S_1$, $S_2$, $S_3$, ..., $S_{n-1}$. The attention’s result is a sequence of context vectors denoted by $c_1$, $c_2$, $c_3$, $c_4$, ..., $c_n$. The context vectors help the decoder to focus on certain parts of the sequence. It consists of an $i$th vector of its input. This allows it to identify the essential parts of the input. Figure 3 depicts the proposed architecture.

### Data preparation and experimental setup

#### Data preparation

Our algorithms were used on two open datasets, CB513 and CullPDB. CullPDB is a non-homologous dataset that has an identity of less than 30%. It consists of 6,128 protein amino residues labeled with $Q_8$ secondary labels. It is randomly divided into 5,600 training, 248 validation, and 280 testing sets. CB513 contains 513 proteins which are filtered CullPDB. CB513 is divided into 410 training samples and 104 validation sets [27].

PSI_BLAST assigns each amino acid to one of the eight secondary structures which are H ($\alpha$-helix), G (3-helix or 310-helix), I (5-helix or P-helix), $\beta$ (residue in isolated beta-bridge), E (extended sheet), T (hydrogen bond turn), S (bend), and “.” (any other structure). These eight secondary structures are reduced to three-state helix (H), sheet (E), and coil (C). The critical assessment of structure prediction

![Figure 3: Architecture of the proposed algorithm.](image)
reduction method assigns \((H, G, I)\) to \(H\), \((E, B)\) to \(E\), and all other states to \(C\).

Eight neurons encode secondary classes at the output layer with the following binary representation:

\[
\begin{align*}
G &= [1 0 0 0 0 0 0 0] \\
H &= [0 1 0 0 0 0 0 0] \\
I &= [0 0 1 0 0 0 0 0] \\
E &= [0 0 0 1 0 0 0 0] \\
B &= [0 0 0 0 1 0 0 0] \\
T &= [0 0 0 0 0 1 0 0] \\
S &= [0 0 0 0 0 0 1 0] \\
L &= [0 0 0 0 0 0 0 1] \\
H &= [1 0 0], \\
E &= [0 1 0], \\
C &= [0 0 1].
\end{align*}
\]

**Experimental set-up**

RNN and GRU in this article have the size of hidden layer 300 and a fully connected output layer consisting of 3 and 8 neurons for \(Q_3\) and \(Q_8\) classification, respectively. Python Spyder, v5.1.1 tool is used for implementation with Keras 2.4.0 for neural network packages where TensorFlow v2.6.0 is running on the top. Our networks consist of a dense layer for dot multiplication and bias function, an activation function to initiate the learning tasks and a dropout layer to avoid overfitting.

**Parameter initialization**

We have started with the random orthogonal matrix weights. We then sampled each element from the random distribution of the mean 0 and variance 0.0012. The dropout value of 0.6 was set to minimize the error. The COCOB optimizer typically learns repeated features by continuously updating the hyperparameter. The batch size 128, the epoch value 200, and the loss/error over the training set are fixed and executed.

**Training procedure**

We trained our entire deep neural network with a COCOB optimizer. As per our previous work, the COCOB optimizer worked better than other optimizers. The training data were passed in 128 sequences to avoid computational inefficiency. Dropout regularization is used to avoid overfitting in this model. Here, we are presenting the training on the Enc–Dec model with an attention mechanism. The network is given amino acid features to get the protein chain’s residue as an output. The attention model predicts the \(Q_8\) state from the input sequence generated by the encoder.

The attention weights are learned through a softmax function and an external network. They represent the importance of \(h_t\) in deciding the next hidden state. A significant attention weight can cause the GRU to focus only on the input \(x_t\). The \(f_c\) network is trained to handle the GRU’s prediction error terms by backpropagation. This process is carried out through the decoder and the \(f_c\) attention network.

This mechanism aims to allow the decoder to pay attention to the various parts of the input sequence. This allows the encoder to avoid encoding all of the information in the sequence into a single vector. The training algorithm is shown as follows:

**Initialization:**

Input data:
\[
\beta, \text{Wealth}_n = 1 \\
x = \{x_1, x_2, ..., x_N\} \text{ protein sequence of length } N \\
S_t = \text{timestamp at } t
\]

Encoder:

Forward propagation:
\[
\begin{align*}
O_{t-1} &= \sigma(W_h S_{t-1} + U_h x_{t-1} + b_h) \\
i_t &= \sigma(W_i S_{t-1} + U_i x_t + b_i) \\
\tilde{S}_t &= \sigma(W_c (O_t \odot S_{t-1}) + U_c x_t + b_c) \\
S_t &= (1 - i_t) \odot S_{t-1} + i_t \odot \tilde{S}_t
\end{align*}
\]

Backward propagation:
for all \(t = 1, 2, ..., N\) do
\[
\begin{align*}
x_t &= \beta_t S_{t-1} \\
i_t &= \beta_t S_{t-1} \\
Z_t &= i_t \beta_t = \frac{z}{1 - \beta_t} \\
\beta_t &= \text{clip} \left[ \beta_t - \frac{z}{1 - \beta_t}, -\frac{1}{2}, \frac{1}{2} \right] \\
S_t &= S_{t-1} - g_t x_t
\end{align*}
\]

Attention:
for \(t = 1, 2, ..., N\) do
\[
\begin{align*}
\epsilon_t &= \sum_{j=1}^{M} \alpha_t h_j \\
a_t &= \frac{\exp(\epsilon_t)}{\sum_{j=1}^{M} \exp(\epsilon_t)} \\
C_t &= \sum_{j=1}^{M} a_t h_j
\end{align*}
\]
Forward Propagation:
for $t = 1$ to $N$
\[ O_{t-1} = \sigma(W_0 S_{t-1} + U_0 C_{t-1} + b_0) \]
\[ i_t = \sigma(W_1 S_{t-1} + U_1 C_t + b_1) \]
\[ O_t = \sigma(W_2 O_{t-1} + U_2 C_t + b) \]
\[ S_t = (1 - i_t) \odot S_{t-1} + i_t S_t \]
end for

Backward propagation:
for all $i \in \{1, 2, \ldots, N\}$ do
\[ C_t = \beta_S S_{t-1} \]
\[ l_i(\beta) = -\log(1 - \beta g_i) \]
\[ Z_i = l_i(\beta) = \frac{s_i}{1 - \beta} \]
\[ \beta_{i+1} = \text{clip}\left(\beta_i - \frac{Z_i}{1 + \sum_{i'=1}^{i} l_i'}, \frac{1}{2}, \frac{1}{2}\right) \]
\[ S_t = S_{t-1} - g C_t \]
end for

Results

Our model evaluated the accuracy of protein secondary structure by $Q_3$, $Q_8$, Mathew correlation coefficients, and SOV. $Q_3$, $Q_8$, and MCC predict individual amino acids. Accuracy based on single residues does not guarantee the overall accuracy of protein secondary structure. Because some secondary structure states depend on adjacent amino acids. We are using SOV scores to guarantee the accuracy of protein secondary structure prediction.

Matrix with a size $3 \times 3$ and $8 \times 8$ is defined to measure accuracy. $R_{mn}$ represented the number of residues observed in state $m$ and predicted as the state $n$. The parameters $m$ and $n$ belong to \{H,E,C\} and \{G,H,I,E,B,T,S,L\} for $Q_3$ and $Q_8$ accuracy respectively.

\[ Q_3 = \frac{1}{i} \sum_{m=1}^{3} R_{mn}, \quad (18) \]
\[ Q_8 = \frac{1}{i} \sum_{m=1}^{8} R_{mn}, \quad (19) \]

$i$ is the total number of residues.

The accuracy of the individual secondary structures is calculated as $Q_i = \frac{1}{i_m} R_{mn}$ where $i_m$ is the total number of residues in state $m$.

Our algorithms are tested on CullPDB and CB513 datasets. $Q_3$ and $Q_8$ accuracies observed for the CullPDB and CB513 dataset by using RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism are shown in Figure 3. We achieved a $Q_3$ accuracy on the CullPDB dataset of 59.23, 62.13, 69.87, and 70.69% for RNN, GRU, Enc–Dec, and Enc–Dec with attention.

Figure 4: Performance of the proposed model.
mechanism respectively. We achieved a $Q_3$ accuracy on the CB513 dataset of 69.17, 73.18, 77.86, and 79.38% for RNN, GRU, Enc–Dec, and Enc–Dec with an attention mechanism. We achieved a $Q_8$ accuracy on CullPDB dataset of 60.34, 63.92, 76.28, and 78.93% for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism, respectively. We have found that Enc–Dec with attention mechanism model gives better results on CullPDB and CB513 datasets which is shown in Figure 4.

The corresponding $Q_H$ (Helix Accuracy), $Q_E$ (Sheet Accuracy), and $Q_C$ (Coil Accuracy) obtained for the CullPDB and CB513 datasets by using our models are shown in Table 1. We achieved corresponding $Q_H$ accuracy on the CullPDB dataset of 61.10, 61.28, 74.42, and 73.23% for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism respectively. We achieved corresponding $Q_E$ accuracy on the CullPDB dataset of 60.23, 64.23, 75.31, and 74.91% for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism respectively. We achieved corresponding $Q_C$ accuracy on the CullPDB dataset of 59.6, 62.10, 70.23, and 76.82% for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism respectively. We achieved 70.3, 69.24, and 71.30% $Q_H$, $Q_E$, and $Q_C$ accuracy, respectively on CB513 dataset for RNN. We achieved 72.10, 73.20, and 70.23% $Q_H$, $Q_E$, and $Q_C$ accuracy, respectively, on the CB513 dataset for GRU. We achieved 78.13, 71.37, and 73.24% $Q_H$, $Q_E$, and $Q_C$ accuracy, respectively, on CB513 dataset for Enc–Dec. We achieved 80.23, 79.14, and 78.92% $Q_H$, $Q_E$, and $Q_C$ accuracy, respectively, on the CB513 dataset for Enc–Dec with an attention mechanism.

SOV scores and MCC measurements observed on CullPDB and CB513 for our models are shown in Table 1. We achieved 79.83, 84.34, 88.10, and 91.3% SOV scores for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism, respectively. We achieved 76.26, 77.13, 80.29, and 80.29% SOV scores for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism on CullPDB, respectively. We achieved 0.81, 0.80, 0.92, and 0.94 corresponding MCCs for helix ($C_H$), respectively, on the CB513 dataset for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism. We achieved 0.84, 0.80, 0.97, and 0.90 corresponding MCCs for sheet ($C_E$), respectively, on the CB513 dataset for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism. We achieved 0.97, 0.81, 0.91, and 0.89 corresponding MCCs for coil ($C_C$) accuracy respectively on the CB513 dataset for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism, respectively. We achieved 0.83, 0.85, 0.81, and 0.88 corresponding

### Table 2: $Q_8$ accuracy comparison of the proposed method with literature methods on CB513

<table>
<thead>
<tr>
<th>Model</th>
<th>$Q_8$ accuracy (%) on CB513</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRNN</td>
<td>51.1</td>
</tr>
<tr>
<td>CNF</td>
<td>64.9</td>
</tr>
<tr>
<td>GSN</td>
<td>66.4</td>
</tr>
<tr>
<td>LSTM</td>
<td>67.4</td>
</tr>
<tr>
<td>SSREDN</td>
<td>68.2</td>
</tr>
<tr>
<td>1D-ConvNet</td>
<td>73.3</td>
</tr>
<tr>
<td>1D-ConvNet with BLSTM</td>
<td>74.45</td>
</tr>
<tr>
<td>Encoder–decoder</td>
<td>75.89</td>
</tr>
<tr>
<td>Encoder–decoder with attention</td>
<td>77.13</td>
</tr>
</tbody>
</table>

### Table 3: $Q_8$ accuracy comparison of proposed method with literature methods on CullPDB

<table>
<thead>
<tr>
<th>Model</th>
<th>$Q_8$ accuracy (%) on CullPDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSpro</td>
<td>66.6</td>
</tr>
<tr>
<td>RaptorX-SS8</td>
<td>69.7</td>
</tr>
<tr>
<td>GSN</td>
<td>72.1</td>
</tr>
<tr>
<td>SSREDN</td>
<td>73.1</td>
</tr>
<tr>
<td>1D-ConvNet</td>
<td>64.48</td>
</tr>
<tr>
<td>1D-ConvNet with BLSTM</td>
<td>75.04</td>
</tr>
<tr>
<td>Encoder–decoder</td>
<td>76.28</td>
</tr>
<tr>
<td>Encoder–decoder with attention</td>
<td>78.93</td>
</tr>
</tbody>
</table>
We achieved 0.87, 0.89, 0.90, and 0.92 corresponding $C_H$ accuracy, respectively, on the CullPDB dataset for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism, respectively. We achieved 0.71, 0.78, 0.76, and 0.79 corresponding $C_C$ accuracy, respectively, on the CullPDB dataset for RNN, GRU, Enc–Dec, and Enc–Dec with attention mechanism, respectively, which is shown in Table 1.

### Table 4: $Q_3$ accuracy comparison of proposed method with literature methods on CB513

<table>
<thead>
<tr>
<th>Model</th>
<th>$Q_3$ accuracy (%) on CB513</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSpro</td>
<td>78.5</td>
</tr>
<tr>
<td>RaptorX-SS8</td>
<td>78.3</td>
</tr>
<tr>
<td>PSIPRD</td>
<td>79.2</td>
</tr>
<tr>
<td>1D-Convnet with BLSTM</td>
<td>66.4</td>
</tr>
<tr>
<td>Encoder–decoder with attention</td>
<td>79.38</td>
</tr>
</tbody>
</table>

Figure 5: Comparison of SOV with literature methods.

Figure 6: Comparison of MCC with literature methods.
Discussion

We have compared all the evaluating parameters of our model with a few of the existing models. It is observed that our model is performing better than the existing models. We have compared our model with BRNN [28], CNF [29], GSN [4], LSTM [3], SSREDN [5], and 1D-Convnet and 1D-Convnet with BLSTM [30]. Q8 comparison on CB513 is shown in Table 2. Our model achieved 77.13% Q8 accuracy and observed significant improvement in the performance as compared to another model in the literature.

We have compared our model with SSpro [31], RaptorX-SS8 [32], GSN, SSREDN, 1D-Convnet and 1D-Convnet with BLSTM. Q8 comparison on CullPDB dataset is shown in Table 3. Our model achieved 78.93% Q8 accuracy and observed significant improvement in the performance as compared to other models in the literature.

We have compared our model with SSpro, RaptorX-SS8, PSIPRD, and 1D-Convnet with BLSTM for Q3 accuracy. Q3 comparison on the CB513 dataset is shown in Table 4. Our model achieved 78.93% Q3 accuracy and observed significant improvement in the performance as compared to another model in the literature.

The Q8 accuracy measures the overall accuracy of predicting the eight different states of protein secondary structure. If the model is good at distinguishing between the various fine-grained states (Alpha Helix, 3–10 Helix, Pi Helix, Beta Strand, Beta Bridge, Turn, Coil, Bend), it may achieve high accuracy on Q8. However, Q3 accuracy measures a more coarse-grained classification into three states (Helix, Strand, Coil). Achieving high Q3 accuracy requires the model not only to differentiate between fine-grained states but also to correctly group them into the broader categories.

The architecture of the Enc–Dec model plays a crucial role. If the model is more focused on capturing the fine-grained details of the input sequence, it might perform well on Q8 but struggle to generalize to the rough Q3 classification. The encoding process in the Enc–Dec model might emphasize capturing the detailed features of the input sequence, making it effective for Q8 but less suited for the simpler Q3 classification. The model might overfit to specific features related to the fine-grained states, making it less robust in grouping them into broader categories. This overfitting could lead to high accuracy on Q8 but lower accuracy on Q3.

Figures 5 and 6 compare the SOV scores and the MCC measurements for CB513 with SPIDER2, JPred4, RptoX, SSPROS, and observed a significant improvement in the performance.

Conclusion and future work

In this article, we have proposed the Enc–Dec model with an attention mechanism to predict the protein structure from amino acid residues. We have started our experimentation with RNN on CB513 and CullPDB open datasets, in which we faced the problem of vanishing gradient. GRU model is used to overcome the vanishing gradient problem. The attention mechanism in our model is used to select prominent features from amino acid residues which helps to increase the performance of our model. We used a modified COCOB optimizer for hypertuning of the hyperparameter. We found significant improvement in Q3 accuracy, Q8 accuracy, SOV, and MCC using our models. Further, we can improve our model’s Q3 accuracy on the CullPDB dataset. As tuning of hyperparameter is the biggest challenge in deep learning, we can further improve the learning ability of our model based on momentum or learning rate.

Acknowledgements: We appreciate the support from the Department of Computer Science and Engineering, Shri Ramdeobaba College of Engineering and Management, Nagpur, for allowing us to experiment on the Nvidia DGX Station and the School of Computer Science and Engineering, Vellore Institute of Technology, Vellore.

Funding information: Authors state no funding involved.

Conflict of interest: Authors state no conflict of interest.

Data availability statement: The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

References


