Protein secondary structure prediction by using deep learning method

Yangxu Wang, Hua Mao*, Zhang Yi

Machine Intelligence Laboratory, College of Computer Science, Sichuan University, Chengdu 610065, People's Republic of China

Keywords:
Deep learning
Secondary structure prediction
Encoder–decoder networks
Recurrent neural networks

A B S T R A C T

The prediction of protein structures directly from amino acid sequences is one of the biggest challenges in computational biology. It can be divided into several independent sub-problems in which protein secondary structure (SS) prediction is fundamental. Many computational methods have been proposed for SS prediction problem. Few of them can model well both the sequence-structure mapping relationship between input protein features and SS, and the interaction relationship among residues which are both important for SS prediction. In this paper, we proposed a deep recurrent encoder–decoder networks called Secondary Structure Recurrent Encoder–Decoder Networks (SSREDNs) to solve this SS prediction problem. Deep architecture and recurrent structures are employed in the SSREDNs to model both the complex nonlinear mapping relationship between input protein features and SS, and the mutual interaction among continuous residues of the protein chain. A series of techniques are also used in this paper to refine the model's performance. The proposed model is applied to the open dataset CullPDB and CB513. Experimental results demonstrate that our method can improve both Q3 and Q8 accuracy compared with some public available methods. For Q8 prediction problem, it achieves 68.20% and 73.1% accuracy on CB513 and CullPDB dataset in fewer epochs better than the previous state-of-art method.

1. Introduction

Discovering protein's structure and biological functions are very important for understanding their biological processes, such as the protein-protein interactions [1], protein complexes identification [2] and protein structure prediction. Protein structure prediction, elucidating the complex relationship between a protein sequence and its structure, is one of the most important challenges in computational biology [3]. The most elemental task of protein structure prediction is protein secondary structure (SS) prediction, which aims to discover the structural states of amino acids. SS represents the local conformation of the polypeptide backbone of proteins and provides a bridge that links the primary sequence and the tertiary structure, which is very helpful for many structural and functional analysis tools [4,5].

Typically, protein secondary structures can either be divided into three states (α-helix (H), β-strand (E) and coil region (C)) or be further classified into eight fine-grained states (3_{10}-helix (G), α-helix (H), π-helix (I), β-strand (E), β-bridge (B), β-turn (T), high curvature regions (S) and irregular loop (L)). SS prediction is usually evaluated by Q3 and Q8 accuracy, which measures the percentage of residues for which 3-state or 8-state SS is correctly predicted. Currently, extensive research efforts have been spent on applying computational methods to address the Q3 prediction problem, but very few to the more challenging Q8 prediction problem.

Hidden markov model (HMM) has been applied to 3-state SS prediction problem [6]. Although HMM can describe the interactions among adjust residues, it's very challenging for HMM to model the complex nonlinear relationship between input protein features and SS. Support vector machine (SVM) [7] can deal with this complex nonlinear mapping, but it's challenging for SVM to take into consideration the interactions among adjacent residues. To our best knowledge, by using a 2-stage neural networks (NNs) method [8], so far the best Q3 accuracy is about 80%. For the Q8 prediction problem, existing methods [9,10] fail to provide promising results. The problem may be that most of these mentioned methods are shallow architectures. The limitation of them is that it's very difficult for a relatively shallow architectures to model well both the complex sequence-structure relationship between input protein features and SS, and the mutual interaction relationship among residues. However, they are both important for SS prediciton [10,11].

Nowdays, NNs with deep architectures, also called deep neural networks (DNNs) become the most powerful machine learning techniques for pattern recognition [12,13]. With the ability of mapping unorganized low-level features into high-level laten data representations which are more suitable for a final classification prob-

* Corresponding author.
E-mail addresses: mellowxu@gmail.com (Y. Wang), huamao@scu.edu.cn (H. Mao), zhangyi@scu.edu.cn (Z. Yi).

lem [14], DNNs can better model the complex sequence-structure relationship for SS prediction. Cheng [15] proposed a typical deep belief networks model for Q3 prediciton, in which each layer is a restricted Boltzmann machine (RBM). Another DNNs approach is reported in [16], which develop a multi-step iterative deep networks model that predicts four different sets of structural properties including 3-state SS. In order to better capture the mutual interactions among residues, a deep convolutional generative stochastic network is proposed in [17] for a 8-state SS prediction problem, which may be the best 8-state predictor as we know. However, the performance of this approach is sensitive to the chosen convolution window size which is difficult to determine. Recurrent neural networks (RNNs) are also adopted for this problem as they can employ contextual information of input sequence and learn the variable-width dependencies in the protein chain [11]. However, the gradient problem still limits the application of these RNN-based approaches [18].

In this paper, we proposed a deep recurrent network, called the Secondary Structure Recurrent Encoder–Decoder Networks (SSREDNs) for the challenge of both Q3 and Q8 prediction. SSREDNs employ the encoder–decoder architecture [19], a typical deep architecture to model the sequence-structure relationship. This architecture allows its encoder part to encode a given protein sequence into a representation layer and the its decoder part decodes this learned representation into new feature space. Compared with classical deep architectures, it enhances the model's learning and representation abilities [19]. Second, different with [17] which use a deep convolutional networks, we use a new-typed recurrent structure, called gated recurrent units (GRUs) [20] in our networks to model the mutual interaction relationship among residues. It overcomes the gradient problem and can better learn both the adjacent and long-range interactions among residues without the problem of choosing the size of the convolution window. The proposed networks can model both the complex nonlinear sequence-structure mapping between input protein features and SS, and the mutual interaction relationship among residues. In particular, a stack auto-encoder (SAE) architecture [21] is also contained in the encoder part of our networks to automatically learn compact and representative features for input. It's especially helpful for proteins as we lack the necessary intuition or knowledge for hand-crafting features involving multiple amino acids [22]. A series of techniques are also adopted to refine the performance of the networks, such as the dropout technique [23] and the optimized algorithm named Adam [24].

For the evaluation purpose, the proposed method is applied to the benchmark dataset CB513 and CullPDB for both Q3 and Q8 prediction. First, SSREDNs outperforms the best Q8 predictor [17], especially on some SS types which are difficult to predict. Furthermore, it also outperforms most of the public available methods for both Q3 and Q8 prediction.

The paper is organized as follows. Section 2 gives an outline of the deep learning architectures and techniques used in this work. In Section 3, the proposed networks model and its learning algorithm are introduced. Experimental details are elucidated in Section 4. Finally, the conclusions and future work are drawn in Section 5.

2. Preliminary

In this section, a brief introduction of DNNs and the SAE are presented. In addition, the GRUs used to deal with the long-term interactions between residues are also described.

2.1. Deep neural networks (DNNs)

DNNs are artificial neural networks with multiple hidden layers between the input and output layers. In DNNs, the deep-seated

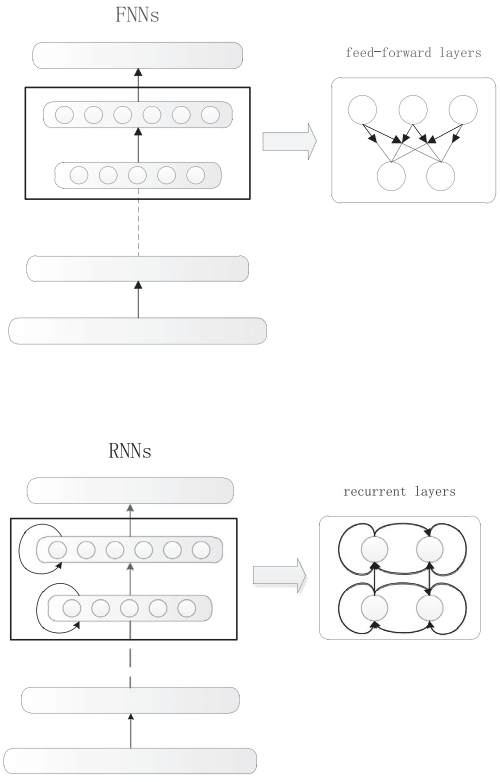


Fig. 1. (a) Deep feed-forward networks. (b) Deep recurrent networks.

layers enable the extraction of more abstract features from lower layers. Although DNNs are typically designed with feed-forward layers, recently researchers have successfully developed DNNs with recurrent structures for applications such as natural language processing [25]. As shown in Fig. 1(a), deep feed-forward networks can represent complex data in a multi-layer network structure. With sufficient layers of learning neurons, it can map unorganized low-level features to a high-level data manifold, which is more suitable for a final classification or regression task.

Recurrent units has connections between neural units to form a directed cycle, which allows them to exhibit dynamic temporal behavior of arbitrary sequential inputs, as shown in Fig. 1(b). Deep networks with recurrent structure, also called deep recurrent neural networks (DRNNs) have already been successfully used for complex sequential data analysis, such as speech recognition, image semantic understanding and online handwritten recognition. With the extra capability of capturing the spatial/temporal dependencies within sequences, DRNNs are more difficult to train due to the vanishing-gradient and over-fitting problems [18].

2.2. Stack auto-encoder (SAE)

The classical auto-encoder (AE) [26] is a three-layer neural network (as shown in Fig. 2(a)): an encoder layer maps input to a representation layer and a decoder layer reconstructs the input from the representation layer. The first two layers are regarded as an encoder, while the later two layers are regarded as a decoder. Given input x and let y denote the state of neurons in the representation layer, an AE's feed-forward process can be formalized as follows:

$$y = \sigma(W_h x + b_h), \tag{1}$$

$$\widetilde{x} = g\left(W_h^T y + b_o\right), \tag{2}$$

where $\widetilde{x}$ is the reconstruct output, and W_h, b_h, W_h^T and b_o respectively denote the weight matrix of the encoder, bias vector of the

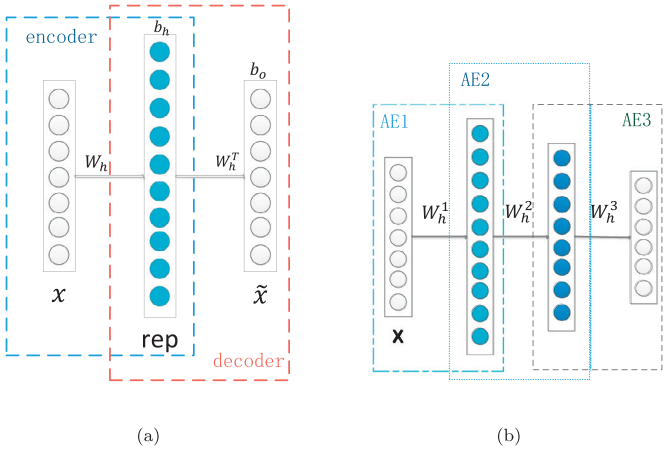


Fig. 2. (a) A typical with three-layer AE. The input x and output $\tilde{x}$ have the same shape. After trained, the weights of both encoder and decoder are fixed. The encoder part can be stacked to an SAE then. (b) A four-layer Stack AE. It is pre-trained layer-by-layer as three stack AEs. The representation learned at the previous AE is used as input for learning the next AE's representation.

hidden layer, weight matrix of the decoder and bias vector for the output layer. As the reconstruction error is minimized in a well trained AE, the cost function J_{AE} can be defined as:

$$J_{AE} = L(x, \tilde{x}), \tag{3}$$

where $L(\cdot)$ is the function measuring the difference between the original inputs and the reconstructed ones.

AE can be trained by the standard back-propagation algorithm [27]. AEs can be further stacked to create an stacked auto-encoder (SAE) [21] (as shown in Fig. 2(b)). In a SAE, the representation learned at the previous level is used as input for learning the next level's representation, which yields a better representation for deeper networks. It can automatically discover and extract useful features or representations in a layer-wise procedure. The training of SAE follows the recently proposed pre-training and fine-tuning paradigm. Automatically learning feature is important for the protein SS prediction as prior knowledge for hand-crafting features involving multiple amino acids is barely available. Therefore, SAE is adopted in SSREDNs to pre-train the feed-forward layers that follow the input layer to learn better features automatically.

2.3. Gated recurrent units (GRUs)

2.3.1. Classical recurrent unit

Recurrent units have connections towards themselves. A classical recurrent unit is shown in Fig. 3(a), its update procedure is formalized as in Eq. 4:

$$h_t = g(W_h \cdot [x_t, h_{t-1}]), \tag{4}$$

where x_t is the input at time t, h_t and h_{t-1} is the hidden state of the recurrent unit at time t and $t-1$, respectively. W_h is the weights matrix. The bias is omit here.

As the nonlinear active function $g(\cdot)$ will act on the recurrent unit repeatedly over time, it is difficult to train a recurrent networks. The gradient tends to either vanish (most of the time) or approach infinity during the training process. This makes gradient-based optimization method difficult, and it becomes even more problematic because the effect of long-term dependencies is hidden (being exponentially smaller with respect to sequence length) by the effect of short-term dependencies [20].

2.3.2. Gated recurrent unit

As classical recurrent units have the gradient problems when training, a new-typed recurrent unit called GRU is used in our work. GRU was proposed to solve the gradient problem in training recurrent networks by allowing each recurrent unit to capture the dependencies of different time scales in an adaptive manner (see Fig. 3(b)). Unlike the classical recurrent unit, the activation h_t of the GRU at time t is a linear interpolation between the previous activation h_{t-1} and the candidate activation $\tilde{h}_t$:

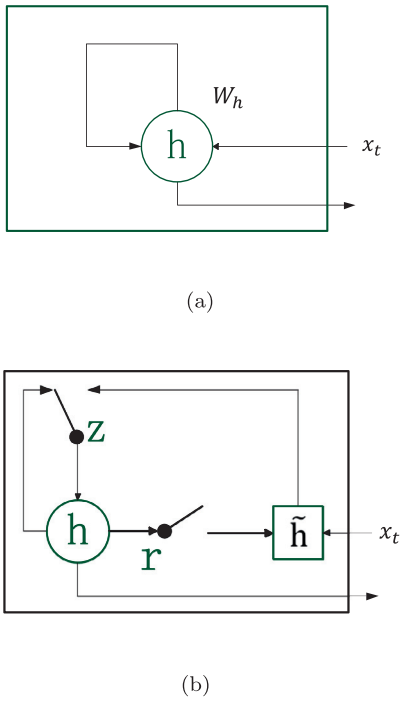


Fig. 3. (a) Classical recurrent unit. (b) Gated recurrent Unit.

$$h_t = z_t \tilde{h}_t + (1 - z_t) h_{t-1}. \tag{5}$$

Here, z_t denotes the update gate of GRU. It decides how much the unit updates its activation or content. This allows the error to easily back-propagate through the unit without vanishing too quickly or blowing up as a result of passing through multiple time steps, and thus reducing the difficulty of training the recurrent networks. z_t is computed as follow:

$$z_t = \sigma(W_z \cdot [x_t, h_{t-1}]), \tag{6}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{7}$$

The candidate activation $\tilde{h}_t$ is computed as [20]:

$$\tilde{h}_t = \tanh(W \cdot [x_t, r_t \odot h_{t-1}]), \tag{8}$$

which is similar to the traditional recurrent unit except for the inclusion of the $r_t \odot h_{t-1}$ term; r_t is a set of reset gates and $\odot$ is an element-wise multiplication. r_t is computed as follows:

$$r_t = \sigma(W_r \cdot [x_t, h_{t-1}]), \tag{9}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{10}$$

We use x_t to denote the sequential input at time t. With GRUs, it is easy for each unit to remember the existence of a specific feature in the input stream over a long series of time steps. Any feature that is decided to be important by the update gate will not be overwritten.

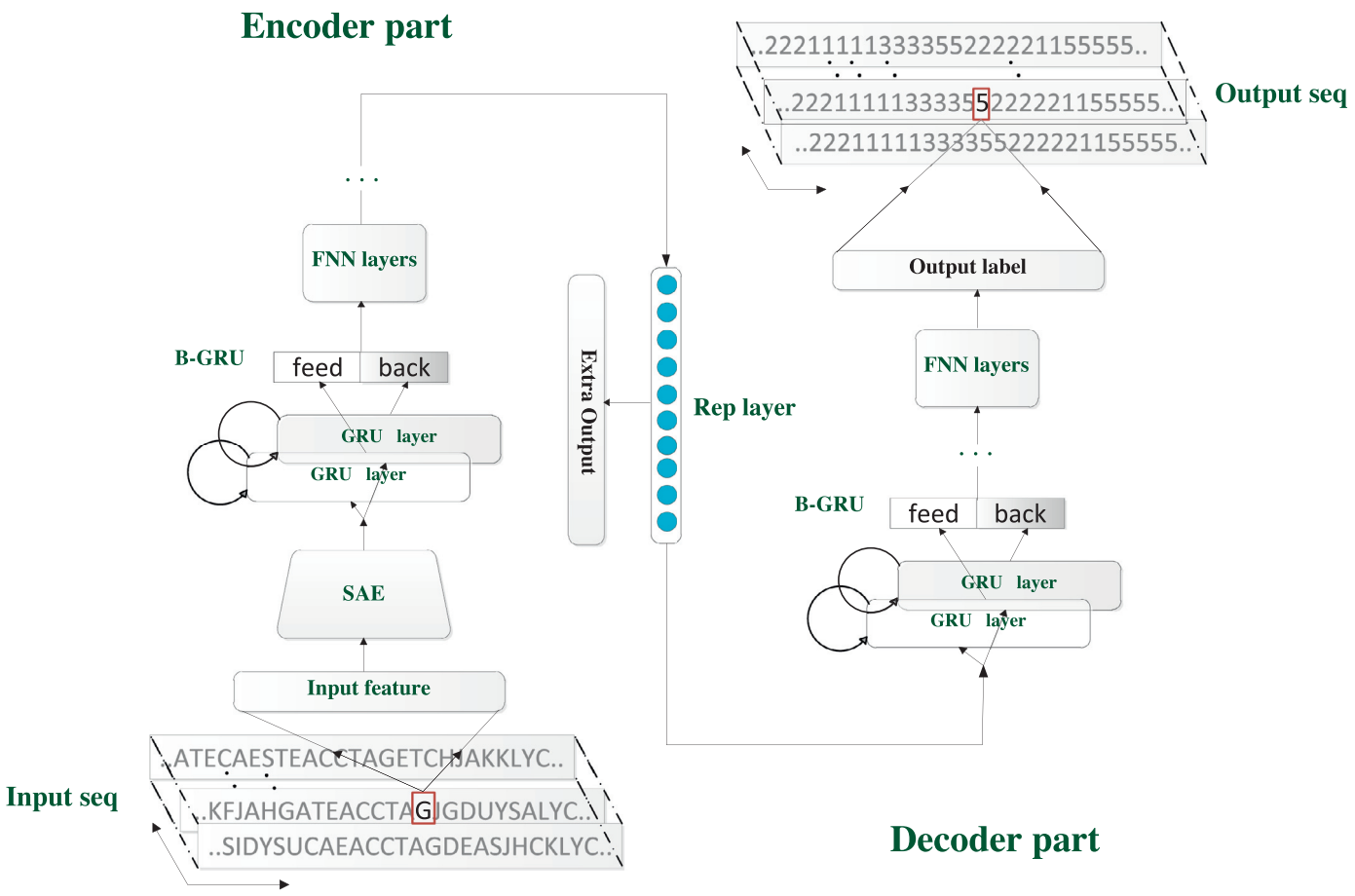


Fig. 4. The proposed secondary structure recurrent encoder–decoder networks.

3. The proposed model and algorithm

As the major part of this article, the proposed secondary structure recurrent encoder–decoder networks (SSREDNs) will be presented in this section. Its architecture and the algorithm for learning parameters from data will be discussed in detail. We use a bidirectional GRU structure here to deal with the spatial interaction information in protein chain. Adam algorithm is used to learn the network weights from data.

3.1. Secondary structure recurrent encoder–decoder networks (SSREDNs)

As shown in Fig. 4, secondary structure recurrent encoder–decoder networks (SSREDNs) is essentially a deep recurrent network with feed-forward layers and recurrent architectures. It is consisted of an encoder part, a decoder part and a representation layer. Both the encoder and decoder part contain feed-forward layers, recurrent layers and special training mechanisms. The encoder part learns a good representation for the input protein feature sequence that reflects both immediate and long-term amino acid dependencies, and the decoder part uses the representation for the final SS prediction which also takes consideration into the spatial dependency. GRUs are used in the encoder and decoder to learn the amino acid interaction information within the protein chain.

3.1.1. Encoder part

In the encoder part, a SAE is first used to pre-train the first few layers of the networks for better feature extraction. This SAE is train by a unsupervised layer-wise strategy. Then, weights from the trained SAE are used to initialize these layers. By using this pre-training strategy, the networks can get more efficient features automatically at the beginning of the training process. Through the encoder part, it allows the encoding of the input sequence into a good representation at the representation layer. Bidirectional GRU layers are used here to capture the context-dependent relationship (interaction among residues) here.

3.1.2. Decoder part

The decoder part decodes the learned representation of the representation layer into the structure space for SS prediction. Bidirectional GRU recurrent layers are also used to learn additional global dependence information for the final prediction. We use a softmax layer as the final layer to output the probability of the eight secondary structural states. Finally, the recurrent encoder–decoder network is trained by a supervised way using the back-propagation learning algorithm.

3.1.3. Training

As shown in Fig. 4, when training we use the amino acid features as input to the network—residue by residue along the protein chain—and the encoder generates a stable representation containing the contextual information of the input sequence. We normalize the activation level of the representation layer through an extra output layer. Next, the decoder predicts the Q8 state for the current input residue using the representation from the encoder. As we use recurrent structures to model the sequence-structure relationship, the window size problem of the convolutional method does not exist. At each time step, the network receives one residue

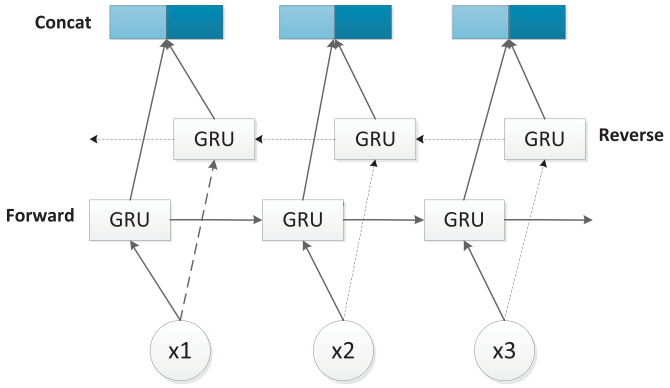


Fig. 5. Bidirectional GRU Layer.

from the protein chain as input and then outputs its secondary structural state.

3.2. Bidirectional GRU

As the protein sequence contains both forward and reverse order dependency information between adjacent positions on the protein chain, we used bidirectional recurrent layers [28] with GRUs in the networks. A bidirectional-GRU (B-GRU) layer is shown in Fig. 5. The B-GRU layer contains two reversed parallel GRU layers which are not connected to each other. Sequential information flows through the two layers in reversed order as time progresses.

Consider a sequence of protein data of length N with the networks' input denoted as $X = (x_1, x_2, \ldots, x_N)$. Each $x_i, i \in (1, 2, \ldots, N)$ is a feature vector that contains the feature information of the acids at position i. The learning algorithm for the B-GRU is shown in Algorithm 1:

The first GRU layer (forward GRU layer) will accept the input from x_1 to x_N at each time step. The second GRU layer (reversed GRU layer) will start at x_N and accept the input from x_N down to x_1; i.e., in the backward time direction. In this way, when processing a sequence input $X = (x_1, x_2, \ldots, x_N)$, the prorsad GRU layer will capture the information $a_t^{prorsad}$ at any time step $t, t \in (1, 2, \ldots, N)$, which contains the dependence relationship of the previous t positions, while the reverse GRU layer captures the reversed information $a_{N+1-t}^{reverse}$, which contains the dependence relationship of the t previous positions in reverse direction. Therefore, once the B-GRU layer is trained, the forward and backward GRU layers' outputs $a_t, t \in (1, 2, \ldots, N)$ may include the full dependence relationships for its input sequence at any position t. Both the forward and backward GRU layers' outputs will pass upward to the output layer of the B-GRU to form a better representation for the subsequent layers. Furthermore, in the back-propagation process, the forward and backward GRU layers are processed in decreasing and increasing time order, respectively, by using the BPTT algorithm [29].

Compared with a generic recurrent layer, the B-GRU can better deal with both the spatial dependencies in protein sequence and the gradient problems that normally occur during network training, which are usually the biggest challenge for training classical recurrent networks.

3.3. Training

The network was trained with Adam, the stochastic gradient-based optimization method proposed by Kingma [24]. It combine the advantages of two recently popular optimization methods: AdaGrad [30], the adaptive gradient algorithm, and RMSProp, which is similar but introduces an additional decay term. Adam

Algorithm 1 B-GRU learning Algorithm.

Require:
Input Data:
$X = \{x_1, x_2, \ldots, x_N\}$: Sequence input X with length N
Ensure:
Feed forward:
for $t = 1$ **to** N **do**
$a_t^{prorsad} = GRU(x_t)$
$a^{prorsad} = \{a_1^{prorsad}, a_2^{prorsad}, \ldots, a_N^{prorsad}\}$
end for
for $t = N$ **to** 1 **do**
$a_t^{reverse} = GRU(x_{N+1-t})$
$a^{reverse} = \{a_N^{reverse}, \ldots, a_2^{reverse}, a_1^{reverse}\}$
end for
for all $t \in (1, 2, .., N)$ **do**
$a_t = concat\left(a_t^{prorsad}, a_{N+1-t}^{reverse}\right)$
$a = \{a_1, a_2, \ldots, a_N\}$
end for
Back propogation:
for all $t \in (1, 2, .., N)$ **do**
Backward pass for output layer, storing the back error δ at each timestep
end for
for $t = N$ **to** 1 **do**
BPTT for reverse GRU layer, using the stored δ from output layer
end for
for $t = 1$ **to** N **do**
BPTT for prorsad GRU layer, using the stored δ from output layer
end for

computes individual adaptive learning rates for different parameters of the networks from estimates of first and second moments of the gradients. It has been shown that Adam performs equal to or better than some other optimization methods, regardless of hyper-parameter setting.

In SSREDNs, the learning of the representation layer is very important for the final performance, because it connects both the encoder and the decoder. As we known, the outputs of SS prediction must be a continuous structural states that adjacent residues form the uniform structure in space, eg. the SS state H may consist of several acids. So a better activation of the representation layer should be that different input features may have similar representation at the hidden representation layer if they have the same output label at the end during the training process. For this purpose, a additional output layer is added followed the representation layer with Mean Square Error cost function to constrain its activation level. It will force different input feature to obtain approximate activations at representation layer if they have the same label. So there are actually two output layers in the training procedure, the gradients flow from both output layers. One updates the whole networks but the other one just reflects the encoder. After the weights of the networks is trained we only use the final output layer (red part in Fig. 6). The cost functions of the two output layers are as follows:

$$L_1(x, \Theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(x_i^j - y_i^j\right)^2, \tag{11}$$

$$L_2(x, \Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (y_i^j \log(P(x_i^j)) + (1 - y_i^j)(1 - \log(P(x_i^j)))), \tag{12}$$

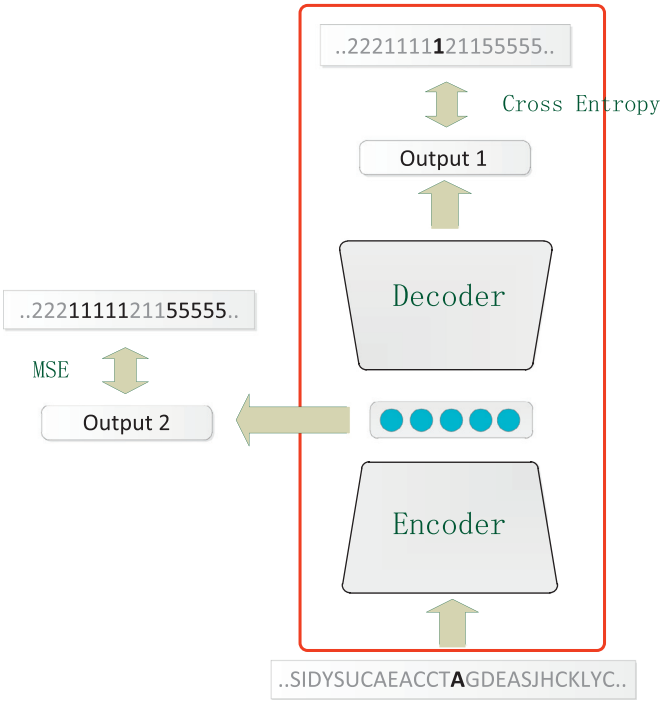


Fig. 6. Multi-output training. A additional output layer with Mean Square Error (MSE) cost is added followed the representation layer during the training process. It help to adjust the activity of the representation layer for the final prediction. It also help to speed up the training process to some extent.

where Θ is the parameter set, and x_i^j, y_i^j denote the j-th element of the i-th sample of the input set x and the target set y, respectively.

We use the Adam algorithm [24] to optimize the SSREDNs in training process. Moreover, the final cost function $L(x, \Theta)$ of SSREDNs is a sum of cost at the two output layers. Θ denotes the whole parameter set of the networks and we use γ to balance the contribution of the two output layers.

$$L(x, \Theta) = L_1(x, \Theta) + \gamma L_2(x, \Theta) \tag{13}$$

4. Experiments

In order to evaluate the performance of proposed SSREDNs, a series of experiments are performed on open datasets CullPDB [31] and CB513 [10]. Detailed data description is first provided in Section 4.1. In Section 4.2, the model training setup is presented. First, variations of network architectures of different layers were tested to refine the model's performance. Among them, we select the one with best performance for later comparison with other existing approaches. We also have evaluated the influence of the pretrain strategy using the SAE on the final prediction performance. The performance analysis is presented in Section 4.3. It shows that our method smoothly converges with fewer training epochs and improves almost all of the 8 statas's sensitivity and precision accuracy on the CullPDB testing set compared with [17] which may be the best 8-state predictor. Our method is also compared with some public methods (SSpro [32], RaptorX [10], PSIPRED [8]) on both CullPDB and CB513 dataset. It outperforms the others on both Q3 and Q8 accuracies.

4.1. Features and dataset

The major purpose of predict the SS types for each amino acid of a given protein sequence. We solve both the Q8 and Q3 prediction problems in this paper. We evaluated the proposed SSREDNs for both Q3 and Q8 on two datasets CullPDB [31] and CB513 [10], which are also used by other related investigations [10]. CullPDB, a large non-homologous dataset (identity less than 30%), contains 6128 protein amino acid sequences labeled with Q8 secondary structure, and has been randomly divided into training (5600), validation (248), and testing (280) sets. A separate evaluation is also performed on CB513 dataset, which contains 513 proteins, while training on CullPDB dataset further filtered to remove sequences with more than 25% identity with the CB513 dataset.

Protein sequence profiles with evolutionary information have become a breakthrough for SS prediction. Thus, Position Specific Scoring Matrix (PSSM) features have been used here, which are widely used features that can be extracted from protein profiles by Define Secondary Structure of Proteins (DSSP) and Position Specific Iterated Basic Local Alignment Search Tool (PSI-BLAST). In [17], the data used for training contained features and labels in 56 channels (22 for PSSM, 22 for amino acid sequence, 2 for terminals, 2 for solvent accessibility labels, 8 for secondary structure labels). The training data include 700 amino acids. It's considered to provides a good balance between efficiency and coverage as the majority of protein chains are shorter than 700 amino acids. When training and testing, shorter sequences (less than 700 amino acids) are padded with 0.

Thorough our experiments, only PSSM features (22) and amino acid sequence features (22) were used here for the 8-state predction. We also further removed the 'Noseq' channel as a convention [17]. So the input profiles in our experiment consist of 50 channels (21 for PSSM, 21 for the amino acid sequence, 8 for secondary structure labels).

4.2. Training setup

4.2.1. Networks setup

In the encoder and decoder framework, the ReLU layers with 50–300 units are used as feed-forward layers. GRU layers always contains 100–300 gated recurrent units. In the B-GRU layer, output from the bidirectional forward and backward layers are concatenated into a single vector to be used as input in the following layers.

Weights of the SAE were initialized in a greedy layer-wise manner, which map the layers' inputs back to themselves. Next, the other weights in each layer are sampled uniformly between -0.05 and 0.05 and biases are initialized at 0. The initial hidden states of the GRU layers $(\tilde{h}, h)$ are all set to 0 and updated during the training procedure. We use 50% dropout at the B-GRU and GRU layers to avoid the over-fitting problem.

A ReLU activation function [33] is used for all the feed-forward layers other than the output layer. When training, a softmax activation function is used on the final classification layer and a sigmoid activation function is used on the middle extra output layer, as shown in Fig. 6. The learning rate for the SAE stage is 0.05. When training the whole encoder–decoder network, the Adam algorithm, which only requires first-order gradients and little memory, is applied to control the parameter updates with the default settings $(\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 1e-8)$. Here, α is the learning rate, β_1, β_2 are parameters for exponential moving averages of the gradient and the squared gradient, respectively, ε is a small parameter used to avoid singularities associated with a zero denominator. We just set γ to 1 for an equivalent contribution between the two output layers in Eq. (13).

All parameters were trained globally by Adam algorithm with the final cost function in Eq. (13). The maximum number of training epochs is 100 and the batch size is 40 in our experiment. All the training procedures are implemented by Python based on Theano and Keras libraries. The training procedure was executed based on Nvidia Tesla K40 GPUs.

Table 1
Prediction accuracies is given for SSREDNs with different architectures.

Model	Q8 Accuracy (%)	Segment overlap score
3SAE,1BGRU,2FNN,1GRU	71.84	77.17
5SAE,1BGRU,2FNN,1GRU	70.51	76.70
3SAE,2BGRU,2FNN,2BGRU	72.51	77.95
3SAE,2BGRU,1GRU,2FNN,1BGRU	72.36	77.64
3SAE,2BGRU,2FNN,1BGRU	**73.14**	**78.20**

Table 2
Comparison of the performance with and without pre-train using SAE.

Model (3SAE,2BGRU,2FNN,1BGRU)	Q8 Accuracy	Segment overlap score
Without pre-train	72.13	77.12
Pre-train using SAE	**73.14**	**78.20**

4.2.2. Architecture analysis

We tried a set of various network architectures to find a suitable model for protein SS prediction problem. Their performances are shown in Table 1. The segment of overlap (SOV) score, which can be interpreted as SS segment-based accuracy, is also used here to evaluate the performance of different network architecture. Empirically, we found that a 3-layer SAE is a suitable choice. The B-GRU layers always get better results than the standard GRU layers during the training process. It verifies that the bidirectional structure can better capture the interaction relations of residues in the protein chain both in the encoder part and the decoder part. Besides, the recurrent layer will also affect the training results (row 4 vs 5 in Table 1). The optimal structure (in row 5) was found to be: {300-256-128 Stacked Auto-encoder } – {256 Bidirectional GRU layers(0.5 dropout) }*2 – {256-128 ReLU layers } – {64 ReLU Rep layer } – {128 Bidirectional GRU layer (0.5 dropout) } – {256-128 ReLU layers} – {8 sigmoid layer}. Furthermore, a {8 sigmoid layer} layer with MSE cost function was added on the {64 ReLU Rep layer }. It obtained 73.14% Q8 accuracy and 78.20% SOV scores on the CullPDB testing (272) set.

4.2.3. Stack auto-encoder

Feature learning is a very important process for SS prediction. The discovery of good features may benefit the prediction process and improve the prediction accuracy. A SAE architecture is used followed the input layer in our model to extract better features from the input protein features automatically. After pre-training, the SAE part learned robust representation from the input features. It will also be fine-tuned when training the whole networks. To confirm if this pre-train strategy using SAE is really helpful for improving the prediction accuracy, we trained a model on the CullPDB training set with and without the pre-train strategy and test it on the CullPDB testing set. The result is shown in Table 2. It improves the final Q8 accuracy from 72.13% to 73.14% and SOV score from 77.12% to 78.20%.

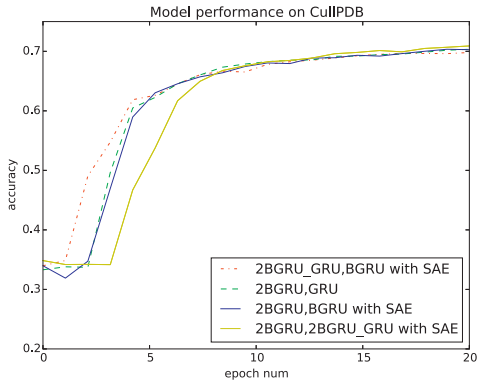
4.3. Performance

4.3.1. Performance on testing set

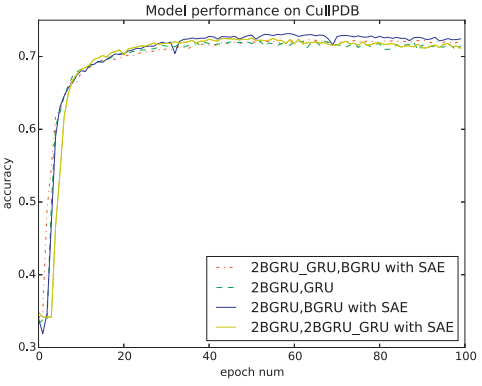
The prediction sensitivities, precisions and frequencies for individual secondary structure states of the CullPDB testing set with 272 sequences are shown in Table 3. Compared with [17], for the four major states, H, E, L and T, our SSREDNs method improves the prediction sensitivity and precision. The improvement for the prediction of state T, i.e., the $\beta - turn$ prediction which depends on the long-range inter-residue interactions, indicates that the SSREDNs model can learn the long-range structure features from the input protein chain sequence. Predictions for the states G, S and B are difficult because of their less frequencies, and the SSREDNs also makes better predictions for them compared with

Table 3
Performance of individual secondary structure state on CullPDB testing set.

Sec.	Sensitivity	Precision	Frequency	Description
H	**94.11**/93.5	**86.96**/82.8	35.8/35.4	α-helix
E	**84.52**/82.3	**78.07**/74.8	24.1/21.8	β -helix
L	**64.01**/63.3	**58.24**/54.1	21.3/18.6	loop
T	**53.97**/50.6	**57.96**/54.8	10.1/11.1	$\beta - turn$
S	**28.29**/15.9	**49.13**/42.3	5.0/7.9	bend
G	**35.99**/13.3	40.48/**49.6**	3.4/4.1	$3_{10} - helix$
B	**7.4**/0.1	45.63/**50**	0.2/1.1	$\beta - brige$
I	– /–	– /00-	0 / 0	$\pi - helix$



(a)



(b)

Fig. 7. Model performance during the training process on CullPDB dataset. (a) The curve converges to a stable state within 20 epoches on CullPDB training dataset. (b) The whole performance curve of SSREDNs within 100 training epochs.

[17] on prediction sensitivities. The lower prediction precisions for states G and B are accounted for by their relative rarity in the training set compared with our test set (3.1 vs 4.1, 0.2 vs 1.1). On the other hand, SSREDNs is essentially a data-driven model, and it can learn the complex interdependencies of residues from a mass of training sequences. The SSREDNs has some limitation for predicting the rarity state G, B and I ($3_{10} - helix$, $\beta - brige$ and $\pi - helix$) due to their few frequencies (only around 5% in total) in the train sequences. These states will also be a focus of future work.

4.3.2. Convergence

Fig. 7(a) shows the training accuracy for the CullPDB dataset. The SSREDNs show a powerful learning ability that always converges within 20 training epochs to a prediction accuracy rate of

Table 4
Q8 accuracy on CullPDB and CB513.

Model	Q8 Accuracy	
	CB513	CullPDB
GSN [17]	66.4	72.1
The optimal SSREDNs	**68.2**	**73.1**

Table 5
Q8 and Q3 accuracy on datasets CB513 and CullPDB.

Method	Q8		Q3	
	CB513	CullPDB	CB513	CullPDB
SSpro	63.5	66.6	78.5	79.5
RaptorX-SS8	64.9	69.7	78.3	81.2
PSIPRED	–	–	79.2	82.5
Ours	**68.2**	**73.1**	**82.9**	**84.2**

about 72%, equal to the best result in [17], which requires 300 epochs. In Fig. 7(b), it shows that SSREDNs will smoothly converges to its optimal result on validation set within about 70 training epochs, and this optimal model is used for the testing set in our experiments.

4.3.3. Comparasion with other methods

As shown in Table 4, we compared the Q8 accuracy of our method with the deep generative stochastic network (GSN) method in [17], which may be the best 8-state predictor. For this validation, we trained a model in which the CullPDB dataset was filtered to remove sequences having homology with CB513 sequences (more than 25% identity). We also just consider the PSSM features and amino acid sequence features in the testing. With the optimal model, we achieve a Q8 accuracy of 68.2% on CB513 and 73.1% on CullPDB, which outperforms the best result of GSN model.

With the same network architecture and parameters set, we also compare our method with the following public available programs: PSIPRED for 3-states SS prediction; SSpro, RaptorX for both 8-states and 3-states SS prediction on the datasets of CB513 and CullPDB. The SSpro package is used without template(i.e., not using a solved structure in PDB as template). All the programs are running with their parameters set according to their respective papers. As listed in Table 5, our method outperforms the others, including the popular PSIPRED on Q3 prediction, SSpro and RaptorX on both Q3 and Q8 prediction. In terms of both Q3 and Q8 accuracy on CullPDB and CB513, we obtains 84.2%, 82.9%, 73.1% and 66.4%, respectively.

5. Conclusion and future work

In this article, we proposed a deep recurrent encoder–decoder network and employed it to predict the secondary structure of residues in amino acid sequences. The combination of encoder–decoder architecture and GRUs is well suited to model both the sequence-structure relationship between input protein features and SS, and the mutual interactions among residues. It also achieves better performance on both Q3 and Q8 accuracy. For further development of the learning ability of SSREDNs, however, a method must be developed for determining the architecture and parameters of the networks, such as layer type, layer size, optimal method and initialization. This is also a challenge for deep learning and machine learning in general. To further improve the learning ability of the present model, multi-task prediction may be an avenue worth pursuing, such as the prediction of both the secondary structure and the solvent accessible surface area.

Acknowledgment

This work was supported by the National Science Foundation of China (Grant nos. 61432012 and 61402306).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.knosys.2016.11.015.

References

[1] X. Luo, Z. Ming, Z. You, S. Li, Y. Xia, H. Leung, Improving network topology-based protein interactome mapping via collaborative filtering, Knowl.-Based Syst. 90 (2015) 23–32.
[2] X. Lei, Y. Ding, H. Fujita, A. Zhang, Identification of dynamic protein complexes based on fruit fly optimization algorithm, Knowl.-Based Syst. 105 (2016) 270–277.
[3] J. Cheng, A.N. Tegge, P. Baldi, Machine learning methods for protein structure prediction, IEEE Rev. Biomed. Eng. 1 (2008) 41–49.
[4] J.K. Myers, T.G. Oas, Preorganized secondary structure as an important determinant of fast protein folding, Nat. Struct. Biol. 8 (6) (2001) 552–558.
[5] Z. Yang, I-Tasser server for protein 3d structure prediction, BMC Bioinform. 9 (3) (2008) 297–315.
[6] Z. Aydin, Y. Altunbasak, M. Borodovsky, Protein secondary structure prediction for a single-sequence using hidden semi-markov models, BMC Bioinform. 7 (1) (2006) 178.
[7] X.D. Sun, R.B. Huang, Prediction of protein structural classes using support vector machines, Amino Acids, 30 (4) (2006) 469–475.
[8] D.T. Jones, Protein secondary structure prediction based on position-specific scoring matrices, J. Mol. Biol. 292 (2) (1999) 195–202.
[9] G. Pollastri, D. Przybylski, B. Rost, P. Baldi, Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles, Proteins: Struct. Funct. Bioinform. 47 (2) (2002) 228–235.
[10] Z. Wang, F. Zhao, J. Peng, J. Xu, Protein 8-class secondary structure prediction using conditional neural fields, Proteomics, 11 (19) (2011) 3786–3792.
[11] S. Babaei, A. Geranmayeh, S.A. Seyyedsalehi, Protein secondary structure prediction using modular reciprocal bidirectional recurrent neural networks, Comput. Methods Programs Biomed. 100 (3) (2010) 237–247.
[12] R. Salakhutdinov, G. Hinton, Deep Boltzmann machines, J. Mach. Learn. Res. 5 (2) (2009) 1967–2006.
[13] Y. Bengio, G. Mesnil, Y. Dauphin, S. Rifai, Better mixing via deep representations, in: Proceedings of International Conference on Machine Learning (ICML), 2012, pp. 552–560.
[14] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science, 313 (5786) (2006) 504–507.
[15] M. Spencer, J. Eickholt, J. Cheng, A deep learning network approach to ab initio protein secondary structure prediction, IEEE/ACM Trans. Comput. Biol. Bioinform. 12 (1) (2015) 103–112.
[16] R. Heffernan, K. Paliwal, J. Lyons, A. Dehzangi, A. Sharma, J. Wang, A. Sattar, Y. Yang, Y. Zhou, Improving prediction of secondary structure, local backbone angles, and solvent accessible surface area of proteins by iterative deep learning, Sci. Rep. 5 (11476) (2015), doi:10.1038/srep11476.
[17] J. Zhou, O.G. Troyanskaya, Deep supervised and convolutional generative stochastic network for protein secondary structure prediction, in: Proceedings of International Conference on Machine Learning (ICML), 2014, pp. 745–753.
[18] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: Proceedings of International Conference on Machine Learning (ICML), 2013, pp. 1310–1318.
[19] K. Cho, B.V. Merrienboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: encoder-decoder approaches, arXiv preprint, arXiv:1409.1259 (2014).
[20] J. Chung, C. Gulcehre, K.H. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, arXiv preprint, arXiv:1412.3555 (2014).
[21] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (6) (2010) 3371–3408.
[22] C.D. Huang, S.F. Liang, C.T. Lin, R.C. Wu, Machine learning with automatic feature selection for multi-class protein fold classification, J. Inform. Eng. 21 (4) (2005) 711–720.
[23] G.E. Dahl, T.N. Sainath, G.E. Hinton, Improving deep neural networks for lvcsr using rectified linear units and dropout, in: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013, pp. 8609–8613.
[24] D. Kingma, J. Ba, Adam: a method for stochastic optimization, in: Proceedings of International Conference on Learning Representations (ICLR), 2015, pp. 254–269.
[25] M. Hermans, B. Schrauwen, Training and analysing deep recurrent neural networks, in: Proceedings of Advances in Neural Information Processing Systems (NIPS), 2013, pp. 190–198.
[26] Q. You, Y.J. Zhang, A new training principle for stacked denoising autoencoders, in: Proceedings of International Conference on Image and Graphics, 2013, pp. 384–389.

[27] R. Hecht-Nielsen, Theory of the backpropagation neural network, Neural Netw. 1 (1) (1988) 65–93.
[28] M. Schuster, K.K. Paliwal, Bidirectional recurrent neural networks, IEEE Trans. Signal Process. 45 (11) (1997) 2673–2681.
[29] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE, 78 (10) (1990) 1550–1560.
[30] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (7) (2011) 257–269.
[31] W. Guoli, R.L. Dunbrack Jr, Pisces: a protein sequence culling server, Bioinformatics, 19 (12) (2003) 1589–1591.
[32] C.N. Magnan, P. Baldi, Sspro/accpro 5: almost perfect prediction of protein secondary structure and relative solvent accessibility using profiles, machine learning and structural similarity, Bioinformatics, 30 (18) (2014) 2592–2597.
[33] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS), 2011, pp. 315–323.