Protein Fold Recognition with Representation Learning and Long Short-Term Memory

Masashi Tsubaki[1,a)] Masashi Shimbo[1,b)] Yuji Matsumoto[1,c)]

Abstract: Predicting the 3D structure of a protein from its amino acid sequence is an important challenge in bioinformatics. Since directly predicting the 3D structure is hard to achieve, classifying a protein into one of the "folds", which are pre-defined structural labels in protein databases such as SCOP and CATH, is generally used as an intermediate step to determine the 3D structure. This classification task is called protein fold recognition (PFR), and much research has addressed the problem of either (i) feature extractions from amino acid sequences or (ii) classification methods of the protein folds. In this paper, we propose a new approach for PFR with (i) learning feature representations with unsupervised methods from a large protein database instead of manual feature selection and using external tools. (ii) learning deep neural architectures, recurrent neural networks (RNNs) with long short-term memory (LSTM) units, and re-training the representations instead of fixing the extracted features. On a benchmark dataset, our approach outperforms existing methods that use various physicochemical features.

Keywords: Protein fold recognition, Deep learning, Representation learning, Recurrent neural network, Long short-term memory

1. Introduction

Predicting the 3D structure of a protein from its amino acid sequence is an important challenge in bioinformatics. However, directly predicting the 3D protein structure is a difficult task. Therefore, classifying a protein into one of the "folds", which are pre-defined structural labels in protein databases such as SCOP and CATH, is used as an intermediate step to determine the 3D structure. This classification task is called *protein fold recognition* (PFR), and the process is divided into two steps: (i) extracting features from amino acid sequences and (ii) classifying the protein folds. The features are generally extracted by using syntactical and physicochemical information of proteins such as hydrophobicity, polarity, and van der Waals volume [1], [2]. With the extracted features, several classifiers have been applied such as support vector machines and ensemble classifiers [3], [4].

In recent natural language processing (NLP), word representation learning methods have become popular [5], [6]. With these methods, vector representations of words can be obtained from a large corpus in an unsupervised fashion. These word representations have many applications, and notably, various deep neural network architectures utilizes them for learning structures of sentences [7]. Such architectures further allow for re-training the word representations simultaneously. Thus, this approach does not depend on feature extractions and consists of two steps: (i) unsupervised learning of representations as pre-training, and (ii) supervised learning of deep neural architectures and re-training of the pre-trained representations.

Inspired by the above NLP approach, this paper proposes a new approach for PFR. Our PFR process is divided into two steps: (i) We obtain features of amino acid sequences with unsupervised representation learning [5], [6] instead of extracting features such as physicochemical-based information. (ii) We design a classifier for protein folds with recurrent neural networks (RNNs) using long short-term memory (LSTM) units [8], train the LSTM, and re-train the features instead of fixing the physicochemical features. Fig 1 shows the overview of our approach.

In more details, our approach is as follows: (i) We first define n-gram amino acids as words of proteins and make a large protein corpus with CATH. To this corpus, we apply three word representation learning methods: skip-gram (SG) and continuous bag-of-words (CBOW) of word2vec [5] and global vectors (GloVe) [6]. Then we obtain the word vectors as pre-trained representations (§see 3.1). (ii) As an input is the word vector sequence of a protein, our LSTM reads the vectors one by one bi-directionally (i.e., from left to right and from right to left) [9], obtain the hidden layers, and then sum the hidden layers as an input for softmax classifier (§3.2). (iii) In addition to the learning of an LSTM classifier for protein folds, we re-train the pre-trained n-gram amino acid representations simultaneously (§3.3).

Our contribution is three-fold: (i) To the best of our knowledge, ours is the first work that combines word representations and LSTMs for PFR. (ii) On a benchmark dataset, our approach outperformed existing methods that use various physicochemical features. This suggests that the capturing of co-occurrence information with word representations and long-term dependen-

[1] Graduate School of Information Science Nara Institute of Science and Technology 8916-5, Takayama, Ikoma, Nara 630-0192, Japan
[a)] masashi-t@is.naist.jp
[b)] shimbo@is.naist.jp
[c)] matsu@is.naist.jp

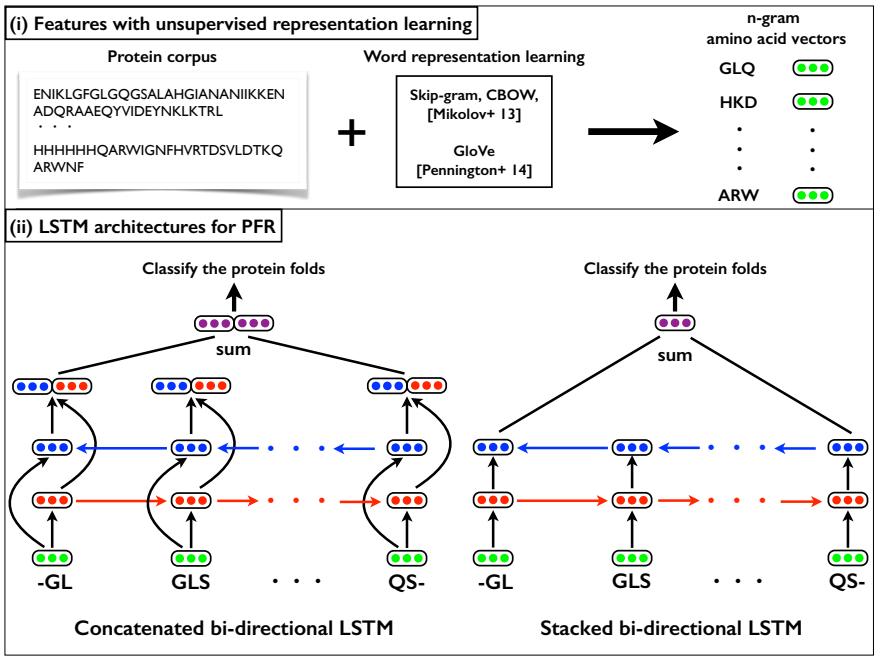

Fig. 1 Our protein fold recognition (PFR) process is divided into two steps: (i) We obtain features of n-gram amino acids with unsupervised representation learning. (ii) We design a classifier for protein folds with RNNs using LSTM units. In this paper, we propose two architectures: concatenated and stacked bi-directional LSTMs (§3.2).

cies with LSTMs is also effective for PFR. (iii) We analyzed the differences of two deep neural architectures, concatenated bi-directional LSTM [9] and stacked bi-directional LSTM [10] (see Fig 1 and §3.2 for detail), in terms of the representational powers and re-training feature representations.

2. Background

In this section, we provide a brief review of word representation learning methods (§2.1) and RNNs with LSTM units (§2.2).

2.1 Word Representing Learning Methods

2.1.1 Skip-gram and continuous bag-of-words of word2vec

The training objective of the skip-gram (SG) model of word2vec [5] is to learn d-dimensional word representations that are useful for predicting surrounding words of a target word in a sentence. Specifically, given a sequence of training words $w_1, w_2, \cdots, w_L$, the objective function J to maximize is the average log probability as follows:

$$J = \frac{1}{L}\sum_{t=1}^{L}\sum_{-\ell\le i\le\ell}\log p(c_{t+i} \mid w_t), \tag{1}$$

where w_t is the target word, ℓ is the window size of the context which is the number of words to consider around of w_t, and c_i is the i-th context word in the window. In the SG model, the probability in equation (1) is defined as follows:

$$p(c_{t+i}|w_t) = \frac{\exp({\mathbf{c}_{t+i}}^\top \mathbf{w}_t)}{\sum_{j=1}^{V}\exp({\mathbf{c}_j}^\top \mathbf{w}_t)}, \tag{2}$$

where V is the number of words in the vocabulary, $\mathbf{w} \in \mathbb{R}^d$ is the d-dimensional vector representation of word w, and $\mathbf{c} \in \mathbb{R}^d$ is the d-dimensional vector representation of context word c. Note that $\mathbf{w}_i \in \mathbb{R}^d$ is the i-th row vector of $\mathbf{W} \in \mathbb{R}^{V\times d}$ and $\mathbf{c}_i \in \mathbb{R}^d$ is the i-th column vector of $\mathbf{C} \in \mathbb{R}^{d\times C}$, where C is the number of context words.

However, the above function is impractical because its computational cost is proportional to V. To reduce the cost, the word2vec toolkit implements negative sampling. Negative sampling approximates the probability in equation (2) by maximizing the inner product of vectors for correct example pairs (w, c) that occur in the corpus, and minimizing it for negative example pairs (w, c') that do not occur in the corpus as follows:

$$p(c_{t+i}|w_t) \approx \sigma(\mathbf{c}_{t+i}^\top \mathbf{w}_t)\prod_{k=1}^{K}\sigma(-{\mathbf{c}'}_k^\top \mathbf{w}_t),$$

where σ is the sigmoid function, $\mathbf{c}'$ is the word vector of negative example, and K is the number of negative sampling words. In contrast to SG, the continuous bag-of-words (CBOW) model, which is another implementation in word2vec, predicts the center word based on the sum of the surrounding context vectors.

2.1.2 Global vectors

Global vectors (GloVe) [6] relies on a global log-bilinear regression model that combines and leverages the advantages of two ideas, (i) a local context window such as used in word2vec and (ii) a global co-occurrence statistics in a corpus. Let w_i be the i-th word in the vocabulary, $\mathbf{w}_i \in \mathbb{R}^d$ be the d-dimensional vector for the word w_i, c_j be the j-th context, and $\mathbf{c}_j \in \mathbb{R}^d$ be the d-dimensional vector of context c_j. The objective function J to minimize is defined as follows:

$$J = \sum_{i,j=1}^{V} f(\#(w_i, c_j))(\mathbf{w}_i^\top \mathbf{c}_j + b_i + b_j - \log(\#(w_i, c_j))^2, \tag{3}$$

where V is the number of words in the vocabulary, $\#(w, c)$ is the number of times that word w occurs in context c, b_i and b_j are word/context-specific bias terms that are also learning parameters in addition to $\mathbf{w}$ and $\mathbf{c}$, and $f(x)$ is a weighting function. In

[6], the authors use the weighting function

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & x < x_{\max} \\ 1 & \text{otherwise,} \end{cases}$$

where $x_{\max} = 100$ in all experiments and report that $\alpha = 3/4$ gives a modest improvement over $\alpha = 1$. From Equation (3), we see that GloVe is fit to minimize the weighted least square loss giving more weight to frequent (w, c) with this weighting function $f(x)$.

2.2 Long Short-Term Memory

In recent NLP, RNNs with LSTM units [8] have re-emerged as a popular architecture for various tasks. In bioinformatics, LSTMs have also been applied to problems such as secondary structure prediction [11], prediction of the number of residue contacts [12], and homology detection [13].

RNNs with LSTM units use memory cells that are in a hidden layer and can store information for a long period of time. Specifically, the memory cells consist of three types of gates that control a flow of information into and out of these cells: an input gate $\mathbf{i}_t \in \mathbb{R}^d$, a forget gate $\mathbf{f}_t \in \mathbb{R}^d$, and an output gate $\mathbf{o}_t \in \mathbb{R}^d$. Given an input vector $\mathbf{x}_t \in \mathbb{R}^d$, the previous hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^d$ and previous cell state $\mathbf{c}_{t-1} \in \mathbb{R}^d$, an LSTM computes the next $\mathbf{c}_t$ and $\mathbf{h}_t$ as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{bmatrix}, \tag{4}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{H} + \mathbf{b}_i), \tag{5}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{H} + \mathbf{b}_f), \tag{6}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{H} + \mathbf{b}_o), \tag{7}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_c\mathbf{H} + \mathbf{b}_c), \tag{8}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{9}$$

where $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o, \mathbf{W}_c \in \mathbb{R}^{2d\times d}$ are weight matrices to learn, $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o, \mathbf{b}_c \in \mathbb{R}^d$ are bias vectors to learn, σ is the element-wise sigmoid function, tanh is the element-wise hyperbolic tangent function, and $\odot$ is the element-wise multiplication of two vectors. In this paper, we represent a series of computation in Equations (4)–(9) by

$$\langle \mathbf{c}_t, \mathbf{h}_t \rangle = lstm(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}). \tag{10}$$

Finally, output $\mathbf{y}_t \in \mathbb{R}^k$ is produced at each time step t from $\mathbf{h}_t$ by

$$\mathbf{y}_t = \mathbf{W}_y\mathbf{h}_t + \mathbf{b}_y,$$

where$\mathbf{W}_y \in \mathbb{R}^{d\times k}$ is the weight matrix and $\mathbf{b}_y \in \mathbb{R}^k$ is the bias vector, with k being the output dimensionality.

3. Method

In this section, we first describe the definition of words in proteins, creation of a large protein corpus for representation learning, and application of word2vec and GloVe to this corpus (§3.1). We then present the bidirectional LSTM architecture (§3.2), followed by the explanation of the training and re-training procedure of our LSTMs and pre-trained word representations (§3.3).

3.1 Words of Proteins and Representation Learning

To apply word representation learning to proteins, we define a word in amino acid sequences as an n-gram of amino acid, and we split the sequences into overlapping of the n-gram amino acids. Since there are 20 types of amino acids that make up proteins, the total number of possible n-grams is 20^n. In this paper, to keep the vocabulary size tractable and to avoid low-frequcency words in learning representations, we set a moderate small n-gram length of $n = 3$. For example, we can split a sequence of protein into an overlapping 3-gram amino acid sequence as follows: $GLSAA\cdots LQS \rightarrow$ *"-GL"*, *"GLS"*, *"LSA"*, $\cdots$, *"LQS"*, *"QS-"*. With this word definition and splitting, we create a large protein corpus from CATH that is a large protein database. Then, to this corpus, we apply SG, CBOW, and GloVe described in §2.1.

Why word representations? In NLP, word representations can capture the meaning and its similarity using the co-occurrence information of words in a corpus. Similarly, it has been pointed out that the co-occurrence information of amino acids in proteins is also important for capturing protein structures [3], [4], [14]. In addition, in most existing methods for PFR, feature vectors of amino acids typically include physicochemical information such as hydrophobicity, polarity, and van der Waals volume. While these features are widely used, the feature vectors themselves are fixed during learning process [1], [2], [15]. In contrast, our feature vectors of n-gram amino acids are re-trained with deep neural architectures described in the following subsection.

3.2 Protein Fold Recognition with LSTMs

Given a protein $P = w_1, w_2, \cdots, w_L$ represented as a word sequence (i.e., an overlapping split sequence with n-gram amino acids), where w_i is the i-th word and L is the length of the word sequence, we first translate all words in the sequence to pre-trained word vectors. Let the obtained word vector sequence be

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_{L-1}, \mathbf{x}_L,$$

where $\mathbf{x}_i \in \mathbb{R}^d$ is the d-dimensional vector of i-th word. Alternatively, we can also consider a vector sequence whose elements consist of concatenated word vectors. For example, the vector sequence composed of concatenation of three contiguous word vectors is as follows:

$$[\mathbf{x}_1; \mathbf{x}_2; \mathbf{x}_3], [\mathbf{x}_2; \mathbf{x}_3; \mathbf{x}_4], \cdots, [\mathbf{x}_{L-2}; \mathbf{x}_{L-1}; \mathbf{x}_L],$$

where $[\mathbf{x}_i; \mathbf{x}_j; \mathbf{x}_k]$ is the concatenation of $\mathbf{x}_i$, $\mathbf{x}_j$, and $\mathbf{x}_k$. This sequence allows us to additionally take into account surrounding features when processing the target word. In our approach, such vector sequence is the input sequence for LSTMs.

Why LSTMs? It has been pointed out that long-term dependencies of amino acids in the sequence is crucial in protein structure prediction [13]. Indeed, LSTMs can find long-term dependencies in the sequence and remember them for a long period of time [8]. However, since vanilla LSTMs use only the past subsequence and ignore the future subsequence, we need an architecture that considers both past and future sequence in proteins. We therefore use bi-directional architectures to overcome this limitation. The original bi-directional architecture proposed by [9] computes two hidden layers, one that processes the input

sequence in a forward direction, and the other that processes the sequence backwards, and concatenates their outputs (Fig 1). In addition the **concatenated** architecture, we attempt to improve the representational power of bi-directional LSTM with **stacked** architectures that assume a hidden layer as an input of the next layer [10]. In later experiments, we compare the performance of the two architectures in PFR.

These two architectures can be formulated as follows. Given a word vector sequence $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_L$ as an input, we apply two layers of the LSTM function given by Equation (10) to obtain a hidden layer sequence $\mathbf{h}_1, \mathbf{h}_2, \cdots \mathbf{h}_L$, where $\mathbf{h}_i \in \mathbb{R}^d$ is the d-dimensional hidden layer of i-th word. Let $lstm_f$ be a forward LSTM function and $lstm_b$ be a backward LSTM function that are given by Equation (10). We distinguish these two functions by subscripts, $forward$ and $backward$, because they do not share parameters ($\mathbf{W}_i$, $\mathbf{W}_o$, etc.). Then, the forward and backward computation of concatenated bi-directional LSTM can be stated as follows:

$$\langle \overrightarrow{\mathbf{c}_i}, \overrightarrow{\mathbf{h}_i} \rangle = lstm_{forward}(\mathbf{x}_i, \overrightarrow{\mathbf{h}_{i-1}}, \overrightarrow{\mathbf{c}_{i-1}}),$$
$$\langle \overleftarrow{\mathbf{c}_i}, \overleftarrow{\mathbf{h}_i} \rangle = lstm_{backward}(\mathbf{x}_i, \overleftarrow{\mathbf{h}_{i+1}}, \overleftarrow{\mathbf{c}_{i+1}}),$$
$$\mathbf{h}_i = [\overrightarrow{\mathbf{h}_i}; \overleftarrow{\mathbf{h}_i}],$$

where $\mathbf{c}_i$ is the i-th cell layer and $[\overrightarrow{\mathbf{h}_i}; \overleftarrow{\mathbf{h}_i}]$ is the concatenation of $\overrightarrow{\mathbf{h}_i}$ and $\overleftarrow{\mathbf{h}_i}$. On the other hand, we represent the stacked bi-directional LSTM, which consider a hidden layer as an input of the next layer, as follows:

$$\langle \overrightarrow{\mathbf{c}_i}, \overrightarrow{\mathbf{h}_i} \rangle = lstm_{forward}(\mathbf{x}_i, \overrightarrow{\mathbf{h}_{i-1}}, \overrightarrow{\mathbf{c}_{i-1}}),$$
$$\langle \mathbf{c}_i, \mathbf{h}_i \rangle = lstm_{backward}(\overrightarrow{\mathbf{h}_i}, \overleftarrow{\mathbf{h}_{i+1}}, \overleftarrow{\mathbf{c}_{i+1}}).$$

With the obtained hidden layer sequence $\mathbf{h}_1, \mathbf{h}_2, \cdots \mathbf{h}_L$, we compute the output layer $\mathbf{y} \in \mathbb{R}^k$ using the summation of all hidden layers as follows:

$$\mathbf{y} = \mathbf{W}_y \sum_{i=1}^{L} \mathbf{h}_i + \mathbf{b}_y,$$

where $\mathbf{W}_y \in \mathbb{R}^{2d \times k}$ (when concatenated bi-directional LSTM) or $\mathbf{W}_y \in \mathbb{R}^{d \times k}$ (when stacked bi-directional LSTM) is the weight matrix to learn, $\mathbf{b}_y \in \mathbb{R}^k$ is the bias vector to learn, and k is the dimensionality of output layer (i.e., the number of protein fold labels to classify). Finally, a softmax layer is added on the top of the output layer $\mathbf{y}$ for modeling multi-class probabilities as follows:

$$p_t = \text{softmax}(\mathbf{y}, t) = \frac{\exp(\mathbf{y}_t)}{\sum_{i=1}^{k} \exp(\mathbf{y}_i)},$$

where t is the index of correct class label and $\mathbf{y}_i$ is the i-th element of output layer $\mathbf{y}$.

3.3 Training Procedure

When a training dataset $\{(P_i, t_i)\}_{i=1}^N$ is given, where P is the amino acid sequence of protein, t is the fold (i.e., label) of protein, and N is the number of training samples, the objective is to minimize the cross-entropy loss plus a L2-regularization term as follows:

$$\mathcal{L}(\Theta) = -\sum_{i=1}^{N} \log p_{t_i} + \frac{\lambda}{2} ||\Theta||_2^2,$$

where p_{t_i} is the probability for the correct class label t of the i-th protein in the training dataset. Standard backpropagation techniques are used to train the set of all parameters Θ. These parameters include the weight matrices and bias vectors of LSTM (§2.2) and pre-trained word representations.

4. Related Work

Many feature extraction techniques have been proposed, and the features are generally extracted by using physicochemical and syntactical information of amino acids. Dubchak et al., 1997 [16] suggested the combination of syntactical and physicochemical features. These features consist of amino acid composition (AAC) as syntactical information and physicochemical information which is extracted from following five attributes: hydrophobicity (H), predicted secondary structure based on normalized frequency of α-helix (X), polarity (P), polarizability (Z), and van der Waals volume (V). The set of these five attributes is represented as "HXPZV" and widely used in PFR as a basic physicochemical feature set [1], [2], [15]. On the other hand, Taguchi and Gromiha, 2007 [14] proposed features that are based on the amino acid occurrence, and argued that only syntactical features should be considered because physicochemical features have no important information for predicting protein structures. In addition, Ghanty and Pal, 2009 [3] introduced features that are based on the pairwise frequency of amino acids separated by one residue (called PF1) and pairwise frequency of adjacent residues (called PF2). Furthermore, Yang et al., 2011 [4] used features that are concatenation of PF1 and PF2 (called PF). While these syntactical features have also been shown useful in PFR, further research has been required in order to explore the potential of physicochemical features. In addition to HXPZV, other various physicochemical attributes have been proposed [2], [15]. With the above extracted features, several classifiers have been developed and employed such as linear discriminant analysis (LDA) [17], neural networks (NNs) [18], and support vector machines (SVMs) [1]. SVMs and ensemble strategy with SVMs exhibit promising results [3], [4].

5. Experiments

5.1 Training Corpus and Evaluation Dataset

Corpus: In order to learn word representations of proteins, we use the CATH database. This database contains 86670 proteins, and we split the all proteins into overlapping 3-gram amino acid sequences.

Dataset: In order to evaluate our method, we use the DD-dataset of Ding and Dubachak, 2001 [1], which contains a training set of 311 proteins, a test set of 384 proteins, and any two proteins have 35–40 % of sequence identity for aligned subsequence longer than 80 residues. This dataset has 27 protein fold labels and the training and test sets are merged to perform 10-fold cross-validation.

5.2 Training Details of Word Representations and LSTMs

Word representations: We set vector dimensionality $d = 100$, window size $\ell = 10$, the number of iterations over a corpus is 10 in CBOW and 100 in SG and GloVe, the number of negative sampling words $K = 5$ (in SG and CBOW), and other hyper-

Existing methods (feature & classifier)	Accuracy (%)
Ding and Dubachak, 2001 [1] (AAC & SVM)	45.1
Ding and Dubachak, 2001 [1] (AAC + HXPZV & SVM)	47.2
Taguchi and Gromiha, 2007 [14] (co-occurrence & SVM)	51.0
Ghanty and Pal, 2009 [3] (PF1 & SVM)	50.6
Ghanty and Pal, 2009 (PF2 & SVM)	48.2
Yang et al., 2011 [4] (PF & SVM)	53.4
Sharma et al., 2013 [2] (HXPZV & SVM)	29.5
Sharma et al., 2013 [2] (HXPZV & NB)	32.8
Sharma et al., 2013 [2] (15 physicochemical attributes & SVM)	40.2
Sharma et al., 2013 [2] (15 physicochemical attributes & NB)	45.3
Sharma et al., 2013 [2] (30 physicochemical attributes & SVM)	43.6
Sharma et al., 2013 [2] (30 physicochemical attributes & NB)	50.9
Our methods	**Accuracy (%)**
Concatenated bi-directional LSTM (GloVe, 7 word concat, not re-training of word rep)	**53.1**
Stacked bi-directional LSTM (GloVe, 7 word concat, re-training of word rep)	**56.9**

Table 1 Accuracies of existing methods and our LSTM architectures.

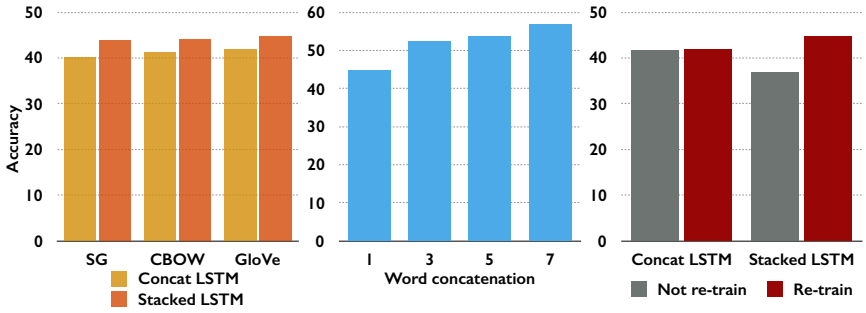

Fig. 2 Effects of model components: initialization of word representations, the number of word concatenations, and re-training of word representations. If not otherwise specified, we show the result of model using GloVe, stacked bi-directional LSTM, one word concatenation (i.e., an input is a word vector), and re-training of word representations.

parameters are set to default values in the toolkits of word2vec[*1] and GloVe[*2]. These word representations are used to initialize our LSTMs but not fixed, they are later re-trained during learning process of LSTMs. We discuss the impact of pre-training and re-training of word representations.

LSTMs: The complete training details of our LSTMs are given as follows:

- Word concatenation (i.e., input words): 1, 3, 5, and 7.
- Optimization method: ADAM [19] with a first momentum coefficient of 0.9 and a second momentum coefficient of 0.999 which are recommended configuration in [19].
- Batch size: 50 sequences for the gradient.
- Dropout ratio: [0.0, 0.1, 0.3, 0.5, 0.7].
- L2-regularization strength: [0.0, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6].

Subsequently, we take the best configuration on the validation set and evaluate only that configuration on the test set. We implemented our LSTMs using Chainer[*3].

5.3 Experimental Results and Discussion

5.3.1 Main Results

Table 1 shows the accuracies of existing methods and our LSTM architectures. The best result is obtained with stacked bi-directional LSTM using 100-dimensional GloVe, 7 word concatenations (i.e., 700-dimensional vectors), and the strategy of re-training of word representations. The accuracy is 56.9 % for the classification task of 27 labels. While the concatenated bi-directional LSTM (the accuracy is 53.1 %) does not outperform the existing methods, it is the first result of representation learning and deep neural architectures achieved the competitive results on the benchmark dataset of PFR, in which no two proteins have sequence identity higher than 35–40%.

Taguchi and Gromiha [14] have argued that features from physicochemical attributes can be ignored due to having insufficient information and only syntactical features should be considered. Our experimental results support the argument. The reason is that our all word representations, SG, CBOW, and GloVe also have the co-occurrence and frequency information of amino acids in proteins.

5.3.2 Effects of PFR Components

Initialization of word representations: In Fig 2, we found that the PFR performance does not depend on initialization of word representations such as SG, CBOW, and GloVe. The reason is that these representations are similar in terms of having the co-occurrence information of amino acids in proteins.

Word concatenation: In contrast, the number of word concatenations is more important factor for improving the performance of PFR. This suggests that surrounding features of the target word are crucial in PFR rather than feature representation of the target word.

Re-training of word representations: We also found that

[*1] https://code.google.com/archive/p/word2vec/
[*2] http://nlp.stanford.edu/projects/glove/
[*3] http://chainer.org/

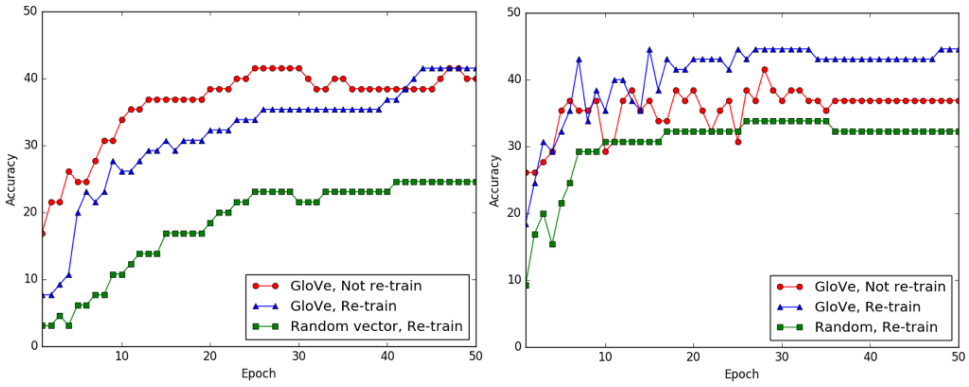

Fig. 3 Learning curve of concatenated bi-directional LSTM (left) and stacked bi-directional LSTM (right).

re-training of word representations does not always give an improvement. While stacked bi-directional LSTM shows the improved accuracy with the help of re-training representations, any substantial improvement was not observed when concatenated bi-directional LSTM is used. In the following subsection, we provide a model analysis of these two LSTM architectures.

5.3.3 Model Analysis

Fig 3 shows the learning curve of concatenated bi-directional LSTM and stacked bi-directional LSTM

Re-training of word representations in two architectures: Fig 3 shows the learning curve. As seen in this Fig, while the learning curve of concatenated bi-directional LSTM is stable, any substantial improvement was not observed with the help of re-training word representations. On the other hand, while stacked bi-directional LSTM can correctly re-train word representations, the learning curve tends to be unstable and overfit the data. This suggests that, in contrast to concatenated architectures that the hidden layer has double (i.e., $2d$) dimensionality, stacked architectures that only assume a hidden layer as the input of the next layer can improve the representational power of bi-directional LSTM.

Random initialization of word representations: Interestingly, Fig 3 shows that even if we use random initialized vectors for word representations, we achieved about 30 % accuracy when the representations are retrained with the stacked bi-directional LSTM. This result shows that stacked architecture can achieve comparable accuracy to existing methods that use basic five physicochemical features without the help of pre-trained representations. This suggests that obtained features, with learning of long-term dependencies of amino acids in the sequence of protein, can cover the information of basic physicochemical attributes.

6. Conclusion

In this paper, we proposed a new approach for PFR and outperformed existing methods on a benchmark dataset. To the best of our knowledge, ours is the first work that combines word representations and LSTMs for PFR.

References

[1] Ding, C. H. and Dubchak, I.: Multi-class protein fold recognition using support vector machines and neural networks, *Bioinformatics*, Vol. 17, No. 4, pp. 349–358 (2001).
[2] Sharma, A., Paliwal, K. K., Dehzangi, A., Lyons, J., Imoto, S. and Miyano, S.: A strategy to select suitable physicochemical attributes of amino acids for protein fold recognition, *BMC bioinformatics*, Vol. 14, No. 1, p. 1 (2013).
[3] Ghanty, P. and Pal, N. R.: Prediction of protein folds: extraction of new features, dimensionality reduction, and fusion of heterogeneous classifiers, *NanoBioscience, IEEE Transactions on*, Vol. 8, No. 1, pp. 100–110 (2009).
[4] Yang, T., Kecman, V., Cao, L., Zhang, C. and Huang, J. Z.: Margin-based ensemble classifier for protein fold recognition, *Expert Systems with Applications*, Vol. 38, No. 10, pp. 12348–12355 (2011).
[5] Mikolov, T., Chen, K., Corrado, G. and Dean, J.: Efficient estimation of word representations in vector space, *arXiv preprint arXiv:1301.3781* (2013).
[6] Pennington, J., Socher, R. and Manning, C. D.: Glove: Global vectors for word representation, *Empirical Methods on Natural Language Processing (EMNLP)* (2014).
[7] Socher, R., Huval, B., Manning, C. D. and Ng, A. Y.: Semantic Compositionality through Recursive Matrix-Vector Spaces, *Empirical Methods on Natural Language Processing (EMNLP)* (2012).
[8] Hochreiter, S. and Schmidhuber, J.: Long short-term memory, *Neural computation*, Vol. 9, No. 8, pp. 1735–1780 (1997).
[9] Graves, A., Jaitly, N. and Mohamed, A.-R.: Hybrid speech recognition with deep bidirectional LSTM, *Automatic Speech Recognition and Understanding (ASRU)* (2013).
[10] Pascanu, R., Gulcehre, C., Cho, K. and Bengio, Y.: How to construct deep recurrent neural networks, *arXiv preprint arXiv:1312.6026* (2013).
[11] Sønderby, S. K. and Winther, O.: Protein secondary structure prediction with long short term memory networks, *arXiv preprint arXiv:1412.7828* (2014).
[12] Jacobsson, A. and Gustavsson, C.: Prediction of the number of residue contacts in proteins using LSTM neural networks, *Halmstad University*, p. 9 (2003).
[13] Hochreiter, S., Heusel, M. and Obermayer, K.: Fast model-based protein homology detection without alignment, *Bioinformatics*, Vol. 23, No. 14, pp. 1728–1736 (2007).
[14] Taguchi, Y. and Gromiha, M. M.: Application of amino acid occurrence for discriminating different folding types of globular proteins, *BMC bioinformatics*, Vol. 8, No. 1, p. 1 (2007).
[15] Dehzangi, A., Sharma, A., Lyons, J., Paliwal, K. K. and Sattar, A.: A mixture of physicochemical and evolutionary–based feature extraction approaches for protein fold recognition, *International journal of data mining and bioinformatics*, Vol. 11, No. 1, pp. 115–138 (2014).
[16] Dubchak, I., Muchnik, I. B. and Kim, S.-H.: Protein folding class predictor for SCOP: approach based on global descriptors., *Ismb* (1997).
[17] Klein, P.: Prediction of protein structural class by discriminant analysis, *Biochimica et Biophysica Acta (BBA)-Protein Structure and Molecular Enzymology*, Vol. 874, No. 2, pp. 205–215 (1986).
[18] Ying, Y., Huang, K. and Campbell, C.: Enhanced protein fold recognition through a novel data integration approach, *BMC bioinformatics*, Vol. 10, No. 1, p. 1 (2009).
[19] Kingma, D. and Ba, J.: Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980* (2014).