Title
Protein sequence labelling by AUC-maximized Deep Convolutional Neural Fields

Author
Sheng Wang, Jianzhu Ma, Siqi Sun, and Jinbo Xu

Affiliation
1Toyota Technological Institute at Chicago, Chicago, IL
2Department of Human Genetics, University of Chicago, Chicago, IL
*To whom correspondence should be addressed to: wangsheng@uchicago.edu, jinboxu@gmail.com

Abstract
Deep Convolutional Neural Networks (DCNN) has shown excellent performance in a variety of Machine Learning tasks. Here we employ Deep Convolutional Neural Fields (DeepCNF), which is a combination of DCNN with Conditional Random Field (CRF), to address the task of protein sequence labelling for the prediction of solvent accessibility (ACC), order/disorder regions (DISO), and 8-state secondary structure element (SS8), which are fundamental problems in computational biology. We show that the label distribution in these real-world linear-chain structured data varies from equally distributed to highly imbalanced. Traditional objective functions, such as maximal log-likelihood and maximal labelwise accuracy, perform well on equally distributed tasks while suffer from highly imbalanced ones. To overcome this challenge, we present a new training algorithm to learn DeepCNF model from imbalanced structured data. In particular, we train DeepCNF by directly maximizing the empirical Area Under the ROC Curve (AUC), which is an unbiased measurement for imbalanced data. To fulfill this, we formulate AUC in a pairwise ranking framework and approximate it by a polynomial function and then apply a gradient-based procedure to optimize this approximation. Experimental results confirm that by maximizing AUC, DeepCNF can obtain significant improvement over the other two traditional objective functions on label imbalanced data in terms of both AUC and Matthews correlation coefficient (Mcc). Furthermore, the results also show that our prediction method greatly outperforms existing popular predictors on all of the three tasks.

Contributions
1. A novel training algorithm that directly maximizes the empirical AUC to learn DeepCNF model from imbalanced structured data is proposed.
2. The behavior of three objective functions, i.e. maximal log-likelihood, maximal labelwise accuracy, and maximal AUC, is studied on real-world protein sequence labelling problem, in which the label distribution varies from equally distributed to highly imbalanced.
3. We achieve the state-of-the-art performance on the prediction of solvent accessibility, order/disorder regions, and 8-state secondary structure element, which are important problems in computational biology.

part I: introduction
Deep Convolutional Neural Networks (DCNN), originated by Yann LeCun at 1998 [cite here] for document recognition, has now been widely used in a plethora of Machine Learning (ML) tasks ranging from speech recognition [cite here], computer vision [cite here], to computational biology [cite here]. This is due to the fact that DCNN could capture medium- and/or long-range structured information in a hierarchical manner. To further exploit such information encoded in the structured data, Chen et.al. [cite here] has integrated DCNN with fully connected Conditional Random Fields (CRF) for semantic image segmentation. Inspired by this idea, we propose Deep Convolutional Neural Fields (DeepCNF) model, which is a combination of DCNN with linear-chain Conditional Random Field (CRF), to address the task
of protein sequence labelling for the prediction of solvent accessibility (ACC), order/disorder regions (DISO), and 8-state secondary structure element (SS8), which are fundamental problems in computational biology [cite here].

A protein sequence can be viewed as a string of amino acids (also called residues in the protein context). The solvent accessibility of protein residues is one of the driving forces of protein folding [cite here], the order/disorder property of each residue plays an important role in many biological processes [cite here], and the protein Secondary Structures (SS) determine the local conformation of the polypeptide backbone of a protein chain [cite here]. However, the label distribution in these real-world problems varies from equally distributed to highly imbalanced. For example, only ~6% of residues are shown to be disorder [cite here]; while a few SS labels, such as 310 helix, beta-bridge, and pi-helix are extremely rare [cite here]. Traditional objective functions, such as maximal log-likelihood [cite here] and maximal labelwise accuracy [cite here], perform well on equally distributed tasks but suffer from highly imbalanced ones [cite here].

Since the distribution of the predicting labels is highly imbalanced for DISO and SS8, we develop a new method to train DeepCNF to further improve prediction performance. Specifically, we train DeepCNF by maximizing AUC (Area Under the ROC Curve), which is a good measure for class-imbalanced data [cite here]. Taking DISO prediction as an example, since by random guess we may have a predictor that has ~94% per-residue accuracy, but it has AUC ~0.5 and is completely useless. Actually, AUC is insensitive to changes in class distribution [cite here] because that the ROC curve specifies the relationship between false positive (FP) rate and true positive (TP) rate, which are independent of class distribution [cite here]. However, it is very challenging to directly optimize AUC. A few algorithms have been developed to maximize AUC on imbalanced unstructured data [cite here], but to the best of our knowledge, there is no such an algorithm for imbalanced structured data (e.g., sequence data addressed here). To maximize AUC of our DeepCNF model, we formulate the AUC function in a ranking framework, approximate it by a polynomial Chebyshev function [cite here] and then use L-BFGS [cite here] to optimize the approximation.

Our experimental results show that if the labels in a prediction task are equally distributed, then there are no large differences between the three training methods; on the other hand, if the label distribution is highly imbalanced, then training with maximum AUC will achieve higher AUC than the other two training methods, especially the maximum labelwise accuracy. Compared with other popular predictors on several publicly available benchmarks, our proposed approach achieves the state-of-the-art performance for all the three prediction tasks. In particular, at a high specificity that is comparable with the other predictors, our method obtains compatible or better precision and sensitivity for each label, especially for those rare labels.

part II: related works

1. related works at algorithmic level

The training algorithm that directly maximizes the AUC is not a new idea. A few researchers have proposed such algorithms as follows: (a) Ferri et al. [cite here] trained a decision tree by using AUC as splitting criteria; (b) Herschtal and Raskutti [cite here] trained a neural network by optimizing AUC; (c) Joachims [cite here] proposed a generalized Support Vector Machines (SVM) that optimizes AUC; and (d) Narasimhan [cite here] explored ways to optimize partial AUC for a structured SVM. However, all these previous methods are trained on un-structured data, while our proposed training algorithm is optimizing AUC for structured data, such as conditional random fields (CRF) and their deep extensions DeepCNF.

Recently, Rosenfeld et.al. [cite here] has proposed a learning algorithm for structured models with AUC loss. There are fundamental differences of our method with theirs: (a) our method targets at a sequence labelling problem with an imbalance label assignment, while Rosenfeld et al at a ranking problem; (b) we consider correlation among the labels of a sequence while Rosenfeld et al treat the ranking of each sample independent of the others; (c) our method is based on CRF, while they used structured SVM; and (d) we also studied deep learning extension of our method, while they did not.
2. related works at application level

Many methods that employ machine learning to integrate protein features for linear-chain label (such as ACC, DISO, and SS8) prediction have been proposed [cite here]. The most common approaches are based on neural networks (NN) [cite here] and support vector machines (SVM) [cite here]. Recently, Cheng et.al [cite here] developed a deep learning method using deep belief network (DBN) [cite here] for DISO prediction; whereas Zhou et.al. [cite here] reported a deep learning approach using a supervised generative stochastic network (GSN) [cite here] for SS8 prediction. Besides the uniqueness of the AUC-maximizing training algorithm, our method differs from them as follows.

For comparison with Cheng’s work on DISO prediction: (a) we use DCNN while Cheng uses a DBN constructed from restricted Boltzmann machine (RBM). DCNN is better than DBN in capturing a longer-range of sequential information; and (b) our method considers the correlation of the “ordered/disordered” states of sequentially-adjacent residues while Cheng’s method does not.

For comparison with Zhou’s work on SS8 prediction: (a) our method places only input features in a visible layer and treats the SS labels as hidden states while Zhou’s method places both the input features and SS labels in a visible layer; (b) our method explicitly models the SS label interdependency while Zhou’s method does not; (c) our method directly calculates the conditional probability of SS labels on input features while Zhou’s method uses sampling; and (d) our method trains the model parameter simultaneously from the input to output layer while Zhou’s method trains the model parameters layer-by-layer.

part III: method

1. Problem definition

We study three sequence labelling problems given a protein sequence consisting of a collection of residues. Here the label is defined as solvent accessibility (ACC), order/disorder (DISO), or 8-state protein secondary structure (SS8), respectively. The prediction of these labels is a fundamental problem in computational biology. Here we describe in brief the calculation of the real labels as follows.

ACC. We applied DSSP [cite here] to calculate the absolute accessible surface area for each residue in a protein. The relative solvent accessibility (RSA) of the residue X is calculated through dividing the absolute accessible surface area by the maximum solvent accessibility which uses Gly-X-Gly extended tripeptides [cite here]. With the RSA value, the classification was divided into three states, say buried (B, with RSA from 0 to 10), intermediate (I, with RSA from 10 to 40) and exposed (E, with RSA from 40 to 100). The background distribution of these three labels B,I,E is close to 1:1:1.

DISO. Following the definition in [cite here], we label a residue as disordered (label 1) if it is in a segment of more than three residues missing atomic coordinates in the X-ray structure. The other residues are labeled as ordered (label 0). The background of these two labels 0,1 is 94:6.

SS8. The 8-state protein secondary structure element was calculated by DSSP. In particular, DSSP assigns 3 types for helix (G for 310 helix, H for alpha-helix, and I for pi-helix), 2 types for strand (E for beta-strand and B for beta-bridge), and 3 types for coil (T for beta-turn, S for high curvature loop, and L for irregular). The background distribution of these eight labels H,E,L,T,S,G,B,I is 35:22:19:11:8:4:1:1.

2. Protein features

Given a protein sequence, we use the same feature set for the prediction of ACC, DISO, and SS8 label. The
A feature set could be divided into residue-related feature and evolution-related feature.

**Residue-related features.** (a) amino acid identity represented as a binary vector of 20 elements; (b) amino acid physic-chemical properties (7 values from Table 1 in [cite here]); (c) propensity of being at endpoints of a secondary structure segment (11 values from Table 1 in [cite here]); (d) correlated contact potential (40 values from Table 3 in [cite here]); and (e) AAindex (5 values from Table 2 in [cite here]). These features may allow for a richer representation of amino acids [cite here].

**Evolution-related features.** We use PSSM (position specific scoring matrix) generated by PSI-BLAST [cite here] to encode the evolutionary information of the sequence under prediction. We also use the HHM profile generated by HHpred [cite here], which is complementary to PSSM to some degree.

3. DeepCNF model

![Figure 1](image.png)

**Figure 1.** Illustration of a DeepCNF. Here $i$ is the position index and $X_i$ the associated input features, $H^k$ represents the $k$-th hidden layer, and $Y$ is the output label. All the layers from the 1st to the top layer form a DCNN with parameter $W^k$ ($k=1,2,...,K$). The top layer and the label layer form a CRF, in which the parameter $U$ specifies the relationship between the output of the top layer and the label layer and $T$ the binary relationship between adjacent labels. Windows size is set to 3 only for illustration.

As shown in Figure 1, DeepCNF has two modules: (i) the Conditional Random Fields (CRF) module consisting of the top layer and the label layer, and (ii) the deep convolutional neural network (DCNN) module covering the input to the top layer. When only one hidden layer is used, DeepCNF becomes Conditional Neural Fields (CNF), a probabilistic graphical model described in [cite here].

Given a protein sequence of length $L$, let $y = [y_1, ..., y_L] \in \Sigma^L$ denote its sequence label where $y_i$ is the label at residue $i$, and $\Sigma$ is the set of all possible labels with either ACC, DISO, or SS8. For instance, if we predict the order/disorder status for each residue, then we may have $\Sigma = \{0, 1\}$. Let $X = [X_1, ..., X_L]$ denote the input feature where $X_i$ is a column vector representing the input feature for residue $i$. DeepCNF calculates the conditional probability of $y$ on the input $X$ with parameter $\theta$ as follows,

$$ P_\theta(y|X) = \exp \left( \sum_{i=1}^L \left[ f_{\theta}(y, X, i) + g_{\theta}(y, X, i) \right] \right) / Z(X) $$  

(1)
Where \( f_{\theta}(y, X, i) \) is the binary potential function specifying correlation among adjacent order/disorder states at position \( i \), \( g_{\theta}(y, X, i) \) is the unary potential function modeling relationship between \( y_i \) and input features for position \( i \), and \( Z(X) \) is the partition function. Formally, \( f_{\theta}(\cdot) \) and \( g_{\theta}(\cdot) \) are defined as follows.

\[
f_{\theta}(y, X, i) = \sum_{a,b} T_{a,b} \delta(y_{i-1} = a) \delta(y_i = b)
\]

(2)

\[
g_{\theta}(y, X, i) = \sum_{a,b} U_{a,b} H_{a,b}(X, i, W) \delta(y_i = a)
\]

(3)

Where \( a \) and \( b \) represent two specific order/disorder labels, \( \delta() \) is an indicator function, \( H_{a,b}(X, i, W) \) is a deep neural network function for the \( h \)-th neuron at position \( i \) of the top layer for label \( a \), and \( W \), \( U \), and \( T \) are the model parameters to be trained. Specifically, \( W \) is the parameter for the neural network, \( U \) is the parameter connecting the top layer to the label layer, and \( T \) is for label correlation. The two potential functions can be merged into a single binary potential function \( f_{\theta}(y, X, i) = \sum_{a,b} T_{a,b} H_{a,b}(X, i, W) \delta(y_{i-1} = a) \delta(y_i = b) \). To control model complexity and avoid over-fitting, we add a L2-norm penalty term as the regularization factor.

Figure 1 shows two adjacent layers of DCNN. Let \( M_k \) be the number of neurons for a single position at the \( k \)-th layer. Let \( X_i|h| \) be the \( h \)-th feature at the input layer for residue \( i \) and \( H^h_k|i| \) denote the output value of the \( h \)-th neuron of position \( i \) at layer \( k \). When \( k = 1 \), \( H^h_k \) is actually the input feature \( X \). Otherwise, \( H^h_k \) is a matrix of dimension \( L \times M_k \). Let \( 2N_k+1 \) be the window size at the \( k \)-th layer. Mathematically, \( H^h_k|h| \) is defined as follows.

\[
H^h_k|h| = X_i|h|, \quad \text{if } k = 1
\]

\[
H^{k+1}_k|h| = \pi \left( \sum_{n=-N_k}^{N_k} \sum_{h=1}^{M_k} \left[ H^h_{i+n}|h| \ast W^n_{i,h} |h| \right] \right), \quad \text{if } k < K
\]

\[
H_k(X, i, W) = H^h_k|h|, \quad \text{if } k = K
\]

(4)

Meanwhile, \( \pi(\cdot) \) is the activation function, either the sigmoid (i.e., \( 1/(1+\exp|-x|) \)) or the tanh (i.e., \( (1-\exp|-2x|)/(1+\exp|-2x|) \)) function. \( W^n_{i,h} \) is a 2D weight matrix for the connections between the neurons of position \( i+n \) at layer \( k \) and the neurons of position \( i \) at layer \( k+1 \). \( W^n_{i,h} \) is shared by all the positions in the same layer, so it is position-independent. Here \( h \) and \( h \) index two neurons at the \( k \)-th and \( (k+1) \)-th layers, respectively. See Appendix about how to calculate the gradient of DCNN by back propagation.

4. objective functions

Suppose we have \( T \) protein sequences for training, and each sequence \( t \) with length \( L_t \). We study the behavior of three different objective functions, namely Log-likelihood, Labelwise accuracy, and the AUC function in this work.

4.1 Log-likelihood
The log-likelihood is the most commonly used objective function for training CRF [cite here]. In this criterion, the loss suffered for a training example \( (y^t, X^t) \) is the negative log probability of the true parse according to the model:

\[
\log_{\text{likelihood}} = \sum_{t=1}^{T} \log P_{\theta}[y^t|X^t]
\]

(5)

Where \( P_{\theta}[y|X] \) is defined in equation (1).

4.2 Labelwise accuracy
[cite here] proposed an object function that could directly maximize the labelwise accuracy defined as
\[ Labelwise = \sum_{t=1}^{T} \sum_{i=1}^{L} \delta(\hat{P}_\theta(y_i^t) > \max_{y_{i',\neq i}} \hat{P}_\theta(y_{i'}^t)) \]  

(6)

Where \( y_i^t \) denotes the real label at position \( i \), \( \hat{P}_\theta(y_i^t) \) is the predicted probability of the real label at position \( i \) being \( (\tau) \), which could be represented by the marginal probability \( \hat{P}_\theta(y_i^t) = \frac{1}{Z(X^t)} \sum_{y_{i',\neq i}} \delta(y_i = (\tau)) \cdot \exp \left( \hat{F}_{1:t}(X^t, y, \theta) \right) \)

where

\[ \hat{F}_{1:t}(X^t, y, \theta) = \sum_{i=1}^{L} f_\theta(y, X, i) \]

Due to the difficulty to take the derivatives of equation (8), we use the following Wilcoxon-Mann-Whitney statistic [cite here] to replace the indicator function with a sigmoidal function with parameter \( \lambda \) (default is set to 15) as \( Q_\lambda^t(x) = 1 / (1 + \exp(-\lambda x)) \). Then it becomes the following form:

\[ Labelwise \approx \sum_{t=1}^{T} \sum_{i=1}^{L} Q_\lambda^t \left( \hat{P}_\theta(y_i^t) X^t - \hat{P}_\theta(\hat{y}_i^t \vee X^t) \right) \]  

(7)

Where \( \hat{y}_i^t \) denotes the label other than \( y_i^t \) that has the maximum posterior probability at position \( i \).

4.3 The AUC function

**Definition.** The AUC of a predictor function \( \hat{P}_\theta \) on label \( \tau \) is defined as:

\[ AUC(\hat{P}_\theta, \tau) = \frac{|P(\hat{P}_\theta(y_i^t) > \hat{P}_\theta(y_j^t)) | \in D^t, j \in D^{t'}|}{|D^t||D^{t'}|} \]  

(8)

where \( P() \) is the probability over all pairs of positive and negative examples, \( D^t \) is a set of positive examples with true label \( \tau \), and \( D^{t'} \) is a set of negative examples with true label not being \( \tau \). Note that the set \( D^t \) and \( D^{t'} \) consist of residues from all \( T \) training sequences, i.e., \( D^t = \hat{t} = 1 \hat{t} T \hat{t} L_{\hat{t}, \hat{t}} \delta_{i,t} \hat{i} \) where \( \delta_{i,t} \hat{i} \) is an indicator function that if the true label of the \( i \)-th position from sequence \( t \) equals to \( \tau \), then \( \delta_{i,t} \hat{i} \) is equal to 1; otherwise 0. Again, \( \hat{P}_\theta(y_i^t) \) could be represented by the marginal probability \( \hat{P}_\theta(y_i^t | X^t) \) from the training sequence \( t \).

Due to the difficulty to take the derivatives of equation (8), we use the following Wilcoxon-Mann-Whitney statistic [cite here] which is an unbiased estimator of **AUC** \( \hat{P}_\theta, \tau \):

\[ AUC^{WWMW}(\hat{P}_\theta, \tau) = \frac{\sum_{i \in D^t} \sum_{j \in D^{t'}} \delta(\hat{P}_\theta(y_i^t) > \hat{P}_\theta(y_j^t))}{|D^t||D^{t'}|} \]  

(9)

Finally, by summing over all labels, the overall AUC objective function is:

\[ \sum_{\tau} AUC^{WWMW}(\hat{P}_\theta, \tau) \]

Approximation. For a large dataset, the computational cost of AUC by equation (9) is high. Recently, [cite here] proposed a polynomial approximation of AUC which can be computed in linear time. The key idea is to approximate the indicator function \( \delta(x > 0) \), where \( x \) represents \( \hat{P}_\theta(y_i^t | X) - \hat{P}_\theta(y_j^t | X) \), by a polynomial Chebyshev approximation. That is, we approximate \( \delta(x > 0) \) by \( \sum_{p=0}^{d} c_p \hat{X}^p \) where \( d \) is the degree and \( c_p \) the coefficient of the polynomial [cite here]. Let \( n_0 = |D^t| \) and \( n_0 = |D^{t'}| \). Using the polynomial Chebyshev approximation, we can approximate equation (9) as follows:

\[ AUC^{WWMW}(\hat{P}_\theta, \tau) \approx \frac{1}{n_0 n_1} \sum_{(l,m) \in Q} \sum_{l=0}^{d} Y_{lm} S(\hat{P}_\theta^{\nu |-l}, D^t) \vee (\hat{P}_\theta^{\nu |-l}, D^{t'}) \]  

(10)
where \( Y_{\mu l} = c \left( \mu \right) \left( -1 \right)^{\mu - l} \), \( s \left[ P^l, D^l \right] = \sum_{i \in D^l} P \left( y_i^l \right) \) and \( v \left[ P^l, D^l \right] = \sum_{j \in D^l} P \left( y_j^l \right) \). Note that we have \( s \left[ P^l, D^l \right] = \sum_{t = 1}^{T} \sum_{i = 1}^{L} \delta_{i, t}^l P \left( y_i^l \right) \) and a similar structure for \( v \left[ P^l, D^l \right] \).

4.4 Complexity analysis

We derive the gradient of the polynomial approximation of AUC function in the Appendix. The gradient of the labelwise accuracy function is derived in [cite here]. While the space complexity is \( O \left( \hat{\Sigma} \sqrt{\nu \cdot L} \right) \) for all the three objectives, the time complexity is different. Specifically, the calculation for log-likelihood, labelwise accuracy, and the polynomial approximation of AUC function requires \( O \left( \hat{\Sigma} \sqrt{\nu^2 \cdot L} \right) \), \( O \left( \hat{\Sigma} \nu^2 \cdot L \right) \), and \( O \left( \hat{\Sigma} \nu^3 \cdot L \right) \), respectively. Since a deep neural network is applied, we may not be able to solve the training problem to global optimum. Instead we use the L-BFGS [cite here] algorithm to find a suboptimal solution.

For training by the AUC function, the running time is linear when protein length is much larger than the number of labels and the degree of the polynomial Chebyshev approximation degree \( d \). When degree \( d \) gets larger, we can approximate the loss function better, but the approximation itself becomes less smooth and more challenging to optimize. A large \( d \) also increases model complexity, which makes it easier to overfit. In our experiments, with the increase of \( d \), the training AUC always improves, but the testing AUC drops after \( d = 15 \).

part IV: result

1. dataset

To select non-redundant proteins for training and test, we pick one representative from each protein superfamily defined by CATH [cite] or SCOP [cite here]. By using test proteins in different superfamilies than the training proteins, we can reduce the bias incurred by the sequence profile similarity between the training and test proteins [cite here]. To fulfill this, we use the publically available JPRED [cite here] training and test data (http://www.compbio.dundee.ac.uk/jpred4/about.shtml), which has 1338 training and 149 test proteins, respectively, each of which belongs to a different superfamily.

For each of the three prediction task, namely ACC, DISO, and SS8, we use this JPRED dataset for comparison of different DeepCNF model hyper-parameters and then determining the optimized ones by 7-cross validation, under the three above mentioned objective functions, i.e., maximal likelihood, maximal labelwise accuracy, and maximal AUC. We then evaluate predictive performance of our methods with the other state-of-the-art methods on CASP10 [cite here], CASP11 [33] targets (merged to CASP dataset) and the recent CAMEO [cite here] (http://www.cameo3d.org/sp/1-year/) hard targets. To remove redundancy between the training/testing data and evaluation data, we filter CASP and CAMEO dataset to make sure that proteins sharing >25% sequence identity with JPRED dataset are removed. In total, the remaining list of CASP and CAMEO dataset is 126 and 147, respectively.

2. evaluation criteria
The simplest measurement of sequence labelling quality is QX accuracy where X is the number of labels. QX is defined as the percentage of residues for which the predicted X labels are correct. For instance, for ACC prediction we use Q3 accuracy whereas for SS8 we use Q8.

We also use some complex measurements based on the confusion matrix consisting of TP, TN, FP, and FN, for a label T under consideration. Specifically, TP (true positives) and TN (true negatives) are the numbers of correctly predicted residues given a label T and those residues not with that label, respectively; whereas FP (false positives) and FN (false negatives) are the numbers of misclassified residues, respectively.

Then we use sensitivity (sens), specificity (spec), precision (prec) and Matthews correlation coefficient (Mcc), defined as

\[
\text{TP} / (\text{TP} + \text{FN}), \quad \text{TN} / (\text{TN} + \text{FP}), \quad \text{TP} / (\text{TP} + \text{FP}), \quad \text{TP} \times \text{TN} - \text{FP} \times \text{FN} / \sqrt{(\text{TP} + \text{FP})(\text{TN} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FN})},
\]

respectively. We also use AUC as the measurement. Note that Mcc and AUC are generally regarded as balanced measures which can be used on class-imbalanced data. Mcc ranges from −1 to +1, with +1 representing a perfect prediction, 0 random prediction and −1 total disagreement between prediction and ground truth. AUC has a minimum value 0.5 and the best value 1.0. Note that in the following sections we may use average Mcc (denoted as \(\text{Mcc}^\prime\)) and average AUC (denoted as \(\text{AUC}^\prime\)) when the number of labels is above two.

3. performance of training methods with different DCNN architectures on different tasks

3.1 DCNN architecture

The architecture of the DCNN in DeepCNF model is mainly determined by the following 3 factors (see Figure 1): (i) the number of hidden layers; (ii) the number of different neurons at each layer; and (iii) the window size at each layer. We compared three different methods for training the DeepCNF model: maximum likelihood, maximum labelwise accuracy, and maximum AUC for the prediction of three-label solvent accessibility (ACC), two-label order/disorder (DISO), and eight-label secondary structure element (SS8), respectively. We conduct 7-fold cross-validation for each possible DCNN architecture, each training method, and each prediction task on JPRED dataset. To simplify the analysis, all the hidden layers have the same number of neurons and the same windows size. The default hyper parameters are 5 hidden layers, 50 hidden neurons and 11 windows size for each layer.

3.2 overall performance

Overall, as shown in Figure 2 to 4, if the labels in a prediction task are equally distributed, then there are no large differences between the three training methods. On the other hand, if the label distribution is highly imbalanced, then training with maximum AUC will achieve higher averaged Mcc and averaged AUC than the other two training methods, especially the maximum labelwise accuracy, while at a slightly cost of QX accuracy. Finally, the maximal likelihood training method is the most robust one. Regarding the model architecture, our model has almost peak performance when it has 4 to 5 hidden layers, 50 to 100 different hidden neurons at each layer, and windows size set to 11. Further increasing the number of layers, number of neurons, and the windows size does not result in significant improvement in QX accuracy, averaged Mcc, and averaged AUC, regardless of the training method.

3.3 performance on each prediction task

We show the performance on the three prediction tasks in details: (i) for ACC prediction, because of the equally distributed three labels, as shown in Figure 2, no matter what training methods are chosen, the optimal performance could obtain 0.69 Q3 accuracy, 0.45 averaged Mcc and 0.82 averaged AUC, respectively; (ii) for DISO prediction, due to the imbalance ratio between order and disorder label is 94:6, as shown in Figure 3, although all three training methods obtain the similar Q2 accuracy at 0.94, the AUC-maximization method obtains 0.51 averaged Mcc and approaches 0.89 averaged AUC, respectively, which greatly outperforms the other two training methods at around 0.49 averaged Mcc and less than 0.86 averaged AUC, respectively; (iii) for SS8 prediction, as shown in Figure 4, since there exists three rare labels (i.e., G for 310 helix, B for beta-bridge, and I for pi-helix), the AUC-maximization method could
greatly enhance the Mcc and AUC of them, and in turn increase the overall averaged Mcc to 0.44 and averaged AUC to 0.86, respectively. This results in the large improvement over the maximum labelwise accuracy training method that achieves 0.41 averaged Mcc and less than 0.8 averaged AUC, respectively.

Figure 2. Dependency of Q3 accuracy, averaged Mcc, and averaged AUC of solvent accessibility (ACC) prediction with respect to the architecture of DCNN. (left) the number of neurons, (middle) window size, and (right) the number of hidden layers. Three different training methods: maximum likelihood (blue), maximum labelwise accuracy (red) and maximum AUC (green).
Figure 3. Dependency of Q2 accuracy, averaged Mcc, and averaged AUC of order/disorder (DISO) prediction with respect to the architecture of DCNN. (left) the number of neurons, (middle) window size, and (right) the number of hidden layers. Three different training methods: maximum likelihood (blue), maximum labelwise accuracy (red) and maximum AUC (green).
Figure 4. Dependency of Q8 accuracy, averaged Mcc, and averaged AUC of 8-state secondary structure element (SS8) prediction with respect to the architecture of DCNN. (left) the number of neurons, (middle) window size, and (right) the number of hidden layers. Three different training methods: maximum likelihood (blue), maximum labelwise accuracy (red) and maximum AUC (green).

4. comparison with state-of-the-art

4.1 programs to compare

Since our prediction of ACC, DISO, and SS8 label is based on the same ab initio protein features, we do not compare our approach with consensus-based [cite here] or template-based [cite here] methods. Instead, we compare our approach with the following ab initio predictors: (i) for ACC prediction, we choose SPINE-X [cite here] and ACCpro5-ab [cite here]. SPINE-X is a neural network (NN) based method while ACCpro5-ab is based on bidirectional recurrent neural network (RNN); (ii) for DISO prediction, we choose DNdisorder [cite here] and DisoPred3-ab [cite here]. DNdisorder applies deep belief network (DBN) while DisoPred3-ab uses support vector machine (SVM) and NN for prediction; (iii) for SS8 prediction, we compare our approach with SSpro5-ab and RaptorX-SS8 [cite here]. SSpro5-ab is based on RNN while RaptorX-SS8 is a conditional neural field (CNF) based method. We cannot evaluate Zhou’s method because it is not publicly available.

4.2 overall evaluation

Having determined the hyper parameters of DeepCNF on JPRED dataset, we evaluate our approach trained on maximum AUC with the other state-of-the-art methods on CASP and CAMEO dataset. As shown in Table 1 to 3, our AUC-maximized DeepCNF model outperforms all other methods on all three prediction
tasks (i.e., ACC, DISO, and SS8) in terms of all evaluation metric, ranging from QX accuracy, Mcc, to AUC, respectively. For prediction tasks with large label imbalance ratio, our method significantly beat the others in Mcc and AUC. Specifically, for DISO prediction on CASP dataset, our method achieves 0.53 Mcc and 0.88 AUC, respectively, which greatly outperforms DNdisorder (0.37 Mcc and 0.81 AUC) and DisoPred3_ab (0.47 Mcc and 0.84 AUC); for SS8 prediction on CAMEO dataset, our method obtains 0.42 Mcc and 0.83 AUC, respectively, which is much better than SSpro5_ab (0.37 Mcc and 0.78 AUC) and RaptorX-SS8 (0.38 Mcc and 0.79 AUC).

4.3 sensitivity, specificity, and precision

Compared with other popular predictors on several publicly available benchmarks, our proposed approach achieves the state-of-the-art performance for all the three prediction tasks. In particular, at a high specificity that is comparable with the other predictors, our method obtains better precision and sensitivity for each label, especially those rare labels.

For different prediction tasks, Table 4 to 5 shows the sensitivity, specificity, and precision on each of the label obtained by our method and the other approaches on the merged CASP and CAMEO dataset. Overall, at a high specificity that is comparable with the other state-of-the-art approaches, our method obtains compatible or better precision and sensitivity for each label, especially for those rare labels such as G, I, B, S, T for SS8, and disorder state for DISO. Taking SS8 prediction as an example, for pi-helix (I), our method has sensitivity and precision 0.18 and 0.33 respectively, while the other method obtains 0.03 and 0.12, to the best. For beta-bridge (B), our method obtains sensitivity and precision 0.13 and 0.42, respectively, while the other method obtains 0.07 and 0.34, to the best.

Table 1. Per-residue performance of solvent accessibility (ACC) prediction on the CASP and CAMEO targets. The evaluation criteria are QX, sensitivity (sens), specificity (spec), precision (prec), the Matthews correlation coefficient (Mcc), and area under the ROC curve (AUC). Note that X is the number of labels for prediction (for ACC prediction, X equals to 3); sens, spec, prec, Mcc and AUC are averaged on X labels. The best values are shown in bold. See text for explanation of all the tested methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>CASP</th>
<th>CAMEO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q3</td>
<td>Sens</td>
</tr>
<tr>
<td>Our method</td>
<td>0.6</td>
<td>0.65</td>
</tr>
<tr>
<td>SPINE-X</td>
<td>0.6</td>
<td>0.59</td>
</tr>
<tr>
<td>ACCpro5_ab</td>
<td>0.6</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Table 2. Per-residue performance of order/disorder (DISO) prediction on the CASP and CAMEO targets.

<table>
<thead>
<tr>
<th>Method</th>
<th>CASP</th>
<th>CAMEO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q2</td>
<td>Sens</td>
</tr>
<tr>
<td>Our method</td>
<td>0.9</td>
<td>0.74</td>
</tr>
<tr>
<td>DisoPred3_ab</td>
<td>0.9</td>
<td>0.67</td>
</tr>
<tr>
<td>DNdisorder</td>
<td>0.9</td>
<td>0.73</td>
</tr>
</tbody>
</table>

Table 3. Per-residue performance of 8-state secondary structure element (SS8) prediction on the CASP and CAMEO targets.

<table>
<thead>
<tr>
<th>Method</th>
<th>CASP</th>
<th>CAMEO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q8</td>
<td>Sens</td>
</tr>
<tr>
<td>Our method</td>
<td>0.7</td>
<td>0.48</td>
</tr>
</tbody>
</table>
Table 4. Sensitivity, specificity, and precision of each solvent accessibility (ACC) label on the CASP and CAMEO targets merged together.

<table>
<thead>
<tr>
<th>ACC label</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0.77</td>
<td>0.74</td>
<td>0.75</td>
</tr>
<tr>
<td>M</td>
<td>0.45</td>
<td>0.36</td>
<td>0.34</td>
</tr>
<tr>
<td>E</td>
<td>0.71</td>
<td>0.67</td>
<td>0.63</td>
</tr>
</tbody>
</table>

*Our method; †SPINE-X; ‡ACCpro5_ab

Table 5. Sensitivity, specificity, and precision of each order/disorder (DISO) label on the CASP and CAMEO targets merged together.

<table>
<thead>
<tr>
<th>DISO label</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.96</td>
<td>0.96</td>
<td>0.89</td>
</tr>
<tr>
<td>1</td>
<td>0.51</td>
<td>0.41</td>
<td>0.55</td>
</tr>
</tbody>
</table>

*Our method; †DisoPred3_ab; ‡DNdisorder

Table 6. Sensitivity, specificity, and precision of each 8-state secondary structure element (SS8) label on the CASP and CAMEO targets merged together.

<table>
<thead>
<tr>
<th>SS8 label</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0.91</td>
<td>0.89</td>
<td>0.90</td>
</tr>
<tr>
<td>G</td>
<td>0.28</td>
<td>0.21</td>
<td>0.19</td>
</tr>
<tr>
<td>I</td>
<td>0.18</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>E</td>
<td>0.84</td>
<td>0.78</td>
<td>0.77</td>
</tr>
<tr>
<td>B</td>
<td>0.13</td>
<td>0.05</td>
<td>0.07</td>
</tr>
<tr>
<td>T</td>
<td>0.56</td>
<td>0.49</td>
<td>0.51</td>
</tr>
<tr>
<td>S</td>
<td>0.29</td>
<td>0.21</td>
<td>0.18</td>
</tr>
<tr>
<td>L</td>
<td>0.61</td>
<td>0.62</td>
<td>0.63</td>
</tr>
</tbody>
</table>

*Our method; †RaptorX-SS8; ‡SSpro5_ab

---

part V: conclusion

In this work we proposed a novel training algorithm that directly maximizes the empirical AUC to learn DeepCNF model (DCNN+CRF) from imbalanced structured data. We also studied the behavior of three objective functions, i.e. maximal log-likelihood, maximal labelwise accuracy, and maximal AUC, on real-world protein sequence labelling problem, in which the label distribution varies from equally distributed to highly imbalanced. Finally, we achieved the state-of-the-art performance in terms of AUC and Matthews correlation coefficient (Mcc) on the prediction of solvent accessibility (ACC), order/disorder regions (DISO), and 8-state secondary structure element (SS8), which are fundamental problems in computational biology.

Instead of using linear-chain CRF, we may model a protein by Markov Random Fields (MRF) which can capture long-range residue interactions [cite here]. As suggested in [cite here], the predicted residue-residue contact information could further contribute to disorder prediction under the MRF model. In addition to the three prediction tasks mentioned in this work, our AUC maximization training algorithm could be applied to many sequence labelling problems with imbalanced label distributions [cite here]. For example, in post-translation modification (PTM) site prediction, the phosphorylation and methylation sites occur much less often than normal residues [cite here].

Acknowledgements
Appendix

S1. Gradient of the polynomial approximation of AUC

The gradient of the approximate AUC with respect to the parameter $\theta$ is as follows,

$$\frac{\partial \text{AUC}^{WWM}}{\partial \theta} = \frac{1}{n_0 n_1} \sum_{i=0}^{d} \sum_{y=0}^{y} \left( \frac{\partial s[P^{i}_o, D_i]}{\partial \theta} + \frac{\partial v[P^{\mu-i}_0, D^\tau_i]}{\partial \theta} \right) \frac{\partial v[P^{\mu-i}_0, D^\tau_i]}{\partial \theta}$$

Note that the calculation of $\frac{\partial s[P^{i}_o, D_i]}{\partial \theta}$ and $\frac{\partial v[P^{\mu-i}_0, D^\tau_i]}{\partial \theta}$ is similar, so we only explain one of them, and suppose there is only one training sequence with length $L$. In particular,

$$\frac{\partial s[P^{i}_o, D_i]}{\partial \theta} = \sum_{i=1}^{L} \frac{\partial [\delta P^{i}_o(y_i^\top X)]}{\partial \theta}$$

Let $Q_i(P_\theta) = [\delta P^{i}_o(y_i^\top X)]^T$, then

$$\frac{\partial s[P^{i}_o, D_i]}{\partial \theta} = \sum_{i=1}^{L} Q_i \frac{\partial P^{i}_o(y_i^\top X)}{\partial \theta}$$

Where $Q_i$ is the gradient of $Q_i$ with respect to the marginal probability $P_\theta$.

Since $P_\theta(y_i^\top X) = \frac{1}{Z(X)} \sum_{y_{i} \in y} \delta(y_i = \tau) \cdot \exp(F_{1:i}[X, y, \theta])$, applying the quotient rule we can compute the gradient of equation (S3) as follows.

$$\frac{\partial s[P^{i}_o, D_i]}{\partial \theta} = \sum_{i=1}^{L} \frac{1}{Z(X)} Q_i \sum_{y_{i} \in y} \delta(y_i = \tau) \cdot \frac{\partial F_{1:i}[X, y, \theta]}{\partial \theta} \cdot \exp(F_{1:i}[X, y, \theta]) \left[ Q_i \cdot P_\theta(y_i^\top X) \right]$$

The second term in equation (S4) could be calculated efficiently using forward-backward algorithm. For parameter $T$ at position $i$, the gradient could be calculated as follow.

$$-\sum_{u} \sum_{u} \alpha[u, i-1] \cdot \beta[u, i] \cdot \frac{\exp(f_\theta[u', u, X, i]) \cdot \delta f_\theta[u', u, X, i]}{Z(X)} \cdot \frac{\partial g_\theta[u, X, i]}{\partial \theta}$$

For parameter $U$ at position $i$, the gradient could be calculated as follows.

$$-\sum_{u} \alpha[u, i] \cdot \beta[u, i] \cdot \frac{\exp(f_\theta[u, X, i]) \cdot \delta f_\theta[u, X, i]}{Z(X)} \cdot \frac{\partial g_\theta[u, X, i]}{\partial \theta}$$

Where $u$ denotes one label, and
by the following recurrences:

\[ \text{forward function } \alpha(u,i) \text{ and backward function } \beta(u,i) \text{ are defined as follows.} \]

\[ \alpha(u,i) = \sum_{u} \delta(y_i = u) \cdot \exp[F_{i+1} \mid X, y, \theta] \]

\[ \beta(u,i) = \sum_{y_i} \delta(y_i = u) \cdot \exp[F_{i+1} \mid X, y, \theta] \]

They can be calculated by dynamic programming as follows.

\[ \alpha(u,i) = \sum_{u} \alpha(u',i-1) \cdot \exp[f_{\theta}(u',u,X,i)] \]

\[ \beta(u,i) = \sum_{u} \beta(u',i+1) \cdot \exp[f_{\theta}(u',u,X,i+1)] \]

The gradient of the inner summation part of the first term in equation (S4) with respect to parameter \( T \) at position \( i \) could be calculated as follows.

\[ \sum_{i} \sum_{u} \phi(u',u,i) \cdot \exp[f_{\theta}(u',u,X,i)] \cdot \frac{\partial f_{\theta}(u',u,X,i)}{\partial \theta} \]

where \( \phi(u',u,i) = Q_i \cdot \delta(y_i = \tau) \cdot \frac{\alpha(u',i-1) \cdot \beta(u,i) + \alpha(u',i-1) \cdot \beta'(u,i)}{Z \cdot X} + \frac{\alpha'(u',i-1) \cdot \beta(u,i) + \alpha(u',i-1) \cdot \beta'(u,i)}{Z \cdot X} \]

Similarly, the inner summation part of the first term in equation (S4) with respect to parameter \( U \) at position \( i \) could be calculated as follows.

\[ \sum_{i} \phi(u,i) \cdot \frac{\partial g_{\theta}(u,X,i)}{\partial \theta} \]

where \( \phi(u,i) = \frac{\alpha'(u,i) \cdot \beta(u,i) + \alpha(u,i) \cdot \beta'(u,i)}{Z \cdot X} \]

Here we define,

\[ \alpha'(u,i) = \sum_{i} \frac{\alpha'(u',i-1) + Q_i \cdot \delta(u = \tau) \cdot \alpha(u',i-1) \cdot \exp[f_{\theta}(u',u,X,i)]}{Z \cdot X} \]

\[ \beta'(u,i) = \sum_{i} \frac{\beta'(u',i+1) + Q_{i+1} \cdot \delta(u' = \tau) \cdot \beta(u',i+1) \cdot \exp[f_{\theta}(u',u,X,i)]}{Z \cdot X} \]

Like the forward matrix \( \alpha(u,i) \) and backward matrix \( \beta(u,i) \), \( \alpha'(u,i) \) and \( \beta'(u,i) \) may also be calculated by dynamic programming. In particular, given the initial conditions \( \alpha'(u,1) = Q_1 \cdot \delta(u = \tau) \cdot \alpha(u,1) \) and \( \beta'(u,L) = 0 \), \( \alpha'(u,i) \) and \( \beta'(u,i) \) can be computed by the following recurrences:

\[ \alpha'(u,i) = \sum_{u} \frac{\alpha'(u',i-1) + Q_i \cdot \delta(u = \tau) \cdot \alpha(u',i-1) \cdot \exp[f_{\theta}(u',u,X,i)]}{Z \cdot X} \]

\[ \beta'(u,i) = \sum_{u} \frac{\beta'(u',i+1) + Q_{i+1} \cdot \delta(u' = \tau) \cdot \beta(u',i+1) \cdot \exp[f_{\theta}(u',u,X,i)]}{Z \cdot X} \]

Let \( a \) and \( b \) denote the labels at two adjacent sequence positions, then the gradient of equation (S4) with respect to parameter \( T \) is:

\[ \frac{\partial s[P_{\theta}^T, D']}{\partial T_{a,b}} = \sum_{i=1}^{L} \tilde{\phi}(a,b,i) \cdot \exp[f_{\theta}(a,b,X,i)] \]

where \( \tilde{\phi}(a,b,i) = \tilde{c} \cdot Q_i \cdot \delta(y_i = \tau) \cdot \frac{\alpha(a,i-1) \cdot \beta(b,i) + \alpha'(a,i-1) \cdot \beta'(b,i) + \alpha(a,i-1) \cdot \beta'(b,i) + \alpha(a,i-1) \cdot \beta(b,i)}{Z \cdot X} \]
The gradient of equation (S4) with respect to parameter $U$ is:

$$
\frac{\partial s[P_{\theta}^j, D^j]}{\partial U_{a,h}} = \sum_{i=1}^{L} \left( \tilde{\phi}[a,i] \cdot H_{a,h}(X,i,W) \right)
$$

(S22)

Where

$$
\tilde{\phi}[a,i] = \frac{\alpha \cdot a \cdot i \cdot \beta \cdot a \cdot i}{Z \cdot X} + \frac{\alpha \cdot a \cdot i \cdot \beta \cdot a \cdot i}{Z \cdot X} - \frac{\alpha \cdot a \cdot i \cdot \beta \cdot a \cdot i}{Z \cdot X}.
$$

(S23)

**S2. More details about the DeepCNF model**

As shown in Fig. 2 in the main text, the DeepCNF has three architecture hyper-parameters: (a) the number of neurons at each layer; (b) the window size at each layer; and (c) the number of the hidden layers. We train the model parameters (i.e., $U$, $T$, $W$) of DeepCNF simultaneously. We first calculate the gradient for parameter $U$, $T$ and then for parameter $W$. Below we explain how to calculate the DeepCNF in a feed-forward way and the gradient by back-propagation.

**S2.1 Feed-forward function of DCNN (deep convolutional neural network)**

Appendix Figure 1 shows two adjacent layers of DCNN. Let $M_k$ be the number of neurons for a single position of the $k$-th layer. Let $X_i[h]$ be the $h$-th feature at the input layer for residue $i$ and $H_i^k[h]$ denote the output value of the $h$-th neuron of position $i$ at layer $k$. When $k=1$, $H_i^1$ is actually the input feature $X$. Otherwise, $H_i^k$ is a matrix of dimension $L \times M_k$. Let $2N_k+1$ be the window size at the $k$-th layer. Mathematically, $H_i^k[h]$ is defined as follows.

$$
H_i^k[h] = \begin{cases} 
X_i[h], & \text{if } k = 1 \\
\pi(\sum_{n=-N_k}^{N_k} \sum_{h'=1}^{M_k} H_i^{k-1}[h'] \ast W_n^k[h,h']) & , \text{if } k < K \\
H_i^{k+1}[h], & \text{if } k = K
\end{cases}
$$

(S24)

Meanwhile, $\pi$ is the activation function, either the sigmoid (i.e., $\pi(x) = 1/(1+\exp(-x))$) or the tanh (i.e., $\pi(x) = (1-\exp(-2x))/|1+\exp(-2x)|$) function. $W_n^k (-N_k \leq n \leq N_k)$ is a 2D weight matrix for the connections between the neurons of position $i$ at layer $k$ and the neurons of position $i+1$ at layer $k+1$. $W_n^k$ is shared by all the positions in the same layer, so it is position-independent. Here $h$ and $h'$ index two neurons at the $k$-th and $(k+1)$-th layers, respectively.

Appendix Figure 1. The feed-forward connection between two adjacent layers of DCNN.
S2.2 Calculation of gradient by back-propagation

Appendix Figure 2. Illustration of how to calculate the gradient of DCNN from layer \( k+1 \) to layer \( k \).

The error function from the CRF part at position \( i \) for a certain label \( u \) is

\[
E_i(u) = \sum_{\mu=0}^{d} \sum_{l=0}^{u} \tilde{\phi}_s(u, i) \cdot v(P_{\theta}^{\mu,l}, D^{lr}) + s(P_{\theta}^{\mu,l}, D^{lr}) \cdot \tilde{\phi}_v^{u-l}(u, i),
\]

where \( \tilde{\phi}_s \) and \( \tilde{\phi}_v \) are derived according to equation (S23) with respect to function \( s(P_{\theta}^{\mu,l}, D^{lr}) \) and \( v(P_{\theta}^{\mu,l}, D^{lr}) \), respectively. As shown in Appendix Figure 2, we can calculate the neuron error values as well as the gradients at the \( k \)-th layer by back-propagation as follows.

\[
\eta(h_i^k) \sum_u H_i[ E_i(u) \ast U_{a,h} ]
\]

if \( k = K \)

\[
E_i^k[ h ] = \ddot{h}_i
\]

\[
\eta(h_i^k) \sum_{n=-N_i}^{M_i} \sum_{h=1}^{M_k} E_{i+n}^{k+1}[ h ] \ast W_n^k[ h, h ]
\]

if \( k < K \) (S25)

Where \( \eta \) is the derivative of the activation function \( \pi \). In particular, it is \( \eta(x) = (1-x) \ast x \) and \( \eta(x) = 1-x \ast x \) for the sigmoid and tanh function, respectively. \( E_i^k \) is the neuron error value matrix at the \( k \)-th layer, with dimension \( L \times M_k \). Finally, the gradient of the parameter \( W \) at the \( k \)-th layer is:

\[
\nabla W_{i,n}(h, h) = \sum_{l=1}^{L} [ E_i^{k+1}[ h ] \ast H_i^k_{i+n}[ h ]]
\]

(S26)