Bioinformatics, 34(17), 2018, 2944–2950
doi: 10.1093/bioinformatics/bty305
Advance Access Publication Date: 16 April 2018
Original Paper

OXFORD

Sequence analysis

Parapred: antibody paratope prediction using convolutional and recurrent neural networks

Edgar Liberis[1,*], Petar Veličković[1], Pietro Sormanni[2,*], Michele Vendruscolo[2] and Pietro Liò[1]

[1]Department of Computer Science and Technology, University of Cambridge, CB3 0FD, UK and [2]Department of Chemistry, University of Cambridge, CB2 1EW, UK

*To whom correspondence should be addressed

Associate Editor: Dr. John Hancock

Received on October 18, 2017; revised on February 20, 2018; editorial decision on March 15, 2018; accepted on April 13, 2018

Abstract

Motivation: Antibodies play essential roles in the immune system of vertebrates and are powerful tools in research and diagnostics. While hypervariable regions of antibodies, which are responsible for binding, can be readily identified from their amino acid sequence, it remains challenging to accurately pinpoint which amino acids will be in contact with the antigen (the paratope).

Results: In this work, we present a sequence-based probabilistic machine learning algorithm for paratope prediction, named Parapred. Parapred uses a deep-learning architecture to leverage features from both local residue neighbourhoods and across the entire sequence. The method significantly improves on the current state-of-the-art methodology, and only requires a stretch of amino acid sequence corresponding to a hypervariable region as an input, without any information about the antigen. We further show that our predictions can be used to improve both speed and accuracy of a rigid docking algorithm.

Availability and implementation: The Parapred method is freely available as a webserver at http://www-mvsoftware.ch.cam.ac.uk/and for download at https://github.com/eliberis/parapred.

Contact: el398@cam.ac.uk or ps589@cam.ac.uk

Supplementary information: Supplementary information is available at *Bioinformatics* online.

1 Introduction

Antibodies are a special class of proteins produced by the immune system of vertebrates to neutralize pathogens, such as bacteria or viruses. They act by binding tightly to a unique molecule of the foreign agent, called the antigen. Antibody binding can mark it for future destruction by the immune system or, in some instances, neutralize it directly (e.g. by blocking a part of a virus essential for cell invasion). Typical antibodies are tetrameric—made of two immunoglobulin (Ig) heavy chains and two Ig light chains—and have a Y-shaped structure, where each of the two identical tips contains a binding site (paratope). The base of the Y mediates the ability of an antibody to communicate with other components of the immune system.

The paratope is typically contained within the hypervariable regions of the antibody which are also referred to as *complementarity determining regions* (*CDRs*). In the structure of an antibody, CDRs are located within binding loops, three on each heavy chain (H1, H2, H3) and three on each light chain (L1, L2, L3). The variability of the CDR sequences allows antibodies to form complexes with virtually any antigen. This binding malleability of antibodies is increasingly harnessed by the biotechnological and biopharmaceutical industry; indeed, monoclonal antibodies are currently the fastest growing class of therapeutics on the market (Ecker *et al.*, 2015; Reichert, 2017).

Novel antibodies that bind a target of interest can be obtained using well-established methods based on animal immunization or on *in vitro* technologies for screening large laboratory-constructed libraries (Leavy, 2010). However, for applications in research, diagnostics and therapeutics, some degree of engineering is required to optimize certain properties, such as binding affinity, stability, solubility or expression yield (Chiu and Gilliland, 2016). Rational engineering decisions become easier if detailed knowledge about an

© The Author(s) 2018. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permissions@oup.com

Downloaded from https://academic.oup.com/bioinformatics/article-abstract/34/17/2944/4972995 by University of Cambridge user on 23 September 2018

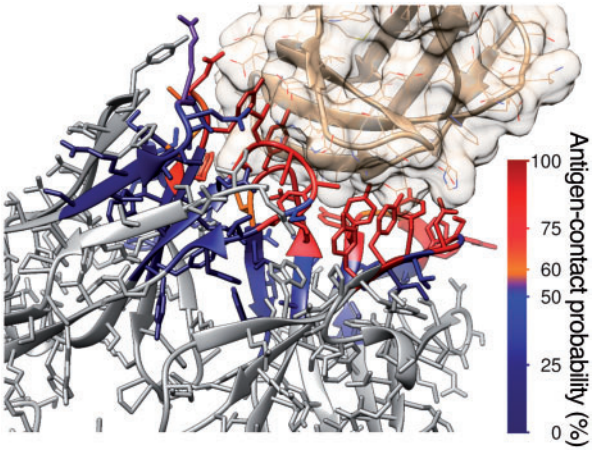


Fig. 1. An example of an antibody–antigen complex (PDB ID 2VXQ) where the antibody CDR loops are shaded according to the binding probabilities calculated by Parapred. Docking was performed using PatchDock with binding site constraints supplied by Parapred

antibody under scrutiny is obtained (Chiu and Gilliland, 2016; Sormanni *et al.*, 2017). However, especially at the early stages of an antibody discovery campaign, only the sequence and an estimate of the binding affinity are usually available. Therefore, computational methods that can accurately predict molecular traits using just the amino acid sequence have a great potential for accelerating antibody discovery by assisting lead selection or facilitating property engineering.

For instance, hypervariable regions contain 40–50 amino acid residues, whereas typically less than 20 actually participate in binding (Esmaielbeiki *et al.*, 2016), and some may even fall outside of the traditional definition of the CDRs (Kunik *et al.*, 2012). The ability to accurately map the paratope would enable to pinpoint residues that are involved in binding, leaving others as candidate mutation sites that can be exploited to optimize other molecular traits, such as solubility or stability, without compromising the binding activity. In addition, as we show in this work, accurate paratope prediction can improve accuracy and speed of docking simulations, making structural models more reliable and easier to obtain (see Fig. 1 for an example).

In this work, we introduce the *Parapred* method for sequence-based prediction of paratope residues. Parapred improves on earlier methods for paratope prediction (Krawczyk *et al.*, 2013; Kunik *et al.*, 2012; Olimpieri *et al.*, 2013; Peng *et al.*, 2014; Tsuchiya and Mizuguchi, 2016) by using deep-learning methods and larger antibody datasets. In particular, we compared Parapred against the following two methods:

- **ProABC.** Olimpieri *et al.* (2013) designed a method for processing antibodies as amino acid residue sequences, obviating the need for a 3D structure. A random forest classifier consisting of 1500 trees was used to predict the probability of a residue being in the paratope. This is the most accurate paratope classifier up to date.
- **Antibody i-Patch.** Krawczyk *et al.* (2013) present a general protein–protein interaction algorithm adapted to cope with antibody–antigen binding specifics. The method computes binding statistics for different areas (patches) of a protein and uses this information to predict the likelihood of each amino acid residue participating in binding.

Our method only requires the amino acid sequence of a CDR and four adjacent residues as its input, which, in contrast to structural data, can be readily obtained experimentally. For simplicity, we only consider antigens that are themselves proteins, which are the vast majority of known antibody targets.

'Deep learning' specifically refers to the process of building machine learning models consisting of multiple layers of non-linear operations, where each successive layer automatically learns more abstract representations (*features*) of the data using the features extracted by the previous layer (Goodfellow *et al.*, 2016, p. 1). A key advantage of deep learning over traditional machine learning methods is that it can perform automated feature extraction directly from raw input data, thus eliminating the need for a domain expert to manually engineer features (Goodfellow *et al.*, 2016, p. 4). Automatically learned features are often found to be superior to manually engineered ones, contributing to the widespread success of deep learning in a range of fields. In particular, Parapred builds upon convolutional and recurrent neural networks (RNNs), which achieved state-of-the-art results in many difficult tasks, such as object recognition (He *et al.*, 2015). Deep learning has already been successfully applied to address problems in protein science, including the prediction of structure (Li *et al.*, 2016), function (Tavanaei *et al.*, 2016) or binding sites (Alipanahi *et al.*, 2015). To the best of our knowledge, this work is the first application of modern deep learning to antibody–antigen interactions.

2 System and methods

2.1 Data acquisition and preprocessing

To train and test our models, we used a subset of the Structural Antibody Database (SAbDab) (Dunbar *et al.*, 2014), which contains antibody and antigen crystal structures. Entries in SAbDab were filtered to obtain a non-redundant set of antibody–antigen complexes with the following properties: (i) antibodies have variable domains of the heavy (V_H) and light (V_L) chains, (ii) structure resolution is better than 3 Å, (iii) no two antibody sequences have >95% sequence identity and (iv) each antibody has at least five residues in contact with the antigen. The final dataset contains 277 bound complexes (Supplementary Material, Section A). Residues in the antibody sequence with missing electron density (i.e. non-resolved in the structure) were assumed to be non-binding, as missing electron density is typically associated with highly dynamic regions.

To construct the input, we identify the CDRs within the sequence of each antibody using the Chothia numbering scheme (Al-Lazikani *et al.*, 1997). We extend the CDR sequences with two extra residues at both ends, as these residues are also known to sometimes engage in binding (Krawczyk *et al.*, 2013; Kunik *et al.*, 2012). These extended CDR sequences are the input of the Parapred method and are processed *individually*. As our initial dataset of antibody–antigen complexes was assembled using a sequence identity cut-off applied to the full antibody sequence, we carried out a sequence identity analysis on the individual CDRs. The results in Supplementary Figure S1 (Supplementary Material, Section B) show that the sequence identity is always below 80% (median ≈ 40%) when comparing whole hypervariable regions and below 90% at the individual loop type level (median across loops of the same type ≈ 30–45%, median across all loop types <20%).

Amino acid sequences have to be encoded as tensors prior to being processed by the model (Fig. 2):

- Each amino acid sequence is encoded as a 'row' in a 3D matrix. As CDR sequences are usually of different length, during training

Downloaded from https://academic.oup.com/bioinformatics/article-abstract/34/17/2944/4972995 by University of Cambridge user on 23 September 2018

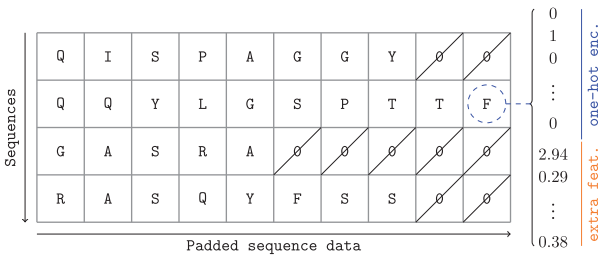


Fig. 2. An example of encoded amino acid sequences. An amino acid residue is represented by a feature vector which consists of one-hot encoding and some extra features. To efficiently process multiple sequences of different lengths during training, each sequence is padded to the length of the longest one. In general, Parapred can process arbitrarily long sequences

they are padded with zero vectors to the length of the longest sequence. This is necessary for fast batch tensor processing provided by deep-learning frameworks. Padding is not required when the model is used for prediction, as the architecture of our neural network does not impose an upper limit on the number of residues in the input sequence. This enables Parapred to process CDRs of arbitrary length, including ones longer than CDRs of sequences from the training set.

- Each element in a matrix encodes an amino acid residue and is itself a vector consisting of two concatenated parts:
  - *One-hot encoding* of the type of the residue (20 possible amino acid types + 1 extra, representing an unknown type). The type is encoded using a 21D vector, where all elements are set to 0 and one element, corresponding to the actual type of the amino acid, is set to 1.
  - Seven additional features, summarized by Meiler *et al.* (2001), which represent physical, chemical and structural properties of each type of amino acid residue (Supplementary Material, Section C).

The final dataset contains 1662 sequences for the algorithm to learn from (277 antibody/antigen complexes × 6 CDRs each).

2.2 Building a deep-learning model

The paratope prediction problem can be formalized as a binary classification problem between two classes of residues: those that do not participate in binding (Class 0) and those that do (Class 1). Following previous conventions (Krawczyk *et al.*, 2013), we define binding residues as those with at least one atom found within 4.5 Å of any of the antigen atoms. The algorithm will output the probability of binding (p) for each residue in the input CDR(s) plus two extra residues per side.

Our model uses several prominent architectural developments in deep learning.

2.2.1 Multilayer perceptrons

Neural networks can be thought of as a set of interconnected units, called *neurons* or perceptrons, each of which performs a simple computation.

Neurons are typically arranged in *layers*, where each neuron in a layer is connected to the output of every neuron in the previous layer. A layer with this kind of connection is called *fully connected*. The neural network itself is constructed as a series of such layers—the data are transformed in turn by every layer as it flows through the network. This architecture is known as a *deep feed-forward neural network* or a *multilayer perceptron* (MLP).

Neural network architectures are extensively used for machine learning tasks that can be reformulated as function approximation problems. We would like a network to learn to approximate some target function $f : X \to Y$ using a set of known input/output pairs for it (*supervised learning* setup). For example, for paratope prediction, $x \in X$ could be a vector encoding a residue and $y \in Y = \{0, 1\}$ could indicate whether the residue participates in binding.

The signals between neurons are real numbers and the neuron computes its output as follows:

- A neuron computes a weighted sum of its inputs ($\mathbf{x}$) and adds a constant term to it. The coefficients by which every input is scaled are called *weights* ($\mathbf{W}$) and the constant term is called the bias (b). The weights and bias constitute a set of adjustable *parameters* of a neuron.
- Some non-linear *activation function* σ is applied to the sum to produce the output. The activation function introduces a non-linearity necessary to model complex functions.

We can compactly write the transformation performed by all neurons in a layer as a single weight matrix multiplication and bias vector addition:

$$\mathbf{y} = \sigma(\mathbf{W}^{\mathrm{T}}\mathbf{x} + \mathbf{b}) \tag{1}$$

2.2.2 Recurrent neural networks

We can design a neural network which processes every element in a sequence in turn. The key idea behind RNNs is to iteratively apply a simple processing block, called RNN *cell*, to obtain a summarized representation of a sequence up to any point. Figure 3 shows a computation graph of an RNN—the cell iteratively consumes inputs ($\mathbf{x}$) by computing a function of $\mathbf{x}$ and the previous state of the cell s.

We use the *Long Short-Term Memory* (*LSTM*) (Hochreiter & Schmidhuber, 1997) cell which is able to learn long-range dependencies in sequences. The computation performed by an LSTM cell consists of the following steps:

- An LSTM cell holds the state s in two vectors: $\mathbf{C}$ ('memory') and $\mathbf{h}$ (previous output). Input $\mathbf{x}$ and state vector $\mathbf{h}$ are concatenated before being processed in four steps:

$$f_t = \sigma\left(\mathbf{W}_f^{\mathrm{T}}[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f\right) \tag{2}$$

$$C'_t = \tanh\left(\mathbf{W}_C^{\mathrm{T}}[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C\right) \tag{3}$$

$$i_t = \sigma\left(\mathbf{W}_i^{\mathrm{T}}[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right) \tag{4}$$

$$o_t = \sigma\left(\mathbf{W}_o^{\mathrm{T}}[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right) \tag{5}$$

where tanh is the element-wise hyperbolic tangent and σ is the logistic sigmoid function $\left(\sigma(x) = \frac{1}{1+\exp(-x)}\right)$. Matrices $\mathbf{W}$ and vectors $\mathbf{b}$ are parameters learned by the network.

- The new cell state C_t and h_t, as well as the output y_t is given by:

$$C_t = C_{t-1} * f_t + C'_t * i_t \tag{6}$$

$$y_t = h_t = \tanh(C_t) * o_t \tag{7}$$

where $*$ is the element-wise vector multiplication.

Capturing dependencies between an output and later inputs is necessary for amino acid sequences because they don't have a

Downloaded from https://academic.oup.com/bioinformatics/article-abstract/34/17/2944/4972995 by University of Cambridge user on 23 September 2018

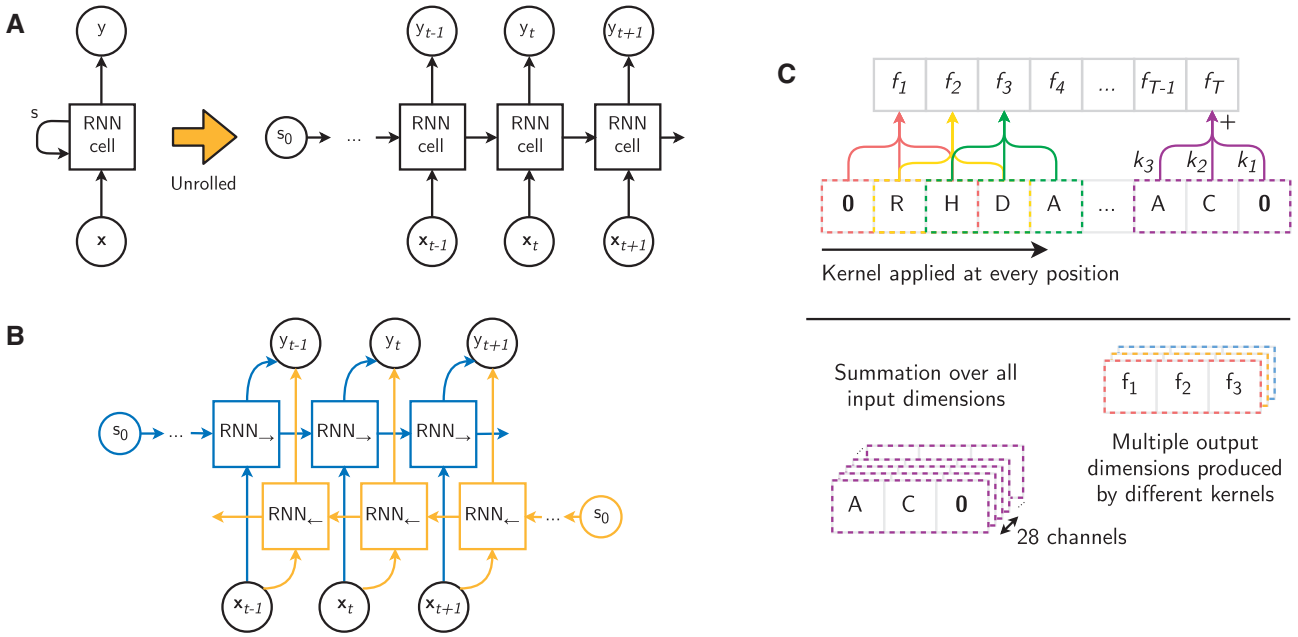


Fig. 3. Neural network constructs used in this work. (**A**) The computation graph of an RNN before (left) and after (right) unrolling. The same RNN cell is used to process every element of the input sequence. (**B**) Unrolled graph of a bidirectional RNN. The inputs are passed through two different RNN cells (one for each direction) and the network's output at time t is an aggregation (here—concatenation) of the two cells' outputs. (**C**) An example of 1D convolution with kernel size 3. Outputs are computed by applying a kernel at each position in the input sequence

canonical direction (reading a sequence left to right is equivalent to reading it right to left). To achieve this, we use a *bidirectional RNN* (Schuster and Paliwal, 1997) which introduces a second pass going in the opposite direction (see Fig. 3). RNNs enable the model to capture features which span the entire input sequence.

2.2.3 Convolutional neural networks

Amino acid residues are known to interact with other residues and prefer some kinds of amino acids more than others as their neighbours (Xia and Xie, 2002). A paratope prediction model can exploit such preferences by processing every residue together with its neighbourhood to learn useful local patterns first and only then use an RNN to learn aggregate features of the entire sequence.

Spatially local features can be extracted using *convolutional layers*, typically found in *convolutional neural networks* (*CNNs*).

A convolutional layer is similar to a single-layer MLP discussed previously, only it uses a convolution operation instead of matrix multiplication. A convolution operation for sequences is defined as:

$$f_t = \sum_{i=-K'}^{K'} \mathbf{k}_{K'+1+i} \cdot \mathbf{i}_{t-i} \tag{8}$$

where $\mathbf{i}_t$ and f_t are elements of the input and output sequences at position t, respectively, and $\mathbf{k} \in \mathbb{R}^{K \times C}$ is a *kernel* of size $K = 2K' + 1$ (w.l.o.g assume that the kernel has an odd number of elements; C refers to the dimensionality of the input). This computation is visualized in Figure 3.

The kernel is applied this way at every position of the input sequence to produce the output sequence. For positions where kernel spans beyond the input sequence, we assume the input is padded with zero vectors: $\mathbf{i}_t = 0$ for $t \leq 0$ or $t > T$. The input and kernel elements themselves are vectors with multiple *channels*—name comes from an analogy with images: each pixel in an image has three dimensions: red, green and blue channels—e.g. an encoded residue would have 28 dimensions/channels ($20 + 1$ amino acid type one-hot encoding + extra 7 features, as described earlier).

Convolution performs a weighted summation over all dimensions of input elements to produce a single number (the sum of vector dot products). The fact that the same small kernel is applied to every position in the input sequence allows it to detect input patterns regardless of their position. Learnable parameters of a convolutional layer are its kernels; multiple output channels can be produced by using several different kernels (*filters*).

2.2.4 Residual connections

Residual connections (He *et al.*, 2015) act as a shortcut connection between inputs and outputs of some part of a network by adding inputs to outputs. Such shortcut can be added around the convolutional feature extractor—if the local feature extractor is supposed to learn some function $h(\mathbf{x})$, with the shortcut connection it only has to learn the residual $h(\mathbf{x}) - \mathbf{x}$ which is often easier to optimize for. The shortcut also enables the rest of the model to learn both from original inputs and extracted local features and acts as a complexity controller by effectively allowing the network to adjust its depth.

2.2.5 Exponential linear units as activation functions

Activation functions introduce a non-linearity which is necessary to model complex functions. Experimenting with the activation function's behaviour can improve the training process. We use the *Exponential Linear Unit* (*ELU*) (Clevert *et al.*, 2015) activation function which makes the network more robust to noise and faster to train. The function is given by:

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x \geq 0, \\ \alpha(e^x - 1) & \text{if } x < 0. \end{cases} \tag{9}$$

We use $\alpha = 1$.

Downloaded from https://academic.oup.com/bioinformatics/article-abstract/34/17/2944/4972995 by University of Cambridge user on 23 September 2018

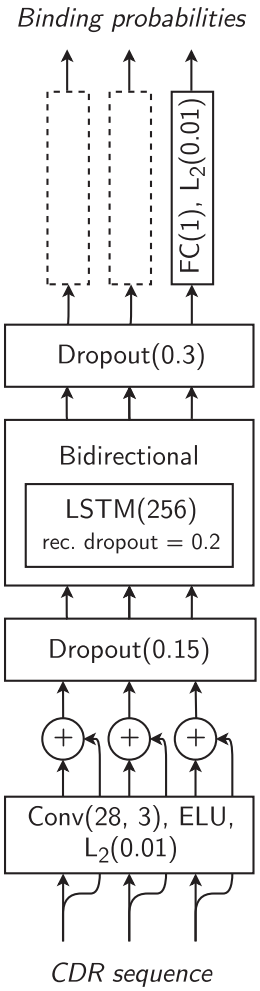


Fig. 4. The architecture of the Parapred method

2.2.6 Model regularization

Deep-learning models often have to be regularized to prevent overfitting—a phenomenon where a network memorizes training examples (and noise) instead of modelling the underlying relationship. We use two regularization methods:

- *Dropout* (Srivastava *et al.*, 2014) is a computationally efficient regularization method. The main idea of Dropout is to discard some intermediate results of the network at every training iteration with a certain probability p. This discourages the network from learning to rely on a particular subset of inputs.
- L_2 regularization (aka *weight decay*) adds an extra term—an L_2 norm of a layer's weights—to network's optimization objective, which penalizes weights if they grow too large during training.

3 Algorithm and implementation

The software was developed in Python using Keras deep-learning framework (Chollet *et al.*, 2015). Overall, the network's computation consists of the following steps (Fig. 4):

1. Encoded sequences (CDRs with two extra residues) are processed by a convolutional layer (regularized with an L_2 term scaled by 0.01) with 28 kernels, each spanning a neighbourhood of three residues. ELU activation is applied to the convolution results.
2. Residual connection is implemented by adding the original input sequences to the convolution output.
3. Resulting features are processed by a bidirectional LSTM with state size 256. The network applies Dropout with $p = 0.15$ to RNNs input and Dropout with $p = 0.2$ to RNNs recurrent connections.
4. Dropout with $p = 0.3$ is applied to the RNNs output and individual feature vectors are processed by a single-output fully connected network with logistic sigmoid activation function (to bring the output to the range of probabilities). Network's weights are regularized using an L_2 term scaled by 0.01.

The model's architecture could be easily augmented with layers that are able to process the 3D structure of an antibody in conjunction with its amino acid sequence. However, such sophisticated architectures would require a much larger training dataset (at least 10× more 3D structures) which is not available at this time. Training this kind of model would also require a way of efficiently exploiting cross-modality during feature extraction (Veličković *et al.*, 2016).

All architectural parameters (aka *hyperparameters*), such as LSTM state size, convolutional layer span, Dropout probabilities, etc. were chosen by evaluating network's performance on a special validation set (subset of the training set not shown to the network during training).

Neural network training is a function optimization problem, where we aim to find a local or global optimum of the optimization target (aka *loss*) with respect to network's parameters. This should be a differentiable measure of how well the neural network approximates the target function. We use the *binary cross-entropy* loss, a popular choice for binary classification problems:

$$\mathcal{L}(\Theta) = \frac{1}{m} \sum_{\mathbf{x}_i, y_i \in \mathrm{TrS}} w_i^s \left(-y_i \log \left(\widehat{f}_\Theta(\mathbf{x}_i) \right) - (1 - y_i) \log \left(1 - \widehat{f}_\Theta(\mathbf{x}_i) \right) \right) \tag{10}$$

where TrS is the training set of size m, $\widehat{f}_\Theta$ is the function computed by the network with parameters Θ and w_i^s is the sample weight (described later).

To find a loss minima, we use the *Adam* (Kingma and Ba, 2014) optimizer with base learning rate setting of 0.01 for the first 10 epochs and 0.001 otherwise. The network is trained with 32 samples at once (aka *batch size*) for 16 epochs (iterations over the entire training set).

The dataset has an uneven number of binding (positive) and non-binding (negative) residues—3.4× more negative samples. The cross-entropy loss function [Equation (10)] equally penalizes misclassified positive and negative samples, which allows the model to keep the overall loss low by preferring to predict that residues will not bind. This achieves good classification accuracy but hinders the model's ability to learn to identify positive samples. This can be improved by penalizing misclassified positive samples more—the per-sample loss is scaled by the *sample weight* w_i^s which we set to a 2.5× higher value for positive samples.

We made Parapred available as a webserver at http://www-mvsoftware.ch.cam.ac.uk/. For convenience, the online interface accepts full V_H and V_L amino acid sequences and uses ANARCI (Dunbar and Deane, 2015) to extract the CDRs.

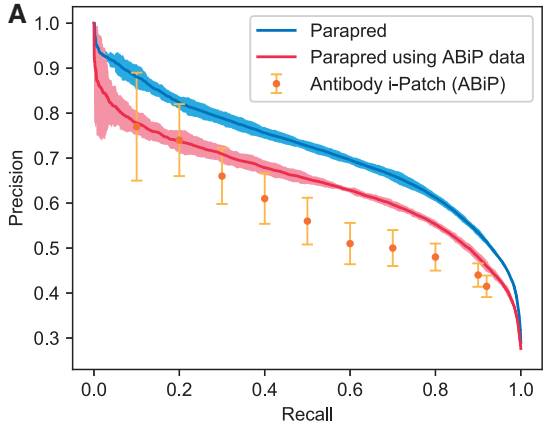
4 Results

4.1 Model results

To ensure an unbiased evaluation, the model has to be tested on data it has not seen during training. To obtain statistically significant results using our small dataset (277 complexes), we used the

Downloaded from https://academic.oup.com/bioinformatics/article-abstract/34/17/2944/4972995 by University of Cambridge user on 23 September 2018

A



B

Model	F-score	MCC	ROC AUC
Parapred	0.690 ± 0.006	0.554 ± 0.009	0.878 ± 0.004
proABC	—	0.522	0.851

C

Loop type	ROC AUC	Loop type	ROC AUC
H1	0.860 ± 0.004	*L1*	0.910 ± 0.005
H2	0.768 ± 0.013	*L2*	0.789 ± 0.010
H3	0.871 ± 0.006	*L3*	0.912 ± 0.006

Fig. 5. Performance characteristics of Parapred. (**A**) Precision-recall curves of Parapred when trained on our dataset and antibody i-Patch's dataset, together with the PR values reported by the authors of antibody i-Patch. Errors show 95% confidence bounds (2 SD). (**B**) Performance indicators of the Parapred with 95% confidence intervals (top row) and of the proABC method (bottom row). To convert predicted binding probabilities to binary labels, we used a threshold of 0.488 [obtained by maximizing Youden's index (Youden, 1950)]; the labels were used to compute the F-score and MCC metrics. (**C**) ROC AUC values of Parapred, separated by loop type. Errors show 95% confidence bounds

10-fold cross-validation technique to assess the model performance on multiple dataset splits. This technique randomly partitions the complexes into 10 subsets and trains the model 10 times. Each time, a different subset is chosen as the test set, and the model is trained from scratch on the complexes belonging to the other nine.

To measure the performance of the binary classifier, we use a number of standard metrics, such as MCC, F-score and precision-recall and ROC curves. Because evaluation results may vary due to the random partitioning of the data, the random initialization of the network parameters, and dropout, the cross-validation process itself is repeated ten times, which enables us to calculate confidence interval of the mean values of each performance indicator.

Figure 5A shows the precision/recall curve of the Parapred method which indicates a statistically significant improvement over Antibody i-Patch for recall values >0.1. This improvement is particularly relevant given that, in contrast to Parapred, Antibody i-Patch requires a structure or a homology model of the antibody and the antigen it binds to.

Figure 5B shows the F-score, MCC and ROC AUC performance metrics of Parapred. Narrow confidence intervals indicate consistent performance across cross-validation rounds. Furthermore, the results show that our model performs significantly better than the current state-of-the-art predictor, *proABC* (Olimpieri *et al.*, 2013) (both MCC and ROC AUC are statistically significantly better), without needing the entire antibody sequence or extra features such as the germline family or antigen volume.

We investigated to what extent the performance improvement originates from using a larger dataset (277 complexes versus 148 of Antibody i-Patch) and to what from the deep-learning-based architecture of Parapred. To assess this, we measured Parapred's performance when trained on the Antibody i-Patch's dataset (Fig. 5A). We find that our method achieves significant precision improvements for recall values >0.5, which is typically the most useful range. We conclude that the deep-learning-based architecture of Parapred is able to capture a richer set of features leading to better classification, even though it uses less explicit information about the antibody (Parapred does not require structural data or any information about the antigen it binds to). The leap in performance, observed when increasing the dataset size, is discussed in Supplementary Material, Section D and is in agreement with the observation that deep models thrive in environments with a larger number of more varied data points to learn from (Goodfellow *et al.*, 2016, p. 430).

Our encoding of an amino acid sequence does not include information about the CDR loop type it originated from, so the model may not be able to capture loop type-specific features. Figure 5C shows the ROC AUC values of our model's predictions, separated by CDR types. The data show that the model's performance varies slightly depending on the loop type; however, our initial attempts at including the loop type information made no appreciable difference to the performance (data not shown).

4.2 Docking improvements

We show the usefulness of Parapred by integrating its predictions with the PatchDock rigid protein docking algorithm (Duhovny *et al.*, 2002).

PatchDock works with two protein molecules in the PDB format and searches for suitable orientations for one of the molecules—conventionally, the antigen—'onto' the other. The algorithm produces several hundred candidate orientations of the antigen, called *decoys*, which are ranked in the output by an internal scoring function. The algorithm also provides facility to guide the search process by pre-specifying potential binding site residues.

Decoys can be classified into 4 quality classes—high (***), medium (**), low (*) or unclassed—based on how close the computed orientation of the antigen is to the true (*native*) orientation recorded in the dataset. The classification uses the CAPRI criteria (see Supplementary Material, Section E).

The usefulness of Parapred was measured by running PatchDock with three potential binding sites of the antibody molecule: (i) the full CDRs, (ii) the actual paratope and (iii) binding residues predicted by Parapred. We picked 30 antibody–antigen complexes at random (highlighted in Supplementary Material, Section A) to be run through the docking algorithm and, to ensure that the model is not tested on data it has been trained on, we used only the remaining structures as the training set for the model (247 structures). For a docking run with Parapred's predictions, residues were assumed to be binding if they scored above 0.67. This threshold was determined as the cut-off value that best recapitulates the total number of residues comprising the predicted binding site with that observed in the actual paratopes of antibodies within the aforementioned 247

Downloaded from https://academic.oup.com/bioinformatics/article-abstract/34/17/2944/4972995 by University of Cambridge user on 23 September 2018

Table 1. The number of high, medium and low quality decoys obtained by running PatchDock with different constraints on a test set of 30 structures

Binding site	Top 10			Top 200			Running time
	***	**	*	***	**	*	
CDRs	0	2	0	1	14	0	3 h 50 min 19.72 s
Paratope	0	7	1	1	21	3	2 h 02 min 22.50 s
Parapred	0	7	0	1	19	2	2 h 15 min 52.26 s
(1) versus (3) speedup							1.70×

structures. To avoid estimating this threshold from structures used in training, its value was determined using the test sets of an extra cross-validation run on just the 247 structures.

We recorded the best class decoy in the top 10 and top 200 decoys, as ranked by PatchDock, for each of the 30 structures. As a hint, we also supplied the antigen's binding region as residues within 5 Å of the real epitope. PatchDock was run with default parameters.

Docking results are shown in Table 1—supplying just the CDR gives the worst performance; however, supplying our model's predictions achieves performance comparable to that obtained when supplying the actual paratope. We conclude that for docking simulations Parapred's predictions are almost as informative as the actual paratope.

We also measured the time taken by PatchDock to produce decoys on a machine with an 'Intel(R) Core(TM) i7-6600U CPU @ 2.60 GHz' processor. We found that specifying Parapred's predictions as a potential binding site produces a 1.70× speedup in PatchDock's computations compared to specifying just the CDRs.

We also attempt to interpret local neighbourhood features learned by the model in Supplementary Material, Section F.

4.3 Concluding remarks

To the best of our knowledge, this work is the first application of modern deep learning (CNN- and RNN-based neural networks) to the paratope prediction problem. Our model is able to generalize using only antibody sequence stretches corresponding to the CDRs (with two extra residues on the either side) and improves on the current state-of-the-art by a statistically significant margin. We also showed that the model's predictions provide speed and quality gains for the PatchDock rigid docking algorithm—decoy quality and time-to-dock were comparable to those obtained when the docking algorithm has knowledge of the actual paratope as assessed from the complex crystal structure.

One of the main benefits of Parapred is that it does not rely on any higher-level antibody features: no full sequence, homology model, crystal structure or antigen information is required. We envisage that the Parapred method, which is freely available to the scientific community, will become a powerful tool in the growing fields of antibody engineering and computational design. In particular, when a bound structure of the antibody is not available, Parapred will enable the accurate identification of residues that are the most important in determining the antibody's activity, leaving other residue positions as available mutation sites, which can be exploited to engineer other essential molecular traits, such as stability or solubility.

Conflict of Interest: none declared.

References

Al-Lazikani,B. *et al.* (1997) Standard conformations for the canonical structures of immunoglobulins. *J. Mol. Biol.*, **273**, 927–948.

Alipanahi,B. *et al.* (2015) Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning (DeepBind). *Nat. Biotechnol.*, **33**, 831–838.

Chiu,M.L. and Gilliland,G.L. (2016) Engineering antibody therapeutics. *Curr. Opin. Struct. Biol.*, **38**, 163–173.

Chollet,F. *et al.* (2015). Keras. *GitHub*.

Clevert,D.A. *et al.* (2015). Fast and accurate deep network learning by exponential linear units (ELUs). arXiv preprint, arXiv: 1511.07289.

Duhovny,D. *et al.* (2002). Efficient unbound docking of rigid molecules. In: Guigó, R., Gusfield, D. (eds) *Algorithms in Bioinformatics. WABI 2002. Lecture Notes in Computer Science*, Vol. 2452. Springer, Berlin, Heidelberg.

Dunbar,J. and Deane,C.M. (2015) ANARCI: antigen receptor numbering and receptor classification. *Bioinformatics*, **32**, 298–300.

Dunbar,J. *et al.* (2014) SAbDab: the structural antibody database. *Nucleic Acids Res.*, **42**, D1140–D1146.

Ecker,D.M. *et al.* (2015) The therapeutic monoclonal antibody market. *MAbs*, **7**, 9–14.

Esmaielbeiki,R. *et al.* (2016) Progress and challenges in predicting protein interfaces. *Brief. Bioinform.*, **17**, 117–131. p

Goodfellow,I. *et al.* (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA.

He,K. *et al.* (2015). Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Vol. 5, pp. 6–14. IEEE, Las Vegas, NV, USA.

Hochreiter,S. and Schmidhuber,J. (1997) Long short-term memory. *Neural Comput.*, **9**, 1735–1780.

Kingma,D. and Ba,J. (2014). Adam: a method for stochastic optimization. *arXiv preprint*, arXiv: 1412.6980.

Krawczyk,K. *et al.* (2013) Antibody i-Patch prediction of the antibody binding site improves rigid local antibody–antigen docking. *Protein Eng. Des. Sel.*, **26**, 621–629.

Kunik,V. *et al.* (2012) Structural consensus among antibodies defines the antigen binding site. *PLoS Comput. Biol*, **8**, e1002388.

Leavy,O. (2010) Therapeutic antibodies: past, present and future. *Nat. Rev. Immunol.*, **10**, 297.

Li,R. *et al.* (2016). Deep convolutional neural networks for detecting secondary structures in protein density maps from cryo-electron microscopy. In: *2016 IEEE International Conference on Bioinformatics and Biomedicine* (*BIBM*), pp. 41–46. IEEE, Shenzhen, China.

Meiler,J. *et al.* (2001) Generation and evaluation of dimension-reduced amino acid parameter representations by artificial neural networks. *Mol. Model. Annu.*, **7**, 360–369.

Olimpieri,P.P. *et al.* (2013) Prediction of site-specific interactions in antibody-antigen complexes: the proABC method and server. *Bioinformatics*, **29**, 2285–2291.

Peng,H.P. *et al.* (2014) Origins of specificity and affinity in antibody–protein interactions. *Proc. Natl. Acad. Sci.*, **111**, E2656–E2665. p

Reichert,J.M. (2017) Antibodies to watch in 2017. *MAbs*, **9**, 167–181.

Schuster,M. and Paliwal,K.K. (1997) Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.*, **45**, 2673–2681.

Sormanni,P. *et al.* (2017) Rapid and accurate in silico solubility screening of a monoclonal antibody library. *Sci. Rep.*, **7**, 8200.

Srivastava,N. *et al.* (2014) Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, **15**, 1929–1958.

Tavanaei,A. *et al.* (2016). Towards recognition of protein function based on its structure using deep convolutional networks. In: *2016 IEEE International Conference on Bioinformatics and Biomedicine* (*BIBM*). pp. 145–149.

Tsuchiya,Y. and Mizuguchi,K. (2016) The diversity of H3 loops determines the antigen-binding tendencies of antibody CDR loops. *Protein Sci.*, **25**, 815–825.

Veličković,P. *et al.* (2016). X-CNN: Cross-modal convolutional neural networks for sparse datasets. In: *2016 IEEE Symposium Series on Computational Intelligence* (*SSCI*), pp. 1–8. IEEE, Athens, Greece.

Xia,X. and Xie,Z. (2002) Protein structure, neighbor effect, and a new index of amino acid dissimilarities. *Mol. Biol. Evol.*, **19**, 58–67.

Youden,W.J. (1950) Index for rating diagnostic tests. *Cancer*, **3**, 32–35.

Downloaded from https://academic.oup.com/bioinformatics/article-abstract/34/17/2944/4972995 by University of Cambridge user on 23 September 2018