UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'Informatique

Protein Residue Contact Prediction based on a Fully-Convolutional Neural Architecture

Antoine Passemiers

Promotor : Prof. Tom Lenaerts
Co-supervisor: Dr. Daniele Raimondi

Master Thesis in Computer Sciences

Academic year 2018 - 2019

TODO:

"*You may also include one or more general quotes related to your topic.*"

Name of the author, date

"*Another quote.*"

Name of the author, date

Acknowledgment

TODO:

I want to thank ...

Contents

Chapter 1

Introduction

1.1 Background and objectives of the thesis

Proteins are large macromolecules in the form of chains of building blocks called amino acid residues. There are 20 common amino acid types, but certain proteins may contain 2 additional amino acid types, namely pyrrolysine and selenocystein. According to Anfinsen's dogma, the three-dimensional structure of a protein is uniquely determined by its underlying amino acid sequence, at least when observed in protein's native environment. When moved from an unfavourable environment (where proper folding conditions are not met) to a solvent at neutral pH, a random coil (a sequence of amino acid residues oriented in random directions) will evolve towards the three-dimensional structure that minimizes Gibbs free energy. This process is called protein folding and has, however, a few exceptions.

Assuming the protein backbone has no structural restriction and is composed of n residues holded together by $n-1$ peptide bonds, then the protein has $2(n-1)$ bond angles that can be each in three different stable states. Therefore, there are at most $3^{2(n-1)}$ possible configurations, and it would take the age of the universe to find the correct folding by enumeration. There is strong evidence that protein folding is a NP-hard problem [29]. However, in practice small proteins are able to fold into a stable conformation in a fraction of a millisecond. This observation is known as the Levinthal's paradox. There has been a long standing perspective that protein folding is guided by heuristics composed of local interactions [43]. Heuristic folding leads to misfolded proteins that can potentially cause genetic diseases. Luckily, some proteins are assisted by molecular chaperones during their folding process [21] to attain their functional conformation. It must be noted that Anfinsen's observations of polypeptide chains refolding spontaneously in an aqueous medium have been made in the framework of in vitro studies: they do not take into account protein-protein interations and thus cannot generalize the self-assembly process well.

Protein Contact Prediction (PCP) can help determining the three-dimensional structure of proteins by limiting the search space to certain conformations constrained by predicted contact maps: this methodology is called contact-assisted protein folding. The problem of predicting the structure of a protein can be reduced to PCP because the latter is

a much simpler problem, and only a few correctly predicted contacts are sufficient to reconstruct the whole structure [39].

Structure-based methods are important in biology, as they help in assigning biochemical or biological functions to proteins. Indeed, the three-dimensional structure of a protein is more conserved than the underlying amino acid sequence across evolution. Prompted by this knowledge, similar functions can be assigned to proteins with low structural dissimilarity. Precisely identifying the role played by each protein in an organism is the first step towards understanding complex body mechanisms like muscle contraction, digestion or perceiving light. Also, determining the static structure of proteins help in detecting misfolded proteins which are possibly involved in diseases like Parkinson's or Alzheimer's, but also in diagnosing those diseases [24]. Finally, solving the protein folding (structure prediction) problem will enable to do better protein design, for example to engineer enzymes like PETase so they have faster plastic-degrading capabilities. There are pipelines designed to predict the structure and then the function of a newly observed protein, such as RaptorX server [52].

Protein structure is organized hierarchically: primary structure, secondary structure, tertiary structure and quaternary structure. Primary structure refers to the chemical composition of the protein, hence the sequence of amino acids present in it. Secondary structure indicates the presence of structures that are local to the amino acids themselves: these structures can generally be α-helices or β-sheets. Tertiary structure contains information about the three-dimensional structure of the protein and results from interactions between side chains of some pairs of amino acids, such as hydrogen bonds, ionic bonds or disulfide bridges. Quaternary structure is specific to proteins having multiple polypeptide chains and describes the structure due to intermolecular interactions between these chains. PCP helps predicting the tertiary structure since three-dimensional models can be reconstructed from protein contact maps (PCM). Also, PCM is a more simplistic and robust description of a protein's geometry because it is invariant to rotations and translations. This simplification helps making deep learning methods perform well on structure prediction.

Most PCP methods can be roughly divided into two categories: the ones based on Evolutionary Coupling Analysis (ECA) and the ones that infer contacts using supervised machine learning. In the former case, amino acid pairwise mutations are statistically modelled and the underlying model's parameters are generally optimized through log-likelihood maximization. In the second case, deep neural architectures are used to refine predictions made by low-level predictors such as ECA, in order to generate high-quality contact maps.

Ultimately, PCP should help making *ab initio* structure prediction. However, most recent methods rely on a whole raft of alignment and prediction tools. Given a protein encoded in FASTA format, ECA is only possible using a Multiple Sequence Alignment (MSA) of this target protein against homologuous proteins. These homologuous proteins come from the same protein family as the target protein, and therefore the most suitable family must be found. This can be done by matching the target sequence to a Hidden Markov Model (HMM) profile representing a family like in Pfam database [20]. Once the homologuous sequences have been retrieved, they have to be aligned to the target sequence using an MSA tool like HHblits or HMMER. In the next step, evolutionary couplings are extracted from the MSA using an ECA predictor like PSICOV [36] or

plmDCA [18]. Eventually, predictions are gathered and refined using a deep neural architecture, necessitating the use of a deep learning framework. These successive layers of dependencies are not making PCP a straightforward process. Therefore, it seems to be a natural choice to set as an objective for this thesis the development of a predictor with minimal requirements and performance close to state-of-the-art techniques [68, 44, 37, 47, 23].

1.2 Structure of the thesis

Let's describe the global view of the thesis itself. Firstly, common state-of-the-art ECA techniques will be described, such as Direct Coupling Analysis (DCA) and Pseudo-Inverse Covariance matrices (PSICOV) (both are statistical methods based on graphical models), as well as the basics of deep learning and backpropagation algorithm. In order to gain a deeper insight on best-performing methods, more details will be given for some specific deep learning methods, such as their architecture, input features and preprocessing. The generic architecture in use for this thesis will be detailed, as well as the hyper-parameter optimization procedure used for cross-validation. Finally, results will be presented in multiple sections:

- Since the proposed deep learning approach relies on DCA predictions as input features, the first step should demonstrate that deep learning is capable of refining contact maps by looking at complex visual patterns that cannot captured by linear models.
- It should be brought to light whether deep learning's performance is less sensitive to the effective number of homologous sequences than DCA methods. The notion of effective number of homologous sequences will be introduced in section 2.4.9.1.2.
- Finally, a benchmark will be established to assess the performance of the proposed method in comparison with other supervised approaches, including state-of-the-art deep learning architectures.

1.3 Contributions

Chapter 2

State-of-the-art

2.1 Protein contact maps

2.1.1 Definition

A **contact** between two residues occurs when two amino acid residues from a same protein are separated by a distance below a given threshold. The distance metric can be either the distance between $C_\alpha - C_\alpha$ atoms or the distance between $C_\beta - C_\beta$ atoms. It should be underlined that, in the case where the first residue is Glycine, $C_\alpha - C_\beta$ is used instead of $C_\beta - C_\beta$. C_α is the first carbon atom attached to a functional group and C_β is the first carbon atom attached to C_α. Functional groups of amino acid residues can be either amine ($-NH_2$) or carboxyl ($-COOH$). In the first case, the amino acid is called alpha amino acid and has an amine group directly attached to the C_α of the carboxyl group. In the second case, it is called beta amino acid and has an amine group attached to the C_β of the carboxyl group.

Most common distance thresholds range between 6 and 12 Å. An angstrom (Å) is an unit of length equivalent to 10^{-10} m, or 10^{-1} nm. Therefore the notion of residue contact depends to a large extent on the threshold used. For example, the average percentage of contacts in the 150 proteins reported in the original PSICOV article [36] is equal to 7%, 14%, 26%, 39% with thresholds 7, 10, 13 and 16 Å, respectively.

By extension, a **protein contact map** can be defined as a symmetric binary matrix C where element C_{ij} is equal to 1 if residues i and j are separated by a distance below the given threshold, and 0 otherwise. Contact maps are invariant to rotations and easier to predict with machine learning methods, contrary to matrices of pairwise distances. Furthermore, the original 3D residue coordinates can be recovered from contact maps [65]. Anfinsen's Dogma postulates that the secondary and tertiary structures of a protein can be inferred from its primary structure: even after disrupting the hydrophobic bonds of a protein, experiments have shown that the latter can recover its original structure with some assisted folding, highlighting the idea that tertiary structure is encoded in the sequence of amino acids itself.

2.1.2 An alternative representation: protein contact networks

Another way of interpreting contact maps is viewing them as adjacency maps of protein contact networks. This sub-section will present the formal definition of protein contact networks as suggested in [14]. Let $G = (V, E)$ be a graph where V is the set of vertices and E the set of edges. Such a graph G can be encoded as a matrix A called the adjacency matrix. Given a set of vertices $\{v_1, \ldots, v_n\}$, adjacency matrix $A \in \{0, 1\}^{n \times n}$ is such that:

$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Using this definition, many adjacency matrices of a same graph exist. Indeed, many matrices can be created by simply making permutations of rows and columns. However, the ordering of vertices is determined by the sequence of amino acids, making it unique. Weighted graphs are slightly different than regular graphs since they are defined not only by their connections but also their weights. Accordingly, the adjacency matrix of a weighted graph is adapted as follows:

$$A_{ij} = \begin{cases} w_{ij} & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

where w_{ij} is the weight of edge (v_i, v_j).

Also, the degree k_i of a vertex v_i is defined as the number of neighbouring vertices, or in other words the number of vertices each sharing an edge with v_i:

$$k_i = \sum_{j=1}^{n} A_{ij} \tag{2.3}$$

This definition of vertex degree also holds for weighted graphs. However, many authors favor minimal representation of protein structure and abandon the use of weights. The diagonal degree matrix D can be defined by the following relation:

$$D_{ij} = \begin{cases} k_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

A **protein contact network** is a graph where the set of vertices is ordered by the primary structure, each vertex is an amino acid itself, and the presence of an edge between two vertices indicates that the two corresponding residues are in contact. Such a network is useful to make a compact representation of a protein structure and metrics such as path length or graph diameter are important for the analysis of long-range residue interactions [14].

Let sp_{v_1,v_2} be the number of vertices located on the shortest path from v_1 to v_2, called the distance between v_1 and v_2. The diameter of a graph $G = (V, E)$ is defined as follows:

$$\text{diam}(G) = \max\{sp_{v_1,v_2} | v_1, v_2 \in V\} \tag{2.5}$$

TODO: Continue with [14]

[45] actual PCNs elaborated from the E. coli proteome Synthetic networks: 1) The effect of backbone on the small-world properties of protein contact maps 2) Exploring community structure in biological networks with random graphs

edges are added among two residues if their Euclidean distance is within the [4, 8] Å range

TODO: Plot: Density of Euclidean distances among native contacts in PCNs

2.2 Direct Coupling Analysis

2.2.1 Potts model

It has long been stated that the three-dimensional structure of proteins has an impact regarding the amino acid composition among homologous proteins. Spatial contacts between residues are related to the variability of amino acid types at fixed pairs of positions. The core idea in Direct Coupling Analysis (DCA) is to disentangle direct correlations and correlations produced at intermediary positions. Potts model allows to perform this disentanglement through inverse statistical mechanics [19].

In Potts model for PCP, evolutionary-related sequences are modelled by the distribution given by the Maximum-Entropy Principle. Among the family of distributions that are suited for proteins (which are discrete sequences), the one that maximizes entropy is the Boltzmann distribution, with the following probability mass function:

$$P(s|J,h) = \frac{1}{Z}\exp\Big(\sum_{i=1}^{L}\sum_{j=i+1}^{L} J_{ij}(s_i,s_j) + \sum_{i=1}^{L} h_i(s_i)\Big) \tag{2.6}$$

where J and h are sets of model parameters, s is an amino acid sequence and Z is a normalization factor called partition function ensuring that the sum $\sum_s P(s|J,h)$ over all lexicographically possible sequences is equal to one.

2.2.2 Exact inference is hard

Given a multiple sequence alignment containing M sequences, a naïve approach would be to maximize its log-likelihood:

$$\begin{aligned}
\log L(J,h|s^{(1)},\dots,s^{(M)}) &= \sum_{k=1}^{M} \log P(s^{(k)}|J,h) \\
&= \sum_{k=1}^{M} \log\left(\frac{1}{Z}\exp\Big(\sum_{i=1}^{L}\sum_{j=i+1}^{L} J_{ij}(s_i^{(k)},s_j^{(k)}) + \sum_{i=1}^{L} h_i(s_i^{(k)})\Big)\right) \\
&= -M\log(Z) + \sum_{k=1}^{M}\Big(\sum_{i=1}^{L}\sum_{j=i+1}^{L} J_{ij}(s_i^{(k)},s_j^{(k)}) + \sum_{i=1}^{L} h_i(s_i^{(k)})\Big)
\end{aligned} \tag{2.7}$$

with the following partial derivatives:

$$\begin{aligned}
\frac{\partial}{\partial J_{ij}(a,b)}\log L(J,h|s^{(1)},\dots,s^{(M)}) &= -M\frac{\partial\log(Z)}{\partial J_{ij}(a,b)} + \sum_{k=1}^{M}\delta\Big(s_i^{(k)},a\Big)\delta\Big(s_j^{(k)},b\Big) \\
\frac{\partial}{\partial h_i(a)}\log L(J,h|s^{(1)},\dots,s^{(M)}) &= -M\frac{\partial\log(Z)}{\partial h_i(a)} + \sum_{k=1}^{M}\delta\Big(s_i^{(k)},a\Big)
\end{aligned} \tag{2.8}$$

where $\delta(a, b)$ denotes the Kronecker delta.

However, there is no straightforward method to compute the partition function Z or its gradient for large systems due to the discrete nature of amino acid sequences. Indeed, Z contains 21^L terms for systems with 21 possible symbols (amino acid types + gap) and sequences of length L. For this aim, several methods like Mean-Field (mfDCA) [49], Message Passing (mpDCA) [70], Pseudo-Likelihood Maximization (plmDCA) [18] or Multivariate Gaussian Modeling (GaussDCA) [4] have been developed.

2.2.3 Pseudo-Likelihood Maximization

plmDCA [18] addresses the problem of estimating the partition function by maximizing the pseudo-loglikelihood instead of the loglikelihood. The pseudo-loglikelihood can be expressed as the sum of loglikelihoods $\log L(J_r, h_r)$, where each $\log L(J_r, h_r)$ is computed at a single site r. The method thus assumes the conditional independence between variables belonging to different sites. However, the partition function at a given site can then be easily computed as a sum of 21 terms since a state can take 21 possible values at a given position. More formally, the penalized loglikelihood at site r is given by:

$$\begin{aligned} \log L^{(reg)}(J_r, h_r) = &- \frac{1}{M_{eff}} \sum_{k=1}^{M} w_k \left(h_r(s_r^{(k)}) + \sum_{i \neq k}^{L} J_{ri}(s_r^{(k)}, s_i^{(k)}) - \log(Z_r) \right) \\ &+ \lambda_h \left\| h_r \right\|_2^2 + \lambda_J \left\| J_r \right\|_2^2 \\ \text{where} \qquad Z_r = &\sum_{a=1}^{q} \exp\Big(h_r(a) + \sum_{i \neq r} J_{ri}(a, s_i^{(k)}) \Big) \end{aligned} \tag{2.9}$$

It can be observed that the formula contains a L_2 penalty term for both fields and couplings, and that each log-probability is weighted by a protein weight w_k. The latter is computed as the protein contribution to set the set of effective sequences, as described in the section of M_{eff} **??**. It must be observed that the optimization procedure is called asymmetric pseudolikelihood maximization because each matrix $J(a, b)$ is supposed to be symmetric but is in practice independently estimated. This allows plmDCA to run in parallel by optimizing $\log L^{(reg)}(J_r, h_r)$ each on a different core.

After maximizing the pseudo-loglikehood in parallel, all remaining information that can be explained by the fields are removed from the couplings by applying an average sum correction w.r.t. the states:

$$\hat{J}_{ij}(a, b) = J_{ij}(a, b) - \frac{1}{q} \sum_{a=1}^{q} J_{ij}(a, b) - \frac{1}{q} \sum_{b=1}^{q} J_{ij}(a, b) + \frac{1}{q^2} \sum_{a=1}^{q} \sum_{b=1}^{q} J_{ij}(a, b) \tag{2.10}$$

Then each matrix $J(a, b)$ is symmetrized by simply averaging it with its transpose:

$$\hat{J}(a, b) \leftarrow \frac{1}{2} \big(\hat{J}(a, b) + \hat{J}(a, b)^T \big) \quad \forall a, b \tag{2.11}$$

Finally, a contact map is predicted by norming $\hat{J}$ over the states and applying an average product correction w.r.t. the sites. plmDCA shows remarkable performance on diverse sets of proteins with running times competitive with mean-field DCA.

2.2.4 Gaussian Direct Coupling Analysis

plmDCA is of state-of-the-art performance, but still requires high computational resources. An alternative method is to use GaussDCA [4], which does exact inference wihout having recourse to iterative algorithms.

In GaussDCA, each variable $x_i \in \{0,1\}$ indicates whether residue located at locus $i\%L \in \{1,\dots,L\}$ is of amino acid type $i/L \in \{1,\dots,22\}$. With such a formalism, each protein is described as a vector $x \in \{0,1\}^{22\,L}$. The key assumption at the core of the method is to approximate each binary variable x_i by a continuous Gaussian variable. Let m be the number of sequences in a MSA, $X \in \{0,1\}^{m\times 22\,L}$ be the matrix representation of the MSA, and $\mu, \bar{x}$ be respectively, the theoretical and empirical mean vectors associated to X. The empirical covariance matrix of X is given by:

$$\bar{C}_{ij}(X,\mu) \triangleq C_{ij}(X,\mu) = C_{ij}(X,\bar{x}) = \frac{1}{m}\sum_{k=1}^{m}(x_i^k - \mu_i)(x_j^k - \mu_j) \tag{2.12}$$

Similarly to PSICOV [36], evolutionary couplings are detected by keeping track of direct interactions between variables of the system, which can be realized by computing the precision matrix, also known as the inverse of the covariance matrix. When matrix $\bar{C}$ is not rank deficient, maximum log-likelihood is attained by setting the theoretical covariance matrix Σ to $\bar{C}$. However, the empirical covariance matrix rarely has full rank due to the limited number of effective sequences in MSAs. The suggested solution is to provide a prior distribution on positive-definite matrices and perform exact Bayesian inference to find an invertible covariance matrix.

2.2.4.1 Bayesian inference

In Bayesian inference, a hypothesis is favoured over others based on its posterior probability, which is proportional to the product of the data log-likelihood under this hypothesis and its prior probability. This relation holds in the parametric formulation of Bayes' rule:

$$P(\theta|X,\alpha) = \frac{P(X,\theta|\alpha)P(\theta|\alpha)}{P(X|\alpha)} \propto P(X,\theta|\alpha)P(\theta|\alpha) \tag{2.13}$$

$P(\theta|\alpha)$ is a prior distribution, hence a distribution over the parameter space without any knowledge about the data X. As more and more data becomes available, the newly observed samples can be incorporated to the model through the likelihood $P(X,\theta|\alpha)$. The likelihood measures how strongly the data is explained by the model, given a set of parameter values. $P(X|\alpha)$ is called evidence or marginal likelihood because it is equal to the data likelihood marginalized over the parameters θ. $P(\theta|X,\alpha)$ is the posterior distribution, giving the probability of some parameters given the observed data.

The marginal likelihood is a constant term w.r.t. the hyper-parameters α. Therefore, the maximum a posteriori estimation (MAP) is simply the maximum product between the likelihood and the prior.

2.2.4.2 Prior distribution

A reasonable choice for the prior distribution over μ and Σ is the normal-inverse-Wishart (NIW) distribution, which is known to be conjugate prior for the Gaussian log-likelihood. The prior and the posterior are said to be conjugate if they are of the same form. In the case of NIW prior, the posterior is also a NIW distribution. The prior is defined as the product $P(\mu, \Sigma) = P(\mu|\Sigma)\, P(\Sigma)$, where the prior of the mean vector is defined as a multivariate Gaussian distribution:

$$P(\mu|\Sigma) = \Big(\frac{2\pi}{\kappa}\Big)^{-\frac{m}{2}} |\Sigma|^{-\frac{1}{2}} \exp\Big(\frac{\kappa}{2}(\mu-\eta)^T\Sigma^{-1}(\mu-\eta)\Big) \tag{2.14}$$

Let's note that the Gaussian distribution is conjugate to itself. The prior over positive-definite matrices (we are interested in invertible covariance matrices exclusively) is defined by the NIW distribution:

$$\begin{aligned} P(\Sigma) &= \frac{1}{Z}|\Sigma|^{-\frac{\nu+m+1}{2}} \exp\Big(-\frac{1}{2}\operatorname{Tr}\Lambda\Sigma^{-1}\Big) \\ \text{where} \qquad Z &= 2^{\frac{\nu m}{2}}\pi^{\frac{m(m-1)}{4}}|\Lambda|^{-\frac{\nu}{2}}\prod_{k=1}^{m}\Gamma\Big(\frac{\nu+1-k}{2}\Big) \end{aligned} \tag{2.15}$$

where Γ is Euler's Gamma function, ν is the degree of freedom and Λ is a parameter matrix called scale matrix.

2.2.4.3 Computing the MAP covariance matrix

The mean values for the NIW prior are known and equal to η and $\Lambda/(\nu-m-1)$, respectively. Assuming η', ν' and Λ' are the corresponding parameters in the NIW posterior, the mean values are given by η' and $\Lambda/(\nu'-m-1)$, respectively. In particular, the matrix Λ' that allows the posterior to be conjugate with the prior is given by the following relation:

$$\Lambda' = \Lambda + n\bar{C} + \frac{\kappa m}{\kappa+m}(\bar{x}-\eta)(\bar{x}-\eta)^T \tag{2.16}$$

Finally, the average covariance matrix is computed as:

$$\begin{aligned} \Sigma &= \frac{\Lambda'}{\nu'-m-1} \\ &= \frac{\Lambda + n\hat{C} + \frac{\kappa n}{\kappa+n}(\hat{x}-\eta)^T(\bar{x}-\eta)}{\nu+n-m-1} \\ &= \lambda\Lambda + (1-\lambda)\bar{C} + \lambda(1-\lambda)(\bar{x}-\eta)^T(\bar{x}-\eta) \end{aligned}$$

As a viable choice for the hyper-parameter matrix Λ, one could choose the less arbitrary one. To this aim, the covariance matrix corresponding to a uniform multivariate distribution has been selected.

2.3 PSICOV

2.3.1 Gaussian graphical models

PSICOV relies on a Gaussian graphical modelization. Such models are represented by undirected graphs that satisfy the pairwise Markov property: two nodes are not connected by an edge if they are conditionally independent. Let $G = (V, E)$ be a graph and let each node k be associated to a random variable X_k. Conditional independence between variables X_u and X_v is formally defined as follows:

$$X_u \perp\!\!\!\perp X_v | X_{V\{u,v\}} \text{ if } \{u, v\} \notin E \tag{2.17}$$

If the graph is not complete, then there exists pairs of conditionally independent variables in the model, resulting in sparsity patterns in the underlying precision matrix. When the covariance matrix of the multivariate Gaussian model is rank deficient, a solution for the precision matrix has to be found with a high prior towards sparse matrices. Such a solution can be achieved through regularization. The solution that follows in the next section relies on pseudo-inverse computation with L_1 regularization.

2.3.2 Evolutionary couplings

The motivation behind PSICOV [36] is that residue contacts produce constraints on the types of residues in the protein at certain pairs of sites: two residues involved in a contact should have complementary physicochemical properties. To capture the covariation of residue types for any pair of sites, the random variable $X_{ia} : \Omega \to \{0, 1\}$ is defined. X_{ia} takes value one if amino acid at site i is of type a. In particular, value $x_{ia}^{(k)}$ is equal to one if the kth sequence in the given MSA has a residue of type a at site i. With this knowledge, the sample covariance matrix $S \in \mathbb{R}^{21L \times 21L}$ can then be computed by:

$$S_{ij}^{ab} = \frac{1}{n} \sum_{k=1}^{n} \left(x_{ia}^{(k)} - \bar{x}_{ia}\right)\left(x_{jb}^{(k)} - \bar{x}_{bj}\right) \tag{2.18}$$

where n is the number of homologous sequences and S_{ij}^{ab} is the sequence identity covariance from amino acid a to amino acid b and from site i to site j. Each matrix S_{ab} is thus a covariance matrix between sites i and j. It is noteworthy that such a covariance matrix is neither positive semidefinite nor symmetric and is, thus, substantially different from regular sample covariance matrices. Precision matrix Θ_{ij} is defined as the inverse of covariance submatrix S_{ij}, but there is no guarantee that such an inverse can be computed directly. Indeed, each submatrix S_{ij} is very likely to be singular due to the absence of correlations for some residue types that are rarely (or never) observed at given sites.

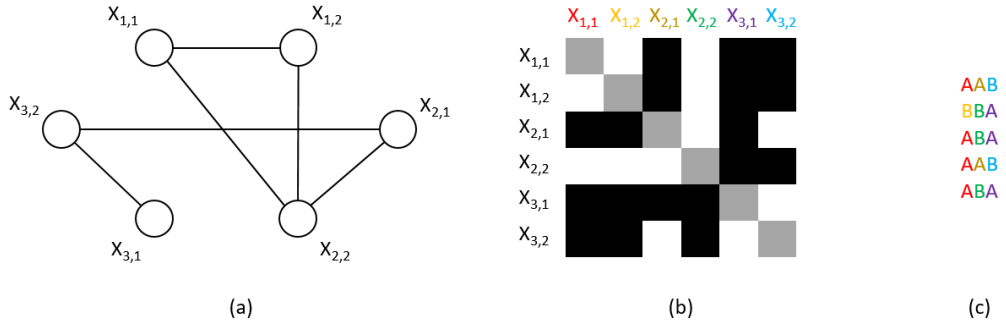

Figure 2.1: Illustration of a small system with only 3 sites and 2 amino acid types A and B: (a) Gaussian graphical model representing the system. (b) Sparsity pattern of the estimated inverse covariance matrix. Cells associated to zero variables are drawn in black. (c) Corresponding MSA.

2.3.3 Inference

The solution chosen here is to estimate sparse inverse covariance matrices instead, by having recourse to Graphical Lasso algorithm [25]. The method assumes that observations follow a multivariate normal distribution characterized by the following probability density function:

$$f(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\Big(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\Big) \tag{2.19}$$

where $d = 21L$ is the number of components in vector x, μ is the theoretical mean and Σ the theoretical covariance matrix. Let's express the log-likelihood of the data as a function of the inverse theoretical matrix Θ:

$$\begin{aligned}
\log L(\Theta) &= \sum_{k=1}^{n} \log f(x^{(k)}) \\
&= \sum_{k=1}^{n} \left(-\frac{1}{2}(x^{(k)}-\mu)^T \Sigma^{-1}(x^{(k)}-\mu) - \log\sqrt{(2\pi)^d |\Sigma|} \right) \\
&= \sum_{k=1}^{n} \left(-\frac{1}{2}(x^{(k)}-\mu)^T \Theta(x^{(k)}-\mu) - \log\sqrt{(2\pi)^d \frac{1}{|\Theta|}} \right) \\
&= -\frac{1}{2}\sum_{k=1}^{n}(x^{(k)}-\mu)^T \Theta(x^{(k)}-\mu) - \frac{1}{2}\sum_{k=1}^{n} \log\left((2\pi)^d \frac{1}{|\Theta|}\right) \\
&= -\frac{1}{2}\operatorname{Tr}\left(S\Theta\right) - \frac{1}{2}\sum_{k=1}^{n} \log(2\pi)^d + \frac{1}{2}\log|\Theta|
\end{aligned} \tag{2.20}$$

The latter expression holds because the empirical mean $\bar{x}$ is equal to μ for any Σ. The

objective function of Graphical Lasso is simply the log-likelihood penalized with L_1 norm. L_1 regularization is used instead of L_2 because of its non-asymptotic behaviour and therefore its ability to produce more zeroes among the parameter values. The solution to the optimization problem is described as:

$$\begin{aligned}\hat{\Theta} &= \underset{\Theta}{\text{argmax}} \quad \log L(\Theta) - \rho \left\|\Theta\right\|_1 \\ &= \underset{\Theta}{\text{argmax}} \quad -\frac{1}{2}\operatorname{Tr}\left(S\Theta\right) - \frac{1}{2}\sum_{k=1}^{n}\log\left(2\pi\right)^d + \frac{1}{2}\log\left|\Theta\right| - \rho\left\|\Theta\right\|_1 \\ &= \underset{\Theta}{\text{argmax}} \quad -\operatorname{Tr}\left(S\Theta\right) + \log\left|\Theta\right| - \rho\left\|\Theta\right\|_1\end{aligned} \tag{2.21}$$

The L_1 norm $\left\|\Theta\right\|_1$ is the sum of absolute values of the elements in Θ. TODO:

TODO: Optimizer, GLassoFast, etc.

TODO: APC

According to Sun et al. [**?**], the solutions found by PSICOV may not be suitable for some proteins. Furthermore, the best solution could be far from the global optimum found by Graphical Lasso since the search space is huge. The suggested solution is to predict multiple contact maps and promote diversity among them by adding a new penalty term.

$$\begin{aligned}&\min_{\Theta^{(m+1)}} \frac{1}{2}\operatorname{Tr}\left(S\Theta\right) + \frac{1}{2}\sum_{k=1}^{n}\log\left(2\pi\right)^d - \frac{1}{2}\log\left|\Theta\right| \\ &s.t. \quad d(\Theta^{(k)}, \Theta^{(m+1)}) \geq \epsilon, \quad k = 1, \ldots, m\end{aligned} \tag{2.22}$$

In that last equation, the distance constraint function d is a convex and differentiable function defined as:

$$d(\Theta^{(k)}, \Theta) = -\sum_{i,j}\delta_0(\theta^{(k)})_{i,j}|\Theta_{i,j}| \tag{2.23}$$

where $\delta_0 : \mathbb{R}^{21L\times 21L} \to \mathbb{R}^{L\times L}$ is such that $d(\Theta_{i,j})$ takes value 1 if submatrix $\Theta_{i,j}$ is 0, and -1 otherwise. The objective function is optimized using a second-order method and updating the solution in the Newton direction at each iteration. Finally, a contact map is obtained by selecting the submatrices among the solutions according to their sparsity. More specifically, submatrices are selected according to their nuclear norm, hence the sum of their singular values.

2.4 Deep learning

2.4.1 A definition of *deep learning*

Behind the trendy words, it is quite difficult to find a consensus on the definition of *deep learning*. According to many, the process of learning deeply can only be achieved by deep neural networks. A deep artificial neural network is composed of a set of parameters and a large stack (or graph) of mathematical operators designed to minimize an objective function. Each operation may be dependent on a subset of the network parameters. In most cases, the network can be described as a stack of operators. The objective function is then a composition of all mathematical operations. Such a network is usually trained using the backpropagation algorithm. The method consists in minimizing the objective function which is usually an average distance between what the network predicts on the basis of an input and what the human supervisor expects. More specifically, it is an iterative algorithm that evaluates the gradient of the objective at each iteration and performs one step in the direction of the steepest descent in the parameter space. The algorithm is expected to stop once a global minimum has been reached.

Deep learning is also often viewed as the capacity of a machine to create a hierarchical modelling of the data. This implies that the model can transform the data to high-level feature maps. According to Yoshua Bengio and Yann LeCun, neural networks only exemplify the notion of deep architectures. They provided a sufficiently good basis for a definition:

> Deep architectures are compositions of many layers of adaptive non-linear components, in other words, they are cascades of parameterized non-linear modules that contain trainable parameters at all levels. Deep architectures allow the representation of wide families of functions in a more compact form than shallow architectures, because they can trade space for time (or breadth for depth) while making the time-space product smaller, as discussed below. The outputs of the intermediate layers are akin to intermediate results on the way to computing the final output. Features produced by the lower layers represent lower-level abstractions, that are combined to form high-level features at the next layer, representing higher-level abstractions [6].

This definition seems to be suitable for stacked models: one can design a cascade of decision tree modules since decision trees are able to construct non-linear decision boundaries. The problem rather lies in determining if there exists a natural way to make them extract high-level features from data. Apart from that, the backpropagation algorithm for neural networks is going to be introduced in more details.

2.4.2 The backpropagation algorithm

A description of backpropagation has to be made in order to understand how modern deep learning works. Let's consider a feedforward neural network containing no cycle. Each of its layers can be viewed as a couple $(f_i(\theta_i, X), b_i(\theta_i, dX))$, where f_i is the forward pass function (prediction function) of layer i, b_i is the backward pass function, θ_i is the set of parameters, and X, Y are input tensors of shapes compatible with f_i and g_i,

respectively, and dX is the signal tensor propagated from next layer to current layer. Let's make the assumption that convolutional layers are two-dimensional and that input instances are image-like data. Also, let's consider a particular case of neural network consisting of a stack of neural layers instead of a graph: a feedforward neural network. Also, let b be the number of instances in the input tensor (more commonly referred to as the batch size), w and h respectively the width and height of the images, and c the number of channels. Finally, let n be the number of layers and m be the number of output neurons in the network. Knowing this, we are able to express the output $Y \in \mathbb{R}^{b \times m}$ of the network as such:

$$Y = (\bigcirc_{i=1}^{n} f_{\theta_i})(X) \tag{2.24}$$

where $X \in \mathbb{R}^{b \times w \times h \times c}$ and $f_{\theta_i}(X)$ is a more convenient notation for $f_i(\theta_i, X)$. We observe that the prediction function of the network is basically a large composition of functions.

Such model is designed to optimize a function reflecting its ability to accurately predict a target value or to abstractly represent the input data in a more general sense. Accordingly, let's introduce a generic loss function $L(Y) : \mathbb{R}^{b \times m} \to \mathbb{R}$ that measures the model's inability to fulfill the given task. Again, the loss function can be rewritten as a composition of the layer forward passes and the loss function, and the objective is to find the set of parameters that minimizes the loss function:

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}}\, \ell((\bigcirc_{i=1}^{n} f_{i,\theta_i})(X)) \tag{2.25}$$

where Θ is the set of all possible values for the parameter set $\theta = (\theta_1, \ldots, \theta_n)$. Since we are considering a loss function, the latter must be minimized. The generic task of minimizing a scalar continuous function can be achieved using numerous continuous optimization techniques among gradient descent algorithms [57] or quasi-Newton methods [9]. In practice, gradient descent approaches require more iterations to converge to a satisfying solution, but are much easier to implement. Also, contrary to quasi-Newton methods, they don't require to implicitly compute the hessian matrix of the loss function according to the network parameters, which makes them less computation-intensive. Let's consider the optimization of the loss function in the gradient descent framework. The loss function is minimized by moving in the parameter space in the direction of the loss gradient, with a step proportional to a parameter either determined empirically or adjusted during optimization phase, called learning rate. Luckily, since we are regarding our neural network as a stack of layers (viewed as nested functions), the gradient computation can be decomposed using the chain rule:

$$(f \circ g)'(X) = \nabla f(g(X)) \cdot g'(X) \tag{2.26}$$

Knowing this, the gradient of $loss(\theta)$ according to the parameters θ_j of layer j (for any layer j with learnable parameters), can be decomposed as the following product:

$$\nabla\ \ell(\theta_m) = \prod_{k=1}^{m} f'_{\theta_k}\left((\bigcirc_{i=1}^{k} f_{\theta_i})(X)\right) \cdot L'\left((\bigcirc_{j=1}^{n} f_{\theta_j})(X)\right) \tag{2.27}$$

Each factor k of the product can be computed using the definition of function f'_k, and the current input to layer k. However, layer k requires the factor from layer $k+1$ in order to compute loss gradient according to its own parameters. Consequently, the signal (the product of factors accumulated from layer n to current layer i) is passed from layer $i+1$ to layer i. In a more general sense, the gradient signal is passed from the output layer to the input layer, hence the name "backpropagation".

The move in the gradient direction with step α (the so-called learning rate) is such that:

$$\theta_k \leftarrow \theta_k - \alpha \cdot \nabla \ \ell(\theta_k) \ \ \forall k \in \{1, \ldots, n\} \tag{2.28}$$

This step is repeated until one of the stop criteria has been met. For example, the algorithm stops when a maximum number of iterations has been reached. However, gradient descent is not the only optimization algorithm that yields satisfying results in practice. For example, Limited-memory BFGS [9] relies on a second order approximation of the loss function given a limited number of past update vectors: this provides a better search direction but in return does not theoretically guarantee that the loss function actually decreases at each iteration.

2.4.3 Fully-connected layers

A *Multi-layer perceptron* is a neural network composed of multiple layers, where each layer's forward pass consists of a linear combination followed by an element-wise non-linear activation function. Let $X^{(p)} \in \mathbb{R}^{n \times m}$ be the input matrix of layer p, $W \in \mathbb{R}^{m \times k}$ the weight matrix, $b \in \mathbb{R}^k$ the bias vector, $n^{(p)}$ the number of inputs to layer p and σ the non-linear activation function of layer p. Each layer can than be formalized as follows:

$$X_{i,k}^{(p+1)} = \sigma\Big(\sum_{j=1}^{n^{(p)}} X_{i,j}^{(p)} W_{j,k} + b_k\Big) \tag{2.29}$$

Backpropagation requires to compute the partial derivatives of layer outputs with respect to current layer parameters:

$$\frac{\partial X_{i,k}^{(p+1)}}{\partial W_{j,k}} = \sigma'\Big(\sum_{j=1}^{n^{(p)}} X_{i,j} W_{j,k} + b_k\Big) \ X_{i,j}^{(p)} \tag{2.30}$$

$$\frac{\partial X_{i,k}^{(p+1)}}{\partial b_k} = \sigma'\Big(\sum_{j=1}^{n^{(p)}} X_{i,j} W_{j,k} + b_k\Big) \tag{2.31}$$

where $Y \in \mathbb{R}^2$ is the output matrix and $\sigma'(x)$ is the derivative of $\sigma(x)$, typically $\sigma(x)(1-\sigma(x))$ for the sigmoid function.

Multi-layer perceptrons have been proved to be Universal Approximators [31], meaning that they can approximate feedforward prediction functions that minimize any training loss (loss function computed on the training set). However, this fact does not inform

about the type of non-linear function to use in order to minimize a given loss function. More importantly, this does not guarantee that the model will perform well on unseen examples. Indeed, high representational power is required when the classification task is abstract. To overcome this problem and lower the validation loss as much as possible, data scientists usually stack more layers on top of each other, but this may imply high computational requirements. Convolutional layers are used instead of dense weight matrices.

2.4.4 Convolutional layers

One of the major advances in semantic segmentation is due to Convolutional Neural Networks (CNNs) [26]. A CNN is an artificial neural network made of a stack of neural layers [41]. One characteristic of CNNs is the presence of convolutional filters that map raw data to more abstract features. Each filter (or kernel) is locally connected to its output unit, which allows the convolutional layer to capture some local information about the inputs, as opposed to fully-connected layers that don't take any spatial information into account when passing data forward. This procedure is inspired by the notion of receptive field introduced by Hubel and Wiesel [33].

Weights are no longer stored in a bidimensional matrix since all inputs are no longer connected to each neuron of the current layer. Instead, each neuron is connected to a certain neighborhood of inputs. In this way, the network drastically reduces its number of parameters but still takes the spatial dependence of the data into account. If the convolutional layer is designed for processing multi-channel images for example, the parameters will be stored in a 4-dimensional tensor. Let $W \in \mathbb{R}^{b\times h\times w\times n_c}$ be the weights of the convolutional filters, $X^{(p)} \in \mathbb{R}^{b\times h_b\times w_b\times n_c}$ the input images of layer p, $b \in \mathbb{R}$ the bias vector and $X^{(p+1)} \in \mathbb{R}^{b\times(\lfloor(h_b-h)/\beta_1\rfloor+1)\times(\lfloor(w_b-w)/\beta_2\rfloor+1)\times n_f}$ the output feature maps. Let's consider the relation between the input images and the output feature maps:

$$X^{(p+1)}_{i,j,k,l} = \sigma\Big(\sum_{\alpha=1}^{h}\sum_{\delta=1}^{w}\sum_{c=1}^{n_c} W_{j,\alpha,\delta,c}X^{(p)}_{i,k+\beta_1\alpha,l+\beta_2\delta,c} + b_j\Big) \tag{2.32}$$

where i is the image identifier, j is the filter index, n_c is the number of channels, h is the filter height, w is the filter width and (β_1, β_2) are the strides (vertical and horizontal distances between neighboring pixels in the neighborhood connected to a same neuron). Partial derivatives are simply given by:

$$\frac{\partial X^{(p+1)}_{i,j,k,l}}{\partial W_{j,\alpha,\delta,c}} = \sigma'\Big(\sum_{\alpha=1}^{h}\sum_{\delta=1}^{w}\sum_{c=1}^{n_c} W_{j,\alpha,\delta,c}X^{(p)}_{i,k+\beta_1\alpha,l+\beta_2\delta,c} + b_j\Big)X_{i,k+\beta_1\alpha,l+\beta_2\delta,c} \tag{2.33}$$

$$\frac{\partial Y_{i,j,k,l}}{\partial b_j} = \sigma'\Big(\sum_{\alpha=1}^{h}\sum_{\delta=1}^{w}\sum_{c=1}^{n_c} W_{j,\alpha,\delta,c}X^{(p)}_{i,k+\beta_1\alpha,l+\beta_2\delta,c} + b_j\Big) \tag{2.34}$$

Just as in the case of fully-connected layers, the computations for the signal propagation are not shown because this report is intended to remain brief. Contrary to neural networks, it must be noted that random forest implementations are rarely equipped with

convolutional filters or even multivariate splits. Even in computer vision applications, univariate decision trees are the most frequently used trees in ensemble learners.

TODO: Double check indices and whether symbols have been correctly defined

2.4.5 Activation functions

An activation function describes the output value of a neuron and is biologically inspired. It is a mathematical representation of the level of action potential sent along its axon. More formally, it is a non-linear scalar function that takes a scalar as input. The presence of activation functions in neural networks along with fully-connected layers allows them to increase their representational power. Indeed, a stack of fully-connected layers without activation functions would have the same representational power as a single fully-connected layer, since a linear combination of linear combinations is itself a linear combination. Thus, activation functions help to actually build a hierarchical representation of the data by curving the hyperplane separating data points multiple times and at each layer.

However, not every activation function is suitable for backpropagation and one of the reasons for the success of deep learning is the low computational requirements for the gradients. Most of the activation functions are non-parametric and element-wise, which makes it easy to compute the signal during backward pass.

The best known activation function is the sigmoid function $\sigma(x)$. It has the property to have a derivative $\sigma'(x)$ expressed as a function of $\sigma(x)$, which speeds up computation times, assuming that the neural outputs are cached.

$$\begin{aligned} \sigma(x) &= \frac{1}{1+\ \exp(-x)} = \frac{\exp(x)}{1+\ \exp(x)} \\ \sigma'(x) &= \sigma(x)(1-\sigma(x)) \end{aligned} \tag{2.35}$$

However, LeCun [42] does not recommend standard sigmoid functions because normalizing activation functions generally ensure better performance. For this reason, the hyperbolic tangent is suitable because its outputs are centered around zero. Also, its derivative $\tanh'(x)$ is expressed as a function of $\tanh(x)$ which is computationally convenient. Finally, an additional linear term can be added in order to avoid flat areas, leading to an activation function of the following form: $f(x) = \tanh(x) + ax$.

$$\begin{aligned} \tanh(x) &= \frac{\exp(x) -\ \exp(-x)}{\exp(x) +\ \exp(-x)} = \frac{\exp(2x)-1}{\exp(2x)+1} \\ \tanh'(x) &= 1-\tanh^2(x) \end{aligned} \tag{2.36}$$

Assuming that target values are in the set $\{-1, 1\}$ in the framework of binary classfication, the hyperbolic tangent can be linearly modified to obtain a new function of the form $f(x) = 1.7159\tanh(\frac{2}{3}x)$. Such an activation function is profitable because, has its second derivative maximized at $x = -1$ and $x = 1$, avoiding saturation effects.

The chain rule informs us that the gradient of a given layer is factorized as a product of vectors/matrices computed by next layers. Because the absolute values of a layer's outputs are always less than one for both tanhand standard sigmoid activation functions, but also the absolute values of the gradient's components, deep architectures are often subject to vanishing gradients. Linear rectifier units (ReLU) are piecewise linear functions designed to solve these issues by keeping positive inputs unchanged. Let's note that ReLU is not differentiable at $x = 0$ but inputs can be reasonaly assumed to be rarely equal to zero in practice.

$$\begin{aligned} \text{ReLU}(x) &= \max(x, 0) \\ \text{ReLU}'(x) &= \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases} \end{aligned} \tag{2.37}$$

The outputs of a neural network are often desired to sum to one, especially when the classification task is to assign each class to a probability conditionally to the network's input. In the case where there are m classes, the output layer is composed of m neurons where the activation function associated to neuron i is given by:

$$\begin{aligned} \sigma(x^{(i)}) &= \frac{\exp(x^{(i)})}{\sum_{k=1}^{m} \exp(x^{(k)})} \\ \sigma'(x^{(i)}) &= \sigma(x^{(i)})(1 - \sigma(x^{(i)})) \end{aligned} \tag{2.38}$$

where $x^{(i)}$ is the component i of the output vector. This function is identical to the Boltzmann distribution introduced in section 2.2.1.

2.4.6 Batch normalization

According to Ioffe and Szegedy [34], deep neural networks are subject to a phenomenon called **internal covariate shift**. When the learning rate is too large, the distribution of a layer's output is drastically altered, making it difficult to train the next layer since the latter is constantly adapting to the new distribution. Batch normalization helps dealing with this issue and allows us to run the optimization algorithm with less careful parameter initialization and a larger learning rate.

When the network is trained with batch learning, its parameters are updated at every batch. Therefore, the distribution of each layer's output is changed at each batch. This is the reason for using the statistics of each batch individually to normalize the data between layers.

TODO: Backpropagating sample gradients in fully-convolution networks

$$\begin{aligned}
\mu_{\mathcal{B}} &= \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|} x_i \\
\sigma^2_{\mathcal{B}} &= \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|}(x_i - \mu^2_{\mathcal{B}}) \\
\hat{x}_i &\leftarrow \gamma\frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} + \beta
\end{aligned} \tag{2.39}$$

Optimize β, γ with backpropagation. TODO:

2.4.7 Regularization

From an optimization perspective, regularization is a penalty used to prevent parameters from growing arbitrarily big during training. According to Occam's law of parsimony, simpler hypotheses should be privileged over more complex ones. Therefore, when the neural architecture involves a large number of free parameters in the presence of relatively few data samples, regularization helps reducing parameters importance and converging to less arbitrary parameter values. From a Bayesian perspective, regularization provides a prior distribution over the model parameters. In Bayes formula, the posterior $P(\theta|X, \alpha)$ is a function of both the prior $P(\theta|\alpha)$ and the likelihood of the data $P(X|\theta, \alpha)$ under model θ.

$$P(\theta|X, \alpha) = \frac{P(X|\theta, \alpha)\, P(\theta|\alpha)}{P(X|\alpha)} \tag{2.40}$$

The relation between the loss function of a neural network and Bayes formula can be established by proving the two following points:

- The log-likelihood of the data is equal to the negative cross-entropy.
- The regularization term is proportional to the prior distribution of the parameters.

The first part is easy to show since negative log-likelihood can be obtained from binary cross-entropy:

$$CE(\hat{y}, y) = -\log\prod_{i=0}^{n} P(\hat{y}_i)^{y_i} \tag{2.41}$$

$$= -\sum_{i=0}^{n} y_i \log\hat{y} + (1 - y_i)\log 1 - \hat{y} \tag{2.42}$$

This allows us to provide a statistical interpretation of the loss function. Regarding priors, L_1 and L_2 regularizations are going to be introduced in the following two sections.

2.4.7.1 L_1 regularization

Adding a L_1 regularization term to the loss function reduces to providing a Laplacian prior on model parameters.

$$\max_{\theta} \log P(\theta|\eta, b) = \max_{\theta} \log \prod_{i=1}^{m} \frac{1}{2b_i} \exp\Big(\frac{-|\theta_i - \eta_i|}{b_i}\Big) \tag{2.43}$$

$$= \max_{\theta} \sum_{i=1}^{m} \frac{-|\theta_i - \eta_i|}{b_i} - \log 2b_i \tag{2.44}$$

$$= \min_{\theta} \sum_{i=1}^{m} |\theta_i - \eta_i| \tag{2.45}$$

By setting vector $\eta \in \mathbb{R}^m$ to 0, the resulting regularization term takes its final well-known form $\sum_{i=1}^{m} |\theta_i|$.

2.4.7.2 L_2 regularization

L_2 regularization acts as a Gaussian prior on model parameters. This can be highlighted by setting the probability density function of the Gaussian distribution as the prior and show that the regularization term is proportional to the logarithm of the product of priors.

$$\max_{\theta} \log P(\theta|\eta, \sigma) = \max_{\theta} \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{(\theta - \eta)^2}{2\sigma^2}\Big) \tag{2.46}$$

$$= \max_{\theta} \sum_{i=1}^{m} -\frac{(\theta_i - \eta_i)^2}{2\sigma^2} - \log\sqrt{2\pi\sigma^2} \tag{2.47}$$

$$= \min_{\theta} \sum_{i=1}^{m} (\theta_i - \eta_i)^2 \tag{2.48}$$

Again, by setting vector $\eta \in \mathbb{R}^m$ to 0, the regularization term takes its final form $\sum_{i=1}^{m} \theta_i^2$.

2.4.8 Optimization algorithms

Gradient descent is a very popular optimization algorithm, but is rarely used as such in practice since state-of-the-art deep learning frameworks offer more advanced gradient-based techniques [57]. What is meant by gradient is the gradient vector obtained by concatenation of the gradients w.r.t. each layer's parameters. This final gradient vector gives an improvement direction, but a decrease of the loss function is only guaranteed by moving by an arbitrary small step in the parameter space.

Gradient descent has three variants: batch, mini-batch and stochastic gradient descent. In batch gradient descent, all training examples are used to compute the improvement direction: this is done by computing the gradient for each training example and averaging across all examples. In mini-batch gradient descent, only a subset of training examples are being considered for the computation of the improvement direction (which can thus be seen as an approximation of the actual gradient). Usually, the ordering of training examples is shuffled at the beginning of each iteration (also called epoch) and then examples are sampled in the resulting order repeatedly, to ensure that each of them is seen by the model exactly once per iteration. In the stochastic variant, only the gradient of a single training example is used to approximate the improvement direction. Due to the high variability of gradients from one example to the other, the improvement direction is changing in a chaotic manner during the optimization process, hence the adjective stochastic. The three types of improvement vectors are summarized in table 2.1.

Name	Number of examples involved	Formula
Average (true) gradient	N	$g_t = \frac{1}{N} \sum_{i=1}^{N} \nabla L(f_{x_t}(Z_i))$
Mini-batch gradient	$\|B\|$	$g_t = \frac{1}{\|B\|} \sum_{i \in B} \nabla L(f_{x_t}(Z_i))$
Sample gradient	1	$g_t = \nabla L(f_{x_t}(Z))$

Table 2.1: Types of gradients and gradient approximations used in common optimization methods. Here the term "sample" refers to a sample of 1 example.

In its most simplistic form, gradient descent optimization consists in update the parameter vector x_t from step t using the following rule:

$$x_{t+1} = x_t + \Delta x_t \tag{2.49}$$
$$= x_t - \eta g_t \tag{2.50}$$

where update vector Δx_t is equal to the negative approximated gradient $-g_t$ multiplied by a learning rate η. Learning rate controls how much model parameters are being updated in the improvement direction.

Stochastic gradient descent has been extended with a so-called momentum term that accelerates the update when gradients approximately point in the same direction from one step to another. TODO: Momentum: cite: Learning representations by back-propagating errors

$$\Delta x_t = \rho \Delta x_{t-1} - \eta g_t \tag{2.51}$$

ρ is a decay parameter that controls how much to keep track of past update vectors. In practice, the landscape of the loss function w.r.t. parameters is likely to be composed of many narrow valleys where standard stochastic gradient vector is inefficient due to the small norm of gradients along valleys. The momentum helps optimizing across such valleys in a smaller number of steps due to its additive effect when gradient vectors are similar from one step to the next one.

TODO: Adam: [40]

TODO: RMSProp: Introduced by Hinton on Coursera, first used in [28]

TODO: AdaGrad: [16]

$$\Delta x_t = -\frac{\eta}{\sum_{\tau=1}^{t} g_\tau^2} g_t \tag{2.52}$$

ADADELTA [72] is an adaptive extension of stochastic gradient descent that is robust to the noise introduced by the high variability of sample gradients. Also, it dynamically selects the learning rate so that no hyper-parameter tuning is required on it.

$$\Delta x_t = -\frac{RMS[\Delta x]_{t-1}}{RMS[g]_t} g_t \tag{2.53}$$

$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon}$

TODO: L-BFGS: [9]

2.4.9 Input features

2.4.9.1 Global features

2.4.9.1.1 Protein length Protein length is computed as the number of residues in the protein sequence. It provides complementary information that convolutional layers cannot capture. Indeed, fully-connected neural networks have the ability to handle arbitrary-sized features maps at the cost of not knowing the dimensionality of their inputs. Injecting protein length as a supplementary global feature may help the model to infer the maximum distance in long-range contacts.

2.4.9.1.2 Effective number of sequences The number of effective sequences is equal, w.r.t. to a given threshold, to the number of non-redundant sequences in the set of homologous sequences. It provides a bound on the potential performance of the DCA methods involved in the pipeline.

$$M_{eff} = \sum_{i=1}^{m} w_i = \sum_{i=1}^{m} \frac{1}{|\{r(s_i, s_j) \geq \tau \ \ \forall j \in \{1, \dots, m\}\}|} \tag{2.54}$$

w_i is the weight associated to homologous sequence i. $r(s_i, s_j)$ is the identity rate between sequences s_i and s_j, in other words the number of matching residues (including gaps) divided by the total number of positions, which is assumed to be equal in both sequences.

2.4.9.2 1-dimensional features

2.4.9.2.1 One-hot encoded sequence Given an amino acid sequence $\{s_1, s_2, \dots, s_L\}$ of size L, its one-hot encoded version is a matrix $X \in \{0, 1\}^{L \times 21}$ where x_{ia} is one if $s_i = a$.

2.4.9.2.2 Solvent accessibility prediction TODO:

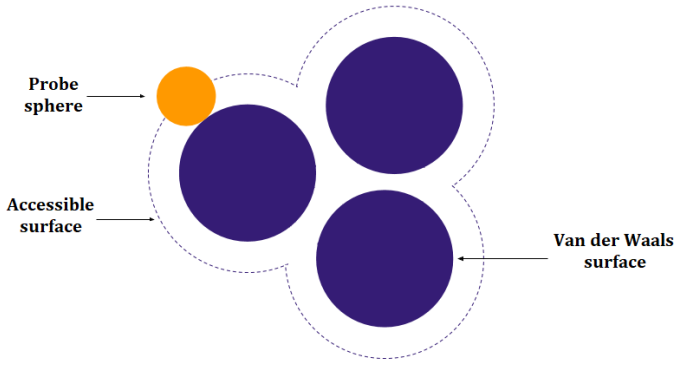

Figure 2.2: Accessible surface, obtained by "rolling" a probe sphere (a molecule of solvent, colored in orange) on the Van der Waals surface of a biomolecule (colored in blue).

TODO: define Relative Solvent Accessibility TODO: cite RaptorX-property RSA prediction is 3-state. A residue is either:

- Buried (B), when RSA is below 10%.
- Intermediate (I), when RSA is between 10% and 40%.
- Exposed (E), when RSA is above 40%.

2.4.9.2.3 Predicted secondary structure prediction As defined by DSSP, there are 8 different states for encoding secondary structure: H (alpha-helix), G (310 helix), I (pi-helix), E (beta-strand), B (beta-bridge), T (beta-turn), S (high curvature loop) and L (irregular loop). In practice, secondary structure prediction is either 3-state or 8-state [66].

- **Alpha-helix** is the most common pattern among secondary structures.
- **310 helix**: TODO
- **Pi-helix**: TODO
- **Beta-strand**: TODO
- **Beta-bridge**: TODO
- **Beta-turn**: TODO
- **High curvature loop**: TODO
- **Irregular loop**: TODO

When the number of labels is limited to three, the predictor only focuses on beta-strands (E), alpha-helices (H) and a third state which is the union of the six remaining states (C). Formally, a m-state secondary structure prediction is a matrix $S \in [0,1]^{L \times m}$ where element $S_{i,j}$ is the probability of residue i being in conformation j, with each row summing to one.

2.4.9.2.4 Region disorder prediction Region disorder prediction is a vector $D \in [0,1]^L$ where D_i is the probability of residue i being in a region of missing residues in the X-ray 3D structure. Residues with high probability are said to be disordered.

2.4.9.2.5 Amino acid frequencies Amino acid frequencies are position-specific features that can be efficiently computed. Let $S \in \{0, \ldots, 22\}^{M \times L}$ be a MSA matrix containing M sequences aligned to a target sequence of length L. Then amino acid frequencies can be arranged in a matrix $F \in \mathbb{R}^{L \times 22}$ where element F_{ia} is computed as follows:

$$F_{ia} = \frac{1}{M} \sum_{k=1}^{M} \delta(S_{ki}, a) \tag{2.55}$$

2.4.9.2.6 Position-Specific Scoring Matrix TODO: PSI-PRED: [35]

2.4.9.2.7 Atchley factors TODO: Atchley: [3]

2.4.9.2.8 Self-information In information theory, self-information is the amount of information, in bits, obtained by observing a random variable. In particular, let $x_{ij} \in \{0, 1\}$ be a binary variable indicating the presence of an amino acid of type j at site i. The self-information suggested by Michel et al [47] can be formalized with the following equation:

$$I_{ij} = \log_2(p_{ij}/\langle p_i \rangle) \tag{2.56}$$

where p_{ij} is the probability of observing amino acid j at site i among all residues of given MSA, and $\langle p_j \rangle$ is the frequency of amino acid j in the Uniref50 dataset.

2.4.9.2.9 Partial entropies TODO: [47]

$$S_i = p_i \log_2(p_i/\langle p_i \rangle) \tag{2.57}$$

2.4.9.3 2-dimensional features

2.4.9.3.1 Mutual Information and Normalized Mutual Information Following the formalism described in [47], MI is described as:

$$MI(x, y) = \sum_{x,y} P(x, y) \log\left(\frac{P(x, y)}{P(x) \cdot P(y)}\right) \tag{2.58}$$

$$NMI(x, y) = \frac{MI(x, y)}{\sqrt{S(x) \cdot S(y)}} \tag{2.59}$$

Average product correction is applied to both MI and NMI.

2.4.9.3.2 Cross-entropy Cross-entropy is computed in [47] using the following formula:

$$H(x, y) = S(x) + S(y) - MI(x, y) \tag{2.60}$$

2.4.9.3.3 Contact potential TODO:

2.4.9.3.4 Evolutionary couplings Predictions from GaussDCA, plmDCA or PSICOV can be used as input to the deep learning approach. Only top predicted contacts are informative, TODO

2.4.9.3.5 Covariance Covariance matrices are computed as in equation 2.18 of the section about Gaussian graphical models (see section 2.3.2). In PSICOV [36], the inferred covariance matrix is averaged across the dimension of amino acid types. In DeepCov [37], the full matrix is used as input to the supervised model.

2.4.10 Features for the proposed approach

The lines which will follow are a discussion about the features to use as inputs to the model. At the outset, it should be noted that all models rely on two-dimensional features. Those features are either predictions from another predictor, covariance matrices or correlated mutations. However, using intermediary predictions requires installing additional software. PSICOV is written in C and can be recompiled. The official implementation of plmDCA was originally made in Matlab, forcing researchers to add a whole raft of external tools. plmDCA, as well as GaussDCA now have a Julia implementation, making them more accessible. The proposed approach incorporates all three predictors in its pipeline.

[

Features		DeepCov	DeepContact	PConsC4	DNCON2	RaptorX	Proposed method
Global	Protein length				×		×
	Meff				×		×
1-dimensional	Column log-entropy		×				
	Predictors stdv		×				
	Encoded sequence			×			×
	Solvent accessibility		×		×	×	×
	Predicted SS		×		×	×	×
	AA frequencies		×	×			
	PSSM				×	×	
	Atchley factors				×		
	Self-information			×			×
	Partial entropies			×			×
2-dimensional	Mutual Information (MI)			×	×	×	×
	Normalized MI		×	×	×		×
	Cross-entropy		×	×			×
	Contact potential					×	
	EVFold		×				
	CCMPred		×		×	×	
	plmDCA				×		
	GaussDCA			×			×
	PSICOV						
	Covariance	×					

Table 2.2: Features used in state-of-the-art deep learning approaches. Feature extraction methods that rely on external tools (excluding MSA tools) are highlighted in bold.

]

2.5 Deep learning and Protein Contact Prediction

2.5.1 Fully-convolutional networks

As a reminder, a fully-convolutional network is a neural network capable of handling variable sized input. Therefore, the dynamic input dimensions cannot be processed by a fully-connected neural layer. An example of such an architecture is DeepCov [37], a deep neural network composed only of 2D convolutional layers and an additional maxout input layer. A maxout layer is made of a convolutional layer, followed by a max-pooling operation. Dimensions of intermediate feature maps are preserved by convolutions using a stride of one and a "same-padding". In most deep learning libraries, padding of type "same" refers to the padding applied by convolutional layers to maintain the sizes of spatial (convoluted) dimensions. In DeepCov, the only features are the covariance matrices as computed in equation 2.18. Couplings matrices predicted by DCA methods can be fed as input to the model instead of covariance matrices, as shown in plmConv study [27].

The approach described in the DNCON2 paper [37] also implements convolutional layers with dynamic spatial dimensions. Additionally to that, the fuzziness of residue contact definition is handled by training one fully-convolutional neural network per contact threshold. More specifically, five networks are trained to output contact maps at 6, 7.5, 8, 8.5 and 10 Å thresholds, respectively. These five are stacked on top of a sixth network in charge of refining and combining the predictions into a final contact map at a threshold of 8 Å.

2.5.2 Residual Networks (ResNets)

As described in section 2.4.2 about backpropagation, the loss gradient with respect to the parameters of a specific layer is computed as the product of many other mathematical entities (vectors, scalars, matrices, etc.), and the number of factors in such a product grows linearly with the number of operations applied after current layer. When this number is too large, some layers may be updated with numerically unstable gradients.

A widely used solution is to add residual connections [30] to the architecture. The latter can thus no longer be viewed as a regular composition of functions and must take into account the residual mapping at the end of each residual block. The output $Y^{(r)}$ of residual block k should now be formalized with a more general form:

$$Y^{(r)} = f\big(X^{(r)}, \{W^{(p)}\}_p\big) \tag{2.61}$$

where $W^{(p)}$ are the weights of a layer p in block k. Residual mappings can be implemented in several ways and figure 2.3 illustrates one of them.

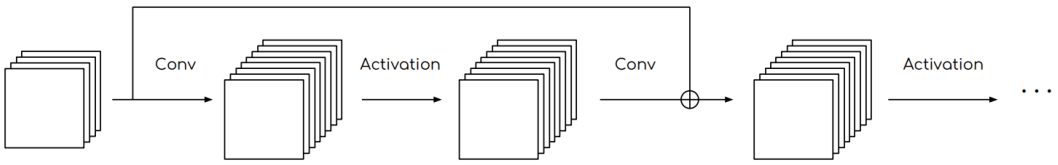

Figure 2.3: Illustration of a residual connection in a convolution neural network. For the element-wise sum to work, the input and output of the residual block are required to be of the same shape.

2.5.3 Deep fully-convolutional residual networks

Most successful methods rely on very deep architectures with residual mappings. One-dimensional features are processed by one-dimensional residual network before being concatenated with two-dimensional features. The resulting tensor is the input of a two-dimensional residual network. Examples of such approach is DeepContact [44] and the state-of-the-art RaptorX-contact predictor [44]. Both methods rely on CCMPred contact prediction, solvent accessibility and secondary structure prediction. Additionally to CCMPred, DeepContact incorporates EVFold predictions together with the rest of the two-dimensional features. Also, global features (e.g. the number of effective sequences) are tiled and concatenated with other features: this does not impact the fully-convolutional property of DeepContact because global features are invariant to the protein length. The largest difference in the two methods lies in the depth of the networks: 9 convolutional layers for DeepContact and 60-70 layers for RaptorX-contact. It must be noted that RaptorX-contact architecture is not fully-convolutional since zero-padding is applied to feature maps when more than one protein is processed in a batch.

2.5.4 U-net architecture

In PconsC4 [47], the model is partly built on top of the U-net architecture [56], and trained on a set of 2891 proteins retrieved from PDB. Features are divided in one-dimensional inputs including one-hot-encoded amino acids, self-information and partial entropies, and two-dimensional inputs including mutual information, normalized mutual information and cross-entropy. For both mutual information and normalized mutual information, average product correction is applied. One-dimensional features are convoluted through one-dimensional residual networks , concatenated and finally reshaped to two-dimensional maps with an outer product. After reshaping, the intermediate feature maps are concatenated with the two-dimensional features and the whole is fed as input to the U-net architecture.

Models designed for semantic segmentation problems (including PCP) have been shown to gain significant performance when additional connections are allowed between layers close to the network's input and layers close to the network's output [32]. U-net architectures develop this idea: they have a somehow symmetric structure made of two sub-networks that compress and decompress the information, respectively. The first sub-network successively alternates between convolutional transformations and max-

pooling, while the second sub-network alternates between convolutional transformations and transposed convolutions (also called upsampling). Similarly to autoencoders, the features maps processed the middle layers or of much smaller dimensionality than input or output tensors. To prevent the whole architecture from forgetting contextual information due to compression, shortcut connections are added between tensors of identical dimensionality. Contrary to residual networks, these shortcut connections are implemented by concatenation instead of addition. Due to the max-pooling and upsampling operations, the width and height of input images should preferrably be a power of two. Because PconsC4 model outputs entire contact maps, and because proteins are of variable length by nature, input feature maps of a particular protein are zero-padded to the smallest power of two that is larger than the protein length. Finally, PconsC4 archicture does not only predict a contact map but predicts multiple contact maps at different distance thresholds.

2.5.5 Dense networks (DenseNets)

Dense networks extend the idea of residual networks by allowing residual mappings between all layers, resulting in an ergodic topology. In DenseNets [32], shortcut connections are implemented with a concatenation operation instead of addition. Nevertheless, spatial dimensions are still constrained to be identical between the source and the destination of a shortcut connection. Figure 2.4 illustrates a dense block composed of 2 convolution layers: it is shown how connections are allowed between all entry points and layer outputs.

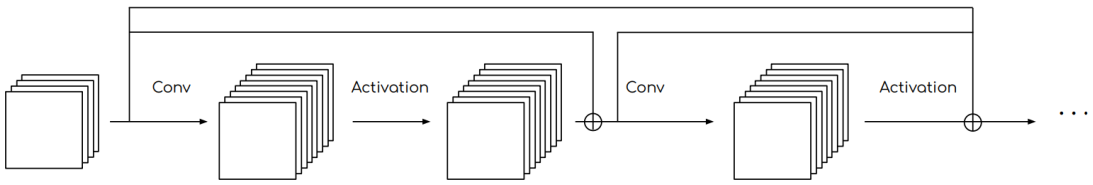

Figure 2.4: Illustration of a dense block inside a DenseNet. All layer entry points and outputs belonging to a dense block are connected by residual mappings.

Each unit of a dense block is the composition of batch normalization, activation and convolution operations.

2.5.6 TiramiProt

TiramiProt is based on the Tiramisu architecture [62] which tries to combine the ideas of both U-net and DenseNet. Like U-net architectures, Tiramisu applies max-pooling for reducing dimensionality and upsampling to restore the input dimensionality. Each dense block output from the downsampling part of the network is, by symmetry, connected to the entry point of the corresponding dense block in the upsampling part of the network. As in U-net, these shortcut connections are implemented by feature map concatenation. Training set and features are the same as in the Pconsc4 study.

2.5.7 DeepConPred2

DeepConPred2 [15] is based on a problem-specific architecture made of three modules. First module consists of a Deep Belief Network (DBN) that predicts contacts between secondary structures from CCMPred coevolutionary information. A DBN is a graphical model implemented as a stack of unsupervised building blocks such as autoencoders or restricted Boltzmann machines. The output of the first module, together with solvent accessibility and secondary structure prediction, is fed as input to each of the DBNs composing the second module: one for short-range contacts, one for medium-range contacts and three for long-range contacts (taken from previous study [71]). Each of these components are used to predict actual residue contacts. Then, each DBN output is fed as input to one of the ResNets of the third module. Each ResNet has 50-80 convolutional layers with Leaky-ReLU activation functions.

2.5.8 Properties of DL approaches

PCP methods that have just been described are summarized in table 2.3 according to some indicators such as training set size, fully-convolutional and fuzzy aspects of the approach and depth of the architecture. RaptorX-contact has not been counted as a fully-convolutional approach since zero-padding is used for batches of more than one protein. Depth of the network has been measured as the number of non-linearities, that is to say the number of layers or blocks preceding an activation function. Additionaly, the table shows which methods incorporate contacts defined at different distance thresholds.

	Training set size	Fully-convolutional	# non-linearities	Fuzzy
DNCON2	1230	-	6	$\top$
DeepCov	6003	$\top$	14	$\bot$
plmConv	231	$\bot$	4	$\bot$
DeepConPred2	3443	-	50-80	$\bot$
DeepContact	-	$\top$	9	$\bot$
PconsC4	2891	$\bot$	-	$\top$
TiramiProt	2891	$\bot$	16	$\top$
RaptorX-Contact	~ 6000	$\bot$	60-70	$\bot$

Table 2.3: Overview of different deep learning models for PCP: number of proteins in the training set, whether the architecture is fully-convolutional, how deep (measured in activation functions) the model is, and whether it learns from contact maps defined with multiple distance thresholds.

In PconsC4 and TiramiProt, a single network predicts multiple contact maps at the same time, while DNCON2 uses multiple networks to each predict a contact map at a different threshold.

2.6 Contact-assisted protein folding

Static protein structure can be recovered with a high resolution solely based on a few true residue contacts [39]. Because predicted contact maps are fuzzy by nature and may contain false contacts, only top $\alpha \cdot L$ contacts (residue pairs associated to highest predicted probabilities) are usually considered. FT-COMAR [64] GDFuzz3D [53] and MODELLER [22] CONFOLD [1] CONFOLD2 [2] CoinFold [67] distance distributions [54]

In the study of Chelvanayagam et al. [10], protein structure is modelled in a distance geometry setting using Gaussian restraints with empirically known mean and variance. Let $E(x_i, x_j)$ be the Euclidean distance between residues i and j, where residues are being represented by the center of their respective C_β (or C_α) atoms. Let $D_{i,j}$ and $V_{i,j}$ be the average Euclidean distance and variance associated with the Gaussian constraint between residues i and j. Under the assumption of independence between residue pairs, maximizing the log-likelihood reduces to minimizing the following objective function:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{(E(x_i, x_j) - D_{i,j})^2}{V_{i,j}} \tag{2.62}$$

where n is the protein size in residues. Constraints are chosen according to predicted secondary structure and surface accessibility. By default, all residues are assumed to be separated by an average distance of 120 Å, with a high variance (120 Å^2) to allow flexibility. For each residue pair, these prior parameters are replaced by a more accurate constraint when sufficient information is available. For example, residues with low surface accessibility are assumed to be separated by a distance of 7.5 Å from the center of mass of the protein, while residues with high surface accessibility or assigned a higher average distance of 12 Å. The Euclidean coordinates of the center of mass are additional variables to be added to the model. Residues participating in the active site are considered to be near in space. Adjacent and almost-adjacent residues are assigned low distances with very small variance since the distance between adjacent C_α atoms is fixed but with a small variability induced by torsion angles. Helices and strands, when predicted secondary structure is available, are modelled as well. All residue pairs with a sequence separation $\in [2, 5]$ in alpha helices are being constrained, and only residue paurs with a sequence separation $\in [2, 3]$ in 3/10 helices are being constrained. Similarly, sequence separation in strands must be in the range $[2, 4]$. Disulfide bonds are modelled by an average distance of 5.5 Å and a variance of 0.2 Å^2 between cysteine residues. In sheets, center residues of adjacent strands are assigned an average distance of 4.54 Å and a variance of 0.1 Å^2. However, sheet topology and disulfide bonds are not always available and the solution suggested in the paper is to have recourse to sheet and disulfide combinatorics

Chapter 3

Materials and methods

3.1 Datasets

Five datasets have been used in the framework of this thesis: one for training the model, one for optimizing the hyper-parameters, and three for benchmarking. The training set is a set of 354 proteins including the 150 families reported in the original PSICOV paper [36], plus a subset of the first benchmark set used by Michel et al. to evaluate PConsC3 [60]. The validation set is composed of the 30 protein families from the validation set of PConsC3, which itself is a subset of the test set of PConsC2 [61].

Three test sets have also been considered in order to make a direct comparison with the state-of-the-art RaptorX-Contact predictor. The first test set embodies the 105 protein domains from the CASP11 experiment, the second test set 76 proteins from the CAMEO project, and the fourth test set 398 membrane proteins.

3.1.1 Homology reduction

A straightforward method for reducing homology between two set of proteins is to remove proteins that have a sequence identity rate above a given threshold. As a rule of thumb, this threshold is usually set to 40%. Identity rates were computed by running Needleman-Wunsch algorithm on each pair of proteins coming from two different datasets. Score matrix was set such that exact matches give a score of 1 and any mismatch gives a penalty of -1. Indeed this approach promotes global alignments with maximum identity rates.

Identity rate	Minimum	Average	Maximum
Training set - CASP11	5.4	22.4	32.5
PSICOV150 - CASP11	7.3	22.9	39.7
PSICOV150 - CAMEO	6.2	21.7	34.4
Training set - CAMEO	4.0	21.3	32.5
PSICOV150 - Membrane	7.6	22.2	74.0

Table 3.1: Identity rates between training and benchmark sets, expressed as percentages.

Similarity rates are more informative than identity rates because amino acids are very likely to mutate across evolution towards amino acids that share similar physico-chemical properties, and such measures take these mutations into account. However, ECOD H-classes [11] were used instead of similarity rates because they can potentially give more evidence on whether two sequences are evolutionary related. Therefore, proteins belonging to different datasets and sharing the same ECOD H-class have been rejected.

TODO: Computation of identity rates, Common ECOD H-classes

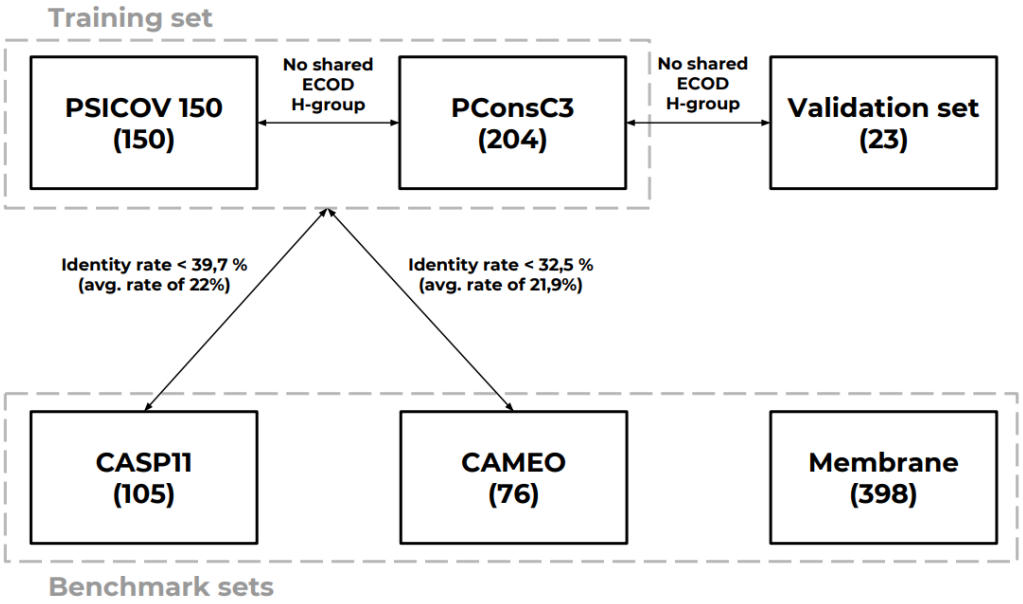

Figure 3.1: Homology reduction between the different datasets

3.1.2 Transmembrane proteins

TODO:

3.1.3 Feature extraction

TODO: Profile HMMs: [17] MSAs have been created using HHblits [55] (version as of the date of 26th February 2016) on the Uniprot20 database with an e-value of 1. Parameters have been set in such a way that all sequences in each of the database MSAs are aligned. The obtained MSAs have been used as input to all other predictors and intermediate predictors, allowing for easier comparability.

All protein sequences, structures, MSAs and intermediate predictions used in this thesis come from the datasets that were publicly available at the address `http://pconsc3.bioinfo.se/pred/download/` as on the date of 28th December 2018. Information available in these datasets is the following:

TODO: PDB parser to obtain distance and contact maps

- Protein sequence in FASTA format
- obtained using HHblits on the corresponding protein family
- Atom 3D coordinates
- PhyCMAP [69] intermediate predictions
- plmDCA [18] intermediate predictions
- GaussDCA [4] intermediate predictions
- Predictions made by PConsC3 [60] at each layer of the model
- CCMPred [58] predictions (only available in the 4 test sets)
- EVFold [59] predictions (only available in the 4 test sets)
- PSICOV [36] predictions (only available in the 4 test sets)
- MetaPSICOV [38] predictions (only available in the 4 test sets)

TODO: Took alignments from PConsC2 and PConsC3 -> model not influenced by the new releases of alignments tools TODO: What about the protein structures?

TODO: Oversampling negative class: [46]

3.2 Summary of input features

As described in section 2.4.9, input features can be split into three categories: global, 1-dimensional and 2-dimensional features. The proposed model takes as input the protein length, the effective number of sequences in the corresponding MSA, position-specific statistics, residue pair-specific statistics, and predictions made by PSICOV, plmDCA and GaussDCA. Additionaly to these features, secondary structure, solvent accessibility and region disorder are predicted by RaptorX-property server and added to the rest of 1-dimensional features. TODO: cite RaptorX-property

Category	Feature name	Dimensionality
Global	Protein length L	scalar
	Effective number of sequences M_{eff}	scalar
1-dimensional	One-hot-encoded sequence	$L \times 22$
	Self-information	$L \times 22$
	Partial entropy	$L \times 2 \cdot 22$
	Predicted secondary structure	$L \times 3$
	Solvent accessibility	$L \times 3$
	Region disorder	L
2-dimensional	Mutual information	$L \times L$
	Normalized mutual information	$L \times L$
	Cross-entropy	$L \times L$
	PSICOV predictions	$L \times L$
	GaussDCA predictions	$L \times L$
	plmDCA predictions	$L \times L$

Table 3.2: Input features of the proposed model. Global features are scalar values, whereas dimensional features are presented in the form of matrices of given shape.

3.3 Proposed architecture

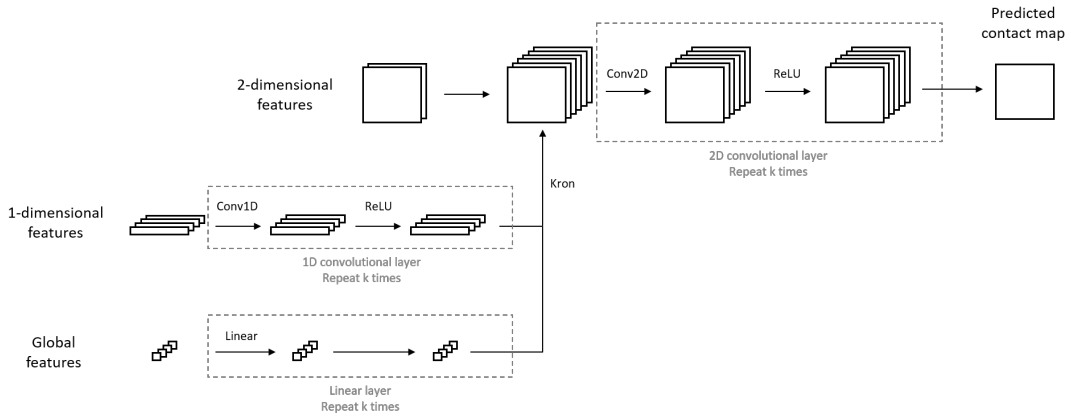

Figure 3.2: Proposed architecture of the deep convolutional neural network for semantic segmentation

3.4 Evaluation

3.4.1 Contact map evaluation

Protein contact maps are imbalanced by nature: they contain very few residue contacts compared to their number of residue pairs. L being the number of residues in a protein, the number of residue contacts increases linearly with L while the number of residue pairs increases quadratically [50]. This is important because one can evaluate a model

only on the L (or even less than L) predicted residue contacts the model is the most confident about. Such evaluation metric is called *best-L/k PPV (Positive Predictive Value)* and can be formulated as follows:

$$\text{Best-L/k PPV} = \frac{\sum_{\substack{(i,j)\in B(L/k) \\ i-j\geq 6}} C_{i,j}}{L/k} \tag{3.1}$$

where $C_{i,j}, N_{i,j} \in \{0,1\}\ \forall i,j, i-j \geq 6$ are boolean values indicating a contact or a non-contact, respectively. $B(L/k)$ is the set of L/k most confident predicted probabilities (with highest values) and $p_{i,j}$ is the predicted probability that residue pair (i,j) forms a contact. Contacts under a residue distance of 6 amino acids are not considered during evaludation phase even though they are used during training phase.

Best-L/k PPV can also be split into three separated metrics: short-range, medium-range and long-range contacts. These three types of contacts can be defined by the residue separations used by Skwark et al. [61]:

- Short-range contacts: 6 - 12 residue separation
- Medium-range contacts: 12 - 24 residue separation
- Long-range contacts: 24+ residue separation

3.4.2 3D model evaluation

TODO: TM-score and RMSD

3.5 Hyper-parameter optimization

In order to ensure the best hyper-parameters are selected for the model that will be evaluated on the benchmark sets, the Hyperopt Python library [7] has been used to explore the hyper-parameter space and fine-tune the model on the validation set. Training and evaluating a deep neural network is very costly and, as a matter of fact, each trial point in the hyper-parameter space should be carefully selected. Techniques based on grid search do not suit the problem because they are uninformed methods.

Hyperopt provides an informed search method called Tree-structured Parzen Estimators (TPE) [8]. In Bayesian hyper-optimization, the posterior probability $P(\alpha|L)$ is defined as a function of the hyper-parameter vector α and the loss L. Contrary to techniques based on Gaussian processes that approximates $P(\alpha|L)$ only, TPE models both posterior $P(\alpha|L)$ and $P(L)$. The prior is iteratively replaced with non-parametric densities based on generated points $\{\alpha_1, \alpha_2, \dots\}$. It this search, TPE is an informed search strategy that refines its prior as new points are observed in the hyper-parameter space. The "tree structure" is due to the way the posterior is computed.

$$P(\alpha|L) = \begin{cases} l(\alpha) & \text{if } L < L^* \\ g(\alpha) & \text{otherwise} \end{cases} \tag{3.2}$$

Let f be the prediction function of the model (see section 2.4.2 about backpropagation algorithm), and let $f(\alpha) \triangleq \text{argmin}_{f(\Theta,\alpha)} c\big(f(\Theta,\alpha)\big)$ be the prediction function of a trained model that minimizes a given loss function c w.r.t. a fixed hyper-parameter vector α. $l(\alpha)$ is the non-parametric density function created by the observations $\{\alpha^i\}$ such that $L = c(f(\alpha))$ is below the threshold L^*, and $g(\alpha)$ is created with the remaining observations. The threshold L^* is set as a quantile of the observed values of L. The value to be optimized in TPE is the Expected Improvement (EI), which is measured as an infinitesimal sum of loss improvements weighted by the posterior. After applying Bayes formule to the posterior, calculus of EI becomes:

$$EI_{L^*}(\alpha) = \int_{-\infty}^{L^*} (L^* - L) P(L|\alpha) dL = \int_{-\infty}^{L^*} (L^* - L) \frac{P(\alpha|L)P(L)}{P(\alpha)} dL \tag{3.3}$$

In the framework of Adaptive Parzen Estimators, to each hyper-parameter is assigned a prior and a density function, and the estimator is built as a weighted mixture of them. For example, a continuous variable can be assigned:

- A uniform prior with lower bound a and upper bound b.
- A function defined as a mixture of Gaussian distributions, each centered on a point of the hyper-parameter space. The standard deviation of a particular distribution can be set as the maximum between distances to the left and right neighbors.

The density function is either $l(\alpha)$ or $g(\alpha)$ depending on whether the loss function associated to current point is below the threshold or not.

Module	Hyper-parameter	Set of values
General	Batch size	$\{1, 2, 4, 8, 16, 32\}$
	Batch normalization	$\{\top, \bot\}$
	Track running state	$\{\top, \bot\}$
	Learning rate	TODO
	L2 penalty	TODO
	Parameter optimization	{ADADELTA, Adagrad, Adam}
	Activation function	{ReLU, ELU, LeakyReLU, Tanh}
	Use global modules	$\{\top, \bot\}$
Global module	Depth	$\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$
1-dimensional module	Depth	$\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$
	Filter size	$\{3, 5, 7\}$
	Number of filters	$\{8, 16, 32, 64, 128\}$
2-dimensional module	Depth	$\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$
	Filter size	$\{3, 5, 7\}$
	Number of filters	$\{8, 16, 32, 64, 128\}$

Table 3.3: Hyper-parameter space for the proposed architecture.

3.6 Implementation

3.6.1 Availability

Main source code is available at: https://github.com/AntoinePassemiers/Wynona.

Template-free contact-assisted 3D modelling algorithm is available at:
https://github.com/AntoinePassemiers/GDE-GaussFold.

3.6.2 Deep learning framework

Methods and results presented in this thesis have been both implemented and produced in Python. The neural architecture has been built on top of PyTorch, which is an open source deep learning framework based on Torch [12].

PyTorch does not natively handle arbitrary-sized inputs, for example fully-convolutional neural networks cannot accept images with variable height/width. For this aim, it is necessary to add a layer of abstraction so the neural models are able to process *virtual batches* of inputs. Let's define a virtual batch as the number of samples a model has to process between each parameter update. A forward pass on a virtual batch then consists in constructing one computational graph per sample and backpropagate the gradients through each one of them separately. Once all the gradients have been computed, they are collected and averaged over the sample dimension. Pytorch allows to explicitly call the forward pass, backward pass and update procedures when needed, which eases the implementation of virtual batch processing.

Model general architecture is fully-convolutional, forcing us to use only deep learning functionalities that are invariant to individual input sizes. These are:

- Element-wise operations like activation functions: ReLU, ELU, Sigmoid, etc.
- Dropout, which preserves the dimensionality of its inputs.
- Convolution, because convolutional filters have a dimensionality that is invariant to the input size (see section 2.4.4).
- Batch normalization (see section 2.4.6).
- Many non-neural transformations like arithmetic operations, Einstein summation, Kronecker product, etc.

3.6.3 Feature extraction

Many features discussed in the present document rely on amino acid counts. Despite the fact that counting algorithms such as histograms are embarassingly parallel and naturally suitable for multiprocessing, they cannot be efficiently vectorized. For this specific reason, the scientific computing library NumPy (which has been extensively used during the experiments) is not sufficient to extract this type of features in reasonable time. Instead, C extensions have been created using the Cython compiler[5].

3.6.4 Contact-assisted 3D modelling

Contact-assisted 3D modelling algorithm makes use of the Multidimensional scaling algorithm as implemented by Scikit-learn [51] and the implementation of Dijkstra's algorithm from NetworkX TODO: cite?. Details about the algorithm design are given in appendix.

Chapter 4

Results

TODO: Deep architectures in PCP: [13]

4.1 Hyper-Parameter Optimization

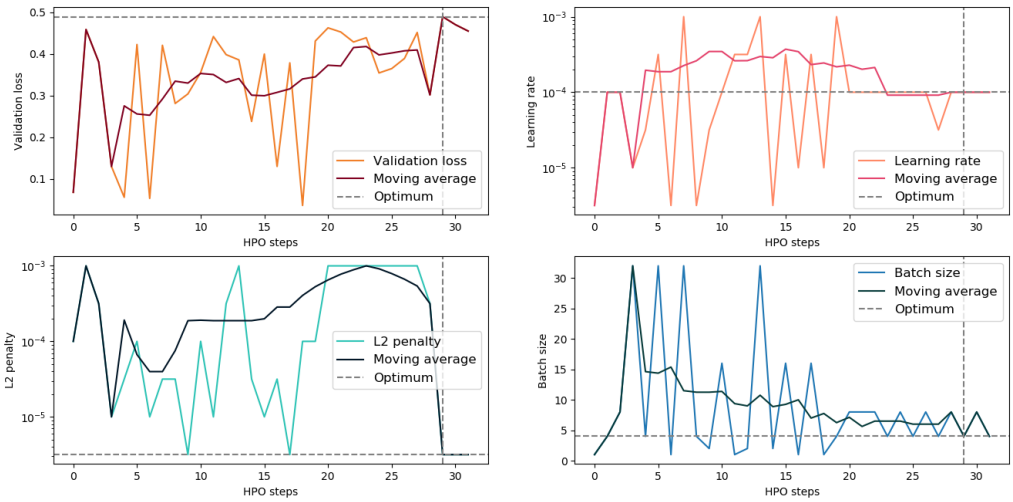

Figure 4.1: Performance and hyper-parameter values as a function of the number of Hyper-Parameter Optimization (HPO) iterations. Top left figure illustrates the optimal point whose value on x-axis is given by the HPO iteration that yields highest validation Best-L PPV. Optimal values for the learning rate, L2 penalty and batch size are denoted by dashed lines in top right, bottom left and bottom right figures, respectively.

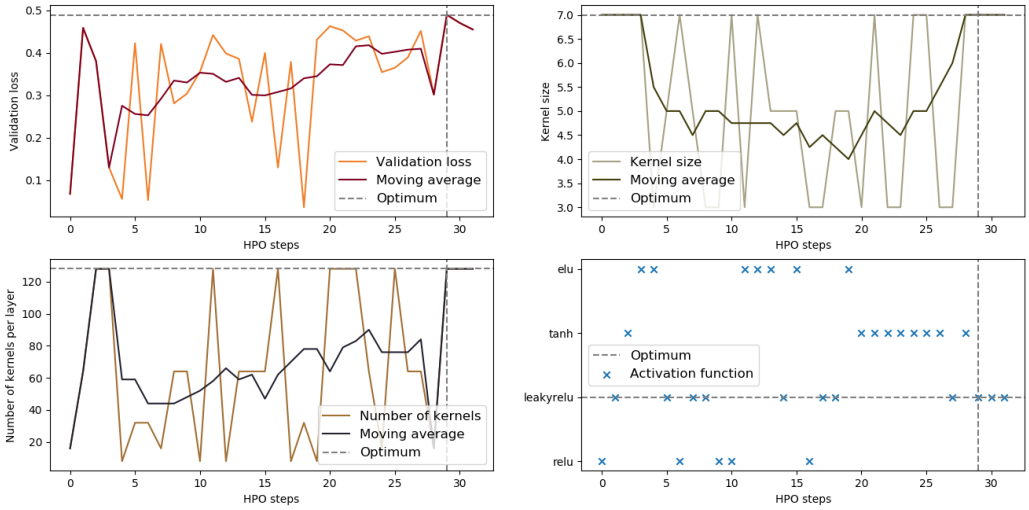

Figure 4.2: Performance and hyper-parameter values as a function of the number of Hyper-Parameter Optimization (HPO) iterations. Top left figure illustrates the optimal point whose value on x-axis is given by the HPO iteration that yields highest validation Best-L PPV. Optimal values for kernel size, number of kernels and activation function are denoted by dashed lines in top right, bottom left and bottom right figures, respectively.

Module	Hyper-parameter	Set of values
General	Batch size	4
	Batch normalization	$\top$
	Track running state	$\bot$
	Learning rate	10^{-4}
	L2 penalty	10^{-4}
	Parameter optimization	Adam
	Activation function	LeakyReLU
	Use global modules	$\top$
Global module	Depth	3
1-dimensional module	Depth	18
	Filter size	7
	Number of filters	128
2-dimensional module	Depth	18
	Filter size	7
	Number of filters	128

Table 4.1: Set of hyper-parameter values obtained at optimal point.

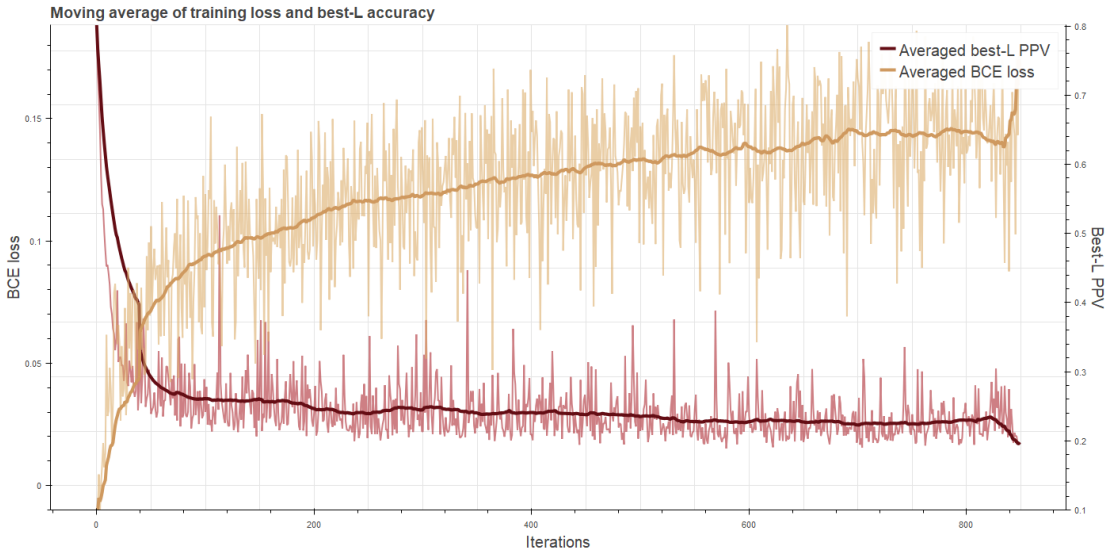

Figure 4.3: Binary Cross-Entropy loss and best-L PPV computed on a rolling window of batches during training phase of the model with the best hyper-parameters.

4.2 Model evaluation on the different benchmark sets

	CASP11			CAMEO			Membrane		
Method	Short	Medium	Long	Short	Medium	Long	Short	Medium	Long
Wynona	0	**0.43**	0.40	0	**0.31**	0.28	-	-	-
PconsC3	0.25	0.29	0.40	0.21	0.23	0.27	0.15	0.19	0.33
RaptorX-Contact	**0.28**	0.35	**0.55**	**0.23**	0.28	**0.42**	0.16	0.22	0.47
MetaPSICOV	0.26	0.31	0.39	0.22	0.22	0.28	0.16	0.21	0.35
PlmDCA	0.14	0.16	0.27	0.11	0.13	0.19	0.08	0.11	0.21
PSICOV	0.14	0.15	0.24	0.13	0.14	0.18	0.09	0.11	0.20
mfDCA	0.13	0.15	0.22	0.10	0.11	0.15	0.09	0.12	0.24

Table 4.2: Best-L PPV of different methods on short, medium and long-range contacts. Results are shown for the three different benchmark sets: CASP11 targets, CAMEO proteins, and the benchmark set of membrane proteins.

4.2.1 CASP11

TODO:

4.2.2 CAMEO

TODO:

4.2.3 Membrane proteins

TODO:

4.3 Sensitivity to the number of homologous sequences

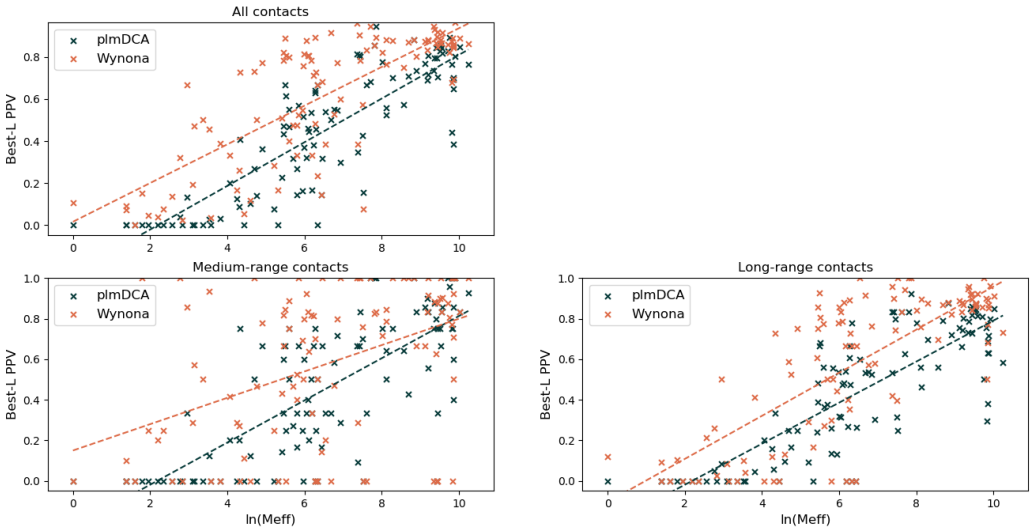

Figure 4.4: Performance as a function of the logarithm of the effective number of homologous sequences. Top figure shows the results on CASP11 targets for all contacts. Bottom left and bottom right figures focus on medium-range and long-range contacts, respectively.

4.4 Model performance by structural class

Automated assignment of CATH C classes: TODO: [48]

4.5 Folding proteins from contact maps

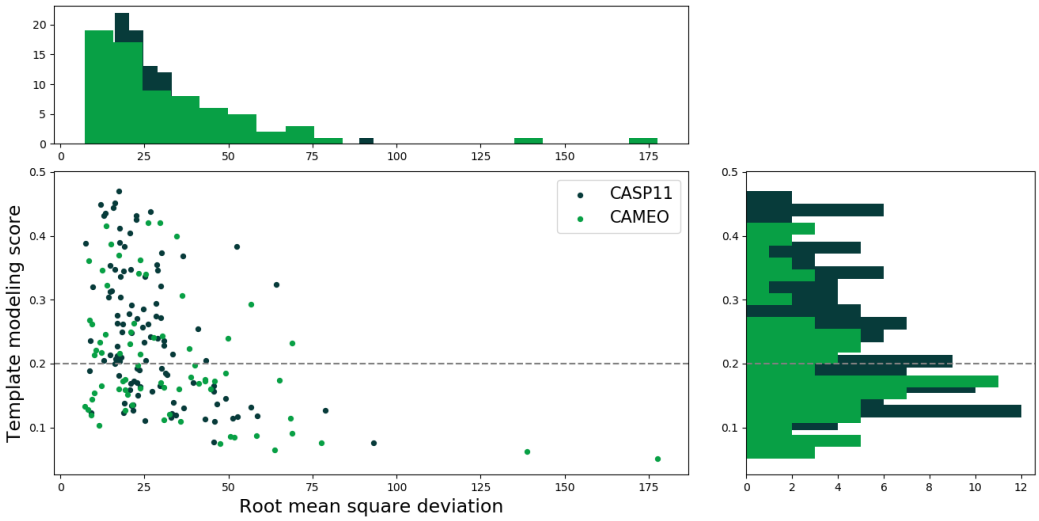

Figure 4.5: Root mean square deviations and template-modelling scores on the 3 benchmark sets.

TODO: tSNE: [63]

Chapter 5

Conclusion

Appendices

Appendix A

3D model assessment

A.1 Contact-assisted 3D modelling

This appendix describes the algorithm used to reconstruct proteins in three dimensions. Like GDFuzz3D [53], it uses graph distances to convert predicted contact maps in order to approximate distance maps. However, the proposed method is template-free and does not make use of MODELLER [22] as in GDFuzz3D.

A.1.1 Graph distances

Let's use the graph representation of contact maps as in section 2.1.2 about Protein Contact Networks. The graph distance between two residues is defined as the length of the shortest path between them. Predicted contact maps are converted to binary adjacency matrices by keeping only the $4.5\,L$ top predicted contacts.

A.1.2 Approximate Euclidean distances

As explained in [53], there is a linear relationship between graph distances and real euclidean distances. Let $GD_{i,j}$ be the graph distance between residues i and j. Then the euclidean distance $\mathrm{d}(\mathbf{x_i}, \mathbf{x_j})$ between the corresponding points x_i and x_j is approximated by:

$$\mathrm{d}(\mathbf{x_i}, \mathbf{x_j}) = 5.72 \times GD_{i,j} \tag{A.1}$$

It must be noted that the error on the euclidean distance also increases with graph distance. For all residue pairs for which no tractable information is available and no protein template is available, this estimated distance remains the best estimator.

A.1.3 Gaussian restraints

Each residue pair may be associated to at most one Gaussian restraint. A Gaussian restraint is defined by its mean and standard deviation, computed empirically over a set of distances with specific properties like graph distances, sequence separation or secondary structure.

As an example, residue pairs with a graph distance of one (residues are in contact) and a sequence separation of one have an average distance equal to the $C_\alpha - C_\beta$ distance (3.82 Å), and a standard deviation equal to 0.35 Å.

Restraint type	Seq. sep.	Mean	Standard deviation
Repulsion	≥ 6	20.00	120.00
Interior	-	5.00	10.00
Adjacent	1	3.81	0.1
Next adjacent	2	5.20	0.55
Next adjacent	3	7.00	0.71
Intra-alpha	1	3.82	0.35
Intra-alpha	2	5.50	0.52
Intra-alpha	3	5.33	0.93
Intra-alpha	4	6.42	1.04
Intra-beta	1	3.80	0.28
Intra-beta	2	6.66	0.30
Alpha/beta	≥ 4	6.05	0.95
Helix/coil	≥ 4	6.60	0.92
All	≥ 4	5.72 $\times$ GD	1.34 $\times$ GD

Table A.1: Gaussian restraints present in the 3D model

The set of points X that best satisfies Gaussian restraints is simply obtained by log-likelihood maximization:

$$\hat{X} = \underset{X}{\operatorname{argmax}} \sum_{i<j,\,(\mu_{i,j},\sigma_{i,j})\in R} \left(\frac{\delta(x_i, x_j) - \mu_{i,j}}{\sigma_{i,j}} \right)^2 \tag{A.2}$$

where R is the set of parameters of the Gaussian restraints. This notation is used because all residue pairs may not be restrained.

TODO: [54]

A.1.4 Evolutionary algorithm

Gaussian log-likelihood is maximized by a vanilla genetic algorithm. The initial population is obtained by adding random noise to the coordinates predicted by the multidimensional scaling algorithm. Parent selection is done by creating two random partitions from current population and keeping the individuals that maximize log-likelihood in each one. A new individual is then created by taking each point from either the first or the second parent, randomly. Mutation is simulated by adding Gaussian noise to each

point with a 50% chance. Finally, the individual with lowest log-likelihood is replaced by the newly created individual.

The set of hyper-parameters is composed of:

- Population size (default value is 2000)
- Partition size (default value is 50)
- Maximum number of iterations (default value is 200000)
- Standard deviation of mutation noise (default value is 10)

A.2 Evaluation metrics

$$\text{TM-score}(X^{(target)}, X^{(aligned)}) = \max_P \left[\frac{1}{L} \sum_{i=1}^{L} \frac{1}{1 + \left(\frac{\delta(x_i^{(target)}, x_i^{(aligned)})}{\delta_0} \right)^2} \right] \tag{A.3}$$

where $\delta_0 = 1.24\sqrt[3]{L-15} - 1.8$, and P is a projection that preserves

The best alignment in 3D is found by determining the projection of $X^{(aligned)}$ that either maximizes the TM-score or minimizes the RMSD. Such a projection has 9 parameters:

- 3 boolean parameters that indicate whether to swap coordinates along the X, Y and Z dimensions, respectively.
- 3 real-valued parameters for translating coordinates along the X, Y and Z dimensions, respectively.
- 3 angles that parametrize the rotation matrices around the X, Y and Z axes, respectively.

$$\begin{aligned} P(x) &= R_\phi^X R_\psi^Y R_\theta^Z x + b \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} x + \begin{pmatrix} b^X \\ b^Y \\ b^Z \end{pmatrix} \end{aligned}$$

A.3 Evaluation of GDE-GaussFold

Figure A.1: TM-scores and RMSD of different folding methods including GDE-GaussFold, with density functions on the sides.

Bibliography

[1] Badri Adhikari, Debswapna Bhattacharya, Renzhi Cao, and Jianlin Cheng. Confold: residue-residue contact-guided ab initio protein folding. *Proteins: Structure, Function, and Bioinformatics*, 83(8):1436–1449, 2015.

[2] Badri Adhikari and Jianlin Cheng. Confold2: improved contact-driven ab initio protein structure modeling. *BMC bioinformatics*, 19(1):22, 2018.

[3] William R. Atchley, Jieping Zhao, Andrew D. Fernandes, and Tanja Drüke. Solving the protein sequence metric problem. *Proc Natl Acad Sci U S A*, 102(18):6395–6400, May 2005. 15851683[pmid].

[4] Carlo Baldassi, Marco Zamparo, Christoph Feinauer, Andrea Procaccini, Riccardo Zecchina, Martin Weigt, and Andrea Pagnani. Fast and accurate multivariate gaussian modeling of protein families: Predicting residue contacts and protein-interaction partners. *PLOS ONE*, 9(3):1–12, 03 2014.

[5] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31 –39, 2011.

[6] Yoshua Bengio and Yann Lecun. *Scaling learning algorithms towards AI*. MIT Press, 2007.

[7] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, jul 2015.

[8] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[9] R. Bollapragada, D. Mudigere, J. Nocedal, H.-J. M. Shi, and P. T. P. Tang. A Progressive Batching L-BFGS Method for Machine Learning. *ArXiv e-prints*, February 2018.

[10] Gareth Chelvanayagam, Lukas Knecht, Thomas Jenny, Steven A Benner, and Gaston H Gonnet. A combinatorial distance-constraint approach to predicting protein tertiary models from known secondary structure. *Folding and Design*, 3(3):149–160, 1998.

[11] Hua Cheng, R. Dustin Schaeffer, Yuxing Liao, Lisa N. Kinch, Jimin Pei, Shuoyong Shi, Bong-Hyun Kim, and Nick V. Grishin. Ecod: An evolutionary classification of protein domains. *PLOS Computational Biology*, 10(12):1–18, 12 2014.

[12] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[13] Pietro Di Lena, Ken Nagata, and Pierre Baldi. Deep architectures for protein contact map prediction. *Bioinformatics*, 28(19):2449–2457, 2012.

[14] L. Di Paola, M. De Ruvo, P. Paci, D. Santoni, and A. Giuliani. Protein contact networks: An emerging paradigm in chemistry. *Chemical Reviews*, 113(3):1598–1613, 2013. PMID: 23186336.

[15] Wenze Ding, Wenzhi Mao, Di Shao, Wenxuan Zhang, and Haipeng Gong. Deepconpred2: An improved method for the prediction of protein residue contacts. *Computational and Structural Biotechnology Journal*, 16, 11 2018.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[17] Sean R. Eddy. Profile hidden markov models. *Bioinformatics (Oxford, England)*, 14(9):755–763, 1998.

[18] Magnus Ekeberg, Tuomo Hartonen, and Erik Aurell. Fast pseudolikelihood maximization for direct-coupling analysis of protein structure from many homologous amino-acid sequences. *Journal of Computational Physics*, 276:341 – 356, 2014.

[19] Magnus Ekeberg, Cecilia Lövkvist, Yueheng Lan, Martin Weigt, and Erik Aurell. Improved contact prediction in proteins: Using pseudolikelihoods to infer potts models. *Phys. Rev. E*, 87:012707, Jan 2013.

[20] Sara El-Gebali, Jaina Mistry, Alex Bateman, Sean R Eddy, Aurélien Luciani, Simon C Potter, Matloob Qureshi, Lorna J Richardson, Gustavo A Salazar, Alfredo Smart, Erik L L Sonnhammer, Layla Hirsh, Lisanna Paladin, Damiano Piovesan, Silvio C E Tosatto, and Robert D Finn. The pfam protein families database in 2019. *Nucleic Acids Research*, page gky995, 2018.

[21] R John Ellis and Saskia M Van der Vies. Molecular chaperones. *Annual review of biochemistry*, 60(1):321–347, 1991.

[22] Narayanan Eswar, Ben Webb, Marc A Marti-Renom, MS Madhusudhan, David Eramian, Min-yi Shen, Ursula Pieper, and Andrej Sali. Comparative protein structure modeling using modeller. *Current protocols in bioinformatics*, 15(1):5–6, 2006.

[23] R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T.F.G. Green, C. Qin, A. Zidek, A. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan, D.T. Jones, D. Silver, K. Kavukcuoglu, D. Hassabis, and A.W. Senior. De novo structure prediction with deep-learning based scoring.

[24] Gianluigi Forloni, Liana Terreni, Ilaria Bertani, Sergio Fogliarino, Roberto Invernizzi, Andrea Assini, Giuseppe Ribizzi, Alessandro Negro, Elena Calabrese, Maria Antonietta Volonté, et al. Protein misfolding in alzheimer's and parkinson's disease: genetics and molecular mechanisms. *Neurobiology of aging*, 23(5):957–976, 2002.

[25] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics (Oxford, England)*, 9:432–41, 08 2008.

[26] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and José García Rodríguez. A review on deep learning techniques applied to semantic segmentation. *CoRR*, abs/1704.06857, 2017.

[27] Vladimir Golkov, Marcin J Skwark, Antonij Golkov, Alexey Dosovitskiy, Thomas Brox, Jens Meiler, and Daniel Cremers. Protein contact prediction from amino acid co-evolution using convolutional networks for graph-valued images. In *Advances in Neural Information Processing Systems*, pages 4222–4230, 2016.

[28] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[29] William E Hart and Sorin Istrail. Robust proofs of np-hardness for protein folding: general lattices and energy potentials. *Journal of Computational Biology*, 4(1):1–22, 1997.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[31] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[32] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[33] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J Physiol*, 160(1):106–154.2, Jan 1962. 14449617[pmid].

[34] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[35] David T Jones. Protein secondary structure prediction based on position-specific scoring matrices. *Journal of molecular biology*, 292(2):195–202, 1999.

[36] David T. Jones, Daniel W. A. Buchan, Domenico Cozzetto, and Massimiliano Pontil. Psicov: precise structural contact prediction using sparse inverse covariance estimation on large multiple sequence alignments. *Bioinformatics*, 28(2):184–190, 2012.

[37] David T Jones and Shaun M Kandathil. High precision in protein contact prediction using fully convolutional neural networks and minimal sequence features. *Bioinformatics*, page bty341, 2018.

[38] David T. Jones, Tanya Singh, Tomasz Kosciolek, and Stuart Tetchner. Metapsicov: combining coevolution methods for accurate prediction of contacts and long range hydrogen bonding in proteins. *Bioinformatics*, 31(7):999–1006, 2015.

[39] David E Kim, Frank DiMaio, Ray Yu-Ruei Wang, Yifan Song, and David Baker. One contact for every twelve residues allows robust and accurate topology-level protein structure modeling. *Proteins: Structure, Function, and Bioinformatics*, 82:208–218, 2014.

[40] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[41] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[42] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[43] C Levinthal. How to fold graciously. mossbauer spectroscopy in biological systems: Proceedings of a meeting held at allerton house. *Monticello, Illinois (Debrunnder JTP, Munck E.., eds.)*, pages 22–24, 1969.

[44] Yang Liu, Perry Palmedo, Qing Ye, Bonnie Berger, and Jian Peng. Enhancing evolutionary couplings with deep convolutional neural networks. *Cell Systems*, 6(1), Jan 2018.

[45] Lorenzo Livi, Enrico Maiorino, Alessandro Giuliani, Antonello Rizzi, and Alireza Sadeghian. A generative model for protein contact networks. *Journal of Biomolecular Structure and Dynamics*, 34(7):1441–1454, 2016. PMID: 26474097.

[46] Grzegorz Markowski, Krzysztof Grabczewski, and Rafal Adamczak. Oversampling negative class improves contact map prediction. *Int J Pharma Med Biol Sci*, 5(4):211–216, 2016.

[47] Mirco Michel, David Menendez Hurtado, and Arne Elofsson. Pconsc4: fast, free, easy, and accurate contact predictions. *bioRxiv*, 2018.

[48] Alex D Michie, Christine A Orengo, and Janet M Thornton. Analysis of domain structural class using an automated class assignment protocol. *Journal of molecular biology*, 262(2):168–185, 1996.

[49] Faruck Morcos, Andrea Pagnani, Bryan Lunt, Arianna Bertolino, Debora S. Marks, Chris Sander, Riccardo Zecchina, José N. Onuchic, Terence Hwa, and Martin Weigt. Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proceedings of the National Academy of Sciences*, 108(49):E1293–E1301, 2011.

[50] Osvaldo Olmea and Alfonso Valencia. Improving contact predictions by the combination of correlated mutations and other sources of sequence information. *Folding and Design*, 2:S25 – S32, 1997.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[52] Jian Peng and Jinbo Xu. Raptorx: exploiting structure information for protein alignment by statistical inference. *Proteins: Structure, Function, and Bioinformatics*, 79(S10):161–171, 2011.

[53] Michal J Pietal, Janusz M Bujnicki, and Lukasz P Kozlowski. Gdfuzz3d: a method for protein 3d structure reconstruction from contact maps, based on a non-euclidean distance function. *Bioinformatics*, 31(21):3499–3505, 2015.

[54] MG Reese, Ole Lund, Jakob Bohr, Henrik Bohr, JE Hansen, and Søren Brunak. Distance distributions in proteins: a six-parameter representation. *Protein Engineering, Design and Selection*, 9(9):733–740, 1996.

[55] Michael Remmert, Andreas Biegert, Andreas Hauser, and Johannes Söding. Hhblits: lightning-fast iterative protein sequence searching by hmm-hmm alignment. *Nature Methods*, 9:173 EP –, Dec 2011.

[56] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[57] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[58] Stefan Seemayer, Markus Gruber, and Johannes Söding. Ccmpred–fast and precise prediction of protein residue-residue contacts from correlated mutations. *Bioinformatics*, 30(21):3128–3130, Nov 2014. 25064567[pmid].

[59] Robert Sheridan, Robert J. Fieldhouse, Sikander Hayat, Yichao Sun, Yevgeniy Antipin, Li Yang, Thomas Hopf, Debora S. Marks, and Chris Sander. Evfold.org: Evolutionary couplings and protein 3d structure prediction. *bioRxiv*, 2015.

[60] Marcin J Skwark, Mirco Michel, David Menendez Hurtado, Magnus Ekeberg, and Arne Elofsson. Accurate contact predictions for thousands of protein families using pconsc3. *bioRxiv*, 2016.

[61] Marcin J. Skwark, Daniele Raimondi, Mirco Michel, and Arne Elofsson. Improved contact predictions using the recognition of protein like contact patterns. *PLOS Computational Biology*, 10(11):1–14, 11 2014.

[62] Nikos Tsardakas Renhuldt. Protein contact prediction based on the tiramisu deep learning architecture. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2018.

[63] Laurens van der Maaten and Geoffrey E. Hinton. Visualizing data using t-sne. 2008.

[64] Marco Vassura, Luciano Margara, Pietro Di Lena, Filippo Medri, Piero Fariselli, and Rita Casadio. Ft-comar: fault tolerant three-dimensional structure reconstruction from protein contact maps. *Bioinformatics*, 24(10):1313–1315, 2008.

[65] Marco Vassura, Luciano Margara, Filippo Medri, Pietro di Lena, Piero Fariselli, and Rita Casadio. Reconstruction of 3d structures from protein contact maps. In Ion Măndoiu and Alexander Zelikovsky, editors, *Bioinformatics Research and Applications*, pages 578–589, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[66] Sheng Wang, Wei Li, Shiwang Liu, and Jinbo Xu. Raptorx-property: a web server for protein structure property prediction. *Nucleic acids research*, 44(W1):W430–W435, 2016.

[67] Sheng Wang, Wei Li, Renyu Zhang, Shiwang Liu, and Jinbo Xu. Coinfold: a web server for protein contact prediction and contact-assisted protein folding. *Nucleic acids research*, 44(W1):W361–W366, 2016.

[68] Sheng Wang, Siqi Sun, Zhen Li, Renyu Zhang, and Jinbo Xu. Accurate de novo prediction of protein contact map by ultra-deep learning model. *PLOS Computational Biology*, 13(1):1–34, 01 2017.

[69] Zhiyong Wang and Jinbo Xu. Predicting protein contact map using evolutionary and physical constraints by integer programming. *Bioinformatics*, 29(13):i266–i273, Jul 2013. 23812992[pmid].

[70] Martin Weigt, Robert A. White, Hendrik Szurmant, James A. Hoch, and Terence Hwa. Identification of direct residue contacts in protein-protein interaction by message passing. *Proc Natl Acad Sci U S A*, 106(1):67–72, Jan 2009.

[71] Dapeng Xiong, Jianyang Zeng, and Haipeng Gong. A deep learning framework for improving long-range residue–residue contact prediction using a hierarchical strategy. *Bioinformatics*, 33(17):2675–2683, 2017.

[72] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.