UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'Informatique

MEMO-F-524 - Master thesis:

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

Antoine Passemiers

Under the supervision of :

Dr. Daniele Raimondi
Co-supervisor of the thesis

Prof. Tom Lenaerts
Supervisor of the thesis

Academic year 2018 - 2019

Contents

Chapter 1

Introduction

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 environmental conditions are met, 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.

Protein Contact Prediction (PCP) can help determining the three-dimensional structure of proteins by limiting the search space to certain conformations forced by predicted contact maps. 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 [27].

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 well 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. 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 pastic-degrading capabilities. There are pipelines designed to predict the structure and then the function of a newly observed protein, such as RaptorX server [38].

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 [15]. 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 [24] or plmDCA [14]. 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.

In this thesis, common state-of-the-art ECA techniques and deep learning models for PCP are going to be described. ECA methods comprise Direct Coupling Analysis (DCA) and Pseudo-Inverse Covariance matrices (PSICOV), which can both be seen as examples of graphical models. TODO: More about what comes in the thesis

TODO: Levinthal's paradox

TODO: DCA are not sufficient -> Show the improvement of DL over DCA TODO: Make a comparison with DL state-of-the-art architectures TODO: Show DL invariance to the number of homologous sequences

TODO: DeepMind: [16]

1.1 Proteins

Some proteins called protein complexes are actually agreggations of multiple polypeptide chains. TODO:

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 carboxyle ($-COOH$). 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 carboxyle group. In the second case, it is called beta amino acid and has an amine group attached to the C_β of the carboxyle 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 [24] 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 [48]. 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.

TODO: Van der Waals radii?

2.1.1.1 Definition

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 connectivities 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 [11].

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:

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

TODO: Continue with [11]

[32] 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

Plot: Density of Euclidean distances among native contacts in PCNs

2.2 Direct Coupling Analysis

2.2.1 Potts model

2.2.2 Exact inference is hard

It has long been stated that three-dimensional structure of proteins have an impact regarding the amino acid composition among homologous proteins. Spatial contacts between residues can thus be inferred by analyzing 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.

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

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. Given a multiple sequence alignment containing B 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 symbol.

However, there is no straightforward method to compute the partition function Z or its gradients 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) [35], Message Passing (mpDCA) [51], Pseudo-Likelihood Maximization (plmDCA) [14] or Multivariate Gaussian Modeling (GaussDCA) [2] have been developed.

2.2.3 Pseudo-Likelihood Maximization

plmDCA [14] 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 $L2$ penalty term for both fields and couplings, and that each log-probability is weighted by a protein weight w_k. The latter is TODO: Also, 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 it requires high computational resources. An alternative method is to use GaussDCA [2], which does exact inference in a single step.

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, $X \in \{0,1\}^{m\times 22\,L}$ the matrix representation of the MSA, and $\mu, \bar{x}$ be, respectively, the theoretical and empirical mean vectors associated to it, 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 [24], evolutionary couplings are detected by keeping track of direct interations between variables of the system, what 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, empirical covariance matrix is rarely 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 inversible 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. $P(\theta|\alpha)$ 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 inversible covariance matrices exclusively) is defined by the NIW distribution:

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

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

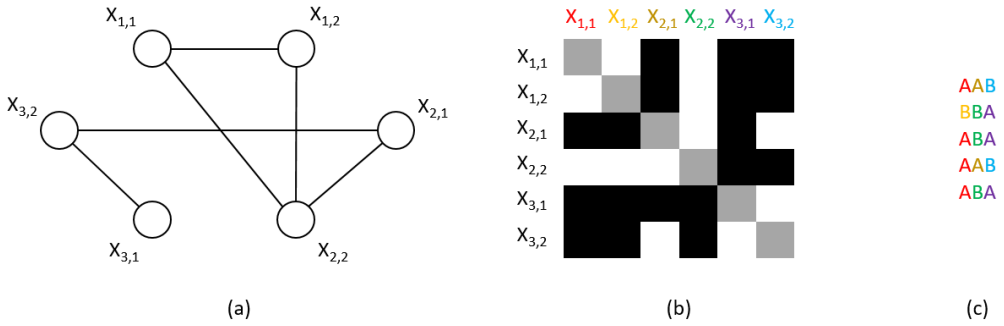

Figure 2.1: Illustration of a small system with only 3 sites and 2 amino acid types A and B: (a) Gaussian graphical model representing the system. (b) Sparsity pattern of the estimated inverse covariance matrix. Cells associated to zero variables are drawn in black. (c) Corresponding MSA.

TODO: Gaussian graphical models, precision matrix, etc.

The motivation behind PSICOV [24] 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. tWith this knowledge, the sample covariance matrix $S \in \mathbb{R}^{21L \times 21L}$ can then be computed by:

$$S_{ij}^{ab} = \frac{1}{n} \sum_{k=1}^{n} \left(x_{ia}^{(k)} - \bar{x}_{ia}\right)\left(x_{jb}^{(k)} - \bar{x}_{bj}\right) \tag{2.17}$$

where sn is the number of homologuous 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.

The solution chosen here is to estimate sparse inverse covariance matrices instead, by having recourse to Graphical Lasso algorithm [17]. The method assumes that observa-

tions follow a multivariate normal distribution characterized by the following probability density function:

$$f(x) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\Big(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\Big) \tag{2.18}$$

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:

$$\begin{aligned}
\log L(\Theta) &= \sum_{k=1}^{n} \log f(x^{(k)}) \\
&= \sum_{k=1}^{n} \left(-\frac{1}{2}(x^{(k)}-\mu)^T \Sigma^{-1}(x^{(k)}-\mu) - \log \sqrt{(2\pi)^d |\Sigma|} \right) \\
&= \sum_{k=1}^{n} \left(-\frac{1}{2}(x^{(k)}-\mu)^T \Theta(x^{(k)}-\mu) - \log \sqrt{(2\pi)^d \frac{1}{|\Theta|}} \right) \\
&= -\frac{1}{2}\sum_{k=1}^{n}(x^{(k)}-\mu)^T \Theta(x^{(k)}-\mu) - \frac{1}{2}\sum_{k=1}^{n} \log\left((2\pi)^d \frac{1}{|\Theta|} \right) \\
&= -\frac{1}{2}\operatorname{Tr}\left(S\Theta\right) - \frac{1}{2}\sum_{k=1}^{n} \log\left(2\pi\right)^d + \frac{1}{2}\log|\Theta|
\end{aligned} \tag{2.19}$$

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:

$$\begin{aligned}
\hat{\Theta} &= \underset{\Theta}{\operatorname{argmax}} \quad \log L(\Theta) - \rho \left\|\Theta\right\|_1 \\
&= \underset{\Theta}{\operatorname{argmax}} \quad -\frac{1}{2}\operatorname{Tr}\left(S\Theta\right) - \frac{1}{2}\sum_{k=1}^{n} \log\left(2\pi\right)^d + \frac{1}{2}\log|\Theta| - \rho\left\|\Theta\right\|_1 \\
&= \underset{\Theta}{\operatorname{argmax}} \quad -\operatorname{Tr}\left(S\Theta\right) + \log|\Theta| - \rho\left\|\Theta\right\|_1
\end{aligned} \tag{2.20}$$

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

$$X_u \perp\!\!\!\perp X_v | X_{V\setminus\{(u,v)\}} \quad \text{if} (u,v) \notin E \tag{2.21}$$

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.

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

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 $\delta_0 : \mathbb{R}^{21L \times 21L} \rightarrow \mathbb{R}^{L \times L}$ 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 articifical 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 [4].

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:

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

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 [41] or quasi-Newton methods [7]. 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_m of layer m (for any layer m with learnable parameters), can be decomposed as the following product:

$$\nabla \prec(\theta_m) = \prod_{i=1}^{m} f'((\bigcirc_{p=1}^{i} f_{\theta_i})(X)) \cdot L'((\bigcirc_{j=1}^{n} f_{\theta_j})(X)) \tag{2.27}$$

TODO: Replace prime symbols by actual partial derivatives

Each factor i of the product can be computed using the definition of function f_i', and the current input to layer i. However, layer i requires the factor from layer $i+1$ in order to compute loss gradient according to its own parameters. Consequently, the signal (the product of factors from layer n to current layer p) is passed from layer $p+1$ to layer p. In a more general sense, the gradient signal is passed from the ouput 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:

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

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 [7] 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:

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

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

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

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

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 [20], 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) [18]. A CNN is an artificial neural network made of a stack of neural layers [29]. 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 [22].

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 layers, since a linear combination of linear combinations is itself a linear combination. Thus, activation functions help to actually build a hierarchical representational of the data by folding the hyperplane containing the 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 computation requirements for the gradients. Most of the activation functions are non-prametric 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 [30] 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 classficaition, 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 previous layers. Because the norm of gradients w.r.t. activation functions is always less than one for both tanh and standard sigmoid, 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 are rarely 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 [23], 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 outputs 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 training algorithm with less careful parameter initialization and larger learning rate.

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

Name	Number of samples involved	Formula
Average (true) gradient	N	$\frac{1}{N}\sum_{i=1}^{N}\nabla L(f_\theta(X_i))$
Batch gradient	$\|B\|$	$\frac{1}{\|B\|}\sum_{i\in B}\nabla L(f_\theta(X_i))$
Sample gradient	1	$\nabla L(f_\theta(x))$

Table 2.1: Types of gradients and gradient approximations used in common optimization methods.

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, L1 and L2 regularizations are going to be introduced in the following two sections.

2.4.7.1 L1 regularization

Adding a L1 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 L2 regularization

L2 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

The gradient vector is obtained by concatenating the gradients w.r.t. each layer's parameter vector. This final gradient vector gives an improvement direction, but a decrease of the loss function is guaranteed by moving by a arbitrary small step in the parameter space.

TODO: Overview: [41]

TODO: Adam: [28]

TODO: Adadelta: [52]

TODO: L-BFGS: [7]

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.

2.4.9.1.3 Predictors standard deviation

2.4.9.2 1-dimensional features

2.4.9.2.1 One-hot encoded sequence Given an amino acid sequence $\{s_1, s_2, \ldots, s_L\}$ 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 TODO:

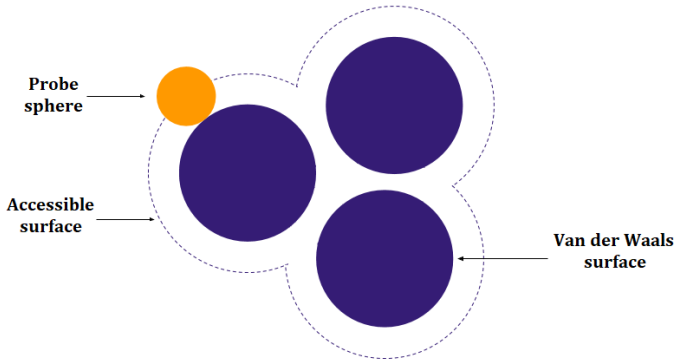

Figure 2.2: Accessible surface, obtained by "rolling" a probe sphere (a molecule of solvent, colored in orange) on the Van der Waals surface of a biomolecule (colored in blue).

2.4.9.2.3 Predicted Secondary Structure Prediction

2.4.9.2.4 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 ma-

trix 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:

$$F_{ia} = \frac{1}{M}\sum_{k=1}^{M}\delta(S_{ki}, a) \tag{2.49}$$

2.4.9.2.5 Position-Specific Scoring Matrix

2.4.9.2.6 Atchley factors TODO: Atchley: [1]

2.4.9.2.7 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 [34] can be formalized with the following equation:

$$I_{ij} = \log_2(p_{ij}/\langle p_i\rangle) \tag{2.50}$$

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.

2.4.9.2.8 Partial entropies [34]

$$S_i = p_i \log_2(p_i/\langle p_i\rangle) \tag{2.51}$$

2.4.9.3 2-dimensional features

2.4.9.3.1 Mutual Information and Normalized Mutual Information Following the formalism described in [34], MI is described as:

$$MI(x, y) = \sum_{x,y} P(x, y)\log\left(\frac{P(x, y)}{P(x)P(y)}\right) \tag{2.52}$$

$$NMI(x, y) = \frac{MI(x, y)}{\sqrt{S(x)S(y)}} \tag{2.53}$$

APC is applied to both MI and NMI.

2.4.9.3.2 Cross-entropy Cross-entropy is computed in [34] using the following formula:

$$H(x, y) = S(x) + S(y) - MI(x, y) \tag{2.54}$$

2.4.9.3.3 Contact potential

2.4.9.3.4 Predictions from DCA and PSICOV

2.4.9.3.5 Covariance

2.4.10 Features, by method

Method			DeepCov	DeepContact	PConsC4	DNCON2	RaptorX	Proposed method
Features	Global features	Protein length				x		x
		Meff				x		x
		Column log-entropy		x				
		Predictors stdv		x				
	1-dim features	Encoded sequence			x			x
		Solvent accessibility		**x**		**x**	**x**	
		Predicted SS		**x**		**x**	**x**	
		AA frequencies		x	x			
		PSSM				x	x	
		Atchley factors				x		
		Self-information			x			x
		Partial entropies			x			x
	2-dim features	MI			x	x		x
		Normalized MI		x	x	x		x
		Cross-entropy		x	x			x
		Contact potential						
		EVFold		**x**				
		CCMPred		**x**		**x**		
		plmDCA				**x**		**x**
		GaussDCA			**x**			**x**
		PSICOV						**x**
		Covariance	x					
		<unknown>					x	

Figure 2.3: Features used in state-of-the-art deep learning approaches. Feature extraction methods that rely on external tools (excluding MSA tools) are highlighted in red.

2.5 Deep learning and PCP

TODO: DNCON2: [25]

TODO: PConsC4: [34], zero-padding, based on U-net: [40]

TODO: DeepContact: [31]

TODO: RaptorX-Contact: [49], zero-padding

TODO: DeepCov: [25]

TODO: TiramiProt: [46]

TODO: [12]

TODO: proteinloopmodeling

Benchmark with both accuracy and standard deviation w.r.t. proteins.

TODO: Deep architectures in PCP: [10]

TODO: ResNet: [19] TODO: DenseNet: [21]

2.5.1 Common features

Chapter 3

Materials and methods

3.1 Datasets

Six datasets have been used in the framework of this thesis: one for training the model, one for optimizing the hyper-parameters, and four for benchmarking. The training and validation sets together compose the 180 protein families used by Michel and Skwark [44]. These 180 families comprise the 150 protein families reported in the original PSICOV article [24], as well as 30 protein families selected from the test set of PConsC2 [45].

The first test set is composed of 210 proteins having no homology to the training set or the validation set. To achieve this goal, homology reduction has been applied to the 383 proteins used in the testing of PConsC2. This homology reduction was conducted by ensuring that no protein in the test set is assigned the same ECOD H-class [8] as any protein from the training or validation set.

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

TODO: Profile HMMs: [13] MSAs have been created using HHblits [39] (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's MSA 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
- MSA obtained using HHblits on the corresponding protein family
- Atom 3D coordinates

- PhyCMAP [50] intermediate predictions
- plmDCA [14] intermediate predictions
- GaussDCA [2] intermediate predictions
- Predictions made by PConsC3 [44] at each layer of the model
- CCMPred [42] predictions (only available in the 4 test sets)
- EVFold [43] predictions (only available in the 4 test sets)
- PSICOV [24] predictions (only available in the 4 test sets)
- MetaPSICOV [26] 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: [33]

3.2 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.

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	$L \times 2 \cdot 22$
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 3.1: 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

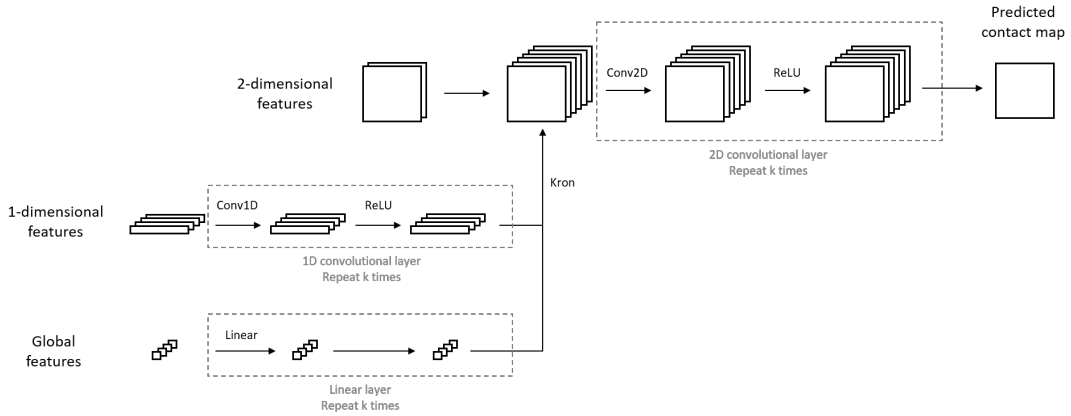

Figure 3.1: Proposed architecture of the deep convolutional neural network for semantic segmentation

3.4 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 [36]. 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. [45]:

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

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 [5] 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) [6]. 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, each hyper-parameter is associated 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	$\{8, 16, 32, 64\}$
2-dimensional module	Depth	$\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$
	Filter size	$\{3, 5, 7\}$
	Number of filters	$\{8, 16, 32, 64\}$

Table 3.2: Hyper-parameter space for the proposed architecture.

3.6 Implementation

3.6.1 Availability

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

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 [9].

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.

TODO: Scikit-learn [37] TODO: Cython [3]

Chapter 4

Results

4.1 Model evaluation on the differents benchmark sets

4.2 Invariance to the number of homologous sequences

TODO: tSNE: [47]

Chapter 5

Conclusion

Bibliography

[1] 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].

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

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

[21] 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.

[22] 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].

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

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

[42] 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].

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

[48] 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.

[49] 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.

[50] 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].

[51] 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.

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