MSc. Eng. Thesis
Master of Science in Engineering

A Deep Learning Approach to Protein Tertiary Structure Prediction

DTU Compute
Department of Applied Mathematics and Computer Science

Author: Lasse Blaabjerg, s132314
Supervisor: Ole Winther

Kongens Lyngby July 2018

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Richard Petersens Plads
Building 324
2800 Kongens Lyngby, Denmark
Phone +45 45 25 30 31
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

The protein folding problem is one of the biggest unsolved challenges in modern biology. Although great advancements have been made, accurate de novo prediction of protein tertiary structure remains notoriously difficult. At the same time, deep learning techniques have been responsible for remarkable breakthroughs within a wide range of computational fields during recent years. In this project, a deep learning approach to protein tertiary structure prediction is presented. This approach leverages the recently introduced concept of end-to-end differentiability, which ensures that a model can be optimized to predict complete protein structures from scratch using only sequence information and a suitable training set. In addition to the concept of end-to-end differentiability, the proposed model utilizes a combination of state-of-the-art deep learning architectures, such as residual (convolutional) neural networks and long short term memory cells, as well as an auxiliary local loss function in order to leverage the advantages of multitask learning. Using this approach, the model is trained and tested on data sets mimicking the conditions of the CASP 11 competition and a number of experiments are carried out in order to investigate the effects of key parameter choices. It is found, that performance is improved by: i) combining recurrent and convolutional neural network architectures, ii) increasing the number of learnable parameters used for angular prediction, iii) balancing the global and local loss signals. A distance based root mean square deviation of 13.2 Å on the CASP 11 test set is reported for the best performing model. The project ends with a brief discussion regarding key challenges suggesting a number of different paths forward towards improved predictive performance.

Preface

This Master's thesis has been prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the official requirements for the degree of Master of Science in Engineering in Bioinformatics and Systems Biology.

Kongens Lyngby, July 2, 2018

Lasse Blaabjerg

Acknowledgements

I would like to thank my supervisor Professor Ole Winther for his careful guidance and valuable input throughout this project. Also, I would like to thank Mohammed Al Quraishi, Systems Biology Fellow at Harvard Medical School, for the detailed mail correspondence in which he helped me understand and apply his novel protein structure prediction model as well as the newly developed ProteinNet data sets.

Contents

CHAPTER 1

Introduction

In the following chapter, the protein folding problem, as well as its relation to the problem of protein tertiary structure prediction, will be introduced. Furthermore, a short motivation regarding the need for protein structure prediction will be given and a brief overview of current methods in protein structure prediction will be presented. Finally, the concept of deep learning, as well as the possible advantages of deep learning in protein structure prediction, will be introduced.

1.1 The protein folding problem

Proteins are large biological molecules ubiquitously present in all living organisms. Within the cell, proteins are responsible for a diverse set of essential tasks including enzymatic activity, cell signaling, signal transduction, replication and regulation of genetic material, transportation and more [4]. The ability of proteins to carry out such a diverse range of tasks is a result of the proteins' ability to bind specifically and tightly to various other biological molecules. These specific binding properties are known to be directly determined by the geometrical structure of the protein, which is uniquely encoded in the proteins' string of amino acid monomers [4].

The importance of proteins within the living cell and the unique relationship between the protein's amino acid sequence, the protein's geometrical structure and the protein's biological properties has made structural biology a centerpoint of modern applied biology. The field was pioneered by Max Perutz and John Kendrew, who were awarded the Nobel Prize in Chemistry in 1962 for their work in determining the structure of globular proteins. Upon seeing the structure of myoglobin at 6 Å resolution for the first time, Kendrew et al. noted that:

> *"Perhaps the most remarkable features of the molecule are its complexity and its lack of symmetry. The arrangement seems to be almost totally lacking in the kind of regularities which one instinctively anticipates, and it is more complicated than has been predicated by any theory of protein structure. Though the detailed principles of construction do not yet emerge, we may hope that they will do so at a later stage of the analysis."* [32]

The unexpected complexity of protein structures led to a myriad of scientific questions, which has been distilled into what is now known as the protein folding problem. The protein folding problem refers to three broad questions: i) What is the physical code by which an amino acid sequence dictates a protein's native structure? ii) How can proteins fold as fast as they do? iii) Can we devise a computer algorithm to predict protein structures from their sequences alone [16]? Today, remarkable progress has been made on the protein folding problem since it was first posed a half-century ago. It is now known, that the protein folding process is governed by free energy minimization and that proteins fold rapidly because random thermal motions

cause conformational changes leading energetically downhill towards the native structure in a funnel-shaped energy landscape [16]. Furthermore, this energy landscape and its related native structure is known to be uniquely determined by the protein's amino acid sequence - a property commonly referred to as Anfinsen's thermodynamic hypothesis [5]. However, a range of more detailed questions remain unanswered e.g. in relation to the nature of different folding routes and the precise mechanisms behind folding diseases such as Alzheimer's. Furthermore, only limited progress has been made on the challenges of protein structure prediction.

1.2 The need for high-accuracy protein structure prediction

Today, the protein folding problem is regarded as one of the greatest unsolved challenges in modern biology [16]. As an example, *Science Magazine* included the protein folding problem in its 2005 list of the 125 most important unsolved problems in science due to its foundational nature and its enormous practical potential. Currently, basic and applied research in fields such as rational drug design and bioengineering is severely burdened by the high cost and labor intensiveness of experimental protein structure determination - particularly for proteins without close homologs. For example, the cost of solving a new unique protein structure with a combination of crystallography and NMR has been estimated to be on the order of $100,000 with modern-day equipment [41]. In contrast, an accurate protein structure prediction tool would be able to predict the same protein structure immediately and at zero marginal cost, thereby allowing researchers to design and alter specific protein structures from scratch at a very low cost. This type of breakthrough would drastically accelerate the pace of innovation within medicine, biotechnology and related fields.

1.3 Approaches to protein structure prediction

Protein structure prediction is notoriously difficult due to the extremely complex interactions between residues and the exponentially growing number of possible structural conformations (see Section 1.5). Historically, protein structure prediction methods have roughly been divided into two major approaches. The first approach attempts to solve the prediction task by directly simulating the folding process using approximations of the physical forces that are known to dominate. These forces include entropy, electrostatics, van der Waals interactions, solvation and others which are typically collected in different types of force-field approximations. Molecular dynamics (MD) is an example of this approach [39]. Such methods have the advantage of being derived from first principles making them able to simulate novel protein structures at high accuracy. However, in practice, MD is only accurate at folding prediction for very small proteins and is currently too computationally expensive for many real-world tasks [39].

As an alternative to the physics-based approach, the second approach attempts to tackle the problem from a statistical perspective. This can be done either on a global scale, such as in template- and threading-based methods, or on a local scale such as in fragment assembly-based methods [18]. Traditional fragment assembly methods work by splitting the protein sequence into fragments, searching various protein databases for close fragment homologs and combining these fragments through energy-minimization of various statistically derived energy functions in combination with some form of scored sampling process [18]. An example of this approach is the widely used Rosetta protocol [21]. Although successful in many cases, fragment assembly

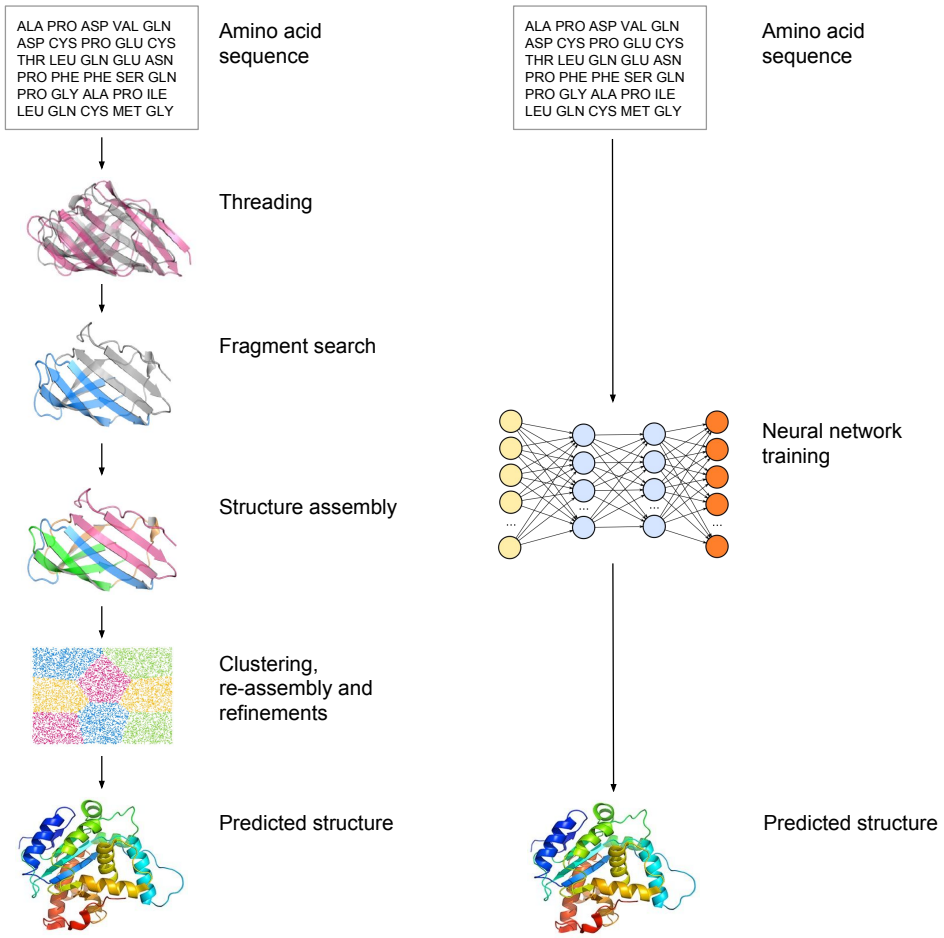

Figure 1.1: Highly simplified illustration of a modern protein prediction pipeline (here: I-TASSER [57]) versus a stand-alone deep learning-based approach.

heavily relies on the presence of close fragment homologs and often requires a great deal of manual refinements [18]. In practice, fragment assembly methods are often combined with MD techniques as well as the addition of so-called co-evolutionary information. When two residues are in close spatial contact in a protein, they will often mutate and evolve under the same evolutionary patterns. By exploiting these subtle relationships, co-evolutionary information can be used to estimate the inter-residue distances of a protein using information from its close homologs [31].

A modern state-of-the-art protein structure prediction pipeline will often combine several of the above mentioned techniques in numerous complicated and labours steps. As an example, a standard protein prediction pipeline will usually begin by analyzing the raw protein sequence to detect structural domains that can be independently modelled and then run a series of algorithms to predict various characteristics such as propensity for secondary structure formation, solvent accessibility and disordered regions. In co-evolutionary methods, a multiple sequence alignment is used to predict a map of intra-protein residue contacts and in template-based methods various databases are searched for structural templates that can serve as a basis

for prediction. These sources of information are then converted into geometric constraints to guide a fragment assembly process where a range of selected protein fragments are evaluated as place-ins on the preliminary structure in terms of some specifically tailored energy function. The entire process can take anywhere from hours to days and span several million lines of code [2].

1.4 Deep learning for protein structure prediction

Due to the great complexity and limitations in modern protein structure prediction pipelines, there has been an increasing interest in more powerful, single-stage prediction techniques. Deep learning has emerged as a promising candidate for in this field (see e.g. the highly simplified comparative illustration in Figure 1.1). Essentially, deep learning is a class of machine learning algorithms that are able to adjust their own parameters by making repeated computations along multiple and numerous (i.e. ”deep”) subsequent layers of non-linear processing units [42]. In recent years, deep learning models have produced breakthrough results within a wide range of computational fields, such as computer vision, speech recognition, natural language processing and more [42]. Although still in its early stages, deep learning techniques have also produced remarkable results within computational biology and protein structure prediction specifically [56][2].

Compared to traditional protein prediction pipelines, deep learning models rely on a great amount of initial work in training and fine-tuning the model. However, once developed, the deep learning models are able to make accurate predictions in milliseconds. This high-speed predictive power also allows deep-learning models to be used in completely new applications, such as docking and virtual screening, where traditional protein prediction pipelines today are too slow to function efficiently [2]. Although this project focuses on the prediction of protein backbone coordinates using sequence information only, any kind of protein structure prediction task that is usually carried out by a traditional protein structure prediction pipeline can in principle be either replaced or combined with a deep learning network. This includes traditional fragment-based methods, side-chain conformations predictions, secondary structure prediction and more.

1.5 The protein linguistic hypothesis

It has been hypothesized that the proven effectiveness of deep learning techniques in protein structure prediction is in part rooted in the common evolutionary relationship between all proteins [2]. It is well established [4] that protein domains can be shared between proteins due to evolutionary pressure. However, the protein linguistic hypothesis postulates that this phenomenon of reuse may go further, where even smaller protein fragments may be shared thus reflecting an underlying evolutionary pressure to reuse working fragments that fold in tried-and-tested ways [2]. In this case, protein structure should exhibit a hierarchical organization spanning small fragments to entire domains. This hierarchy of structures would be similar to a form of protein grammar that would be highly abstract but most importantly discoverable and learnable [2]. The protein linguistic hypothesis coupled with recent experimental results suggest that deep learning may play a crucial part in finally solving the missing part of the protein folding problem.

CHAPTER 2

Theory

In the following chapter, basic theory relevant to this project will be introduced. First, fundamental theory regarding protein structure classification including the geometrical constraints of protein folding will be introduced. Then, the Natural Extension Reference Frame algorithm will be introduced as a method for efficiently transforming protein angles to Cartesian coordinates. Finally, basic theory behind the deep learning techniques used in this project will be introduced, including different network architectures, the advantages of multitask learning, global and local loss functions and the optimizer algorithm.

2.1 Protein structure

2.1.1 Primary, secondary, tertiary and quarternary structure

Traditionally, protein structure characterization is divided into four hierarchical layers: Primary, secondary, tertiary and quarternary structure [4]. Each of these will be introduced briefly below.

On the primary structure level, proteins are viewed as biopolymers made out of sequences of amino acid monomers linked by amide bonds. In nature, a total of 20 different amino acids are observed, each containing a free amino group (NH_2) and a free carboxyl group (COOH) both bound to a central carbon atom (C_α). The central carbon atom is bound to a hydrogen atom as well as a side chain group (R) that uniquely determines the identity of the amino acid. During protein synthesis, amide bonds are formed through a condensation reaction (thereby producing an amino acid "residue") [4]. It is this sequence of bonded amino acid residues that determine the primary structure of the protein and thereby the secondary, tertiary and quarternary structure of the protein in accordance with Anfinsen's thermodynamic hypothesis [5].

On the secondary structure level, proteins consist of local segments of a few well-defined structural elements. The number of secondary structure elements depends on annotation, with either three or eight elements used namely the helix, the β-strand and the random coil or the alpha-helix, the 3_{10}-helix, the π-helix, the β-strand, the β-bridge, the turn, the bend and the coil [4]. Each type of protein secondary structure element is defined by its repeating set of local bond angles as illustrated in the so-called Ramachandran plot in Figure 2.2.

On the tertiary structure level, proteins are viewed as polypeptide chains that fold freely in three-dimensional space. The final protein fold is usually complex and unorganized, however, recognizable structural motifs and domains (e.g. the beta hairpin) is often observed across different types of proteins. These tertiary (or "super-secondary") structure motifs are often reused across different kinds of proteins due to their ability to accomplish certain biological

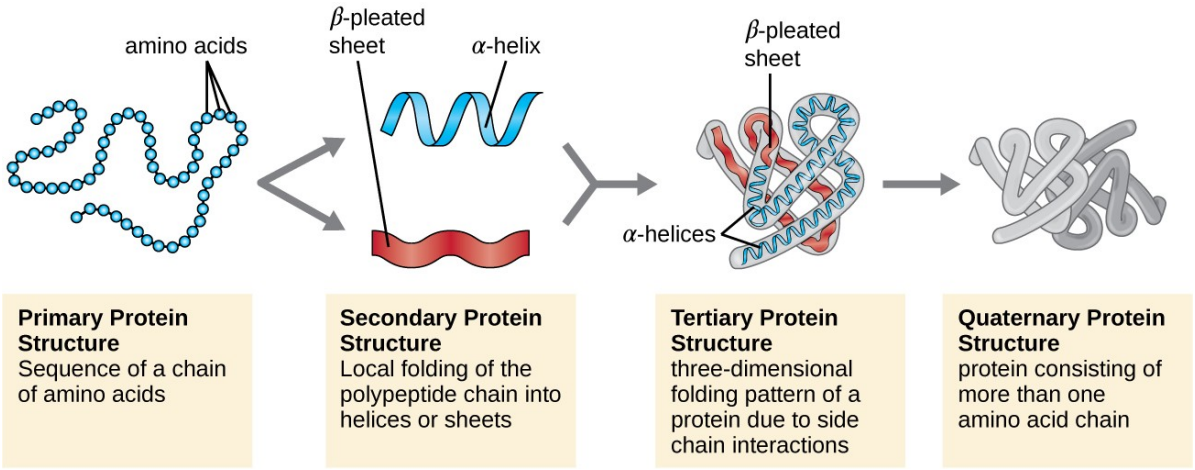

Figure 2.1: Illustration of the primary, secondary, tertiary and quarternary structure of a protein [37].

functions. One example is the zinc-finger motif consisting of two beta strands with an alpha helix folded over to bind a zinc ion - a motif that is very important in various DNA binding proteins [4].

On the quarternary structure level, two or more protein molecules can on occasion bond together in large multi-subunit complexes. The sizes of these complexes range from simple dimers (two subunits) to large oligomers usually found in some form of symmetrical arrangement. This arrangement of different protein molecules into a single unit can help accomplish various biological tasks, such as enhanced stability, coordinated regulation and cooperativity [4].

2.1.2 Dihedral angles

The end-to-end curvature of the protein backbone is what defines the geometric shape of the protein. This curvature can be uniquely defined by three dihedral angles: The ϕ dihedral angle between the C_i, N_{i+1}, $C_{\alpha,\,i+1}$ and C_{i+1} atoms (indices referring to the residue number), the ψ dihedral angle between the N_i, $C_{\alpha,\,i}$, C_i and N_{i+1} atoms and the ω dihedral angle between the $C_{\alpha,\,i}$, C_i, N_{i+1} and $C_{\alpha,\,i+1}$ atoms. These angles are illustrated in Figure 2.3. In nature, only a subset of possible ϕ and ψ angles are energetically favorable. These energetically allowed dihedral angles define the different secondary structure elements as illustrated in the Ramachandran plot in Figure 2.2. In the Ramachandran plot, the ω angle is usually omitted as it is almost always found in nature at a fixed 180 degrees, except in rare cases where the ω angle is close to 0 degrees [4].

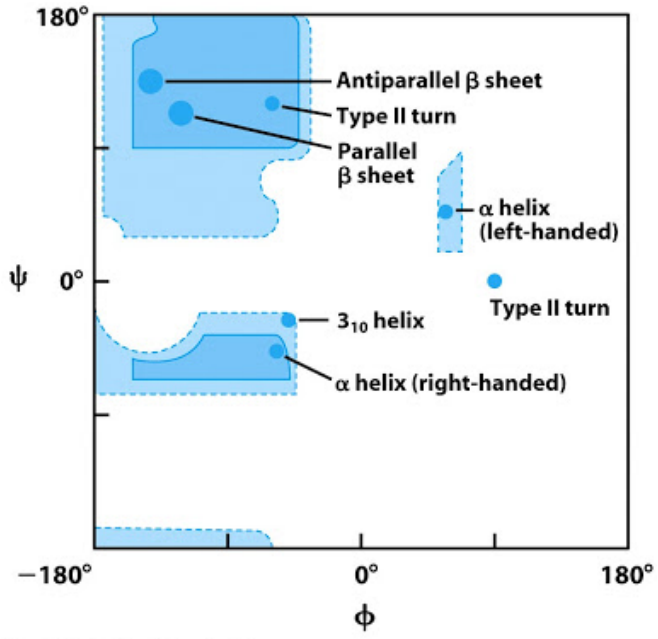

Figure 2.2: Ramachandran plot of naturally occurring ϕ and ψ angles and related secondary structure elements [27].

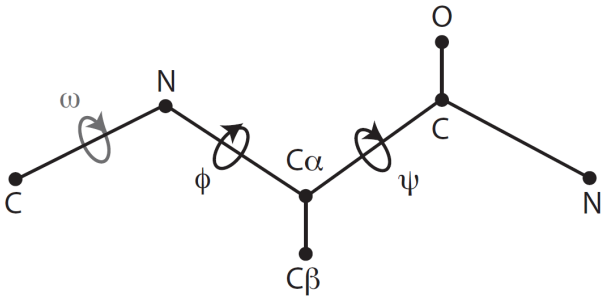

Figure 2.3: Illustration of the three dihedral angles of the protein backbone: ϕ, ψ and ω [10].

2.2 The Natural Extension Reference Frame algorithm

The so-called Natural Extension Reference Frame (NeRF) algorithm was used in order to convert a predicted set of dihedral angles to atomic backbone coordinates in a computationally efficient manner [44]. In practice, this is done by building the protein chain from scratch, placing each new backbone atom after the other in a step-wise fashion. Intuitively, this is done by first placing the new atom a bond length away from the previous atom in line with the previous bond axis. Then this new bond is rotated so that the position agrees with the desired bond and dihedral angles. The rotation is done in two steps, first setting the angle between the two bonds, then rotating about the previous bond to the correct dihedral angle. Examples of such two-step algorithms are General Rotations, Quaternions and the Rodrigues-Gibbs formula [44].

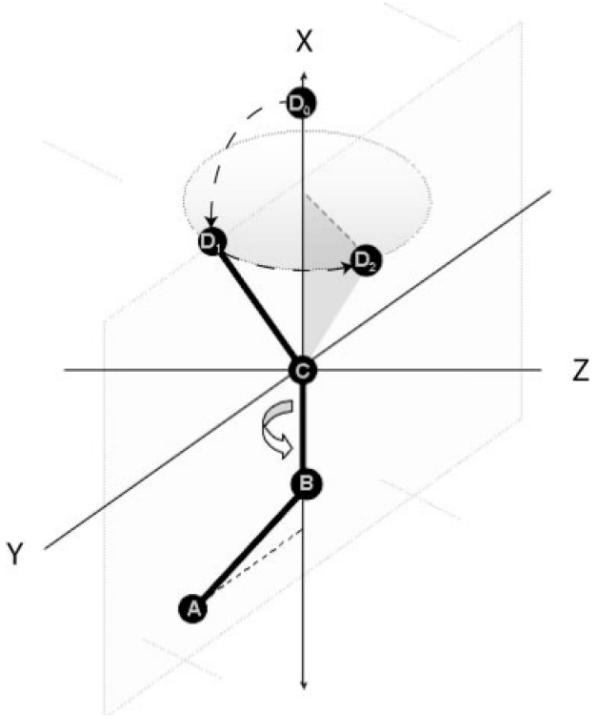

Figure 2.4: The dihedral angle to Cartesian coordinate transformation. In two-step approaches, the new atom is first placed at position $\mathbf{D}_0$. It is then first rotated around **C** by the bond angle to position $\mathbf{D}_1$ and then rotated around the **BC** bond axis by the dihedral angle to position $\mathbf{D}_2$. In the NeRF method, the atom is placed directly into an xyz coordinate system at position $\mathbf{D}_2$ (i.e. $\mathbf{D}_{init}$ in Equation 2.1). This coordinate system is then transformed so that the xy plane lies in the ABC plane. This transformation is applied to $\mathbf{D}_2$ [44].

A different way of performing this task is to first place the new atom in an default coordinate system where the position can be easily computed, and then rotate this coordinate system in a single step into the reference frame of orientation defined by the three previous atoms [44]. This approach is what defines the NeRF algorithm. In comparison to traditional two-step approaches, the NeRF algorithm is superior in that it requires significantly fewer floating point operations and demonstrates the best intrinsic numerical stability. The NeRF method is illustrated in Figure 2.4. Assuming a frame of reference where atom **C** is at the origin, atom **B** on the negative x-axis and atom **A** lies in the xy-plane, the atom **D** is placed at the initial position [44]:

$$\mathbf{D}_{init} = (l\cos(\theta)\,,\quad l\cos(\phi)\sin(\theta)\,,\quad l\sin(\phi)\sin(\theta))\,, \tag{2.1}$$

where

$$l = \text{bond}_{CD}\,,\quad \theta = \text{angle}_{BCD}\,,\quad \phi = \text{torsion}_{BC}\,. \tag{2.2}$$

This initial placement $\mathbf{D}_{init}$ is independent of the positions of **A**, **B** and **C** and depends solely on the bond length and bond angles.

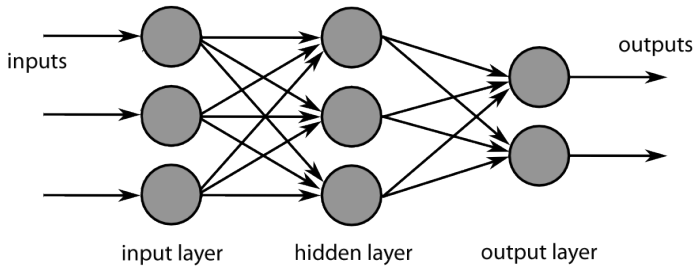

Figure 2.5: Illustration of a simple FNN architecture with three input units (i.e. three input dimensions), a single hidden layer with three hidden units and two output units (e.g. for use in binary classification).

To obtain the final position $\mathbf{D}$, the initial frame is transformed to the protein's frame of reference via the rotation matrix $\mathbf{D}$:

$$\mathbf{D} = \mathbf{M}\,\mathbf{D}_{init} + \mathbf{C}\,, \tag{2.3}$$

where

$$\mathbf{M} = [\hat{\mathbf{bc}}\,, \quad \hat{\mathbf{n}} \times \hat{\mathbf{bc}}\,, \quad \hat{\mathbf{n}}] \tag{2.4}$$

and

$$\hat{\mathbf{bc}} = \frac{\hat{\mathbf{BC}}}{\|\hat{\mathbf{BC}}\|_2}\,, \quad \hat{\mathbf{n}} \times \hat{\mathbf{bc}} = \frac{\hat{\mathbf{n}} \times \hat{\mathbf{bc}}}{\|\hat{\mathbf{n}} \times \hat{\mathbf{bc}}\|_2}\,. \tag{2.5}$$

In this project, the NeRF algorithm was implemented from scratch using residue-specific bond lengths and bond angles obtained from textbook references [17].

2.3 Deep learning

2.3.1 Feed-forward neural networks

The most simplistic of all deep learning network architectures is the feed-forward neural network (FNN). The FNN makes a prediction by processing the input values through a series of sequential (hidden) layers, each containing a number of non-linear processing units that are fully connected to every other unit in the two neighboring layers. The activation of a unit in a standard FNN with bias in an arbitrary layer l is defined as:

$$\mathbf{z}_{l+1} = \mathbf{h}_l \mathbf{W}_{l+1} + \mathbf{b}_{l+1}\,, \tag{2.6}$$

$$\mathbf{h}_{l+1} = a(\mathbf{z}_{l+1})\,, \tag{2.7}$$

where $\mathbf{h}_l$ is the activation of the l'th layer, $\mathbf{W}_l$ is the weight matrix and a is the activation function [42]. An illustration of a FNN architecture is presented in Figure 2.5.

In this project, the leaky rectified linear unit (leaky ReLU) was used as the preferred activation function:

$$a(x) = \max(x,\ \alpha x) \tag{2.8}$$

with $\alpha = 0.1$. This activation function was chosen due to its non-vanishing gradient properties compared to the classical sigmoid function and its inability to get stuck at zero gradient values unlike the standard ReLU function [25].

2.3.2 Softmax classification

In a standard classification task, the final prediction is made by normalizing and the activation of each output unit via the softmax function [42]:

$$\text{softmax}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}\,, \tag{2.9}$$

where j is the index of a particular output unit and K is the total number of output units [42]. Besides normalizing the output, the softmax function is useful as it accentuates the largest activation value in a way that is differentiable (compared to e.g. the argmax function).

In practice, the log-softmax variation is used in this project in order to improve numerical stability:

$$\text{log-softmax}(z)_j = \log\left(\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}\right) = z_j - \log\sum_{k=1}^{K} e^{z_k} = z_j - \left(b + \log\sum_{k=1}^{K} e^{z_k - b}\right), \tag{2.10}$$

where $b = \max_{k=1}^{K} z_k$. Specifically, this helps address issues with overflow as the largest value exponentiated after shifting is now zero.

2.3.3 Backpropagation

In order to learn, neural networks must first make prediction by letting the input values transverse through the network layers (a procedure known as the ”forward pass”). Then, it must assign an error term to each prediction made and then subsequently update each of the weights in the network in a way that is likely to reduce the error on further predictions (a procedure known as the ”backward pass”). In practice, this is done via the backpropagation algorithm [19]. No single general expression exists for the backpropagation algorithm, as it depends on subsequently stepping ”back” through the computational layers of the network by continuously applying the chain rule of derivation from the initial error term. After completion, the backpropagation algorithm outputs the set of error gradients with respect to each weight used in the network:

$$\nabla\mathbf{E} = \begin{bmatrix} \dfrac{\partial E}{\partial w_1} \\ \dfrac{\partial E}{\partial w_2} \\ \vdots \end{bmatrix}. \tag{2.11}$$

This set of gradients is used to update each weight in the network according to the chosen optimizer algorithm.

2.3.4 The Adam optimizer

The Adam optimizer is a first-order optimization algorithm that utilizes an adaptive learning rate defined by the following update rule defined for each parameter:

$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}\,, \tag{2.12}$$

where η is the learning rate and $\hat{m}_t$ and $\hat{v}_t$ are the parameter-specific, bias-corrected first- and second-order momenta that are computed from an exponential moving average of the gradient and the gradient squared using decay-rates β_1 and β_2 [34]:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2\,, \tag{2.13}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t\,, \tag{2.14}$$

where g_t is the error gradient of a particular parameter at time step t. The Adam optimizer holds several advantages over more classical optimzers such as stochastic gradient descent, namely that it is parameter-specific, incorporates momentum and employs an adaptable learning rate [34].

2.3.4.1 The AMSGrad correction

Recently, experimental results in e.g. object recognition [28] and machine translation [30] has suggested that the Adam optimizer in some cases fails to converge to an optimal solution and is outperformed by stochastic gradient descent with momentum. It has been suggested, that the reason for this suboptimal behavior is due to the use of exponential moving averages of past squared gradients. This moving average can become problematic in cases where only a few mini-batches hold valuable information, as these can be quickly diminished by the exponential average. In order to mitigate this issue, the AMSGrad correction to the Adam algorithm has been proposed [47]. Instead of using v_t directly from Equation 2.13, the previous v_{t-1} is only used if it is larger than the current one, i.e. for a single parameter the second-order momenta is chosen as:

$$\widetilde{v}_t = \max(v_{t-1},\ v_t) \tag{2.15}$$

2.3.5 Batch normalization

Training deep neural networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This effect, called internal covariate shifts, slows down the training by requiring lower learning rates and careful parameter initialization [29]. In this project, batch normalization is implemented in all models in order to reduce the problem of internal covariate shifts. The normalization occurs by transforming each activation x_i of a given layer according to the formula:

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\,, \tag{2.16}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta\,, \tag{2.17}$$

where $\mathcal{B}$ is the batch ensemble and γ and β are learned parameters [29]. During validation, the batch statistics are replaced by the population statistics in order to make inferences more reliable.

2.3.6 Multitask learning

In standard machine learning applications, the goal is to reduce the error of a particular metric. However, by only focusing on a single metric, there is an inherent risk of ignoring valuable information that might be useful in solving the task at hand. Multitask learning attempts to incorporate as much information as possible about the task by combined optimization of several metrics in parallel [49]. In this project, a particular kind of multitask learning is used in a way in which all learnable parameters are shared between two tasks and where there is a direct and unique mapping between the first and the second tasks. This type of multitask learning can be understood to work through three main mechanisms [11]:

- Attention focusing: If a task is noisy and high-dimensional, it can be difficult for a model to differentiate between relevant and irrelevant features. Multitask learning can help the model focus its attention on the features that actually matter, as other tasks will provide additional evidence for the relevance or irrelevance of those features [49].
- Eavesdropping: Some features G are easy to learn for some task B, while being difficult to learn for another task A. This might either be because A interacts with the features in a more complex way or because other features are impeding the model's ability to learn G. Through multitask learning, the model is allowed to eavesdrop, i.e. learn G through task B [49].
- Representation bias: Multitask learning biases the model to prefer representations that other tasks also prefer. This will also help the model generalize to new tasks, as a hypothesis space that performs well for a sufficiently large number of training tasks will also perform well for learning novel tasks assuming that they are from the same environment [7].
- Regularization: Finally, multitask learning acts as a regularizer thereby reducing the risk of overfitting [49].

2.3.7 Local vs. global loss functions

In this project, two tasks are solved in parallel both of which are assigned their own loss function. First, the dihedral angles of the protein backbone are predicted and evaluated through the angular deviation loss function. Second the Cartesian coordinates of the protein backbone are predicted and evaluated through the distance-based root mean square deviation (dRMSD) loss function. Both the angle-level as well as the coordinate-level representation of the protein backbone is unique and direct implying that there is a direct mapping between the two representations. However, the two loss functions exhibit slightly different behaviors and is therefore able to compliment each other. The angular loss function will tend to favor a strong local representation (useful in e.g. secondary structure motif prediction), with the downside that the global representation can be easily disturbed by a single incorrect angle, which would only affect the angular loss function slightly. On the other hand, the dRMSD loss function will tend to favor a strong global representation with the downside that correct local representations, such as secondary structure motifs, are neglected.

2.3.7.1 Angular deviation

The deviation between two angles is defined analogously to the usual root mean square deviation measure taking into account the periodic degrees of angular freedom [10]:

$$D(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\frac{1}{n}\sum_i (\min\left(|x_{2,i} - x_{1,i}|, 2\pi - |x_{2,i} - x_{1,i}|\right))^2}\,, \tag{2.18}$$

where $\mathbf{x}$ is a list of angular values with length n. Defined in this way, the angular deviation loss equally weights the deviation of each ϕ, ψ and ω prediction along the protein backbone.

2.3.7.2 dRMSD

Each backbone residue is described by a total of three atomic coordinates yielding the sequences $(c_1, c_2, ..., c_{3L})$ where L is the length of the protein. For two such sequences (i.e. prediction and target), the dRMSD loss is defined as follows [2]:

$$\text{dRMSD} = \frac{\|\mathbf{D}\|_2}{\sqrt{L(L-1)}}\,, \tag{2.19}$$

where

$$D_{j,k} = \widetilde{D}^{pred}_{j,k} - \widetilde{D}^{tar}_{j,k} \tag{2.20}$$

and

$$\widetilde{D}_{j,k} = \|c_j - c_k\|_2\,. \tag{2.21}$$

The dRMSD loss thereby defines the l_2 norm of the element-wise differences between the two inter-atomic Euclidean distance matrices of the predicted and the targeted atomic structures. Compared to other structural similarity metrics, the dRMSD has the advantage of not requiring the predicted and the targeted structure to be globally aligned, which eases computational costs.

2.3.8 Convolutional neural networks

An alternative to the FNN architecture is the convolutional neural network (CNN) architecture. Instead of connecting every unit in every neighboring layer with each other as in an FNN, a CNN works by sliding a fixed filter across subsets of the input space (called the receptive field) in order to extract characteristic patterns in the input. By learning specific values of the filter, the CNN is able to extract different types of features in the input and by using a number of different filters the CNN is able to search for a number of different patterns across the input. A convolutional operation can be defined for an arbitrary number of dimensions, however, this project will only make use of convolutions across one-dimensional input (i.e. the length of the protein sequence).

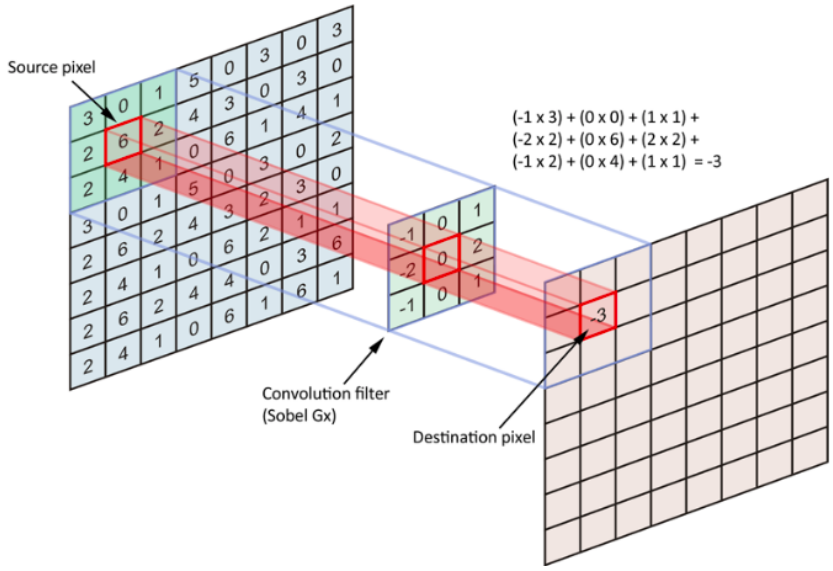

Figure 2.6: Overview of 2D convolution. The fixed convolutional filter slides over the input image with a pre-set stride length [14].

The activation of the i'th hidden unit in a single one-dimensional convolutional layer with input $\mathbf{x}$ can be defined as:

$$z_{i,d} = \sum_{k=0}^{K} w_{k,d}\, x_{j+k} + b_d\,, \tag{2.22}$$

$$h_{i,d} = a\,(z_{i,d})\,, \tag{2.23}$$

where $w_{k,d}$ is the k'th weight of the d'th filter, K is the filter size and j is the index of the current position in the input sequence [35]. The convolutional process is straightforwardly extended to the two-dimensional example as illustrated in Figure 2.6.

Convolutional architectures leverage three important ideas that can help improve learning in a deep learning network: i) sparse interactions, ii) parameter sharing, iii) shift invariance [19]. Sparse interactions refers to the fact that not all neighboring units are connected (as in the FNN) but that a small set of weights can be used to map interactions between layers. Parameter sharing refers to the fact that the same weights are used for several different functions in the network, as the fixed filter weights slides across the input sequence. Shift invariance refers to the fact that the CNN output is shifted in direct proportion to translational changes in the input. These three key characteristics helps the CNN use dramatically fewer learnable parameters than would otherwise be needed as well as making the CNN robust to translational variances in its input [19].

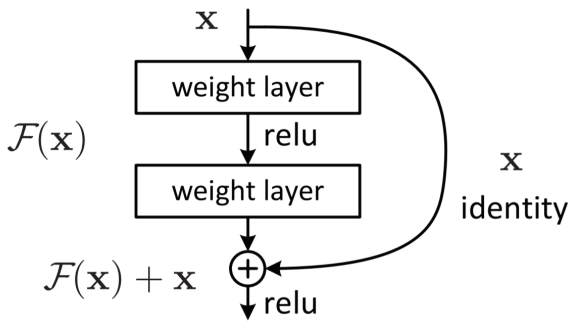

Figure 2.7: A two-layer residual block in a residual neural network [24].

2.3.8.1 Residual (convolutional) neural networks

Although deep neural networks have powerful representational capabilities they are often notoriously difficult to train. One of the main issues in this regard is the degradation problem, in which the performance of deep networks saturate prematurely and then degrade. It has been proven, that such degradation is not caused by overfitting the training data and that adding more layers to a suitably model leads to even higher saturation [24]. At a first glance, adding additional layers should never degrade performance as the additional layers can simple perform an identity mapping on the output of earlier layers (i.e. do nothing). Recent research suggests that traditional deep learning architectures might have difficulties in approximating identity mappings, possibly due to the fact that neurons use non-linear activation functions and therefore tend to overfit when tasked to fit a linear function (intuitively, this situation resembles using polynomials to fit a straight line).

In order to mitigate the degradation problem, a residual architecture has been proposed in which stacked residual blocks of weight layers are implemented as illustrated in Figure 2.7. The residual architecture is governed by the following equations:

$$\mathbf{y}_l = h(\mathbf{x}_l) + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)\,, \tag{2.24}$$

$$\mathbf{x}_{l+1} = a(\mathbf{y_l})\,, \tag{2.25}$$

where $\mathbf{x}_l$ and $\mathbf{x}_{l+1}$ are the input and output of the l'th layer, $\mathcal{F}$ is a residual function, $h(\mathbf{x}_l) = \mathbf{x}_l$ is an identity mapping used to match dimensions and a is the activation function (in our case; the leaky ReLU function defined in Equation 2.8).

The central idea of the residual network architecture is therefore to learn the additive residual function $\mathcal{F}$ via the residual block in order to add it to the input $\mathbf{x}$ via a skip connection. This concept of learning a series of additive residual functions ensures that the gradients in a residual network architecture are able to propagate back in the the network freely without interference.

This idea of freely propagating gradients can be illustrated by assuming that both h and a are identity mappings such that: $\mathbf{x}_{l+1} \equiv \mathbf{y}_l$. In this case, the general network equation is obtained:

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)\,, \tag{2.26}$$

where L is the total number of layers in the network. Denoting the loss function E the chain rule of backpropagation yields [26]:

$$\frac{\partial E}{\partial \mathbf{x}_l} = \frac{\partial E}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial E}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)\right). \tag{2.27}$$

It is seen that the residual architecture allows the gradient to be split into two additive terms, with the term $\partial E/\partial \mathbf{x}_L$ ensuring that gradients can be propagated directly backward to any shallower layer. Usually, residual networks, such as the one described, are implemented such that each residual block consists of 2-3 convolutional layers, although other types of architectures are theoretically possible.

2.3.8.2 Residual block pre-activation vs. post-activation

In this project, a "pre-activation" architecture is implemented similar to the one presented by He et al. [26]. In this architecture, the activation function and batch-normalization is placed before each convolutional layer in the residual block as opposed to after it. The use of pre-activation as opposed to traditional post-activation ensures that the output of the activation function only affects the $\mathcal{F}$ path, which has been shown to ease optimization and reduce overfitting [26].

2.3.9 Recurrent neural networks

Recurrent neural networks (RNNs) are a different kind of deep learning architecture defined by its ability to feed the activation of one layer back into itself in a series of sequential time-steps. By doing so, the RNN is able to make predictions of each position in a data sequence using information found at previous steps in the sequence. RNNs are primarily used for sequential data, although it can be shown, that for any computable function there exists a finite RNN that can compute it [53].

2.3.9.1 Long short term memory cells

Despite the attractiveness of traditional RNNS, they are known to be difficult to optimize due to the effects of unstable gradients [42]. The long short term memory (LSTM) cell is a specialized type of RNN providing a solution to this problem. The LSTM cell works by introducing a cell state $\mathbf{c}_t$, which is passed along the sequence. This cell state is then controlled by so-called input, forget and output gates. Each of these gates can be though of as a a conventional artificial neuron that computes an activation controlling how new sequence information is incorporated compared to previous input.

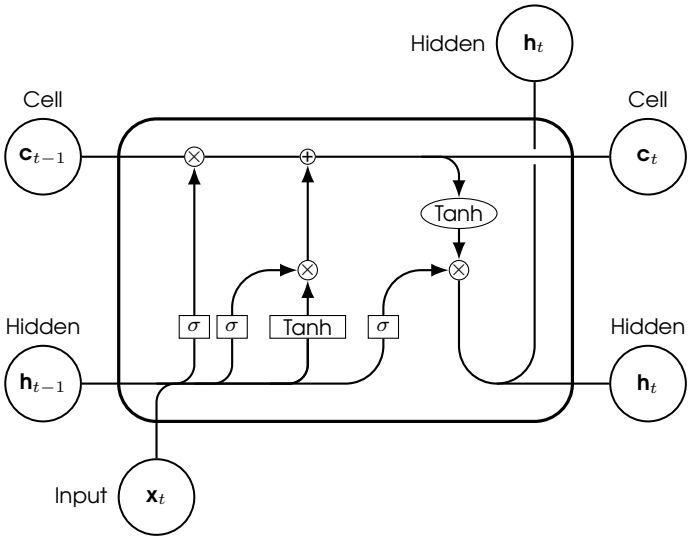

Figure 2.8: Overview of LSTM-cell operations. Illustration inspired by [43] and [23].

The LSTM cell without peepholes is defined by the equations:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i)\,, \tag{2.28}$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)\,, \tag{2.29}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)\,, \tag{2.30}$$
$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{W}_{xg}\mathbf{x}_t + \mathbf{W}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_g)\,, \tag{2.31}$$
$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \widetilde{\mathbf{c}}_t\,, \tag{2.32}$$
$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t)\,, \tag{2.33}$$

where $\mathbf{i}_t$ is the input gate, $\mathbf{f}_t$ is the forget gate, $\widetilde{\mathbf{c}}_t$ is the proposed new cell state, $\widetilde{\mathbf{c}}_t$ is the new cell state and $\mathbf{o}_t$ is the output gate with all gates and states defined at time step t [20]. A graphical representation of the LSTM cell is presented in Figure 2.8.

2.3.9.2 Bidirectional LSTMs

Traditionally, the LSTM cell analyzes the input sequence in a single backward-to-forward direction, as is relevant for predictive tasks where strong sequential causality is implied [20]. However, in the case of proteins, the folding of a single amino acid in the polypeptide chain is influenced by the physical forces acting on both sides of the residue and bidirectional sequence analysis is therefore needed. A solution to this problem is the use of a bidirectional LSTM. A biLSTM consists of two memory cells that analyze the sequence from opposing directions [20] with the final hidden state being a concatenation of the two separate hidden states from each direction [52]:

$$\mathbf{h}_t = \begin{pmatrix} \overrightarrow{\mathbf{h}_t} \\ \overleftarrow{\mathbf{h}_t} \end{pmatrix}. \tag{2.34}$$

CHAPTER 3

Data

In the following chapter, the data set used throughout this project will be introduced. First, the general ProteinNet family of data sets will be introduced including a brief summary of the unique advantages that it holds over other relevant data sets. Then, the particular data set used in this project - the ProteinNet-CASP11 data set - will be introduced along with an overview of its features and their definitions. Finally, key summary statistics will be presented.

3.1 The ProteinNet data sets

ProteinNet is a newly created family of data sets specifically created for machine learning tasks within protein structure prediction. The development of ProteinNet has been motivated by the need for a clean and standardized data set for protein structure prediction that could help spur innovation within the field - analogous to e.g. the ImageNet data set within computer vision [3]. The preliminary version of ProteinNet was released in March 2018 by Mohammed Al Quraishi, Department Fellow at the Department of Systems Biology at Harvard Medical School[1]. The main idea behind ProteinNet is to mimic the conditions of the biennial CASP competitions, wherein state of the art protein structure prediction methods are tested on newly identified and unpublished structures. The CASP structures thereby provide a standardized benchmark for how well predictive methods perform at a given moment in time. In mimicking these conditions, ProteinNet has been organized into a series of data sets covering CASP 7 through CASP 12, each of which contains a number of differently sized training sets as well as a validation set and a test set.

3.1.1 Splitting methodology

In many machine learning applications, the splitting of data into training, validation and test sets is a relatively straightforward task, since samples can usually be assumed to be independent and identically distributed. However, this assumption breaks down in the field of protein structure prediction where proteins often share some level of evolutionary relationship with each other. This shared homology usually manifests itself as an information leakage from the validation and/or test set into the training set thereby increasing the risk of inflated performance metrics and overfitting.

In addition to the risk of information leakage between the test set and the training set, there is a risk of overfitting on a particular set of evolutionarily related proteins due to a poorly constructed training set. This issue is mitigated by ensuring that the training set consists of a diverse range of evolutionarily distinct proteins, as this will enable the model to generalize to a broad range of test proteins. However, construct such a diverse and unbiased training set

[1]Data sets and related code is available at: `https://github.com/aqlaboratory/proteinnet`

is often complicated due to the subtle nature of homology assessment. Specifically, traditional ways of homology assessment such as sequence-sequence alignment scores tend to suffer from two main issues: i) evolutionarily unrelated proteins may register as having a sequence identity of e.g. 10% when in reality their relationship is spurious, ii) two evolutionarily related proteins may not be detectable as such even though their actual sequence identity is 10% [3].

In order to mitigate these challenges, homology in the ProteinNet data sets is assessed via special profile-profile alignment scores that are able to detect deep sequence homology. These alignment scores have been computed using the MMseqs2 software [55]. By assessing homology via profile-based alignment scores, each ProteinNet training set can be constructed from a set of evolutionarily distinct proteins separated via clustering. Furthermore, differently-sized training sets can be constructed by controlling the maximum permitted level of homology between training set proteins [3]. An example of this for the ProteinNet-CASP11 set is presented in Table 3.1.

3.1.2 Features

The features included in each protein record of the ProteinNet data sets are:

- Amino acid sequence: The primary sequence is represented by a an alphabet of size 20. In this project, the primary sequence is one-hot encoded during pre-processing.
- Position-specific-scoring (PSSM) information: The PSSSM matrices summarize the propensity of each residue position along the protein chain to mutate to other amino acids and are represented by a sequence of real-valued 20-dimensional vectors, normalized to range between 0 and 1. An additional dimension, corresponding to the information content of a residue, is concatenated with each vector to bring the total number of dimensions to 21. The PSSMs contained in the preliminary release of ProteinNet are derived using JackHMMer from UniParc and metagenomic sequences [3].
- Tertiary structure: The preliminary release of ProteinNet only contains the coordinates of the backbone atoms, corresponding to the sequential chain of N, C_α and C' atoms. The full protein backbone is thus represented by a sequence of $3N$ 3-dimensional vectors, where N is the number of amino acids in a protein [3].
- Masks: ProteinNet protein records come with their own masking vectors indicating the positions along the residue chain, where the atomic coordinates are not precisely defined due to either experimental or intrinsic reasons [3].

3.1.2.1 Added feature: Dihedral angles

In addition to the above features of the original ProteinNet data, each protein record used in this project have been augmented with the three dihedral angles necessary to compute the angular deviation loss. The dihedral angles have been computed using the praxeolitic formula, which requires fewer numeric operations than similar methods.

Assuming four coordinate points in space $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$ the dihedral angle can be computed as [15]:

$$\text{dihedral angle} = \text{arctan2}(\mathbf{b}_1 \times \mathbf{v}, \mathbf{v} \cdot \mathbf{w}), \tag{3.1}$$

where the arctan2 function is defined as:

$$\text{arctan2}(y,x) = \begin{cases} \arctan(y/x) & \text{if } x > 0\,, \\ \arctan(y/x) + \pi & \text{if } x < 0 \text{ and } y \geq 0\,, \\ \arctan(y/x) - \pi & \text{if } x < 0 \text{ and } y < 0\,, \\ \pi/2 & \text{if } x = 0 \text{ and } y > 0\,, \\ -\pi/2 & \text{if } x = 0 \text{ and } y < 0\,, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0\,. \end{cases} \tag{3.2}$$

The vector quantities are defined as:

$$\mathbf{v} = \mathbf{b}_0 - \mathbf{b}_0 \cdot \hat{\mathbf{b}}_1 \otimes \hat{\mathbf{b}}_1\,, \quad \mathbf{w} = \mathbf{b}_2 - \mathbf{b}_2 \cdot \hat{\mathbf{b}}_1 \otimes \hat{\mathbf{b}}_1 \tag{3.3}$$

and

$$\mathbf{b}_0 = -(\mathbf{p}_1 - \mathbf{p}_0)\,, \quad \hat{\mathbf{b}}_1 = \frac{\mathbf{p}_2 - \mathbf{p}_1}{\|\mathbf{p}_2 - \mathbf{p}_1\|_2}\,, \quad \mathbf{b}_2 = \mathbf{p}_3 - \mathbf{p}_2\,, \tag{3.4}$$

such that $\mathbf{b}_i$ are the vectors connecting the atomic positions $\mathbf{p}_j$ in space, $\mathbf{v}$ is the projection of $\mathbf{b}_0$ onto a plane perpendicular to $\mathbf{b}_1$ and $\mathbf{w}$ is the projection of $\mathbf{b}_2$ onto a plane perpendicular to $\mathbf{b}_1$.

3.1.3 ProteinNet-CASP11 data sets

In this project, the ProteinNet-CASP11 data set has been used for training and testing. The ProteinNet-CASP11 data set has been organized into a total of 7 subsets summarized in Table 3.1. It is seen how the training set is available in multiple thinnings corresponding to how densely the training proteins are sampled in sequence space (as discussed in Section 3.1.1). In regards to the test set, target proteins have been additionally categorized into template-based modeling targets (TBM), corresponding to structures with known protein topologies, and free modeling targets (FM), corresponding to targets with novel topologies. This classification is derived from the official CASP classification. In the CASP11 case, the test set consists of a total of 81 proteins categorized into 62 TBM proteins, 15 FM proteins and 4 unclassified proteins.

Type	**Deep sequence identity threshold %**	**No. proteins**	**Data size [MB]**
Training	30	22,344	2,119
	50	29,936	2,990
	70	36,005	3,646
	90	42,507	4,265
	100	87,573	8,923
Validation	NA	224	21
Test	NA	81	8

Table 3.1: Overview of ProteinNet-CASP11 data sets. Listed data sizes are for raw, non-padded text records only - workable data formats result in significantly increased data sizes.

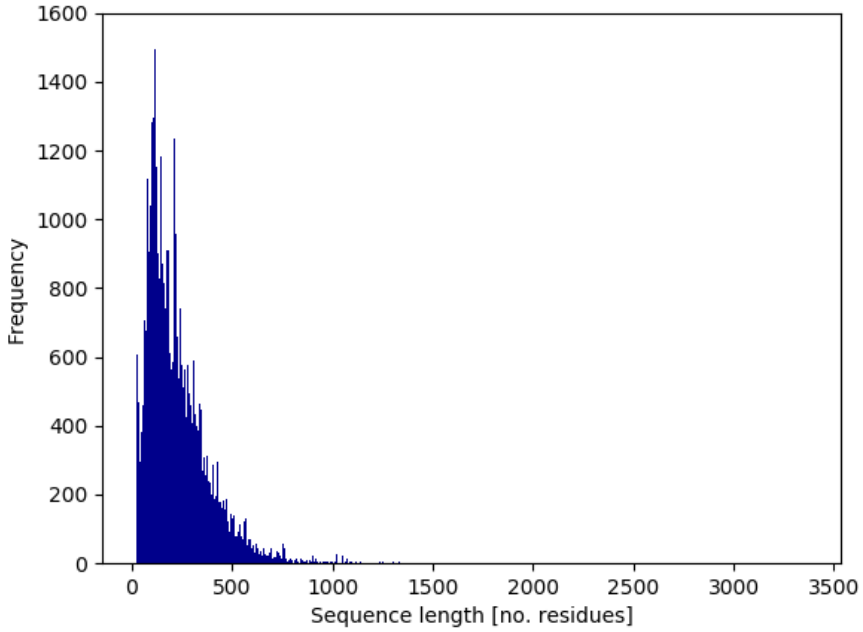

Figure 3.2: Distribution of sequence lengths in the ProteinNet-CASP11 training_100 training set. Only 38 proteins have sequence lengths above 1,500 residues.

3.1.3.1 Sequence lengths

In regards to sequence lengths, most proteins are generally a few hundred residues long with a small handful of proteins having of length of 1,500 residues or more. The distribution of sequence lengths for the largest CASP11 training set (i.e. the training_100 set) is shown in Figure 3.2.

3.1.4 Implementation: h5py for efficient RAM management

All computations were performed on the DTU Compute GPU cluster which consists of a cluster of accessible GPUs of types Titan X, Titan Xp and GTX 1080 Ti[2]. All cluster GPUs have RAM of either 11 or 12 GB, which is usually enough for most applications. However, this amount of GPU memory turned out to be insufficient in the case of the ProteinNet-CASP11 training sets, which increased dramatically in size after pre-processing and padding (compared to the sizes listed in Table 3.1). This increase in data sizes, combined with large memory requirements for model parameters etc., introduced a RAM shortage in the model implementation. In order to mitigate this issue, the h5py package was successfully employed. h5py works by letting the user load small portions of the data set selectively (e.g. batches of data) into the RAM in NumPy format without needing to load the entire data set [12]. In short, the h5py package can be described as a Pythonic interface to the HDF5 binary data format, which lets the user store huge amounts of numerical data and easily manipulate the data from NumPy with only limited overhead.

[2]Further technical specifications available at `https://itswiki.compute.dtu.dk/index.php/GPU_Cluster`

CHAPTER 4

Related work

In this chapter, the deep learning-based protein tertiary structure prediction model recently proposed by Mohammed Al Quraishi is introduced. Key ideas leveraged in this project has been inspired by the work of Al Quraishi and his model therefore serves as a benchmark for the results obtained in this project.

4.1 *"End-to-end differentiable learning of protein structure"*

A related approach to protein structure prediction via deep learning was proposed by Al Quraishi in his recent paper *"End-to-end differentiable learning of protein structure"* published as a pre-print in February 2018 [2]. In the paper, Al Quraishi is the first (to this author's knowledge) to present a successful working model of a network that predicts Cartesian coordinate structures from protein sequence input in a way that is fully differentiable and therefore end-to-end optimizable. It is this notion of end-to-end differentiability that the final model in this project attempts to build on. In order to achieve his predictions, Al Quraishi uses[1] a bidirectional LSTM with two cells (each having 800 hidden units) to predict the input of an angularization layer defined by the following equations (Al Quraishi's notation)[2]:

$$p_t = \text{softmax}\left(W_\phi \left[h_t^{(f)}, h_t^{(b)}\right] + b_\phi\right), \tag{4.1}$$

$$\phi_t = \arg\left(p_t \exp(i\,\Phi)\right) = \text{arctan2}\left(p_t \sin(\Phi),\ p_t \cos(\Phi)\right), \tag{4.2}$$

where W_ϕ is a weight matrix, b_ϕ is a bias vector and Φ is a matrix containing forty dihedral angle triplets (ϕ, ψ, ω). The arctan2 function is defined as presented in Equation 3.2 and is used to compute a weighted average of the estimated angles. The output of the angularization layer is fed into a series of geometric units that transform the predicted angles into Cartesian coordinates through a modified version of the NeRF algorithm [44]. Finally, the dRMSD loss is computed and the error is backpropagated through the computational graph.

Using this approach, Al Quraishi was able to achieve a mean FM dRMSD of 8.7 Å and a mean TBM dRMSD of 7.4 Å on the ProteinNet-CASP11 test set - corresponding to a first place in the FM category. Despite these impressive results, personal conversations with Al Quraishi revealed that his network is extremely difficult to train. Specifically, training times were approximately 2 months on a single Titan Xp GPU using a batch size of 32. Furthermore, Al Quraishi's model is very seed sensitive such that only 1 out of a 1,000 training sessions were able to achieve the reported results. Lastly, as the model is trained to optimize for the dRMSD loss only, the predicted angular values of the model is often of low quality, making the model bad at predicting e.g. secondary structures.

[1] Source: Al Quraishi, personal communication.

CHAPTER 5

Results

In the following chapter, two models are presented corresponding to a preliminary model as well as a final model. For each model, a section will be dedicated to the implementation and intuition behind the model, and a separate section will be dedicated to the results obtained from the model including a brief discussion of their implications.

5.1 Preliminary model: Soft-label prediction of dihedral angles using pre-trained angular matrix

5.1.1 Implementation

In the preliminary model, the target protein structures are represented using the dihedral angles only. A high-level overview of the model is presented in Figure 5.1. In order to predict the dihedral angles, an external mixture model is first pre-trained and then used in the preliminary model to pre-process the target dihedral angles into soft-label representations. Specifically, each (ϕ, ψ) angle pair of the protein backbone was transformed into a vector of mixture component weights (i.e. soft-labels):

$$\begin{pmatrix} \phi \\ \psi \end{pmatrix}_i \longrightarrow \begin{pmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_K \end{pmatrix}_i , \tag{5.1}$$

with i being the index of the i'th residue of the protein backbone and K being the total number of mixture components. Notice, that the ω angles are omitted from Equation 5.1 as these were assumed to be fixed at $\omega = 180$ degrees. The predictive network was then trained to predict all vectors $(\pi_0, \pi_1, \ldots, \pi_K)^T$ via the soft-label cross-entropy loss function:

$$H(\boldsymbol{h}, \boldsymbol{t}) = -\sum_i t_i \, h_i + \log \sum_j \exp(h_j) \, , \tag{5.2}$$

where $\boldsymbol{h}$ is the predicted weight vector before softmax normalization and $\boldsymbol{t}$ is the target weight vector. A complete derivation of the soft-label cross-entropy loss is presented in Appendix 1.

The mixture model used to pre-process the angles was constructed using a mixture of cosine bivariant von Mises distributions. The bivariant von Mises distribution exhibits Gaussian-like properties in periodic space and is therefore well-suited to capture angular degrees of freedom [38]. The cosine variant was chosen from the family of bivariate von Mises distributions, as this variant has been developed specifically for protein structure prediction purposes by Mardia et al. [38].

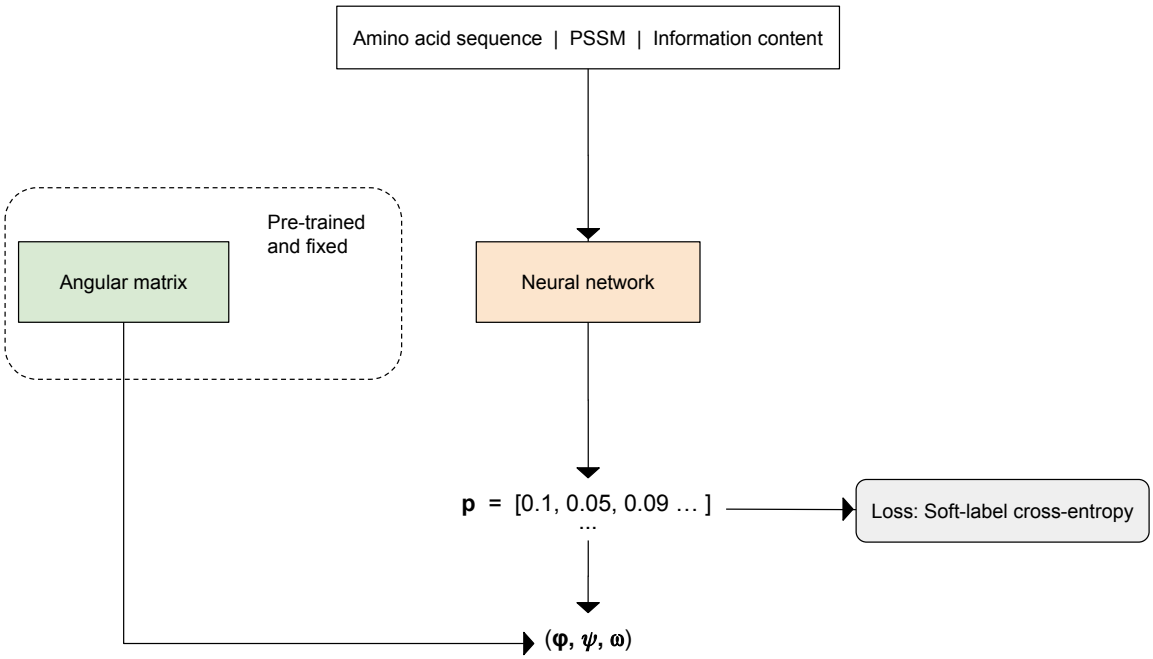

Figure 5.1: High-level overview of the preliminary model. Colored boxes correspond to learnable parameters.

The density function of the cosine variant of the bivariate von Mises distribution is given by:

$$f(\phi, \psi \,|\, \mu, \nu, \kappa_1, \kappa_2, \kappa_3) = \\ c(\kappa_1, \kappa_2, \kappa_3) \exp\left[\kappa_1 \cos(\phi - \mu) + \kappa_2 \cos(\psi - \nu) - \kappa_3 \cos(\phi - \mu - \psi + \nu)\right], \tag{5.3}$$

where c is a normalization constant, μ and ν are the respective means of ϕ and ψ, κ_1 and κ_2 are their concentrations and κ_3 is related to their correlation [38]. A superposition of cosine variant von Mises distributions was used to construct the external mixture model in the usual manner:

$$p(\phi, \psi) = \sum_{k=1}^{K} \pi_k \, f(\phi, \psi \,|\, \mu_k, \nu_k, \kappa_{1,k}, \kappa_{2,k}, \kappa_{3,k}) \,. \tag{5.4}$$

A visualization and sampling-plot of a trained mixture model with fifty mixture components is provided in Figure 5.2 and Figure 5.3. It is seen that the component distribution generally matches well with the expected Ramachandran plot (see e.g. Figure 2.2 for reference).

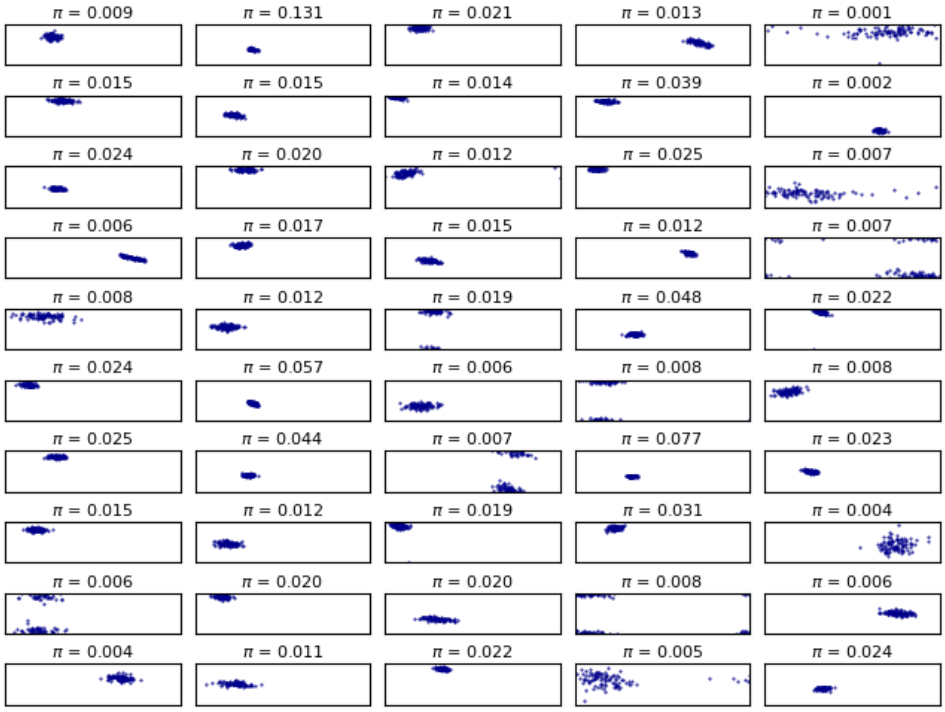

Figure 5.2: Visualization of trained cosine bivariant von Mises mixture model with 50 mixture components. Mixture weights are printed above each mixture component plot. The x- and y-axis of each plot measure the ϕ and ψ angle respectively in the range $[-180; 180]$ degrees.

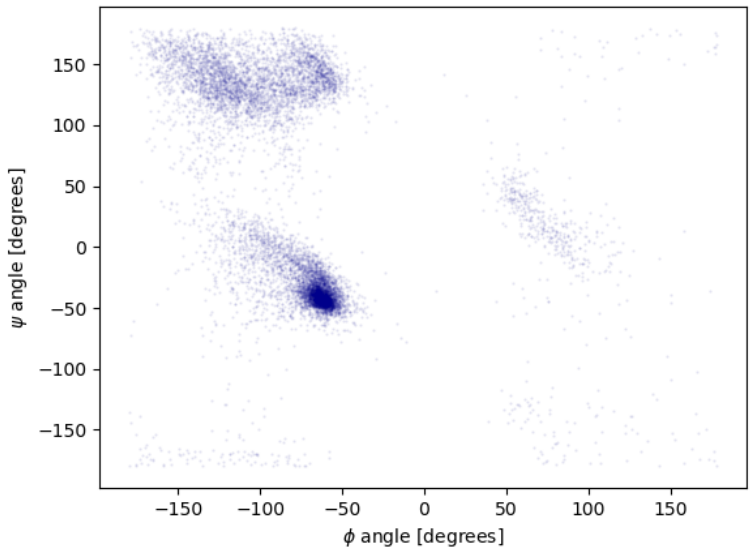

Figure 5.3: Scatterplot of 10,000 angles sampled from the cosine bivariant von Mises mixture model with 50 mixture components.

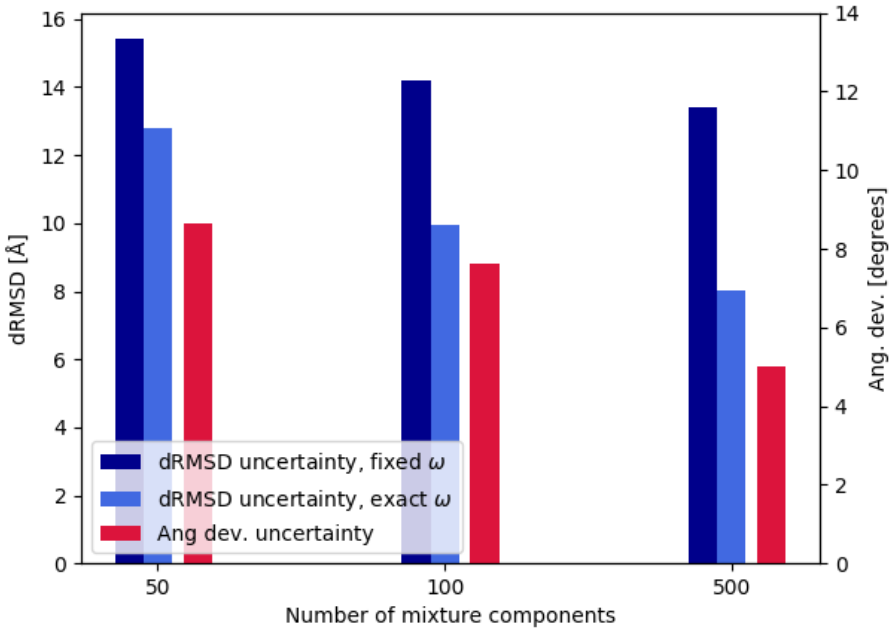

Figure 5.4: Illustration of inherent angular deviation and dRMSD uncertainty in the preliminary model approach. "Fixed ω" means all $\omega = 180$. "Exact ω" means that true ω values were used together with the doubled-transformed ϕ, ψ angles.

After training, the predicted soft-label vectors are transformed back to their dihedral angle representation via the arctan2 method. The arctan2 method serves to compute a weighted circular mean of the predicted mixture component weights (i.e. the predicted soft-labels):

$$\begin{pmatrix}\hat{\phi}\\ \hat{\psi}\end{pmatrix}_i = \text{arctan2}\left(\sum_{k=1}^{K} p_{k,i} \sin\left[\begin{pmatrix}\mu\\ \nu\end{pmatrix}_k\right], \sum_{k=1}^{K} p_{k,i} \cos\left[\begin{pmatrix}\mu\\ \nu\end{pmatrix}_k\right]\right), \tag{5.5}$$

where k is the index of the k'th mixture component, $p_{i,k}$ is the predicted soft-label weight of the k'th mixture component of the i'th residue (softmax-normalized such that $\sum_{k=1}^{K} p_{i,k} = 1$) and $(\mu, \nu)_k^T$ is the ϕ and ψ means of the k'th mixture component. The set of all $(\mu, \nu)_k^T$ is denoted the "angular matrix". It should be noted that the values in the angular matrix remain fixed during training in this preliminary model (as opposed to the final model presented later in Section 5.3).

Essentially, the arctan2 method estimates a circular mean of a set of angles by first transforming the angles to corresponding points on the unit circle (e.g. α to $\{\cos(\alpha), \sin(\alpha)\}$ thus converting polar coordinates to Cartesian coordinates), it then computes the weighted arithmetic mean of these points and then finally converts the estimated mean point back to polar coordinates via the arctan2 function (as defined in Equation 3.2). This method produces a reasonable estimate of the circular mean, however, it is not entirely precise as it disregards the curvature of the underlying sample space. An alternative to this approach will be discussed in Section 7.5

5.1.2 Results

Despite the initial attractiveness of the preliminary model approach, various experiments revealed that the described transformation from soft-label prediction to dihedral angles carried with it a non-reducible error-term that effectively put a limit on the minimum obtainable loss. The mean of this non-reducible error-term is denoted the "uncertainty" of the angular deviation and the dRMSD respectively. This inherent uncertainty can be traced back to an unexpected consequence of the fixed and limited number of mixture components used in the pre-trained angular matrix. The situation is illustrated in Figure 5.4. It is seen how the angular uncertainty decreases as a function of the number of mixture components, however, the uncertainty is still high in the case of 500 mixture components after which increasing the number of components becomes computationally unfeasible using the described set-up in the preliminary model.

The angular uncertainty is computed by the procedure:

1. Transform true ϕ, ψ angles to soft-label representations via external mixture model.
2. Transform these soft-label representations back to ϕ, ψ angles via the arctan2 weighted mean (see Equation 5.7).
3. Compute the angular deviation (see Section 2.3.7.1) between the double-transformed and the true dihedral angles.

The dRMSD uncertainty also depicted in Figure 5.4 is a direct consequence of the angular uncertainty which compounds for each residue in the protein backbone chain. As expected, this compounding behaviour makes the dRMSD uncertainty an increasing function of the protein sequence length. This relationship is illustrated in Figure 5.5.

It was hypothesized that the angular uncertainty was a consequence of a larger-than-expected number of ω angles different from 180 degrees. This hypothesis was investigated by examining the actual distribution of ω angles as shown in Figure 5.6. It is seen, that the fixed ω assumption is fairly inaccurate. In fact, the distribution seems to be approximately exponential and symmetric around $\omega = 180$ with a few ω angles close to zero. However, the mean angular uncertainty was only partially reduced by replacing the fixed ω angles with the true ω angles in step 3 as shown in Figure 5.4.

Further analysis revealed that the underlying reason for the angular uncertainty in the preliminary model was due to a mismatch between how the pre-trained mixture model is optimized versus how it is actually used in the model. Fundamentally, the mixture model is trained to maximize likelihood by varying all free mixture component parameters, while the predictive model only uses the mixture component means to transform predictions from soft- to hard-label (i.e. through Equation 5.7). An example of how this mismatch leads to angular uncertainty is presented in Figure 5.7. It is seen how the easy target is associated with mixture components with means close to the data point, while the hard target is associated with a single mixture component with a mean far from the data point. Referring to visualization of mixture components in Figure 5.2, it is seen that the primary mixture components of the easy target are very localized (components 6, 12 and 48), while the primary mixture component of the hard target is more "spread out" (component 5).

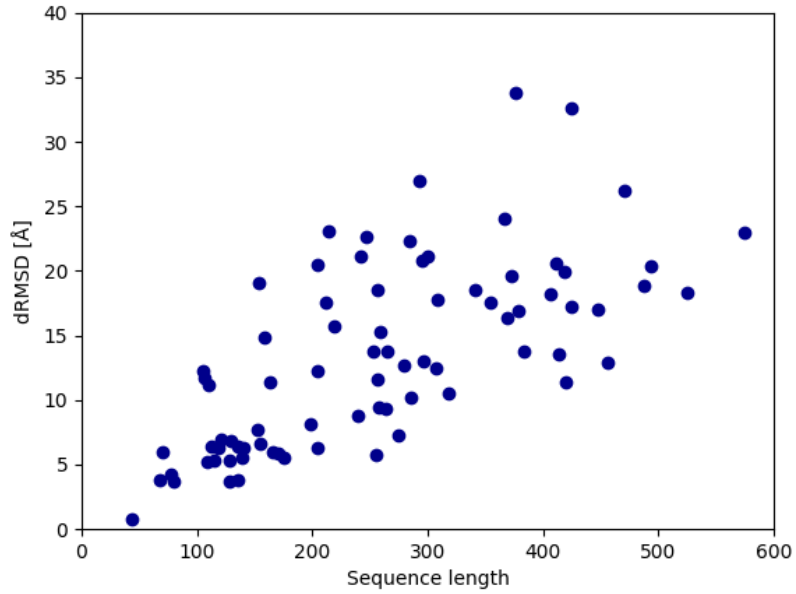

Figure 5.5: Example of coordinate-wise (dRMSD) uncertainty as a function of sequence length. In this case, 100 mixture components and fixed ω values corresponding to an angular uncertainty of 7.6 degrees.

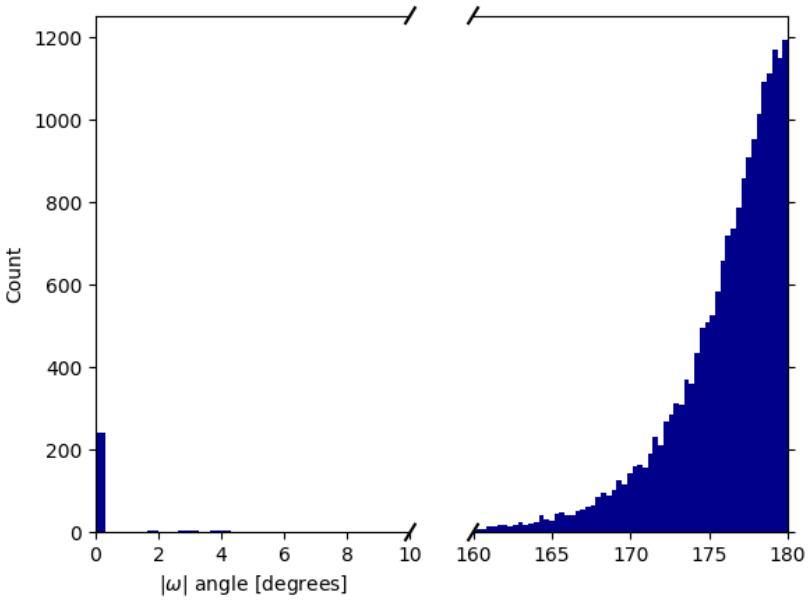

Figure 5.6: Distribution of absolute-valued ω angles obtained from the ProteinNet-CASP11 test set.

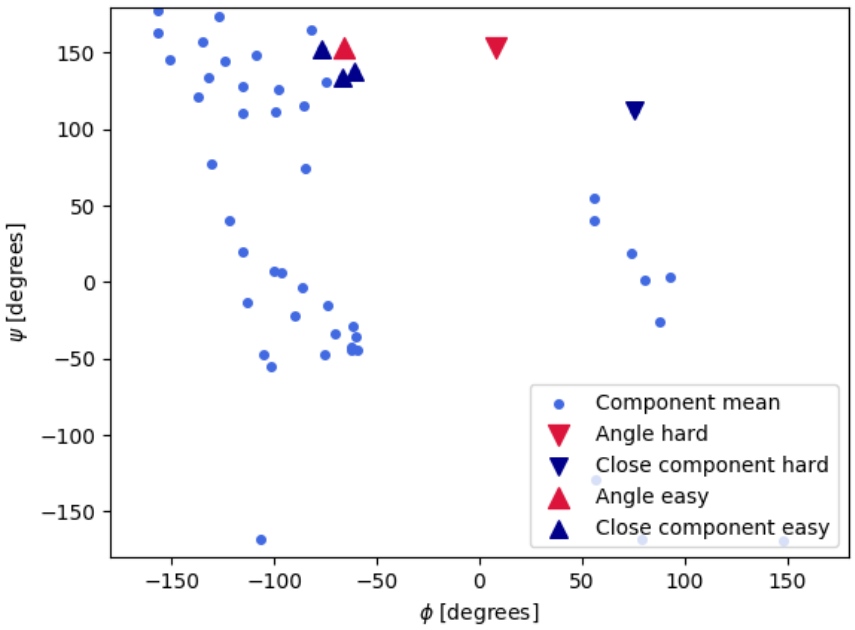

Figure 5.7: Means of mixture components alongside two angles from the data set. Angular uncertainty of hard and easy target is 1.5 and 55.7 degrees respectively. Close components have been selected such that $\sum \pi_{(\phi,\psi)} > 0.95$.

To summarize, it was found that the use of an external mixture model introduced a non-reducible error-term in our preliminary model, which impaired its local (i.e. angular) representation of the proteins. This local (i.e. angular) uncertainty translates into significant deviations in the global (i.e. coordinate-wise) representation. This inherent uncertainty in both the local and the global representation ultimately led to a rejection of the preliminary model approach. It is concluded, that the global structure of the protein seems to be highly sensitive to the accuracy of its local representation and that alternatives to the use of the pre-trained angular matrix seem to be needed.

5.2 CNNs vs LSTMs for sequence modeling

Traditionally, sequence modeling tasks such as time series analysis (but also protein sequence prediction) has been synonymous with the use of recurrent neural networks [20]. On the contrary, CNNs have usually been associated with tasks in computer vision. However, recent experimental results have indicated that CNNs can outperform recurrent networks on a number of classical sequence modelling tasks, such as audio synthesis and machine translation [6]. It has been hypothesized that part of the reason for this surprising effect is that the theoretically unlimited sequence memory of LSTMs is in practice obscured by the effects of unstable gradients, such that deep CNN architectures (possibly with the use of dilation layers) often are able to maintain much longer sequence memories compared to their recurrent counterparts [6].

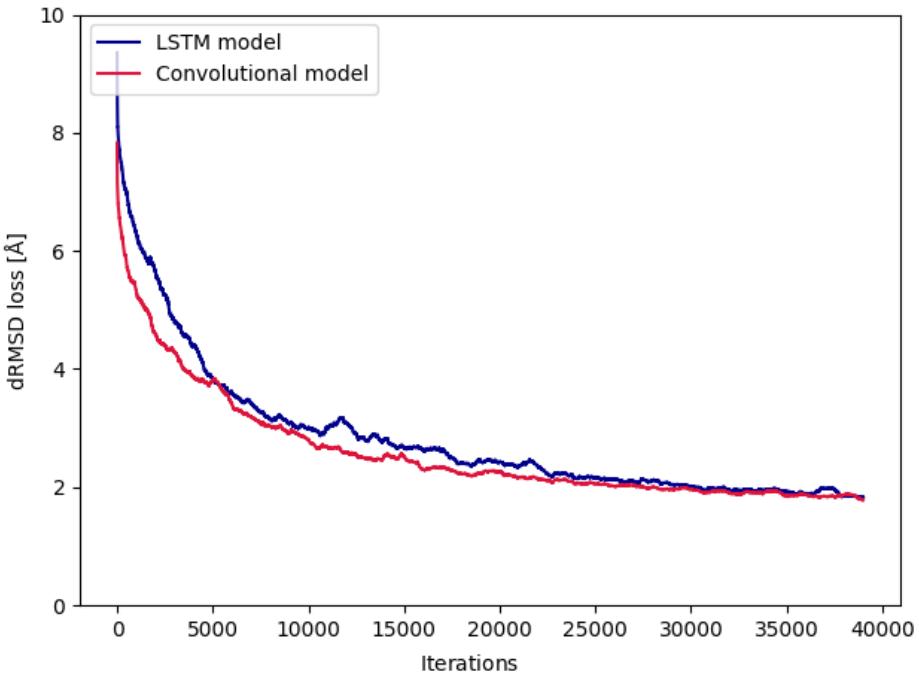

Figure 5.8: Comparison of dRMSD loss for 5 short proteins during training using an LSTM model and a convolutional model (residual architecture) with comparable number of learnable parameters. Smoothed via moving average.

This is relevant to the case of protein tertiary structure prediction, as long sequence memory is critical for capturing long-range dependencies such as β-strand interactions and other motifs that fold back onto themselves. In addition to the possibility of longer sequence memory, CNNs offer different advantages such as manual control over receptive field sizes while also having lower computational requirements than traditional LSTMs/GRUs [6].

In order to test this hypothesis, a number of different experiments were carried out. One example is shown in Figure 5.8. The figure shows a simple LSTM model an a simple convolutional model (residual architecture) with comparable number of parameters. Both models were trained to optimize the dRMSD loss of five proteins using the final model approach as detailed in Section 5.3. It is seen that the convolutional model matches the performance of the LSTM while also converging slightly faster. These findings were validated experimentally in multiple scenarios and with multiple sets of hyperparameter settings. It is concluded, that convolutional models can indeed match (and maybe surpass) the performance of traditional LSTMs in our limited case of protein sequence prediction.

5.3 Final model: Multitask prediction of dihedral angles and Cartesian coordinates using integrated angular matrix

5.3.1 Implementation

In the final model of this project, the issues related to the angular uncertainty of the preliminary model are mitigated by incorporating the angular matrix into the predictive network as a set of learnable parameters. This includes incorporating the ω angle, which was discovered to be flexible (see e.g. Figure 5.6). In addition to this, the final model is specifically developed in order to leverage the concept of end-to-end differentiability recently introduced by Al Quraishi (see Section 4). However, in addition to the main dRMSD loss used by Al Quraishi, the final model introduces the angular deviation as an auxiliary loss function used to guide the global structure prediction. A high-level overview of the model is illustrated in Figure 5.9. It is seen that the predictive network consists of a residual CNN network and a separate LSTM network connected via a skip connection. This combination of predictive network architectures is implemented in order to leverage the potential advantages of CNNs over LSTMs in sequence analysis (see Section 5.2). Separate models were trained using only the CNN network, only the LSTM network as well as the two networks combined.

The convolutional network is implemented with a residual network architecture as described in Section 2.3.8.1. A total of 8 residual blocks are used with 2 convolutional layers in each block plus an initial convolutional layer for transforming the raw input initially. A 1x1 convolution layer is used at the end of each residual block in order to match the input dimensions to the output of the residual function. Kernel widths were fixed at 3 for each convolutional layer using a stride of 1. The number of filters in each convolutional layer is set to increase linearly for each residual block from 200 to 600.

The LSTM network is implemented as a bidirectional LSTM with two cells stacked after each other, as described in Section 2.3.9.1. The number of hidden units in each cell is set to 600.

The skip connection between the CNN and the LSTM is implemented such that the CNN takes the raw sequence information as input whereas the LSTM takes the raw sequence information as well as the output of the CNN as input. This is done under the hypothesized assumption that the CNN might be better able to recognize and extract small structural motifs (such as the β hairpin) due to its fixed receptive field. These CNN motif identification-features could then be further processed and serve as valuable input to the LSTM. However, as demonstrated in Section 5.2, the CNN and the LSTM might work interchangeably during sequence analysis. Because of this, the input to the LSTM is made as a concatenation of the CNN output and the raw sequence features (as opposed to a summation such as in a residual architecture) in order to make each network theoretically able to bypass the other completely.

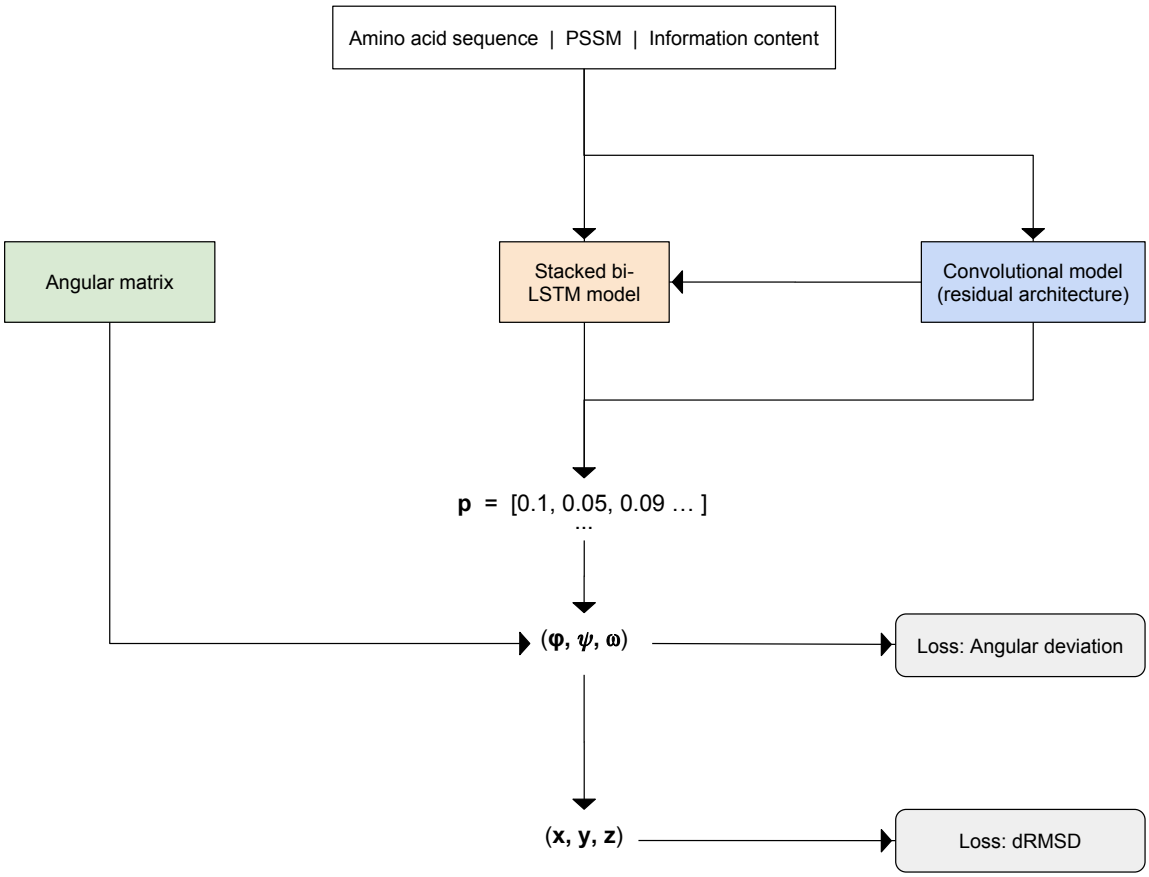

Figure 5.9: High-level overview of the final model architecture. Colored boxes correspond to learnable parameters.

Predictions are made in the final model as follows. For a given residue i, the predictive network outputs a softmax probability $p_{t,i}$ estimating the probability that the dihedral angles belong to the t'th angular triplet of the angular matrix $(\tilde{\phi},\ \tilde{\psi},\ \tilde{\omega})_t^T$:

$$\boldsymbol{p}_i = \text{softmax}(\mathbf{W}_{l,i}\ \mathbf{x}_{l,i} + \mathbf{b}_{l,i})\,, \tag{5.6}$$

where l is the final layer of the network.

The weights of the different softmax predictions are averaged together using the arctan2 circular mean method (as presented in Section 5.1.1):

$$\begin{pmatrix}\hat{\phi}\\ \hat{\psi}\\ \hat{\omega}\end{pmatrix}_i = \text{arctan2}\left(\sum_{t=1}^{T} p_{t,i}\ \sin\left[\begin{pmatrix}\tilde{\phi}\\ \tilde{\psi}\\ \tilde{\omega}\end{pmatrix}_t\right],\ \sum_{t=1}^{T} p_{t,i}\ \cos\left[\begin{pmatrix}\tilde{\phi}\\ \tilde{\psi}\\ \tilde{\omega}\end{pmatrix}_t\right]\right), \tag{5.7}$$

where T is the total number of angular triplets in the angular matrix (analogous to the number of mixture components used in the external mixture model in the preliminary model).

After the dihedral angles have been predicted by the network, a number of steps are performed:

1. The angular loss is computed as described in Section 2.3.7.1
2. The dihedral representation of the protein backbone is transformed into the Cartesian coordinate representation via the NeRF algorithm described in Section 2.2.
3. The dRMSD loss is computed as described in Section 2.3.7.2
4. Weights in the predictive network $\{\mathbf{W},\ \mathbf{b}\}$ as well as angular triplets in the angular matrix $(\tilde{\boldsymbol{\phi}},\ \tilde{\boldsymbol{\psi}},\ \tilde{\boldsymbol{\omega}})^T$ are updated.

During the implementation, the following configurations were chosen in relation to hyperparameters and related elements :

- Optimizer: The Adam optimizer with AMSGrad correction was used in all models. Experiments were carried out using stochastic gradient descent with momentum. However, the Adam optimizer seemed to generalize performance slightly better. The use of the AMSGrad correction also seemed to improve performance slightly in some cases.
- Learning rate: The learning rate for all models was selected manually and heuristically at $\eta = 0.001$. A random hyperparameter search should have been employed for all models, but was eventually omitted due to a lack of time and computational resources. However, it was found that the model performance was generally robust to small variations in η, most likely due to the adaptive nature of the Adam algorithm.
- Batch size: Experiments were performed with batch sizes ranging from 1 to 5 proteins (using the appropriate loss norm scaling). Batch sizes larger than 5 proteins were not immediately possible due to limitations in GPU RAM. In the end, a batch size of 5 proteins were selected for all models, however, the overall results did not seem very sensitive to batch sizes. Perhaps surprisingly, results comparable to those posted in Section 5.3.2 were obtained using a batch size of 1 protein (with the number of training iterations scaled accordingly). These findings are supported by recent experimental results, which suggest that very small batches are sometimes able to generalize better than large batch sizes due to an avoidance of sharp minima [33].
- Initialization: Angular triplets were initialized such that: i) ϕ and ψ values in the angular matrix matched the mixture component centroids μ and ν of the externally trained mixture model from the preliminary model (see Section 5.1), ii) ω values exhibited a 10/90 split between values in the ranges [0;60] and [110;180] degrees respectively in order to match the observed distribution from Figure 5.6.

In addition to the above, the following observations were made during the development and testing phase:

- Training set: All models were trained on the training_30 set in order to ensure maximum information per minibatch as well as a reasonable number of achievable epochs. The number of iterations used in a standard training session was 14,000, with one iterations corresponding to the processing of a single batch of 5 proteins. Due to the large size of the training set (22,344 proteins), a standard training session therefore only corresponded to 0-3 epochs depending on the random sampling process. In comparison, a similar training session using the training_100 set (87,573 proteins) would amount to less than a single epoch. Total training times were typically 2-3 days depending on the size of the model using the computational set-up described in Section 3.1.4.
- Batch normalization: Batch normalization on the LSTM output $\boldsymbol{p}$ was critical in order for the LSTM to learn at an acceptable rate. Batch normalization is known to reduce covariate shifts, which appeared to be critical in the interface between the LSTM and the angular matrix parameters. This final layer of batch normalization was not strictly necessary in the convolutional model due to the multiple batch normalization layers throughout each residual blocks.
- Numerical stability: A significant amount of time was spent on initiatives related to increasing the numerical stability of the models. In practice, model weights would become `NaN` at some point during training - often after having trained for 1-2 days. The following actions were taken in order to increase the numerical stability of the models:

  a) Gradient value clipping to range [-5;5].
  b) Changing softmax layer to log-softmax layer (as described in Equation 2.10).
  c) Adding fixed $\epsilon = 10^{-4}$ value to denominator of the arctan2 function (see Equation 3.2.
  d) Excluding proteins with a sequence length lower than 10 residues.
  e) Increasing batch-size from a single protein to several.
  f) Limiting the number of angular triplets T to 500 and less.

  Each of the above actions b-f increased the numerical stability of the models slightly. Late in the project, a bug was found in the gradient clipping method such that the clipped values were returned but not re-fed to the network. Following this discovery, all `NaN` errors were removed completely after a proper gradient clipping method had been implemented.

Model	T	Primary loss	FM dRMSD [Å]	TBM dRMSD [Å]
CNN-LSTM	500	Global & local	11.8	13.5
CNN-LSTM	2,000	Global	12.2	13.8
CNN-LSTM	500	Global	12.3	13.9
CNN	500	Global	13.1	15.1
CNN-LSTM	100	Global	14.6	16.8
LSTM	500	Global	16.0	18.1
CNN-LSTM	500	Local	53.7	58.8

Table 5.10: Sorted overview of final model experimental results.

- Data issues: It was found during debugging that the masking vectors of the ProteinNet-CASP11 data did not fit perfectly for a small number of proteins in the training set. Specifically, it appeared that for 134 proteins in the training_30 set, the tertiary structure of the masked protein data had an unmasked coordinate row at the first atomic position, such that the first atom appeared to be placed at origin. As the rest of the atomic positions were located at coordinate points with values in the 200s or 300s Å, there appeared to be an unnaturally long bond length between the first atomic position and the rest of the atomic backbone (typical bond lengths for most residues are around 1.5 Å [17]). Effectively, this produced very large dRMSD loss values for the 134 erroneous protein targets, which seemed to impair training. In order to mitigate this issue, a check was implemented in the dRMSD loss function in order to mask the first atomic position in the case of an erroneous protein target.

5.3.2 Results

A number of different experiments were carried out during the course of this project in order to optimize the performance of the final model. A summary of these results are presented in Table 5.10.

It should be noted that the predictive results of each trained model varied slightly from session to session (roughly in the range 0-2 Å for the dRMSD loss). As a result, the presented results have therefore been selectively picked in order to represent the observed general trend. Ideally, standard deviations should have been included for all presented results, however this was not possible due to a lack of time and computational resources.

A complete code-base for all relevant models as well as as a pre-processed version of the ProteinNet-CASP11 test and validation set is available online at: `https://github.com/lblaabjerg/Master`. Relevant training sets have been omitted due to size restrictions. All models have been implemented in PyTorch 0.4.0 [45] using standard PyTorch and Python libraries.

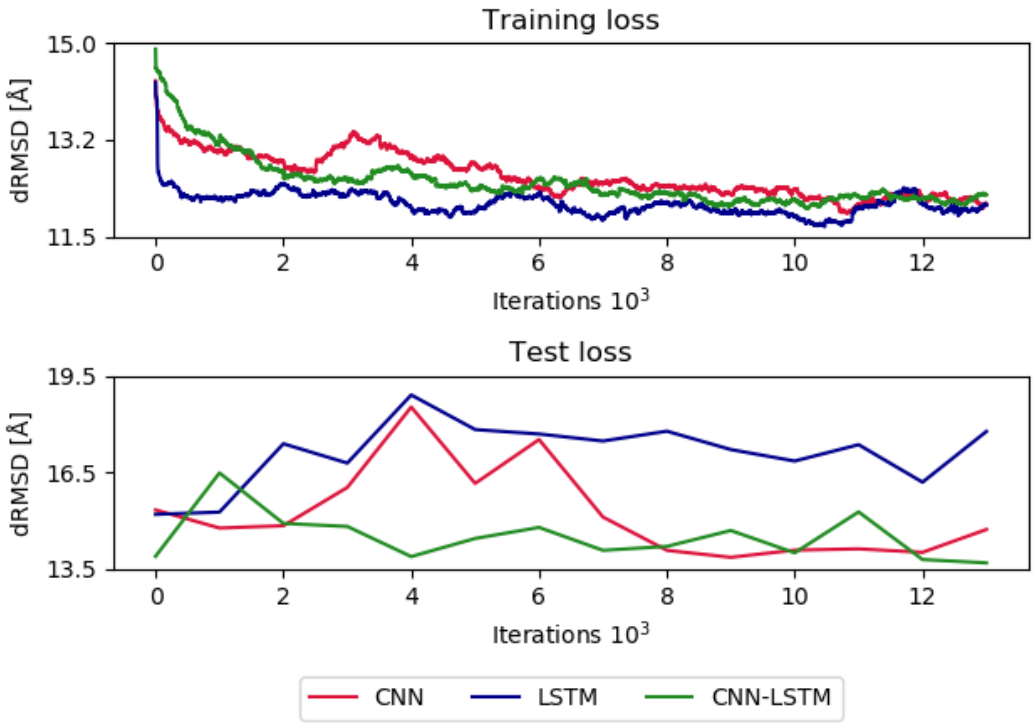

Figure 5.11: Comparison of training and test dRMSD loss using CNN, LSTM and CNN-LSTM models. Training loss smoothed via moving average.

5.3.2.1 Network architectures

The performance of three different network architectures were tested using: i) only the CNN model, ii) only the LSTM model, iii) the combined CNN-LSTM model from Figure 5.9. All three models were tested with an angular matrix size of $T = 500$ (i.e. 500 angular triplets). The number of learnable parameters in the LSTM model and the CNN model were made to be of a comparable size with 12,342,300 learnable parameters in the LSTM model and 13,486,500 learnable parameters in the CNN model respectively. The results from the experiment is presented in Figure 5.11. It is seen, that the difference between the three different models is generally small and that all three models exhibit the same type of learning behaviour in which convergence is reached very fast after which the learning seems to reach a plateau. Despite this behaviour, it seems like the LSTM converges the fastest on the training data but generalizes the worst, whereas the CNN converges slower on the training data but generalizes better. The CNN-LSTM seems to converge relatively slowly on the training data as well but generalize the best. Although difficult to see, the training loss is in fact decreasing very slowly for all three models. These results suggest that the CNN-LSTM model generally works slightly better than either the CNN model alone or the LSTM model alone, but that both of these models have enough representational power individually to produce a comparable level of predictions. It can be tentatively concluded, that the size of the model employed (i.e. the total number of learnable parameters) only seems to be a minor driver of increased performance.

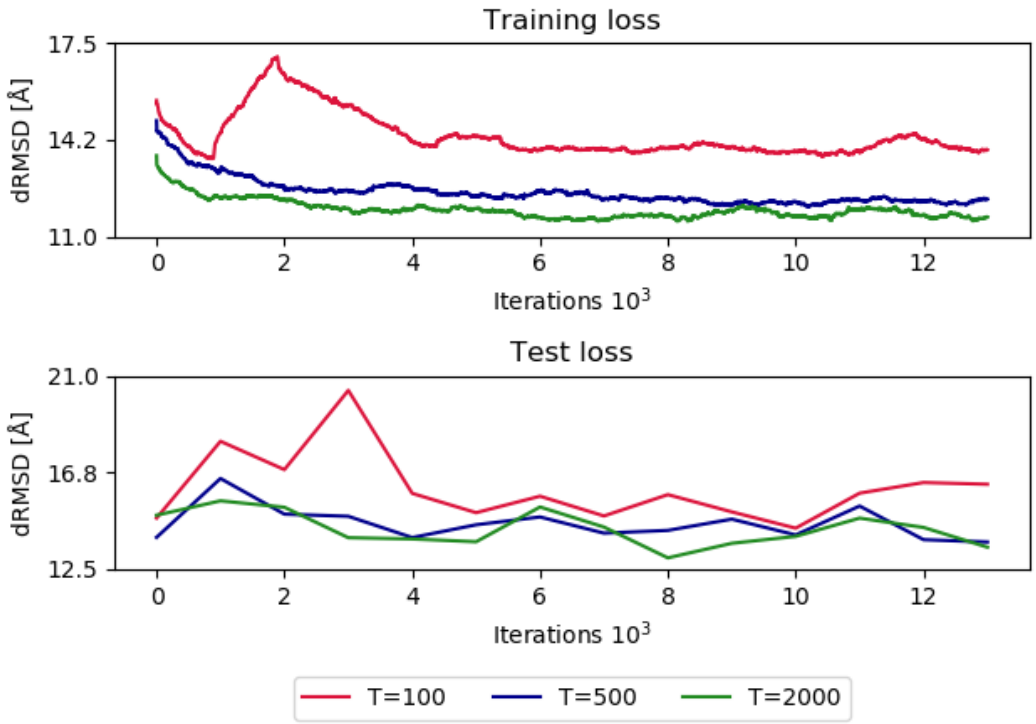

Figure 5.12: Comparison of training and test dRMSD loss using a CNN-LSTM model with 100, 500 and 2,000 angular triplets. Training loss smoothed via moving average.

5.3.2.2 Angular matrix size

As discovered during the development of the preliminary model, a strong local representation seems important in producing an accurate global representation of the protein structure. In order to test this hypothesis, three different variations of the CNN-LSTM model was developed using different angular matrix sizes, i.e.: i) $T = 100$ angular triplets, ii) $T = 500$ angular triplets, iii) $T = 2,000$ angular triplets. The results are presented in Figure 5.12. A small increase in both training and test performance is observed when increasing the size of the angular matrix, although the absolute difference in losses from $T = 100 \rightarrow T = 500$ is greater than the absolute difference from $T = 500 \rightarrow T = 2000$. These results suggest that the optimal model should have at least 2,000 angular triplets in its computational set-up and perhaps slightly more for optimal performance, although the gain in performance seems to saturate above $T = 2000$. The results of this experiment is noted to be in good agreement with the findings in Figure 5.4.

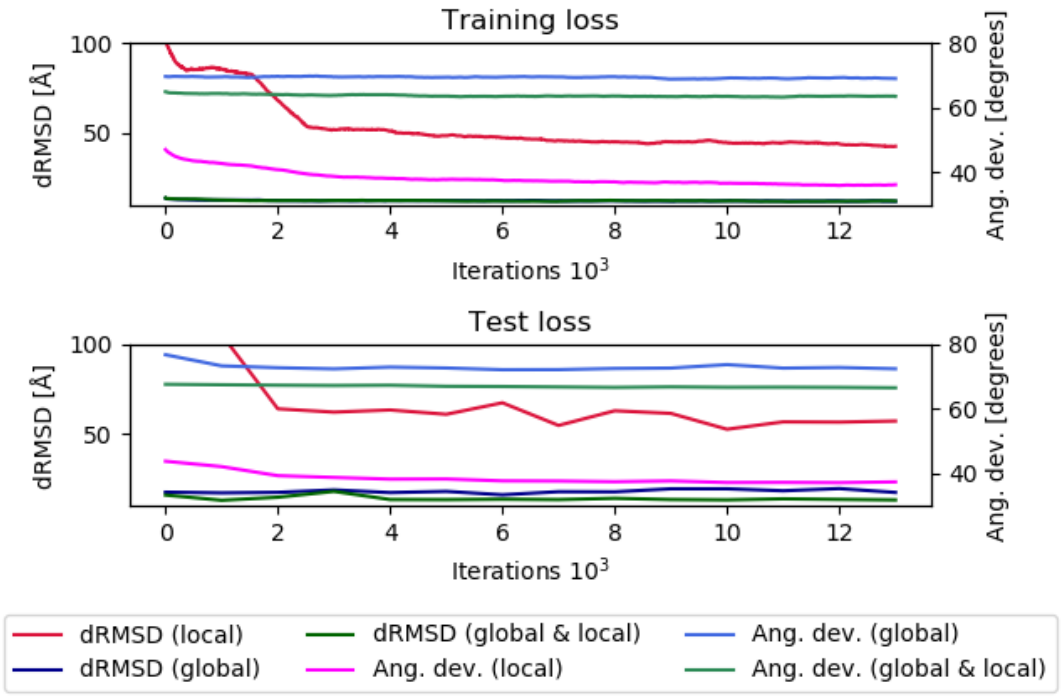

Figure 5.13: Comparison of training and test dRMSD loss and angular deviation loss using a CNN-LSTM model with $T = 500$ and multitask weights as: i) ang. dev loss approximately 1/10 of dRMSD loss (global), ii) dRMSD loss approximately 1/10 of ang. dev. loss (local), iii) dRMSD loss approximately equal to ang. dev. loss (global and local). Training loss smoothed via moving average.

5.3.2.3 Multitask loss weights

Multitask loss weights. As discussed in Section 2.3.6, multitask learning can sometimes be a powerful technique for overcoming learning plateaus. In order to test this assumption, three different CNN-LSTM models with $T = 500$ angular triplets was trained using different sets of global and local loss weights: i) dRMSD loss set to approximately 1/10 of angular deviation loss (global model), ii) angular deviation loss set to approximately 1/10 of dRMSD loss (local model), iii) dRMSD loss set approximately equal to angular deviation loss (global & local model). The results are presented in Figure 5.13. As expected, the global model achieves a very fast convergence to a relatively low dRMSD loss plateau and a relatively high angular deviation loss plateau. Conversely, the local model converges fast to a relatively low angular deviation loss plateau with the dRMSD loss slowly converging but remaining at a high value. Perhaps surprisingly, it is observed that the global & local model consistently achieves lower test losses on both dRMSD and angular deviation compared to the global model. A possible explanation could be that only a relatively strong local loss signal is able to assist the global loss function in our model (i.e. unlocking the eavesdropping functionality presented in Section 2.3.6). These findings suggest that the optimal model should have multitask weights such that the angular deviation loss is approximately equal to the dRMSD loss (although the exact optimal balance needs to be further analyzed).

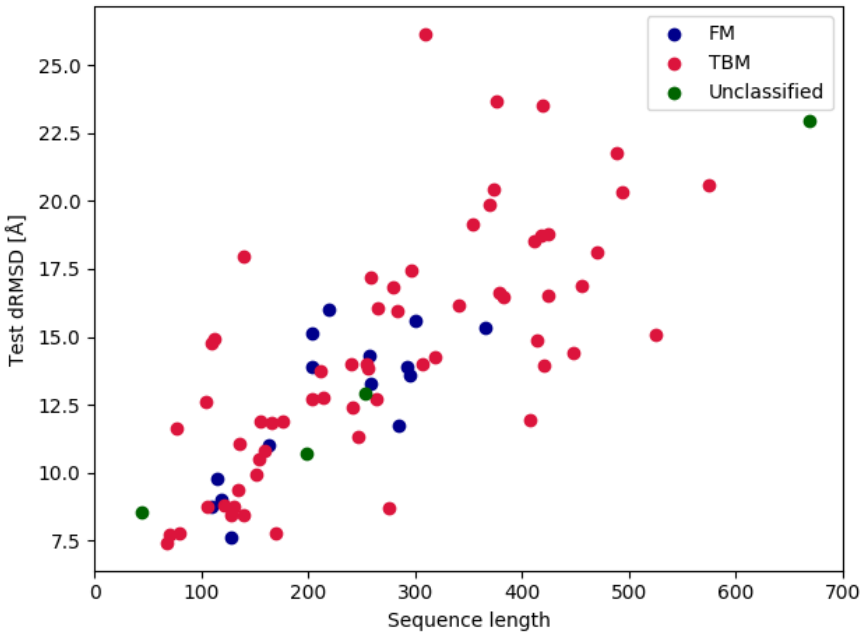

Figure 5.14: Overview of test dRMDS values versus sequence length and target class for best performing CNN-LSTM model.

Looking at the loss curves depicted in Figure 5.11, Figure 5.12 and Figure 5.13 one might immediately suspect that the generally observed premature convergence is caused by a too-high learning rate. However, experiments were carried out replicating listed experiments using significantly lower learning rates of $\eta = 0.0001$ and $\eta = 0.0005$ with no significant improvement in convergence and/or performance. In addition to the experiments listed above, it was also investigated whether scaling the dRMSD loss function could be used to increase performance. In order to test this, three global-loss CNN-LSTM models with $T = 500$ were trained using: i) $\text{dRMSD}^{1/2}$, ii) dRMSD, iii) dRMSD^2. However, this scaling did not seem to affect the performance of the models significantly.

5.3.2.4 Details on best model performance

It is seen from the results in Table 5.10 that the CNN-LSTM model with a balanced global and local loss performed the best in the listed experiments. Detailed results for this particular model is therefore presented below. In Figure 5.14, the distribution of test dRMSD results are presented as a function of protein sequence lengths and protein target class. It is seen, that long sequences are generally harder to predict than short sequences and how a large part of the predictive results can be traced back to this trend alone. Differences in test protein category does not seem to influence results significantly.

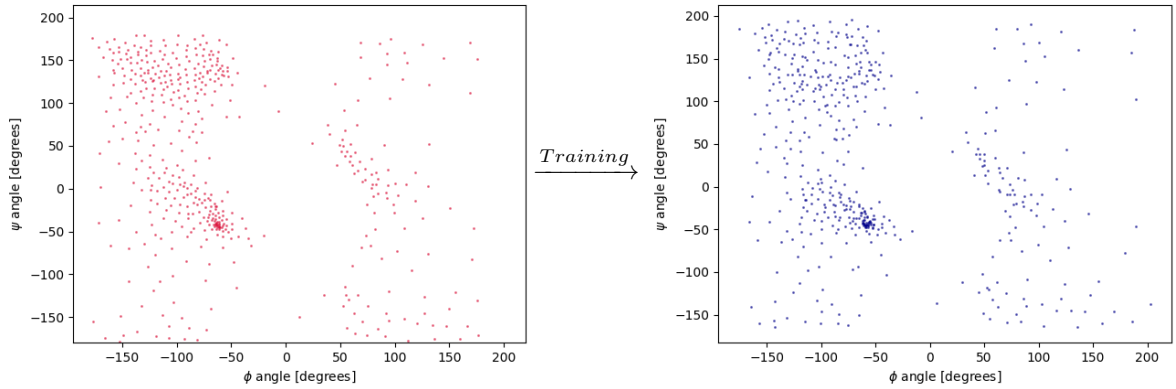

Figure 5.15: Visualization of 500 angular triplets (without ω angles) at initialization (Left) and after training (Right) for best performing CNN-LSTM model.

In Figure 5.15, a visualization of the angular triplets (without ω angles) is presented before and after training of the global & local CNN-LSTM model with angular matrix size $T = 500$. It is seen, that the angular matrix values travel outside the usual range $[-\pi,\ \pi]$ at which they are initialized. However, this is expected due to the periodicity of the arctan2 computations. Taking this periodicity into account, the angular matrix values before and after training are roughly the same, although there seems to be a slight trend towards more dispersion of the cluster regions during training.

A structural visualization of the best predictions of the global & local CNN-LSTM model is presented in Figure 5.16. These visualizations have been created using the standard PyMOL visualization tool [51] together with specifically developed scripts in the final model code-base. It is seen from the visualizations that the network is able to capture some of the very large-scale global structure of the protein targets (i.e. approximate positions of N- and C-terminus) but that the overall performance of the network is poor. Specifically, it seems that network has a bias towards only predicting different variations of the same type of twisted alpha-helix. This behaviour could be a sign of premature convergence, meaning that the network has only just started to recognize the most prevalent global structure (the alpha-helix), which it then uses as a preferred prediction with only small variations. Presumably, the network should be able to diverge from this type of single-minded prediction with more training and/or more advanced techniques designed to speed up the learning process. These observations, as well as their implications for further improving the performance of the network, will be discussed in-depth in the following chapters.

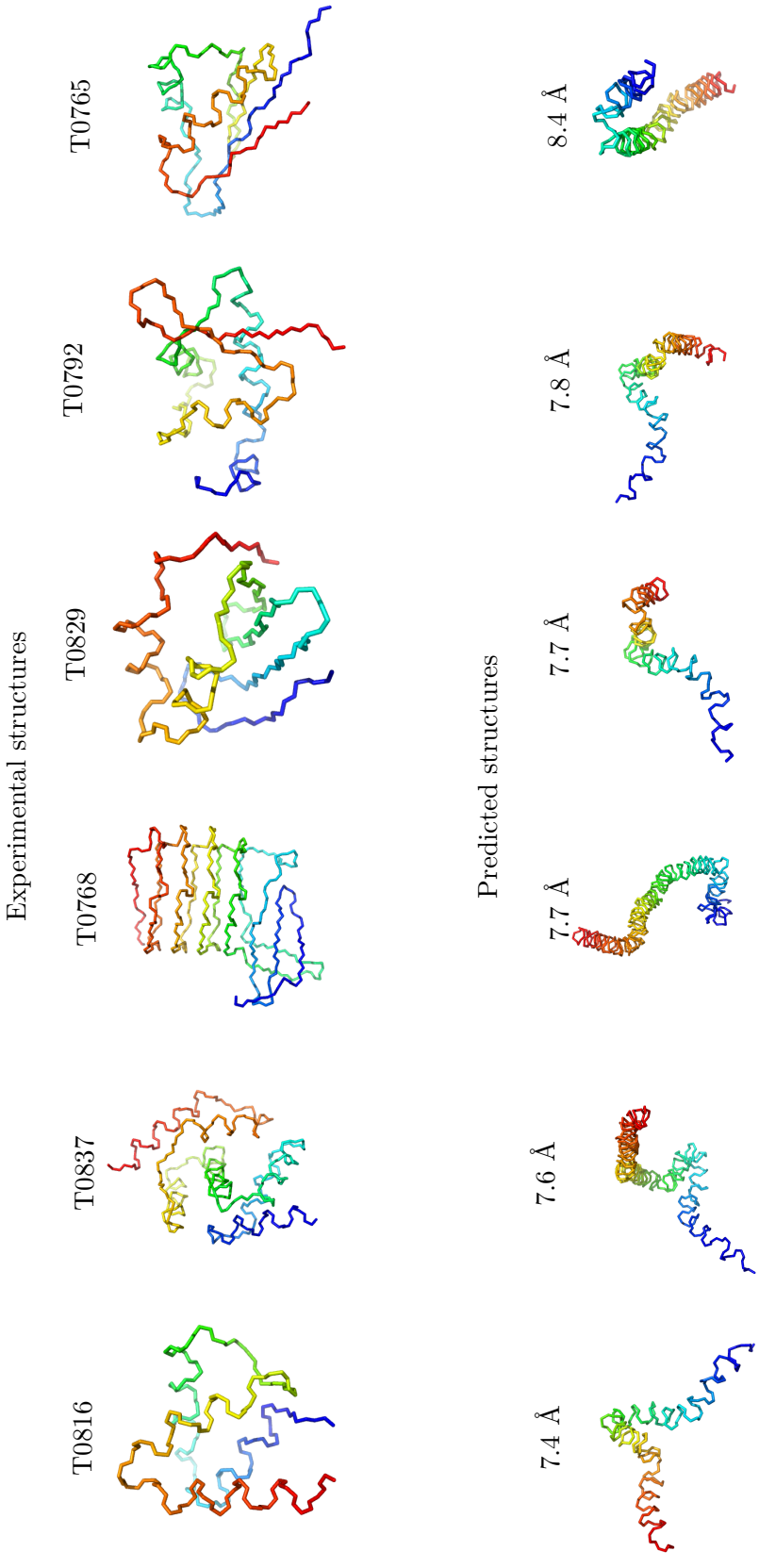

Figure 5.16: Top predictions and associated targets for the best performing CNN-LSTM model. Target IDs as well as dRMSD loss values are printed above each structure. All proteins have been colorized in a spectrum that spans the length of the structure in order to aid visualization.

CHAPTER 6

Discussion

The results presented in Table 5.10 are generally promising giving the notorious difficulty of protein tertiary structure prediction. The results obtained in this project are higher than the results presented by Al Quraishi [2], however, training times were also significantly shorter. Assuming an average training time of 3 days for our final model versus 2 months for Al Quraishi's model, our proposed model is then approximately faster than Al Quraishi's model by a factor of 20. Furthermore, our final model seems to be more robust than Al Quraishi's model, which was only able to reach the reported state-of-the-art performance in 1 out of 1,000 initializations.

Computationally, one major point of divergence between the proposed final model and Al Quraishi's model is the number of angular triplets T used. Reportedly, Al Quraishi's model uses only 40 angular triplets and is in general not very sensitive to the size of the angular matrix. Conversely, we have reported small increases in performance for up to 2,000 angular triplets and beyond (see Figure 5.12). It can be argued - at least theoretically - that only three angular triplets are needed in order to span the three-dimensional parameter space (i.e. the Ramachandran space plus the ω dimension). However, this seems unlikely in practice, as an extremely precise weighting of the three orthogonal angular triplets would be needed in order to obtain an accurate angular prediction for a given residue. Further analysis is needed in order to elucidate this issue.

As mentioned in Section 5.3.2, it is observed that almost all of the proposed final models exhibit some variation of same training curve that converges quickly and then reaches a plateau on both training and test loss. Different parameters - namely size of the angular matrix and relative local loss weight - seems to affect performance slightly, however, these parameters did not change the fundamental pattern of fast, early (and perhaps premature) convergence.

It seems that the key challenge in further improving the performance of the proposed CNN-LSTM model is related to discovering an efficient method to break the apparent loss plateau. It is seen from earlier results (e.g. Figure 5.8) that the proposed model is able to learn highly accurate predictions of protein structures for small data sets. This suggests, that the problem is not related to any immediate errors or bugs in the software or general implementation. Furthermore, experiments presented in Figure 5.11 and Figure 5.12 suggest that the underlying problem is not related to representational power (i.e. number of learnable parameters). Finally, serious issues with data quality seems unlikely since the ProteinNet-CASP11 data sets have been specifically tailored in order to meet the demand for "clean" biological data and have already been successfully used by Al Quraishi in his published work.

A likely hypothesis regarding the nature of the loss plateau could therefore be that the slowdown in network learning is caused by a highly non-convex error surface with large near-flat areas and a likely proliferation of saddle points. This hypothesis would help explain the extremely long training time of Al Quraishi's model, as saddle points are known to be notoriously slow to escape from using first-order optimizers (although not impossible, assuming a random component to the step direction of the optimizer [19]). Further intuition regarding this hypothesis, as well as three proposed ways to mitigate this issue, is presented in Section 7.4.

CHAPTER 7

Future work

In the following chapter, a number of concrete suggestions to improve the performance of the presented final model is introduced. These ideas range from alternative training strategies to improved mathematical operations and completely new problem definitions. It should be noted that neither of these ideas have been tested on real data and that all proposed ideas therefore need further analysis and evaluation before implementation.

7.1 Further optimization and parallelization

In general, hyperparameter optimization of the final model was carried out manually and heuristically. This approach was chosen primarily due to a lack of time and computational resources. However, it is expected that significant improvement in performance can be obtained through a systematic, large-scale, random search process. Furthermore, simply training the models for longer should help increase performance as suggested by the results of Al Quraishi (see Section 4).

In order to speed up training times, computational parallelization of training data batches could be implemented. Generally, parallelization works by splitting a batch of training examples into separate examples, which are then run through the computational graph separately and in parallel using multiple GPUs. This task is easily accomplished in PyTorch using the `torch.nn.DataParallel` container developed specifically for this purpose [46]. Parallelization together with enough free GPUs could help decrease the wall-clock time needed to reach state-of-the-art performance.

7.2 Pre-training on short proteins

As seen in Figure 5.14, short protein structures are usually easier to predict than long protein structures. Theoretically, this relationship could be exploited in order to improve performance by pre-training the network on a subset of the training set containing only short proteins (e.g. sequence lengths below 200 residues) before training the network on the full training set. This would allow the network to learn the basic motifs of protein tertiary structures (e.g. beta hairpins, alpha helices and so on) in an easier and less complicated learning environment. After reaching a certain pre-training performance, the model could then be allowed access to the full training set. In theory, this approach would allow the network to get a head start in predicting the more complicated structures in the training set, since most long proteins are simply extensions and/or multi-domain collections of the basic structural motifs. However, parameters such as size and composition of the pre-training set needs to be carefully tuned in order to avoid degradation of the networks ability to generalize after training.

7.3 Local loss weight schedules

Increases in performance might be obtained by using a gradually increasing local loss weight schedule during training. Ideally, this would allow the network to optimize the global structures as much as practically possible via the global loss function, after which the local loss weight would then gradually increase. In theory, this would help the network leverage the eavesdropping ability of multitask learning (as introduced in Section 2.3.6) in a more sophisticated way, thereby allowing the local loss to guide the global loss through the apparent loss plateau. Although plausible, it is presently unknown whether or not a local loss weight schedule would work better than the fixed local loss weight approach which has been implemented in this project. Further analysis is needed in order to determine the optimal weight schedule function (e.g. exponential decay, cosine decay, with or without warm restarts and so on).

7.4 Attacking saddle points

As mentioned in the discussion, a highly non-convex error surface with large near-flat areas and a likely proliferation of saddle points could help explain the existence of the observed loss plateau. Intuitively, the proliferation of saddle points can be demonstrated through a simple toy-model showing the low probability of encountering a minima (or maxima) in high-dimensions. Assuming a diagonalized Hessian with n parameters:

$$\mathbf{H} = \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix},$$

where a minima corresponds $\mathbf{H}$ being positive-definite, i.e.: $d_1 > 0, ..., d_n > 0$. Assuming that the partial derivatives $d_1, ...d_n$ are not biased, we can assume a probability: $P(d_i > 0) = 1/2$. Furthermore, we can assume every d_i to be independent due to the strong non-linearity of the Hessian in complex neural networks. Using these simple assumptions, the probability of encountering a minima (or maxima) becomes: [40]:

$$P(minima) = P(d_1 > 0, ...d_n > 0) = P(d_1 > 0) \, ... \, P(d_n > 0) = \frac{1}{2^n} \, . \tag{7.1}$$

As n is typically in the range of thousands or millions for modern deep learning networks, the probability of encountering a minima (or maxima) in high dimensions quickly becomes very low. A more correct and rigours derivation of why saddle points are a bigger challenge than local minima for neural networks is found in [13].

Many different strategies have been proposed in order to mitigate this issue. The straightforward solution is simply to extend the training time of the network, as saddle points have been proven to be negotiable for stochastic first-order optimizers given enough training time [19]. However, in practice, this solution is cumbersome and requires a significant amount of computational resources. Instead, the two most promising strategies seem to be: 1) Implementation of advanced optimization techniques 2) Implementation of warm-restart learning rate schedules.

In the case of advanced optimization techniques, there are several candidate solutions available. Typically, these optimizers work by scaling the update step with the inverse of the curvature of the error surface, such that flat areas are transversed faster. This approach is what defines the Newton-Raphson update [9]:

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \mathbf{H}^{-1}\mathbf{g}_t \,, \tag{7.2}$$

where $\mathbf{g}_t$ is the error gradient with respect to $\boldsymbol{\theta}_t$ at time step t and $\mathbf{H}^{-1}$ is the inverse Hessian matrix. However, the classical Newton-Rapshon update suffers from two main issues: i) the exact Hessian is almost always too computationally expensive to implement in practice, ii) the unmodified Newton-Raphson update will converge to any type of stationary point (i.e. minima, maxima or saddle point) in a non-convex setting. The update must therefore be modified in such a way that steps are always taken in a direction aligned with the usual gradient step direction [13]. Mitigating these two challenges is an ongoing are of research and careful thought must be given to the exact choice of optimizer. A promising solution has been proposed by Reddi and Zaheer et al. in a recent paper in which they introduce an optimizer that efficiently alternates between first- and second-order optimization, with the latter mode used only close to saddle points [48].

Alternative, cyclical learning rate schedules have also proven effective in attacking saddle points [54]. This is due to the fact that periodically increasing the learning rate can help the optimizer transverse the nearly-flat regions of a saddle point instead of being effectively stuck using a small learning rate. An example of a cyclical learning rate schedule is the cosine annealing schedule with warm-restarts, where the learning rate is given by [36]:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{current}}{T_{max}}\pi\right)\right). \tag{7.3}$$

where η_t is the current learning rate, η_{min} is the minimum learning rate, η_{max} is the maximum learning rate and T_{max} is the number of iterations at which $\eta_t = \eta_{min}$.

7.5 A curvature-aware angular mean

A well-known problem in directional statistics is that standard approaches typically disregard the curvature of the underlying sample space [22]. This too is an issue in our approach, where a predicted angle is computed as a weighted mean of angular triplets via the arctan2 method (as discussed in Section 5.1.1). It can be shown, that the arctan2 method is the maximum likelihood estimator of the mean parameter of the von Mises distribution [9]. However, the von Mises distribution is fundamentally flawed in that it is constructed by using Euclidean metrics to restrict an isotropic normal distribution to lie only on the unit n-sphere. The general von Mises distribution can be written as [22]:

$$\begin{aligned} \mathrm{vM} &\propto \exp\left(-\frac{\kappa}{2}\|\mathbf{x} - \boldsymbol{\mu}\|^2\right), \\ \|\mathbf{x}\| &= \|\boldsymbol{\mu}\| = 1, \quad \kappa > 0\,. \end{aligned} \tag{7.4}$$

It is seen that the von Mises distribution is constructed using the Euclidean distance measure $\|\mathbf{x} - \boldsymbol{\mu}\|^2$.

A more suitable distance measure is the arc-length of the shortest connecting curve on the n-sphere, which amounts to the angle between two points, i.e. $\text{dist}(\mathbf{x}, \boldsymbol{\mu}) = \arccos(\mathbf{x}^T \boldsymbol{\mu})$ [22]. From this measure, the spherical normal distribution is obtained [22]:

$$\begin{aligned} \text{SN} \propto \exp\left(\frac{-\lambda}{2}\arccos^2(\mathbf{x}^T \boldsymbol{\mu})\right), \\ \|\mathbf{x}\| = \|\boldsymbol{\mu}\| = 1, \quad \kappa > 0\,. \end{aligned} \tag{7.5}$$

By maximizing the likelihood of the spherical normal distribution instead of the von Mises distribution, a different estimate of the angular mean is obtained. Algorithm 1 present a procedure for computing this mean [22].

Algorithm 1: Offline maximum likelihood

1 Initialize: $\boldsymbol{\mu}$ as a random point;
2 **repeat**;
3 $\quad \nabla_{\boldsymbol{\mu}} \leftarrow -\frac{1}{N}\sum_{n=1}^{N} \text{Log}_{\boldsymbol{\mu}}(\mathbf{x}_n)$;
4 $\quad \boldsymbol{\mu} \leftarrow \text{Exp}_{\boldsymbol{\mu}}\left(-\frac{1}{2}\nabla_{\boldsymbol{\mu}}\right)$;
5 **until** $\|\nabla_{\boldsymbol{\mu}}\| \leq 10^{-5}$;

where the logarithm map is defined as:

$$\begin{aligned} \text{Log}_{\boldsymbol{\mu}}(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}\,(\mathbf{x}^T \boldsymbol{\mu}))\frac{\theta}{\sin(\theta)}\,, \\ \theta = \arccos(\mathbf{x}^T \boldsymbol{\mu}) \end{aligned} \tag{7.6}$$

and the exponential map is defined as:

$$\text{Exp}_{\boldsymbol{\mu}}(\mathbf{v}) = \boldsymbol{\mu}\cos(\|\mathbf{v}\|) + \frac{\sin(\|\mathbf{v}\|)}{\|\mathbf{v}\|}\mathbf{v}\,. \tag{7.7}$$

The general idea in Algorithm 1 is to minimize the negative log-likelihood of the data by a gradient descent approach. This is done by iteratively mapping the points on the n-sphere to a linear tangent space $\mathcal{T}_{\boldsymbol{\mu}}$. This tangent space mapping is useful, as it linearizes the spherical space enabling the computation of the Euclidean average (see Figure 7.1 for an illustration). The procedure works as follows. First, the points on the sphere $\mathbf{x}$ is mapped to the linear tangent space $\mathcal{T}_{\boldsymbol{\mu}}$ via the logarithmic map Log_{μ}. Then, the average tangent is computed $\nabla_{\boldsymbol{\mu}}$. Finally, this tangent is mapped back to the sphere via the exponential map Exp_{μ} and the mean is updated. Algorithm 1 is straightforwardly converted to a weighted mean by replacing the uniform weight $1/N$ with a specified weight p_n (as is relevant to our case) and a relatively accurate estimate of the mean $\boldsymbol{\mu}$ can in most cases be obtained within 5-6 iterations[1].

[1] Source: Hauberg, personal communication.

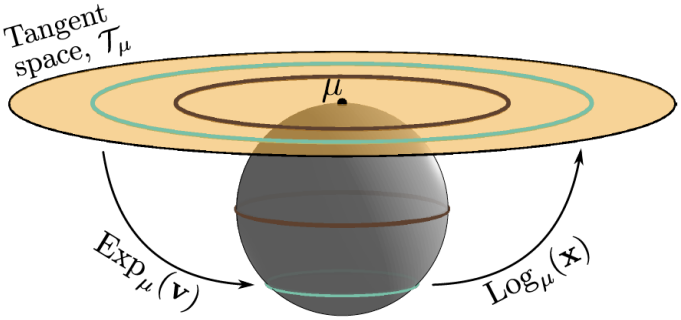

Figure 7.1: The tangent space $\mathcal{T}_\mu$ of the sphere and related operators [22].

7.6 Alternative representations of protein structure data

In this project, the protein structure prediction problem has been formulated as a classification problem using a discrete number of angular triplets used to compute a continuous weighted mean. However, several other alternative structure representations could have been used. For example, all dihedral angles could have been represented categorically, such that each possible angle is binned to a subinterval in the range $[-\pi;\ \pi]$. In this set-up, the network is trained to predict the relevant dihedral angle-bin for each residue and the mean value of the predicted bin is then used in future transformations (i.e. to Cartesian coordinates or something else). However, the results in Section 5.1 suggest that a highly accurate local representation is needed in order to achieve a reasonable global representation thereby necessitating a very large number of bins. Furthermore, it is unclear how the large number of largely unpopulated bins outside the usual Ramachandran areas would impact training dynamics.

Alternatively, the problem could be reformulated as a regression problem using the sine/cosine values of the raw angular values as prediction targets. This approach has - at least in some cases - been shown to outperform the discrete methods [50] and would also eliminate the need for a trainable angular matrix thereby reducing model complexity. However, in general, neural network models are more naturally applied to classification problems and further analysis (i.e. in the use of an appropriate activation function) is needed in order to determine the feasibility of this approach.

Finally, completely different ways of representing the protein structure could be tested. One example is the Parinello-Behler (PB) descriptor [8]. However, although the PB descriptor requires significantly fewer computational resources than the bin-type approach, recent work has suggested that the PB descriptor and related approaches, such as the Coloumb matrix, tend to produce inferior results compared to the straightforward bin-type approach [1].

CHAPTER 8

Conclusion

This project has attempted to tackle the problem of accurate protein structure prediction through the use of novel deep learning techniques. It has been shown that a potential breakthrough in protein tertiary structure prediction could help advance a wide range of scientific fields, such as drug discovery and biotechnology. Furthermore, it has been shown that deep learning seems to be a promising candidate in solving this challenge. Throughout this project, various protein tertiary structure prediction models have been developed and implemented and a number of insights have been gathered.

In the first part of this project, a preliminary model was developed and evaluated. The preliminary model relied on a pre-trained mixture model used to predict ϕ and ψ angles of the protein backbone. Although promising, the preliminary model was shown to suffer from a non-reducible error term in its local (i.e. angular) representation of the protein structure. It was suggested that the source of this error-term originated in the pre-trained mixture model in which the mixture components used in the predictive process were fundamentally fixed and limited in numbers. As a consequence of these discoveries, the preliminary model approach was ultimately rejected.

In the second part of this project, a final model was developed in accordance with the novel technique of end-to-end differentiability recently proposed by Mohammed Al Quraishi. In addition to the techniques introduced by Al Quraishi, the final model introduced a number of minor changes as well as two major improvements: i) a combination of two separate network architectures, i.e. a residual CNN combined with an LSTM using a skip connection, ii) an auxiliary loss function (the angular deviation loss) used to leverage the known advantages of multitask learning. In order to test the limits of this final model, a number of different experiments were carried out. It was discovered that the performance of the model was slightly improved by utilizing a high number of angular triplets in the angular matrix as well as a balanced weighting between local and global loss feedback. However, learning was in every experiment impaired by a surprising dynamic in which convergence was attained very fast after which learning seemed to plateau.

In order to mitigate this issue of premature convergence, three alternative strategies have been proposed. Furthermore, a small list of techniques have been introduced which could help improve the performance of the final model going forward. Summarizing, the results presented in this project seem promising, although currently far from state-of-the-art, with future improvements expected through continued analysis and development.

CHAPTER 9

Appendix

9.1 Appendix 1: Derivation of soft-label cross-entropy loss

Assuming the final prediction is made in the network via a softmax layer:

$$p_i = \frac{\exp(h_i)}{\sum_j \exp(h_j)}\,, \tag{9.1}$$

where h_i is the activation of the i'th output unit.

The cross-entropy of a single prediction becomes:

$$\begin{aligned} H(\boldsymbol{h}, \boldsymbol{t}) &= -\sum_i t_i \, \log(p_i)\,, \\ &= -\sum_i t_i \, [h_i - \log \sum_j \exp(h_j)]\,, \\ &= -\sum_i t_i \, h_i + \log \sum_j \exp(h_j)\,. \end{aligned} \tag{9.2}$$

Bibliography

[1] Absalonsen, J. *Evaluation and Comparison of Several Representations of Spatial Protein Data For Use In Deep Learning*. Bachelor's project, unpublished. University of Copenhagen. Supervisor: Boomsma, W. 2018.

[2] Al Quraishi, M. *End-to-end Differentiable Learning of Protein Structure*. bioRxiv preprint, doi: http://dx.doi.org/10.1101/265231. 2018.

[3] Al Quraishi, M. *ProteinNet*. `https://github.com/aqlaboratory/proteinnet`. 2018.

[4] Alberts, B. et al. *Molecular Biology of the Cell*. Garland Science, 2014.

[5] Anfinsen, C. B. et al. "The Kinetics of Formation of Native Ribonuclease During Oxidation of the Reduced Polypeptide Chain". In: *Proceedings of the National Academy of Sciences of the United States of America* 47.9 (1961), pages 1309–1314.

[6] Bai, S., Kolter, J. Z., and Koltun, V. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. arXiv preprint: 1803.01271v2 [cs.LG]. 2018.

[7] Baxter, J. "A Model of Inductive Bias Learning". In: *Journal of Artificial Intelligence Research* 12 (2000), pages 149–198.

[8] Behler, J. "Atom-centered Symmetry Functions For Constructing High-Dimensional Neural Network Potentials". In: *Journal of Chemical Physics* 134.7 (2011), page 074106.

[9] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.

[10] Boomsma, W. et al. "A Generative Probabilistic Model of Local Protein Structure". In: *Proceedings of the National Academy of Sciences* 105.26 (2008).

[11] Caruana, R. "Multitask learning". In: *Machine Learning* 28.1 (1997), pages 41–75.

[12] Collette, A. *Python and HDF5*. O'Reilly Media, 2013.

[13] Dauphin, Y. N. et al. "Identifying and Attacking the Saddle Point Problem In High-Dimensional Non-Convex Optimization". In: *Neural Information Processing Systems* (2014).

[14] Dertat, A. *Applied Deep Learning - Part 4: Convolutional Neural Networks*. Blog-post, available at `https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2`. 2017. (Visited on June 13, 2018).

[15] *Dihedral/Torsion Angle From Four Points in Cartesian Coordinates in Python*. URL: `https://stackoverflow.com/questions/20305272/dihedral-torsion-angle-from-four-points-in-cartesian-coordinates-in-python` (visited on March 12, 2018).

[16] Dill, K. A. and MacCullum, J. L. "The Protein-Folding Problem, 50 Years On". In: *Science* 338.6110 (2012), pages 1042–1046.

[17] Engh, R. A. and Huber, R. “Accurate Bond and Angle Parameters For X-Ray Protein Structure Refinement”. In: *Acta Crystallographica Section a* 47.4 (1991), pages 392–400.

[18] Gajda, M. J., Pawlowski, M., and Bujnicki, J. M. “Protein Structure Prediction: From Recognition Of Matches With Known Structures To Recombination Of Fragments”. In: *Multiscale Approaches To Protein Modeling: Structure Prediction, Dynamics, Thermodynamics and Macromolecular Assemblies* (2011), pages 231–254.

[19] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. The MIT Press, 2016.

[20] Graves, A. *Supervised Sequence Labelling With Recurrent Neural Networks*. Springer, 2012.

[21] Handl, J. et al. “The Dual Role of Fragments In Fragment-Assembly Methods For De Novo Protein Structure Prediction”. In: *Proteins-structure Function and Bioinformatics* 80.2 (2012), pages 490–504.

[22] Hauberg, S. *Directional Statistics With the Spherical Normal Distribution*. Pre-print. 2018.

[23] Havtorn, J. *How do I draw an LSTM cell in Tikz?* Forum-post, available at `https://tex.stackexchange.com/questions/432312/how-do-i-draw-an-lstm-cell-in-tikz`. 2018. (Visited on June 12, 2018).

[24] He, K. et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition* (2016), pages 770–778.

[25] He, K. et al. “Delving Deep Into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *2015 IEEE International Conference on Computer Vision* (2015), pages 1026–1034.

[26] He, K. et al. “Identity Mappings in Deep Residual Networks”. In: *2016 European Conference on Computer Vision* (2016), pages 630–645.

[27] Horton, R. B. et al. *Principles of Biochemistry*. Pearson Prentice Hall, 2006.

[28] Huang, G. et al. “Densely Connected Convolutional Networks”. In: *Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE Computer Society Conference* 2017-January (2017), pages 2261–2269.

[29] Ioffe, S. and Szegedy, C. “Batch Normalization: Accelerating Deep Network Training By Reducing Internal Covariate Shift”. In: *32nd International Conference on Machine Learning* 1 (2015), pages 448–456.

[30] Johnson, M. et al. “Google’s Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation”. In: *Transactions of the Association for Computational Linguistics* (2017), pages 339–351.

[31] Juan, D. de, Pazos, F., and Valencia, A. “Emerging Methods In Protein Co-Evolution”. In: *Nature Reviews Genetics* 14.4 (2013), pages 249–261.

[32] Kendrew, J. C. et al. “A Three-Dimensional Model Of the Myoglobin Molecule Obtained By X-Ray Analysis”. In: *Nature* 181.4610 (1958), pages 662–666.

[33] Keskar, N. S. et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”. In: *International Conference on Learning Representations* (2017).

[34] Kingma, D. P. and Ba, J. L. “Adam: A method For Stochastic Optimization”. In: *International Conference on Learning Representations* (2015).

[35] Krizhevsky, A., Sutskever, I., and Hinton, G. E. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pages 1097–1105.

[36] Loshchilov, I. and Hutter, F. "SGDR: Stochastic Gradient Descent with Warm Restarts". In: *International Conference of Learning Representations* (2017).

[37] *Lumen Microbiology*. URL: `https://courses.lumenlearning.com/microbiology/chapter/proteins/` (visited on March 26, 2018).

[38] Mardia, K. V., Taylor, C. C., and Subramaniam, G. K. "Protein Bioinformatics And Mixtures of Bivariate von Mises Distributions For Angular Data". In: *Biometrics* 63.2 (2007), pages 505–512.

[39] Marx, D. and Hutter, J. *Ab Initio Molecular Dynamics: Basic Theory and Advanced Methods*. Cambridge University Press, 2009.

[40] Masip, D. *Local minima vs saddle points in deep learning*. Forum-post, available at `https://datascience.stackexchange.com/questions/22853/local-minima-vs-saddle-points-in-deep-learning`. 2017. (Visited on June 15, 2018).

[41] Maxfield, K. *Why Structure Prediction Matters*. 2015. URL: `https://www.dnastar.com/blog/structural-biology/why-structure-prediction-matters/` (visited on March 26, 2018).

[42] Nielsen, M. *Neural Networks and Deep Learning*. Determination Press, 2015.

[43] Olah, C. *Understanding LSTM networks*. Blog-post, available at `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. 2015.

[44] Parsons, J. et al. "Practical Conversion From Torsion Space To Cartesian Space For In Silico Protein Synthesis". In: *Journal of Computational Chemistry* 26.10 (2005), pages 1063–1068.

[45] Paszke, A. et al. *Automatic differentiation in PyTorch*. 2017.

[46] *PyTorch DataParallel*. URL: `https://pytorch.org/docs/stable/_modules/torch/nn/parallel/data_parallel.html` (visited on June 15, 2018).

[47] Reddi, S. J., Kale, S., and S., Kumar. "On the Convergence of Adam and Beyond". In: *International Conference of Learning Representations* (2018).

[48] Reddi, S. J. et al. "A Generic Approach for Escaping Saddle points". In: *Proceedings in Machine Learning Research* 84 (2018).

[49] Ruder, S. *An Overview of Multi-Task Learning in Deep Neural Networks*. arXiv preprint: 1706.05098 [cs.LG].

[50] Schreiner, J. M. *Predicting Phi and Psi Angles On The Protein Backbone Using Artificial Neural Networks*. Bachelor's project, unpublished. Technical University of Denmark. Supervisor: Frellsen, J. 2018.

[51] Schrödinger, LLC. "The PyMOL Molecular Graphics System, Version 1.8". 2015.

[52] Schuster, M., Paliwal, K., and General, A. "Bidirectional Recurrent Neural Networks". In: *IEEE Transactions on Signal Processing* (1997).

[53] Siegelmann, H. T. and Sontag, E. D. "On The Computational Power Of Neural Nets". In: *Journal of Computer and System Sciences* 501.1 (1995), pages 132–150.

[54] Smith, L. N. “Cyclical Learning Rates for Training Neural Networks”. In: *IEEE Winter Conference on Applications of Computer Vision* (2017).

[55] Steinegger, M and Soeding, J. “MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets”. In: *Nature Biotechnology* 35.11 (2017), pages 1026–1028.

[56] Wang, S. et al. “Accurate De Novo Prediction of Protein Contact Map by Ultra-Deep Learning Model”. In: *PLOS Computational Biology* 13.1 (2017), e1005324.

[57] Zhang, Yang. “I-TASSER server for protein 3D structure prediction”. In: *BMC Bioinformatics* 9.1 (2008), page 40.