Adapting AlphaFold for RNA Structure Prediction

by

Candidate: KZVK3, Student Number: 20069950

September 2021

A Dissertation submitted in part fulfilment of the

Degree of Master of Engineering:

Machine Learning

Institute for Environmental Design and Engineering

Bartlett School of Environment, Energy and Resources

University College London

LIST OF FIGURES

3.8.1 Figure from [Belkin et al., 2019]. 16

3.8.2 A toy regression example of fitting 10 samples from a $\sin(\cdot)$ function (plus Gaussian noise) using a function with N basis functions. This example shows that after the classical "overfitting" stage, the excess risk decreases as the model complexity increases. Figure from [Fang et al., 2020]. 16

3.8.3 Plot of two univariate functions fitted to 10 data points using Random ReLU features $\phi(x; (v_1, v_2)) := \max(v_1 x + v_2, 0)$. The data points are shown in red circles. The fitted function with $N = 40$ random ReLU features is the blue dashed line; the coefficient vector's norm (scaled by $\sqrt{N}$) is ≈ 695. The fitted function with $N = 4000$ random ReLU features is the black solid line; the coefficient vector's norm is ≈ 159. Figure from [Belkin et al., 2019]. 17

4.2.1 An example of the graph defined by an random asymmetric distogram over four points, a $16(= 4^2)$ dimensional object. I have attempted to visualise this in 2D space by representing different distances by bending the edges between them, for a longer distance. I did not include self-edges to avoid clutter. $D(\boldsymbol{z})_{ij}$ represents the distance from i to j. I used Mathcha [Nha, 2021] to create this diagram. 19

4.2.2 Alphafold function, this represents the forward pass of data through the model without relaxation. I created this diagram to summarise the full mathematical function AlphaFold represents. I used Mathcha [Nha, 2021] to create this diagram. 21

4.2.3 Alphafold training iteration. High level view of computational graph, modules with parameters are in blue/purple, loss functions in red and inputs in green. I used Mathcha [Nha, 2021] to create this diagram. 22

4.6.1 A prediction of titin, stitching together predictions from a 1400 amino acid crop size every 1200 residues and super-imposing then deleting the overlapping 200 residues [Goddard, 2021]. 25

5.1.1 A flowchart showing the entire data pipeline. Online resources are in purple, local files are in blue and orange, orange boxes make up the raw file of the final dataset. I used Mathcha [Nha, 2021] to create this diagram. 27

5.1.2 The proportion of the examples in the full dataset (y-axis) shorter than length (x-axis). If sequence search gave a hit to an RFAM family then it is coloured green, else red. The bottom plot is as above, zoomed in on the x-axis. 28

5.2.1 A plot of the lengths of the hits and the logarithm of the e-value to the family. The bottom plot is as above, zoomed in on the x-axis. 29

6.3.1 The backbone of a protein is shown on the left, the backbone of RNA is shown on the right. The relevant atoms are labelled using the PDB codes with a preceding superscript indicating the order they are listed in the PDB file for a given structure. I have recreated the PDB figures from [PDB, 1992] in Mathcha [Nha, 2021]. 33

6.3.2 Constructing the frame for a base in the RNA. This is the same as in AlphaFold [Jumper et al., 2021] but with different atoms of course. Note that this gives a transformation (translation and rotation) from the PDB basis to the base-basis. The atoms correspond to basis $\mathcal{B}_1$, the process is the same for $\mathcal{B}_2$. I used Mathcha [Nha, 2021] to create these diagrams. 34

7.1.1 A module is shown in red if there is no weight sharing of its submodules, if there is weight sharing it is shown in purple. I have coloured the distogram head in orange as it is interchanged for other variations of distogram heads. 36

7.2.1 The probability mass functions p_1 (left) p_2 (right) over resolution bins at each epoch. The function generating these bars is given in equation (7.1) (left) and equation (7.2) (right) with $c = \frac{1}{2}$. The plot on the right is the same function but clipped to onefor resolutions above 3.5Å. 38
7.2.2 Weighting function over RNA length. The pmf is proportional to the weights. 38
7.2.3 Full dataset partitioned by length (less than 80) and whether an MSA exists. 38
7.2.4 A histogram with bin size equal to the integers of the number of sequences in the dataset with lengths 1 to 80. 38
7.2.5 In the above figures the top row of plots are the loss components during training (left) and validation (right). The middle row of plots are the metrics for a subset of training examples (the subset is randomly picked before experiments and kept constant). The final row is of validation metrics. The violin plots are normalised by maximum mode (width), not by area or counts. 40
7.2.6 This experiment was initialised with new parameters. The violin plots are normalised by maximum mode (width), not by area or counts. 41
7.3.1 Distogram predictions of the chain 6D9J-4 using just relpos, left-right-outer embedding and a distogram head with the original distogram loss. The leftmost plot is early in training, iteration 100, the following plot is iteration 10000, then the target distogram and the MSE loss. 43
7.3.2 Distogram predictions of the chain 6D9J-4. The model consists of relpos, left-right-outer embedding, triangle attention, triangle multiplication, pair representation transition block and a distogram head with the original distogram loss. The leftmost plot is early in training, iteration 100, the following plot is iteration 2000, next is the target distogram, and the MSE over iterations of gradient decent. 43
7.3.3 Distogram predictions of the chain 6D9J-4. The model consists of relpos, left-right-outer embedding, triangle attention, triangle multiplication, pair representation transition block and a distogram head with the delta-distogram loss. The leftmost plot is early in training, iteration 100, the following plot is iteration 1000, next is the target distogram, and the MSE over iterations of gradient decent. 44
7.3.4 This is the RNA chain 1MMS-C, visualised in green is the true structure, in red is the prediction of model trained after setting the distogram loss to 0. This model is trained only on this example for 3000 gradient steps using a learning rate of 10^{-3}. This experiment was done to test that it could overfit successfully. The structure model loss curve is shown on the bottom, it has been smoothed with a window of 100 iterations and standard deviations calculated for each window. 46
7.3.5 A selection of the Evoformer output distograms from the disto-pretrained model, the predictions are on the top row and the true clipped distograms are on the bottom row. 47
7.3.6 Visualisations of the many of the model predictions from the final-model, the predictions are superimposed onto the true structure using the Kabsch algorithm [Kabsch, 1976] which minimises RMSD over a rigid transformation, the per-base origin to atom vectors are re-inserted from the ground truth. 48
7.3.7 This experiment corresponds to the final-model. The top row of plots are the metrics for a subset of training examples (the subset is randomly picked before experiments and kept constant). The final row is of validation metrics. The violin plots are normalised by maximum mode (width), not by area or counts. 48

LIST OF TABLES

7.2.1 Initial experiments, each is run for 12 epochs apart from version 14 which is run for 32 epochs. 42

Abstract

In this project I briefly review the history and problem of structure prediction for biological molecules. I discuss previous methods used for the protein structure prediction task and RNA structure prediction task. I give the background machine learning knowledge required. I then focus on the state of the art method for protein structure prediction, AlphaFold [Jumper et al., 2021]. I discuss how to apply AlphaFold to the task of RNA structure prediction. I then outline my experiments, and discuss future work. My main contributions in this project are: (1) PyTorch implementation of the core components of AlphaFold. (2) A version of the AlphaFold architecture adapted to predict RNA with training scripts. (3) A novel distogram loss function. (4) A methods section detailing how to overfit the model to RNA data. Additionally, my lab partner and I contributed a clear dataset pipeline to construct an RNA dataset. Unfortunately I could not perform the desired amount of training. Future work would involve reducing the effects of overfitting in the network.

Acknowledgements

I would like to thank my supervisor David Jones for all his help with my project, I would also like to thank my lab partner for help creating the dataset.

1 Introduction

1.1. INTRODUCTION

In this project I explore the field of protein and RNA structure prediction machine learning algorithms. Following the recent release of the state-of-the-art protein structure prediction method AlphaFold [Jumper et al., 2021] I have structured my project to focus on how AlphaFold can be applied to predict the structure of RNA molecules. Unfortunately I could not complete my full plan for this project. Partly due to the timing of AlphaFold's release, not allow much time to act in response to it. Another particularly challenging element was the very limited data, the fraction of the human genome transcribed to RNA is approximately 30 times as large as that coding for proteins, but the number of available RNA structures is less than 1% of that for proteins.

The project is structured with an introduction to experimentally resolving structures for proteins and RNA. This is followed by the different methods currently used for protein and RNA structure prediction and folding using machine learning. I provide a short chapter with the necessary machine learning background. The following chapter outlines the architecture and training procedure for AlphaFold [Jumper et al., 2021]. My lab partner and I create a dataset and pipeline scripts for an RNA dataset. The next chapter describes how I have adapted AlphaFold to work with RNA data. This is followed by the experiments chapter which outlines a training framework and discusses a number of training experiments. The future work section discusses all of the shortcomings, how to improve the model and the possible future applications of machine learning in bioinformatics. I have included all of the source code for this project here [KZVK3, 2021].

1.2. HISTORY OF BIOMOLECULAR IMAGING

Almost all of the machine learning methods that are applied to the structure prediction task rely on data in the form of atom coordinates, which is typically very challenging to obtain. It is important to understand the significant landmarks that have shaped our understanding of these biological molecules. This section is largely summarised from [Brooks-Bartlett and Garman, 2015a].

X-ray crystallography has become an indispensable technique for structural analysis over the last century allowing us to view the world on a scale that was previously not possible. X-ray crystallography provides us with three-dimensional information into the structural mechanisms and functions of biological molecules and materials such as steel. Automation of the crystalisation screening process using robots now allows scientists to test thousands of crystallisation conditions with very little effort. Many synchrotron facilities exist around the world, these generate bright sources of X-ray radiation suitable to obtain improved diffraction from smaller crystals. Crystallography has become an essential tool and its use is integral to certain industries, such as the pharmaceutical, mining and aeronautical industries.

I will briefly outline the main miles stones in the development of biomolecular imaging. Wilhelm Conrad Röntgen was an experimental physicist who investigated the properties of gases and crystals, he discovered X-rays in 1895. Max von Laue was awarded the Nobel Prize in 1914 for discovering the diffraction of X-rays by crystals. A year after Laue's discovery William Henry Bragg built the first X-ray spectrometer combined this with an ion chamber to measure the strength of individual reflections. Bragg's son, William Lawrence Bragg, developed theory around X-ray diffraction patterns, Bragg's law, $n\lambda = 2d\sin(\theta)$. Further theory developed includes the concepts of the reciprocal lattice and the sphere of reflection (Ewald sphere) [Ewald, 1913]. This makes it possible to determine when the diffraction conditions will be satisfied for a given crystal lattice. This allows accurate prediction of the positions of reflections on a detector. Additionally, Charles

G. Darwin's papers [Darwin, 1914b] [Darwin, 1914a] described calculation of the total intensity of a reflection. The intensity then can be measured by integrating it over the reflection [Bragg, 1914], this has been standard practice for crystallography experiments since. Now Fourier analysis is a valuable component of structure solution methods, first suggested by William Henry Bragg [Bragg, 1915]. Peter Josephus Wilhelmus Debye investigated the effect of atomic vibrations in the crystal lattice on the diffraction pattern [Eckert, 2012]. In 1916 Debye and Paul Scherrer developed the technique of powder crystallography [Davies, 1984]. The advantage of this method is that it does not require relatively large single crystals as did the X-ray spectrometer which was in widespread use at the time.

In 1917 James Sumner set out to isolate the first enzyme, urease, hoping it would be a globulin for easy precipitation from dialysis. It took nine years for Sumner to obtain crystals of urease. Arthur Lindo Patterson outlined how to give the interatomic distances within the molecule without any knowledge of the actual phase contributions from each term in the Fourier series [Patterson, 1934]. John Desmond Bernal and Hodgkin took the first X-ray diffraction images of a protein in its mother liquor inside a capillary [Bernal and Crowfoot, 1934]. Previously protein crystals were exposed to air prior to X-ray exposure, which dehydrated the protein crystals and resulted in no observable diffraction. This breakthrough meant that a method of performing X-ray analysis on crystals of proteins was now available.

During the 1930s DNA could not be crystallised so X-ray diffraction images were generated by irradiating fibres of DNA. X-ray diffraction images from DNA were first taken by William Astbury and Florence Bell [Astbury and Bell, 1938]. Rosalind Franklin and Gosling collected X-ray diffraction data from both A and B forms of DNA that would be vital to the solution of its structure. Wilkins presented Franklin's data to James Watson, allowing Watson and Crick to solve the structure of DNA [Watson and Crick, 1953].

Analysis of diffraction patterns from protein crystals was still a complex and unsolved problem. The ionisation spectrometer was not suited to efficiently record large numbers of reflections. The Weissenberg camera [Weissenberg, 1924] became widely used for recording reflections from diffraction patterns. However, it was still an open question as to how to determine the phase of each reflection directly from the X-ray diffraction data. In 1937 Max Perutz started work on X-ray analysis of crystalline proteins. John Kendrew began research with Max Perutz as a PhD student in 1945 and worked on myoglobin, a protein which is roughly a quarter of the size of haemoglobin. In 1953 Perutz managed to successfully use the method of isomorphous replacement to solve the phase problem for the haemoglobin diffraction patterns. Then in 1957 Kendrew solved the first protein structure, myoglobin, by X-ray diffraction analysis, and by 1959 he had achieved atomic resolution of the myoglobin structure. In the same year Perutz solved the structure of haemoglobin.

The number of parameters required to solve protein structures soared [L, 1961] (Bragg 1961), computers became necessary in order to process the data. Additionally, the use of one heavy atom derivative to determine the phases of the reflections in the diffraction pattern (single isomorphous replacement) led to an ambiguity in the possible phases of the reflections. Perutz overcame this problem by creating multiple heavy atom derivatives (multiple isomorphous replacement). Bijvoet suggested that choosing an X-ray wavelength similar to the absorption edge of the heavy atoms would change the intensity of certain reflections (anomalous scattering) [Bijvoet, 1954]. The additional intensity changes could be used to solve the phase ambiguity. A proposed phasing method for finding non-crystallographic symmetry from the reflection intensities [Rossmann and Blow, 1962]. The method was developed into what is now known as [1]molecular replacement.

Dorothy Crowfoot Hodgkin was approached to determine the structure of penicillin. Determination of its structure would provide a means for production of synthetic penicillin and this became a top priority during the war. Hodgkin achieved this in 1945.

Synchrotron radiation sources, where radiation is emitted from highly energetic electrons being deflected in a roughly circular path by a magnetic field, began to be used for protein crystallography experiments in 1974 [Mitchell et al., 1999]. Synchrotrons are crucial to the study of protein structures at atomic resolution because the greater X-ray beam flux increases the intensity of reflections in the diffraction pattern. Additionally, the ability to tune the wavelength of these X-rays has allowed crystallographers to take advantage of anomalous dispersion to solve the phase problem [Lindley, 1995]. Photographic film was replaced by electronic area

[1]Molecular replacement allows the phases of an unknown protein structure to be estimated given a known, partially homologous structure

detectors since they provide better dynamic range, improved information per photon and better time resolution [Schulz and Rosenbaum, 1978].

Alongside the developments of X-ray diffraction techniques, the electron diffraction phenomenon was also being exploited for structural analysis investigations, since the discovery that electrons could be diffracted by crystals[Brooks-Bartlett and Garman, 2015b]. Electron microscopy can be performed on samples unsuitable for X-ray crystallography [Crowther and Klug, 1975]. Since electrons interact more strongly with atoms than X-rays, the crystal samples for electron microscopy are much thinner than those used in X-ray crystallography. However this can lead to problems such as increased radiation damage and loss of three-dimensional order [Henderson and Unwin, 1975]. By applying two-dimensional spatial filtering and three-dimensional image reconstruction, Aaron Klug was able to interpret the data and combine them with X-ray diffraction data to determine the three-dimensional structure of biologically important molecules such as viruses, chromatin and the nucleosome [Klug, 1983].

In 1985 Herbert A. Hauptman and Jerome Karle shared the Nobel Prize in Chemistry for their achievements in the development of direct methods for the determination of crystal structures. The term direct methods is used to refer to the fact that the crystal structure is determined directly from the diffraction data.

All of the protein structures solved before 1975 were of soluble proteins. However, membrane proteins represent around a quarter of the proteomes of most organisms [Carpenter et al., 2008], however they are difficult to crystallize because they are particularly insoluble. The first low resolution membrane protein structure was solved by Richard Henderson and Nigel Unwin in 1975 by electron diffraction, and the first at high resolution, the photosynthetic reaction centre, was solved by Johann Deisenhofer, Hartmut Michel, and Robert Huber in 1985 using X-ray diffraction [Deisenhofer et al., 1985].

Third generation synchrotron and cryo-protection techniques to reduce the effects of radiation damage were introduced in the late 1980s and early 1990s [Garman and Schneider, 1997], Hope 1988, [Teng, 1990] the majority of X-ray crystallography diffraction experiments now take place with the crystals held at around 100K.

In 1997 Paul Boyer and John Walker shared the Nobel Prize in Chemistry 'for their elucidation of the enzymatic mechanism underlying the synthesis of adenosine triphosphate (ATP)', the energy currency of biology. This was followed in 2003 by Rob MacKinnon's work on the structural and mechanistic studies of ion channels. This was followed by Roger Kornberg studies of the molecular basis of eukaryotic transcription, 2006. The determination of the structure of the ribosome was a huge achievement for crystallographic methods, since it consists of around 280,000 non hydrogen atoms compared with the 1260 in myoglobin solved in 1957 (Figure 4) [Garman, 2014]. Robert Lefkowitz and Brian Kobilka studies of G-protein-coupled receptors, an integral membrane protein family that account for about 40 per cent of current pharmaceutical drug targets [Filmore, 2004].

X-ray crystallography is an essential tool in developing our understanding of materials, enabling the discovery of the anti-influenza neuraminidase inhibitors, Relenza and Tamiflu. Next steps for crystallography, may involve membrane proteins as working with them is challenging, they constitute a high percentage of pharmaceutical drug targets. The number of unique membrane protein X-ray structures deposited in the PDB is currently rising exponentially, a new method of crystallizing membrane proteins in lipidic cubic phase (LCP) is contributing to this success. Electron microscopy (EM) has traditionally been used to study the overall shapes of large molecules and complexes, however, cryo-EM techniques both for diffraction experiments and for single particle imaging are now approaching the resolution of X-ray data. Recent developments in serial femtosecond crystallography (SFX) using X-ray free electron lasers (XFELS) have shown promising results in reducing or even totally avoiding radiation damage and also in generating interpretable diffraction from membrane proteins.

Turning specifically to RNA, X-ray crystallography has allowed scientists to better understand the detailed three-dimensional structure of RNA, and the tertiary interactions which are laid upon its secondary structure. The tertiary structure is essential to the biological activity of RNA and due to its single strand formation RNA folding has been revealed to be closer to protein folding than the predictable repetitive folded structure of DNA double helix. Up to 1995, the number of X-ray RNA structures (alone or in complex with protein) in the Protein Data Bank amounted to about 1% of the present content of total RNA and RNP structures [Westhof, 2015]. Developments in X-ray crystallography allowed scientists to further their understanding the structural biology of RNA. In

1996 the crystal structure of the P4-P6 domain of the Tetrahymena group I intron, a structure twice the size of a tRNA with novel types of RNA contacts was shown (labs of Tom Cech and Jennifer Doudna). This development, P4-P6, showed that large RNAs could also be crystallized starting from in vitro synthesis and production [Westhof, 2015]. After this RNA architectures expanded at a much quicker speed. The late 1990s and early 2000s were important for research into ribozymes; however, after the P4-P6 structure, structures of group I introns (eg catalystic RNA) from various species appeared at a slower pace [Westhof, 2015]. Important chemical factors needed to be understood; such as the interplay between folding and catalysis, the distinction between structural and catalytic divalent ions, and the roles of bases through protonation and deprotonation in the chemical steps [Westhof, 2015]. The discovery of riboswitches in 2002 yeided further important information.

In summary Westhof states that RNA structural research has shown: (i) that all hydrogen bonding capabilities of the bases are exploited for forming the base pairs necessary for the intricate architectural folds that RNA can adopt to bind other RNAs, proteins, or encapsulate ligands, (ii) that protonation and deprotonation of bases contribute to the catalysis of nucleolytic ribozymes, (iii) that tautomeric forms of bases lead to Watson–Crick-like base pairs that fool the translational fidelity mechanisms of ribosomes and polymerases [Westhof, 2015].

It is important to note that although mathematical or chemical safeguards exist, errors in stereochemistry, in RNA folds or in solvent or ligand identifications do occur and are, unfortunately, rarely corrected in the main depository databases used by scientists in their research [Westhof, 2015]. ML scientists need to be aware of this issue.

2 | Machine Learning for Protein and RNA Structure Prediction

2.1. FOLDING LABS

2.1.1. DMPFold, Jones Lab

DMPfold [Greener et al., 2019] is developed from DeepMetaPSICOV [Kandathil et al., 2019] (DMP) contact predictor. DMPfold predicts inter-atomic distances, torsion angles and hydrogen bonds to build models as opposed to predicting contacts. It works by iteratively generating models and refining constraints, filtering the unsatisfied ones. DMPfold out-performs CONFOLD2 [Adhikari and Cheng, 2018] and Rosetta [Bender et al., 2016] for the CASP12 free modelling domains.

At a high level the model consists of initially embedding inputs features such as covariance data and contact predictions by a network which directly predicts the distances, hydrogen bonds and torsion angles for a set of models. These inputs are also embedded using another network for residual, iterative updates to the predictions. A set of models are constructed using Crystallography and NMR system [Brunger, 2007] used to determine structures experimentally. After passing through these networks the first prediction is produced, models clustered, this is recycled as inputs to the second network with more of the initial inputs. The process of recycling is repeated for input for three iterations.

2.1.2. Recurrent Geometric Network, AlQuraishi Lab

Recurrent Geometric Network, RGN [AlQuraishi, 2019a], was the first end-to-end differentiable model for learning of protein structure. It uses little coevolutionary data, no templates, has fast inference and managed to achieve state-of-the-art performance on the prediction of novel protein folds without the use of co-evolutionary information. On the task of predicting known protein folds without the use of structural templates the model is competitive with methods that use experimental protein structures as templates, achieving $3 - 7$Å accuracy.

The model relies on a recurrent neural architecture to encode the internal representation of protein sequence, local structure parameterisation in the form of torsion angles, coupling of local protein structure to its global representation via recurrent geometric units and a differentiable loss function. A geometric unit takes in three torsion angles for the current residue, and the partially completed backbone resulting from the geometric unit upstream of it. The output is a new backbone extended by one residue. Feeding the outputs of each unit into the next until the final backbone prediction is made. The units perform translations and rotations to perform their function.

2.1.3. Rosetta, Baker Lab

There are many sub groups within the Baker lab. Traditionally many approaches at the Baker lab have relied on fragment-based methods, and have performed well in CASP experiments. These methods require large amount of computing power. Additionally, they produce a variable fraction of native-like models and do not generalise well [Kim et al., 2005]. Rosetta suite is biomacromolecular modeling software consisting of feature de novo folding, comparative modeling, loop construction, protein docking, small molecule docking, and protein design [Bender et al., 2016].

In CASP14 the Baker group entered a deep learning framework, DeepAccNet, to estimate per-residue accuracy and residue-residue distance signed error in protein models and uses these predictions to guide Rosetta protein structure refinement [Hiranuma et al., 2021]. The network involves 3D convolutions for local atomic environments and 2D convolutions to provide global context.

Before AlphaFold was published, there were some details made available. Not knowing whether DeepMind would publish AlphaFold many groups tried to recreate is from the information available. The Baker lab wrongly assumed that it relied on an SE3 transformer (more details below), however, still managed to achieve reasonable performance. The architecture involved a 3-track network in which information at the 1D sequence level, the 2D distance map level, and the 3D coordinate level is successively transformed and integrated [Baek et al., 2021].

2.1.4. Other Labs

Other notable labs include the Zhang lab with the I-TASSER server [Yang et al., 2015]. The I-TASSER Suite pipeline consists of threading template identification, iterative structure assembly simulation, model selection and refinement, and structure-based function annotation. Another successful group at CASP14 managed to combine AlphaFold1 models with state-of-the-art physics-based refinement via molecular dynamics simulations [Heo and Feig, 2020].

2.2. CO-EVOLUTIONARY METHODS

Random mutations in DNA occur in members of a population over time. Mutations that occur in a region of DNA that codes for a protein will, in most cases, change the amino acid that it codes for (however it is a redundant encoding so some wont). This change may or may not change the structure of the protein. If it does change the structure, then, in the vast majority of cases the protein will cease to function as intended and the mutated genome will not be able to pass on the change to the next generation. However, if the change does not affect the function, then the new variant of the protein can pass on to the next generation. A perfect MSA would contain all possible sequences of amino acids which would fold up into the same structure. Or an uniform sample from them. MSA data can be thought of as a random sample of mutations which do not change the structure and function of the protein.

Techniques which utilise Multiple-Sequence Alignments (MSA) have been shown to be essential for high performance in the structure prediction task. Using multiple sequence alignments as features for protein structure prediction was popularised in the machine learning community after [Jones et al., 2011]. This turned out to be a foundation and core source of information which AlphaFold relied on.

2.3. MOLECULAR DYNAMICS PHYSICS SIMULATION

Protein folding comes under the field of molecular dynamics, closely related to the structure prediction task. Molecular simulations complement structures with energetics and dynamics. Computational experiments can establish cause–effect relationships beyond mere correlation. In silico experiments provide validation of simulation. Free-energy calculations test putative mechanisms and allow comparison to experiments [Bock et al., 2018]. Molecular dynamics simulations is attractive because it has sound theoretical foundations yet challenging due to computational intractability. The structure of the ribosome has been resolved at atomic resolution in many functional and conformational states. Molecular dynamics simulations have added information on dynamics and energetics to the available structural information, bridging the gap from kinetics obtained from single-molecule to bulk experiments [Bock et al., 2018].

Molecular dynamics (MD) simulations monitor time-resolved motions of macromolecules. So far, executable and easily reproducible protocols and metadata to generate and analyse MD studies are only rarely made available [Hildebrand et al., 2019]. Files from MD simulations can be divided into input files, used for production, and output files. Depending on the software used such as GROMACS [Berendsen et al., 1995], NAMD [Phillips et al., 2005], AMBER [Case et al., 2005], DESMOND [Bowers et al., 2006], or ESPResSo [Weik et al., 2019], many different input and output file formats are required and generated which are often incompatible. Additional. MD

simulation can generate huge file sizes making it difficult to distribute and difficult to work with. Some open source databases do exist, such as BIGNASIM [Hospital et al., 2015] for nucleic acids, Cyclo-lib [Mixcoha et al., 2016] for cyclodextrins, GPCRmd [Rodríguez-Espigares et al., 2020] for G protein-coupled receptors, MemProtMD [Newport et al., 2018] for membrane proteins, MoDEL [Meyer et al., 2010] for monomeric soluble proteins and NMRlipids database [Bacle et al., 2021] for lipid bilayers. Most of these databases contain hundreds of trajectories and 10's of nanoseconds worth of simulation time. However, larger projects such as Dynameomics [van der Kamp et al., 2010] have not open sourced their work. This database contains molecular dynamics simulations of the native state and unfolding pathways of over 1000 proteins, representing the majority of folds in globular proteins.

Separate simulations for constituent substructures of a protein allow for a 1-millisecond simulation of the folded protein BPTI, revealing a small number of structurally distinct conformational states which reversible interconversion differs on time scales over 1000 to local relaxations [Shaw et al., 2010].

The slowest-folding proteins folded ab initio by all-atom simulations have folding times ranging from nanoseconds to microseconds. Longer time scale simulations have been achieved for NTL9(1-39) a protein which has a folding time of $\sim 1.5ms$ [Voelz et al., 2010]. This method uses data to seed adaptive resampling simulations to achieve statistically converged descriptions of folding landscapes at longer time scales. Molecular dynamics simulation usually involves investigating the different folding pathways of a protein to understand more about the mechanisms that cause the final fold. Sequence design methods can determine differences in protein folding mechanisms [Brown and Head-Gordon, 2004]. Platforms such as DeepChain [Carranza and Skwark, 2020] offer easy access to a powerful NLP (NLP methods are discussed in the next section) based structure prediction algorithm and molecular dynamics simulation.

To simulate on longer time scales to understand protein dynamics, it is possible to approximate side chain free energy at every integration step [Jumper et al., 2018], this method is called 'Upside'. Having detailed, well-parameterised forcefields restricts the amount of simulation that can be done. However, this can act as a bias and can be traded-off for resolution. Using a contrastive divergence technique it is possiible to train from protein simulations [Jumper et al., 2017].

2.3.1. Energy Function Methods

Single-molecule force spectroscopy has proven extremely beneficial in elucidating folding pathways for membrane proteins. It is possible to simulate many unfolding trajectories and measurements to reproduce key experimental features that may be missed using all-atom methods [Wang et al., 2019b]. Tuning many parameters at once makes finding optimal parameters for force fields used in molecular simulation challenging. It is possible to run a simulation, obtain gradients of a loss function with respect to all the parameters, and use these to improve the force field. retaining the interpretability of force fields [Greener and Jones, 2021].

2.4. NATURAL LANGUAGE PROCESSING MODELS

Language models can distill information in massive protein sequence databases. Discovering evolutionary, structural, and functional organization across protein space. Recent advances in protein language modeling and their applications to downstream protein property prediction problems is discussed in Bepler and Berger [2021].

Many of these language models are influenced by GPT-3 [Brown et al., 2020], a transformer architecture which uses a quadratic complexity attention operation in the sequence length. This has been a bottleneck for scaling the sequence length, this is particular disadvantageous for proteins as it means proteins must be cropped or use approximate versions of attention. More computationally tractable transformers are reviewed here [Tay et al., 2020] such as the Reformer, Linformer, Performer and Longformer. Since then many more efficient architectures have come out, notable ones include Nyströmformer, using clustered attention [Vyas et al., 2020], and methods which do not use attention such as LambdaNets [Bello, 2021] and MLP mixers [Tolstikhin et al., 2021].

One way to help with the bottleneck of quadratic attention is to decrease the effective length of sequences is to use byte-pair encoding schemes [Sennrich et al., 2016] this means more tokens can be created to represent common short subsequences of amino acids (or words)

for example sub-sequences of length 3 would require $21^3 = 9261$ tokens instead of $21 = \{N_{\text{amino acids}} + N_{\text{unknown}}\}$, and would decrease the sequence size by a factor of 3. However, token alphabet size increases exponentially for a linear reduction in sequence length.

An important architecture that has good properties in terms of computational complexity and model capacity is the Perceiver [Jaegle et al., 2021]. This uses cross attention to scale the number of key value pairs, while keeping a fixed number of latent queries.

Unsupervised contact prediction is central to uncovering physical, structural, and functional constraints for protein structure determination and design. It has been shown that Transformer attention maps learn contacts from the unsupervised language modeling objective and can outperform a state-of-the-art unsupervised contact prediction pipeline, suggesting these pipelines can be replaced with a single forward pass of an end-to-end model [Rao et al., 2020].

A similar model [Rives et al., 2021] uses representation learning to produce features that generalize across a range of applications, enabling state-of-the-art supervised prediction of mutational effect and secondary structure. This is trained in an unsupervised manner on 250 million protein sequences spanning evolutionary diversity. The learned representation space encodes structure from the level of biochemical properties of amino acids to remote homology of proteins. Information about secondary and tertiary structure is encoded in the representations and can be identified by linear projections.

Even larger protein language models have been trained since [Elnaggar et al., 2021] on 2.1 billion protein sequences on data from UniRef and BFD. The train two auto-regressive language models (Transformer-XL, XLNet) and two auto-encoder models (Bert, Albert). They show that dimensionality reduction LM-embeddings extracts important biophysical properties governing protein shape. As well as good performance on sub-cellular localisation for 10 cellular compartments and whether a protein is membrane-bound or water-soluble.

Another unsupervised model [Villegas-Morcillo et al., 2020] performed competitively on the supervised task of protein molecular function prediction. The feature representation outperformed hand-crafted features such as one-hot encoding of amino acids, k-mer counts, secondary structure and backbone angles. A two-layer MLP achieved competitive performance in the third Critical Assessment of Functional Annotation benchmark.

2.5. GRAPH NETWORKS AND EQUIVARIANCE

Parameterising neural networks in ways which account for symmetries/invariance in the data with respect to the training objective has been crucial for state of the art results whhen applying deep learning to datasets. Particularly, in computer vision convolutional neural networks (CNNs) [LeCun et al., 1989], using convolutional features/filters which treat different spacial positions in images equally allows for efficient weight sharing and larger models to be used with greater network capacity available for the training task. CNNs have since been adapted to have rotational invariance [Cohen and Welling, 2016]. The QUVA Lab have since progressed the field of invariant and equivariant neural network architectures [Wagstaff et al., 2021], notably focusing on graph neural networks (GNNs) producing the SE(3) transformer [Fuchs et al., 2020]. A good resource for the mathematical background for these papers is the lectures on geometric anatomy for theoretical physics [Schuller, 2013] which covers topics such as set theory, tensor spaces, Lie groups, Lie algebra and Fibre bundles.

Graph neural nets are particularly well suited to molecular prediction. The dominating models on the Quantum Machines 9 (QM9) dataset [Blum and Reymond, 2009] are graph neural networks [Zhang et al., 2020], [Shui and Karypis, 2020], [Klicpera et al., 2020]. Speculation about AlphaFold led many to believe that an SE3 transformer was a component of the architecture, following Fabian's blog post [Fuchs, 2020]. Similarly after Deepmind's Lie-transformer paper [Hutchinson et al., 2021] was released people believed that could have been a component of AlphaFold.

Since then many methods would incorporate SE(3)-equivariant components into deep learning models. For example, SE(3)-Fold [Wu et al., 2021b], which uses SE(3)-equivariant energy-based models for end-to-end protein folding. They characterised conformation space using a SE(3)-equivariant graph neural network, modified to embed the protein-specific domain knowledge. This succeeds earlier work also done by the Tencent AI Lab [Wu et al., 2021a] training a generative network in a denoising manner to optimise the structure of proteins. Specifically by attempting to predict the correction signal from corrupted distance matrices between Cα atoms. Then Langevin dynamics based sampling is adopted to gradually optimise structures from random initialisations.

2.6. RNA GEOMETRY

The primary structure of RNA is the sequence of the bases in the nucleic acid chain. The intramolecular base pairs between complementary base pairs determine the secondary structure of the molecule (e.g. helices, bulges, loops and junctions). The tertiary structure refers to the 3-dimensional arrangement or shape of RNA. In the RNA molecule the pairing of the four bases (A, C, G, U) via hydrogen bonds gives rise to the secondary structure. Each base pair usually pairs with one other base (A-U & G-C) and wobble pair (G-U), often resulting in the formation of a nested structure; multiple stacked base pairs form a helix, one or multiple unpaired base pairs form a loop. Three special base pairs [Bellaousov and Mathews, 2010] occur in secondary structure noncanonical, base triples [Almakarem et al., 2011] and pseudoknots [van Batenburg, 2001]. The backbone of RNA has a greater number of torsion angles than for proteins. RNA can be described in terms of a 2D structure, which describes the Watson–Crick (WC) base pairs. The structural elements of RNA can be reasoned about at a hierarchy of levels and how it contributes to the 3D structure [Miao and Westhof, 2017].

The secondary structure of RNA is a planar structure formed by paired bases, which performs crucial roles in fundamental biological processes and complex diseases. RNA secondary structure profile determines which base is paired with which other base. The RNA secondary structure profile can be obtained from biological experiments, either using a chemical reagent or biological crystallisation. The chemical regent method has high cost but can obtain a large amount of prediction data. This method also contains significant noise which can make it challenging to get results of all bases on RNA due to the limited of sequencing coverage. Biological crystallisation is costly in terms of effort and equipment, but can obtain accurate results.

2.7. RNA SECONDARY STRUCTURE PREDICTION

As the secondary structure of RNA is important in determining the function of noncoding RNAs it is important to research RNA secondary structures. Computational prediction has become the mainstream approach for predicting RNA secondary structure. Now that more RNA sequence data is available and improvements in machine learning techniques, especially deep learning techniques, these methods are increasingly being used for RNA secondary structure prediction due to their increased accuracy and applicability. For example, deep neural networks have been used to identify contacts between RNA bases [Jian et al., 2019]. A number of machine learning algorithms have been applied to the problem of RNA secondary structure prediction, namely RPRes [Wang et al., 2021] which consists of a bidirectional LSTM [Hochreiter and Schmidhuber, 1997] and residual neural network. This method significantly improved the performance than the previous state-of-the-art calculation method CROSS [Delli Ponti et al., 2017]. Machine learning methods for RNA secondary prediction can be classed into three categories: score scheme; pre-processing and post-processing; and prediction process [Zhao et al., 2021]. In all these categories the models were trained in a supervised way.

Firstly looking at the machine learning based score scheme in which score schemes are generated and then replaced. The structure prediction is still formulated as an optimisation problem, where the estimated parameters are used for the evaluation of the scores of possible conformations. Within the score scheme there are three approaches: free energy parameter-refining approach, weighted approach, and probabilistic approach.

- The free energy approach is a popular approach in which the calculation free energy of a secondary structure is the sum of the free energy values of its elements. Researchers focus on the problem of assigning free energy values to the elements of RNA molecules. Methods such as [Xia et al., 1998], [Andronescu et al., 2007], [Andronescu et al., 2010] focus on estimating the parameters for the free energy.

- The weighted approaches consider the parameters of RNA structure elements as weights instead of free energy changes. The weighted approach can utilise ML models to determine thousands of weights for more comprehensive RNA structure elements instead of obtaining a few energy parameters from a large number of wet lab experiments. Machine learning based weights can complement the gaps in the thermodynamic parameter approach [Zakov et al., 2011], [Akiyama et al., 2017], [Sato et al., 2021]. An advantage of the weighted

approach is that it decouples structure prediction from energy estimation, possibly beneficial for both tasks.

- Probabilistic approaches based on machine learning include methoods such as stochastic context-free grammars (SCFGs) [Sakakibara et al., 1994], [Knudsen and Hein, 2003], [Do et al., 2006], [Yonemoto et al., 2015]. SCFGs specify formal rules and joint probability distribution over possible RNA structures for a given sequence; with a probability parameter for each production rule and assigns a probability to each sequence it derives. The probability parameters are learned from datasets of RNA sequences annotated using known secondary structures. The probabilistic approach gives less accurate predictions than the best free-energy based models.

Pre-processing methods based on machine learning include assigning folding rules to RNA [Zhu et al., 2018], they show that there is significant folding diversity among different RNA families. Pre-processing methods can also extract and rank features from the target RNA sequence [Hor et al., 2013], they propose a method to combine feature selection and classifier fusion in an incremental manner. Post-processing methods based on machine learning such as [Koessler et al., 2010] where they model the bonding of two RNA secondary structures to form a larger secondary structure with a graph merge operation.

Predicting process based on machine learning can be done in an end-to-end manner, or a with a hybrid approach. Examples of the end-too-end manner include a Hopfield network [Liu et al., 2006], neural networks [Steeg, 1993], mean-field theory [Apolloni et al.], unrolled algorithms [Chen et al., 2020] or convolution neural networks [Calonaci et al., 2020]. Examples of hybrid approaches include: performing k-nearest neighbour classification from sequence alignments [BINDEWALD, 2006], using an adaptive recurrent neural network wiith an energy-based filter [Lu et al., 2019], using a convolutional neural network and dynamic programming [Zhang et al., 2019], base pair maximisation principle and deep learning [Wang et al., 2019a] and using state inference with deep recurrent neural networks [Willmott et al., 2020].

2.8. RNA TERTIARY STRUCTURE PREDICTION

Unlike proteins, the RNA backbone has numerous degrees of freedom, eight, if one counts the sugar pucker. This makes RNA structure prediction a highly complex problem. The RNA backbone torsion angles can be represented as η and θ, analogous to ψ and ϕ in proteins. For a review of the statistical validation of the η/θ formalism, benefits in RNA motif discovery, structural comparison, RNA structure building and tertiary structure prediction, see [Keating et al., 2011].

RNA molecules can also fold in a similar way to proteins, though with a larger dependence on electrostatic interactions. As with proteins, evolutionary information can provide useful constraints on the overall structure of the chain. Evolutionary couplings derived from sequence co-variation using a global maximum entropy model can predict 3D contacts in RNA, including long-range tertiary contacts and non-Watson-Crick base pairs [Weinreb et al., 2016].

In the past RNA tertiary structure prediction typically relies on base-pairing constraints derived from phylogenetic covariation analysis. However, an approach, inspired by the Rosetta low-resolution protein structure prediction method, that seeks the lowest energy tertiary structure for a given RNA sequence without using evolutionary information differs from this [Das and Baker, 2007]. The model is tested using 20 RNA sequences with known structure and lengths of roughly 30 bases. The method reproduces over 90% of Watson–Crick base pairs, approaching the accuracy of secondary structure prediction methods. The method recapitulates over 30% of non-Watson–Crick base pairs in the native structures. One or more of the top five models agrees with the native structure to within 4Å RMSD over the backbone in over 50% of the cases.

A powerful method trained on just 18 known RNA structures enables identification of accurate structural models without assumptions about their defining characteristics [Townshend et al., 2021]. The resulting scoring function, Atomic Rotationally Equivariant Scorer (ARES), outperforms previous methods in community-wide blind RNA structure prediction challenges. Learning effectively from a small amount of data is difficult procedure when using deep neural networks. The method only atomic coordinates as inputs and incorporates no RNA-specific information, hence the approach is applicable to a much wider range of problems.

2.9. RNA PUZZLES

RNA-Puzzles is a community-wide effort on the assessment of blind prediction of RNA tertiary structures [Magnus et al., 2019]. RNA-Puzzles acts as a benchmark to compare the performance of 3D structure prediction of RNA.

The RNA-Puzzles toolkit includes computational resources such as:

- Decoy sets generated by different RNA 3D structure prediction methods.
- 3D structure normalisation, analysis, manipulation, visualisation tools.
- 3D structure comparison metric tools.

2.9.1. Xiao Group

The Xiao group [Zhao et al., 2012] develop a method called 3dRNA, which can build ncRNA 3D structure based on sequence and 2D structure. Constructing 3D RNA structure from the smallest secondary elements (SSE): helix, hairpin loop, internal loop, bulge loop, and junction loop. In the first step, it uses a tree to represent the RNA secondary structure. Each node in the tree corresponds to an SSE. Then the tree is traversed to match one or more suitable templates for each node from the 3D structure templates library. The tree is traversed again to assemble the matched templates of the node and its parent node resulting in a complete tertiary structure. Each SSE can have multiple templates, this can produce many tertiary structures from using different templates for each SSE. If no appropriate templates exist for a certain SSE, 3dRNA builds a raw template for it using a distance geometry method. Finally, all the structures are clustered and scored, then the best is selected.

2.9.2. Adamiak Group

The Adamiak group [Popenda et al., 2012] develop a method called RNAComposer. It uses a knowledge-based, fragment assembly method using a tree representation of the secondary structure and homology of structural elements. The secondary structure is translated into the corresponding 3D structure using a dictionary based on the. RNA FRABASE database [Popenda et al., 2008].

Initially, the RNA secondary structure is divided into fragments using the tree graph representation. The secondary structure elements are single strands, stems and loops (apical, bulge internal, and n-way junctions). Dictionary search for related 3D structure elements is performed for each secondary structure element resulted from fragmentation. The most suitable 3D structure elements are selected. Then these elements are superimposed and assembled depending on common canonical base pairs. Finally, energy minimisation is performed in torsion angle space [Güntert et al., 1997] and then in atom coordinate space [Schwieters et al., 2003] is performed.

2.9.3. Chen Group

The approach used by the Chen group [Zhao et al., 2017] uses Vfold2D to find the minimum free energy and suboptimal 2D structures. Relying on Vfold-derived motif-based loop free energies. If homologous sequences are available from BLAST or Rfam, this MSA can be input to Vfold2D. Given the sequence and 2D structure output from Vfold2D, Vfold3D predicts the 3D structures by assembling A-form helices and motif templates from known RNA structures. Motif templates can be searched for in the VfoldMTF database using sequence similarity scores. By zipping/unzipping of the terminal base pairs of the helix stems the restrictions on the lengths of the loop branches can be relaxed and this enlarges the search space for the fragments. It may be necessary to perform subtle adjustments of the spatial arrangement of motifs to eliminate inter-motif clash. The last step involves refining the 3D structures by all-atom energy minimisation through AMBER or NAMD, an example is shown here [Xu and Chen, 2015]. Sometimes manual efforts are needed to remove steric clashes.

2.9.4. Bujnicki Group

The Bujnicki group [Boniecki et al., 2015] typically use a hybrid modeling strategy. Initially, the RNA target sequence is used to generate a multiple sequence alignment and template search is performed. If valid templates are returned, these are used to build a model of parts of the target molecule using a comparative, template-based modeling method ModeRNA. If no templates are returned, SimRNA, a coarse-grained method, is used to perform template-free folding simulations. Spacial restraints can be obtained from experimental analyses of contact maps or predictions of the secondary structure, can aid the template-free folding. Finally, the predictions are refined using the AMBER force field extended with the QRNAS method containing energy terms for explicitly modeling hydrogen bonds, idealised base pair planarity and regularising the backbone conformation. An example is demonstrated in [Piatkowski et al., 2016].

3 | Machine Learning Background

Parametric machine learning refers to finding values of the parameters θ of a prediction function $f(x;\theta)$ such that it minimises a loss function $\ell(y, f(x;\theta))$ of a sample of unseen validation data y and its predictions $f(x;\theta)$. So variable size datasets are summarised by a fixed number of parameters, non-parametric methods scale the model complexity as more examples are observed.

3.1. 'LINEAR' LAYERS

Linear operations in machine learning typically refer to the mapping of a vector to a single scalar unit as follows $f_{\text{Lin}}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ where $\boldsymbol{w} \in \mathbb{R}^m, b \in \mathbb{R}$ are the parameters. 'Linear layers' typically refer to performing this operation of a sequence of output scalars, a single output vector,

$$\boldsymbol{f}_{\text{Lin}}(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{b}$$

while $\boldsymbol{x} \to W\boldsymbol{x}$ is a linear transformation of $\mathbb{R}^m$ the translation by the bias vector $\boldsymbol{b}$ makes the overall transformation affine. Note that the gradient with respect to an element of W is $\partial_{w_{ij}} \boldsymbol{f}_{\text{Lin}}(\boldsymbol{x}) = \partial_{w_{ij}} \sum_k W_{lk} x_k = x_j$, and $\partial_{b_i} \boldsymbol{f}_{\text{Lin}}(\boldsymbol{x}) = 1$.

3.2. FEED-FORWARD MULTI-LAYERED PERCEPTRON

A feedforward MLP (multi-layered perceptron, or, neural network) is the parametric functions $f : \mathcal{X} \to \mathcal{Y}$ defined by parameter matrices $\mathcal{W} = [W_1, ..., W_d]$ of dimensions $[(n_0, n_1), (n_1, n_2), ..., (n_{d-1}, n_d)]$, parameter vectors $\mathcal{B} = [\boldsymbol{b}_1, ..., \boldsymbol{b}_d]$ of dimensions $[n_1, n_2, n_3, ..., n_d]$ and a non-linear function $\sigma : \mathbb{R} \to \mathbb{R}$ which acts elementwise. Define the recursive function,

$$\boldsymbol{\psi}(\boldsymbol{\xi}, i) = \sigma\big(\boldsymbol{\psi}(\boldsymbol{\xi}, i-1)^\top W_i + \boldsymbol{b}_i\big),$$

let $\boldsymbol{\psi}(\boldsymbol{\xi}, 0) = \boldsymbol{x}$ and $\hat{\boldsymbol{y}} = \boldsymbol{\psi}(\boldsymbol{\xi}, d)$. Common non-linear functions $\sigma(z)$ used are the rectified linear unit, ReLU$(z) = \max(0, z)$ and the sigmoid function $S(z) = (1 + e^{-z})^{-1}$. Note that an MLP consists of alternating 'linear' layers with non-linear functions. Computing the full function $f_{\text{MLP}}(\boldsymbol{x}) = \sigma\big(\big(...\big(\sigma\big(\boldsymbol{x}^\top W_1 + \boldsymbol{b}_1\big)^\top W_2 + \boldsymbol{b}_2\big)...\big)^\top W_d + \boldsymbol{b}_d\big)$ is referred to as the forward pass. The backward pass refers to computing the gradient of $f_{\text{MLP}}(\boldsymbol{x})$ with respect to the parameters $(\mathcal{W}, \mathcal{B})$. Note, that this has the same time complexity as the forward pass, as each operation can make use of the chain rule. Consider computing the gradient through a non-linearity in the function $\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{g}(\sigma(\boldsymbol{z}))$, so $\partial_{z_i} \boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{g}'(\sigma(\boldsymbol{z}))\partial_{z_i}\sigma(\boldsymbol{z})$, we already computed $\sigma(\boldsymbol{z})$ on the forward pass, and $\boldsymbol{g}'$ denotes the derivative of function $\boldsymbol{g}$ with respect to its argument, since $\boldsymbol{g}$ is typically the linear layer above, we know it's gradient. All functions implemented in autodiff software must have the corresponding gradient function.

Due to the nature of inference in deep models, which can require significant sequential computation (forward and backward passes) it is common to pass a sample of data points through at once. This is because the passing through individual layers can easily make use of parallel computation. This allows for the amortisation of significant amounts of computation.

3.3. LAYER NORMALISATION

After applying the linear and non-linearity in a neural network a latent vector of activations $\boldsymbol{z}$ is obtained. Let the vector $\boldsymbol{z} \in \mathbb{R}^D$, the layer dimension/axis is $d = 1, ..., D$, a common operation is to use layer-normalisation [Ba et al., 2016]. This simply corresponds to computing the following equation

$$f_{\text{LN}}(z_d) = \frac{z_d - \mu}{\sqrt{s^2 + \epsilon}} \cdot \gamma + \beta$$

where $\mu = \frac{1}{D}\sum_i z_i$, $s = \sqrt{\frac{1}{D}\sum_i (z_i - \mu)^2}$, ϵ is a small positive number for numerical stability and (γ, β) are two scalar parameters.

3.4. MULTI-HEAD ATTENTION

Given a sequence of N_q (query) vectors $Q = [\boldsymbol{q}_1, ..., \boldsymbol{q}_{N_q}]$, a sequence of N_v (key-value) vector-'object' pairs $[(\boldsymbol{k}_1, \boldsymbol{v}_1), ..., (\boldsymbol{k}_{N_v}, \boldsymbol{v}_{N_v})]$ with $\boldsymbol{q}_i, \boldsymbol{k}_j \in \mathbb{R}^n$, where 'object' must be an element of a set where any element can be linearly combined and also be in the set. Let the keys be $K = [\boldsymbol{k}_1, ..., \boldsymbol{q}_{N_v}]$ and let the values be also in a vector space $V = [\boldsymbol{v}_1, ..., \boldsymbol{v}_{N_v}]$ with $\boldsymbol{v}_i \in \mathbb{R}^m$, then the attention operation [Bahdanau et al., 2016] is defined as

$$\boldsymbol{f}_{\text{Attn}}(Q, K, V) = \text{softmax}\left(\frac{Q^\top K}{\sqrt{d_k}}\right) V,$$

where $\text{softmax}(\boldsymbol{z}) = \exp(\boldsymbol{z}) / \sum_i \exp(z_i)$.

Multi-head attention [Vaswani et al., 2017] refers to performing

$$\boldsymbol{f}_{\text{MHA}}(Q, K, V) = \text{cat}[\boldsymbol{f}_{\text{Attn}}(QW_1^Q, KW_1^K, VW_1^V), ..., \boldsymbol{f}_{\text{Attn}}(QW_H^Q, KW_H^K, VW_H^V)]W$$

where H is the number of heads, cat is the concatenate operation and all W are parameters.

3.5. COMMON LOSS FUNCTIONS

A typical loss function used for a continuous variables is the mean square error loss

$$\ell(\{\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n\}_{n=1}^N) = \frac{1}{N}\sum_{n=1}^N (\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2$$

where N is the number of examples in the batch, $\hat{\boldsymbol{y}}$ is the prediction and $\boldsymbol{y}$ is the true value.

For multi-class classification problems a common loss function is the cross-entropy loss which corresponds to using a log-softmax operation and then the negative log likelihood loss. The weighted cross-entropy loss is

$$\ell(\hat{\boldsymbol{y}}, c) = \boldsymbol{w}_c\left(-\hat{\boldsymbol{y}}_c + \log\left(\sum_j \exp(\hat{\boldsymbol{y}}_j)\right)\right),$$

where $\boldsymbol{w}_c$ is the weight of class c. Datasets often have an imbalance of examples in each class, it is common to set the weights proportional to a function of the number of examples in each class.

Often surrogate loss[1] functions can be designed to allow for efficient optimisation.

3.6. OPTIMISATION BY GRADIENT DECENT

For a scalar function $\ell(\boldsymbol{\theta})$ of the vector $\boldsymbol{\theta} \in \mathbb{R}^D$ the gradient $\boldsymbol{g}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta})$ lies in $\mathbb{R}^D$ for a given $\boldsymbol{\theta}$. Particularly, $\boldsymbol{g}(\boldsymbol{\theta})$ points in the direction that changing $\boldsymbol{\theta}$ increases $\boldsymbol{g}$ the fastest. Similarly, $-\boldsymbol{g}(\boldsymbol{\theta})$ is the direction to change $\boldsymbol{\theta}$ to get the greatest decrease in ℓ. Performing the update

[1] Surrogate loss functions correspond to a smooth function approximating the discontinuous or badly behaved true function, for example an indicator function, the 0-1 loss. In general it requires initially mapping $\mathcal{Y}$ to a linear surrogate space and then decoding back into the original space $\mathcal{Y}$.

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \delta \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$ repeatedly is referred to as gradient descent (δ is a small step size, which must be chosen to be small enough). However, we consider samples of data from a distribution, therefore we refer to this optimisation algorithm as stochastic gradient descent (SGD).

Many other optimisers also use information from the moments of the gradient. ADAptive Moment estimation (Adam) [Kingma and Ba, 2017] uses running averages of the first and second moments of the gradient. The biased estimate of the first order moment of the gradient is $m \leftarrow m\beta_1 + (1-\beta_1)g$, where g is the gradient and $\beta_1 \in [0,1]$ usually $\beta_1 = 0.9$. The biased estimate of the second order moment of the gradient is $v \leftarrow v\beta_2 + (1-\beta_2)g^2$ and $\beta_2 \in [0,1]$ usually $\beta_2 = 0.999$. The biases are corrected $\hat{m} \leftarrow m/(1-\beta_1)$, $\hat{v} \leftarrow v/(1-\beta_2)$ and the final update is $\theta \leftarrow \theta - \hat{m}\delta/(\sqrt{\hat{v}} + \epsilon)$, where δ is a small scalar, the learning rate.

3.7. STATISTICAL LEARNING THEORY

Machine learning has well-studied foundations in statistical learning theory [Vapnik, 1998]. Learning about the relationship between arbitrary variables $(X,Y) \sim \mathbb{P}(X,Y) = \mathbb{P}(Y|X)\mathbb{P}(X)$ (using Bayes rule), we can denote our predictions as $\hat{y} = A(x)$ where A is the algorithm. The mean square error is a standard loss function and measure of how close continuous predictions are to the true values of a sample of data distributed from $\mathbb{P}(X,Y)$. This is an empirical estimate of $\mathbb{E}_{(x,y)}[(A(x)-y)^2]$, the expected error. Let $f(x) = \mathbb{E}_{Y|x}[\mathbb{P}(Y|x)]$. Then we can decompose the expected error as follows,

$$\begin{aligned}
\mathbb{E}_{(x,y)}[(A(x)-y)^2] &= \mathbb{E}[A(x)^2 - 2A(x)y + y^2], \\
&= \mathbb{E}[(y-f(x))^2] + f(x)^2 - 2f(x)\mathbb{E}[A(x)] + \mathbb{E}_x[(A(x) - \mathbb{E}_z[A(z)])^2] + \mathbb{E}[A(x)]^2, \\
&= \underbrace{\mathbb{E}[(y-f(x))^2]}_{\text{Bayes Error}} + \underbrace{(f(x) - \mathbb{E}[A(x)])^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}[(A(x) - \mathbb{E}[A(x)])^2]}_{\text{Variance}}. \qquad (3.1)
\end{aligned}$$

Where the Bayes error is irreducible noise, the bias is the difference in the algorithm and the truth and the variance captures the variance of the algorithm between training datasets.

The expected risk of a predictor $\hat{f}$ is

$$e(\hat{f}) = \mathbb{E}_{X,Y}[\ell(\hat{f}(X), Y)]$$

If the training set is $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^n \sim \mathbb{P}(X,Y)$, then the empirical risk is

$$e_{\mathcal{T}}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\hat{f}(x_i), y_i).$$

The heart of statistical learning theory is the process of empirical risk minimisation over functions in a hypothesis space $\mathcal{H}$

$$f_{\mathcal{T}} = \inf_{f_{\mathcal{T}} \in \mathcal{H}} (e_{\mathcal{T}}(\hat{f}))$$

which gives $f_{\mathcal{T}}$, an approximation of f. The approximation converges in the limit of the dataset size, hence the 'big data' trend in machine learning.

3.8. OVERFITTING

An estimator $f_{\mathcal{T}}$ has overfit the training data $\mathcal{T}$ if

$$\mathbb{E}_{\mathcal{T}}[e_{\mathcal{T}}(f_{\mathcal{T}})] \leq e_{\mathcal{T}}(f) \qquad \text{and} \qquad \mathbb{E}_{\mathcal{T}}[e(f_{\mathcal{T}})] > e(f).$$

So the estimator will get better performance as measured by the empirical risk than the true function. This is because the predictor is modelling some of the variance of each specific training set. This causes the variance component of the expected error to decrease, equation 3.1. However this causes the bias component to increase.

Effectively balancing the bias-variance trade-off with respect to model complexity can be challenging in deep learning. This is because neural networks can approximate any function given a sufficiently large network. Instead of controlling the complexity of the network architecture

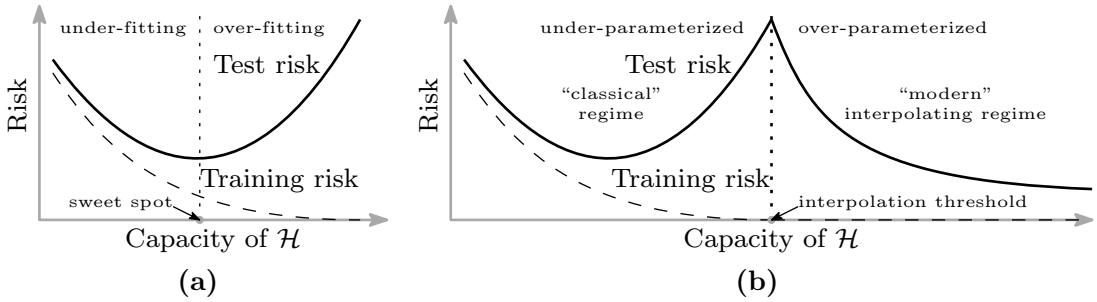


Figure 3.8.1: Figure from [Belkin et al., 2019].

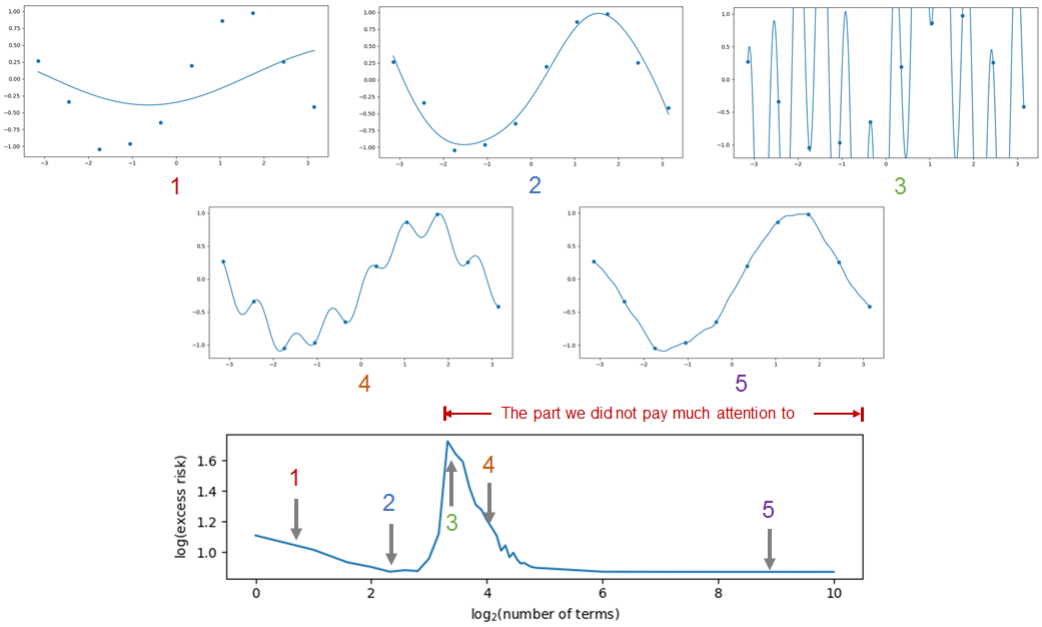


Figure 3.8.2: A toy regression example of fitting 10 samples from a $\sin(\cdot)$ function (plus Gaussian noise) using a function with N basis functions. This example shows that after the classical "overfitting" stage, the excess risk decreases as the model complexity increases. Figure from [Fang et al., 2020].

we typically control the amount of training, saving the parameters of the model which achieve lowest validation error. This is referred to as early stopping. Alternatively it is possible to place a zero prior on the weights. This is known as regularisation, it is often done with half the L2 norm as the gradient is equal to the weights.

Classically, a model that achieves zero training error will be treated as having overfit the training data. However, the learnt function in plot 5 in Figure 3.8.2 would generally not be considered as having overfit the training data. Certainly not to the extent as plot 3 in Figure 3.8.2, even though both achieve zero training error. There have been surprising empirical results that continue to change the view of the 'classical' overfitting theory. Namely 'double descent' [Belkin et al., 2019]. This effect can be seen in the bottom plot of Figure 3.8.2 and the right plot in Figure 3.8.1. There is increasing evidence that neural networks that generalise well will behave more like plot 5 than plot 2 in Figure 3.8.2. We will see later that similar "double-descent" curves (a term introduced in [Belkin et al., 2019]) can be widely observed in other models as well. These results are the main justification for using large flexible models. Considering bigger hypothesis spaces with more candidate predictors explaining the data, we are able to find interpolating functions that have smaller parameter norms and are thus 'simpler' (simpler roughly used to mean smoothness, simpler functions are traditionally preferred as they are usually more interpretable and robust to deviations in the input). See Figure 3.8.3.

A theory as to why the double descent behaviour occurs is that it is due to the non-convexity

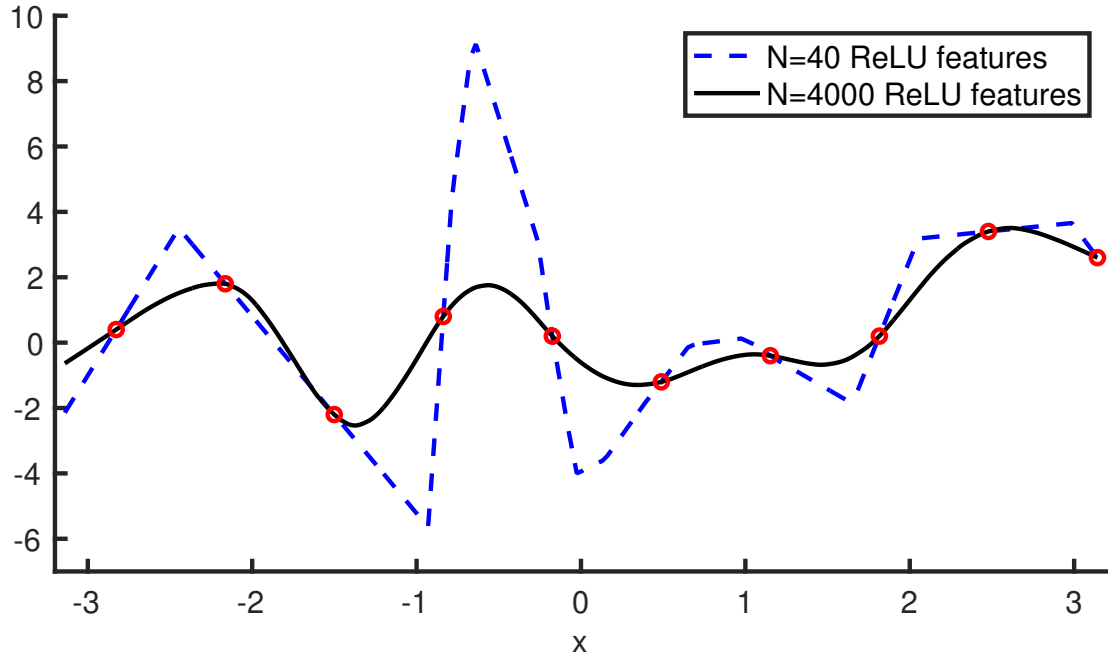


Figure 3.8.3: Plot of two univariate functions fitted to 10 data points using Random ReLU features $\phi(x;(v_1,v_2)) := \max(v_1x + v_2, 0)$. The data points are shown in red circles. The fitted function with $N = 40$ random ReLU features is the blue dashed line; the coefficient vector's norm (scaled by $\sqrt{N}$) is ≈ 695. The fitted function with $N = 4000$ random ReLU features is the black solid line; the coefficient vector's norm is ≈ 159. Figure from [Belkin et al., 2019].

of neural network optimisation. An under-parametrised network is highly sensitive to initialisation. Therefore, the interpolation peak can only be observed within a small subspace of parameter space, which sampling of the parameter space can easily miss. Good generalisation in an over-parameterised model is not explained by traditional complexity measures such as VC dimension or Rademacher complexity. These complexity measures only consider the worst predictor in the in the function space (in terms of minimising the empirical risk). This bound is uninformative for over-parameterised models, since there will be many predictors that generalise poorly.

Neural networks, trained using gradient descent on the empirical risk, approximate kernel machines [Domingos, 2020]. Kernel methods operate in an infinite dimensional reproducing kernel Hilbert space. It is infinite dimensional because it works in the dual domain of the dataset, hence can utilise arbitrary linear combinations of a kernel function over the data points. The generalisation properties of kernel methods are therefore informative with regard to neural networks. The generalisation properties of kernel methods are investigated in [Wei et al., 2018]. They show that instead of explicit regularisation, early-stopping, an implicit regularisation technique, can be used to obtain a bound on the worst case generalisation error. The algorithm used is an iterative kernel boosting algorithm analogous to iterations of gradient descent. The optimal stopping criteria they define is a function of a statistical error term (local Gaussian complexity of the kernel class) [Wei et al., 2018].

Lastly, an interesting result from the Facebook AI lab showed that random search over neural architectures gets near state-of-the-art results on PTB (RNNs) and CIFAR-10 (ConvNets) [Li and Talwalkar, 2019]. This suggests that the machine learning community show maybe spend less time on coming up with neural network architectures that tailor to the individual problem at hand.

4 | AlphaFold

AlphaFold [Jumper et al., 2021] managed to achieve state-of-the-art accuracy at the protein structure prediction task, with experts declaring it to be solved. In CASP14[1] AlphaFold managed to achieve a median backbone accuracy of 0.96Å RMSD (root mean squared deviation), the next best team achieved a median backbone accuracy of 2.8Å RMSD. AlphaFold is available to run for free in Google Colab notebooks [Mirdita et al., 2021] even with molecular dynamics simulation [Arantes et al., 2021].

There are a number of exciting novel architectural innovations to neural network modules introduced in AlphaFold. I will provide a high level description of them below, paired with Figure 4.2.2 which I created in order to fully describe the function computed by AlphaFold during inference.

$$(\overrightarrow{\boldsymbol{x}}_i^{C^\beta}, \boldsymbol{r}_i^{\text{pLDDT}}) \leftarrow \text{AlphaFold}(\boldsymbol{f}_i^{\text{amino}}, N_{\text{res}}, \boldsymbol{f}_{s,i}^{\text{msa}}, \boldsymbol{f}_{s,i}^{\text{extra_msa}}, \boldsymbol{f}_{s,i}^{\text{angle}}, \boldsymbol{f}_{s,i}^{\text{templ_pair_feat}}),$$

where $\boldsymbol{f}_{s,i}^{\text{msa}}, \boldsymbol{f}_{s,i}^{\text{extra_msa}}, \boldsymbol{f}_{s,i}^{\text{angle}}, \boldsymbol{f}_{s,i}^{\text{templ_pair_feat}}$ are optional. Indices i, j are used to index the residue dimension, indices s, t index the sequences in MSA. $\boldsymbol{f}_{s,i}^{\text{msa}}$. The predicted atom positions are $\overrightarrow{\boldsymbol{x}}_i^{C^\beta}$ and the confidencec in the atom position is $\boldsymbol{r}_i^{\text{pLDDT}}$.

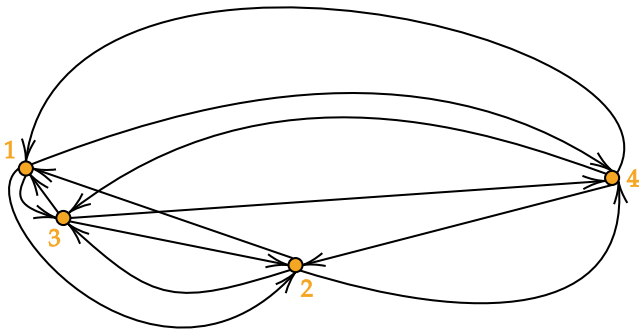
4.1. DATA PIPELINE

Given an arbitrary sequence of amino acids represented as a string, it is searched against a database of proteins. Specifically, JackHMMer [Potter et al., 2018] on the UniRef90 database [Suzek et al., 2014], HHblits [Remmert et al., 2012] on Uniclust [Mirdita et al., 2016], JackHMMer on the MGnify database [Mitchell et al., 2019], HHSearch [Söding, 2005] on the PDB70 database. This collects all of the raw data that is used as inputs. This data is then processed into the input tensors to the model. The sequence is represented using a one-hot encoding. The deletion matrix for the raw MSA data is computed.

During training the input data goes through data transformations, ensembled and non-ensembled functions. The ensembled functions include masking the MSA data, creating profiles for the MSA. Profiles use a feature F constructed from the inverse tan function on the deletion matrix M, $F = \arctan(\frac{1}{3}M)\frac{\pi}{2}$. Then the MSA is clustered into the fixed size MSA buffer using block deletion. Block deletion removes contiguous sections from the ordered MSA. Any left over MSA is assigned to its nearest neighbour in the sampled MSA and fills up the Extra-MSA buffer. Then the MSA is cropped, features are calculated for the MSA, and the templates are cropped. The non-ensembled functions include, BERT-style augmentation of the MSA data and a HHBlits profile. BERT-style augmentation involves corrupting the MSA, replacing a character with another picked uniformly at random with 0.1 probability and masking with 0.1 probability.

Computing y-features is done by loading a PDB file. The coordinates of all atoms, a mask for all atoms, b-factor, the amino acid type and the residue index can all be extracted. These are all straight-forward to process for training since no data transforms take place, other than the same sequence crop as performed for the corresponding x features.

[1] Critical Assessment of Techniques for Protein Structure Prediction



$$D(z) = \begin{bmatrix} & 10 & 2 & 14 \\ 6 & & 6 & 9 \\ 1 & 5 & & 12 \\ 17 & 7 & 13 & \end{bmatrix}$$

Figure 4.2.1: An example of the graph defined by an random asymmetric distogram over four points, a $16(= 4^2)$ dimensional object. I have attempted to visualise this in 2D space by representing different distances by bending the edges between them, for a longer distance. I did not include self-edges to avoid clutter. $D(\boldsymbol{z})_{ij}$ represents the distance from i to j. I used Mathcha [Nha, 2021] to create this diagram.

4.2. NEURAL NETWORK MODULES

Embedding Modules

The following modularisation is not exactly the same as the paper or the original code [Jumper et al., 2021].

- **Input_Embedder**$(\boldsymbol{f}_i^{\textbf{amino}}, N_{\textbf{res}}, \boldsymbol{f}_{s,i}^{\textbf{msa}}) \rightarrow \boldsymbol{m}_{s_c,i}, \boldsymbol{z}_{ij}$

  This module computes latent variables $\boldsymbol{m}_{s_c,i}, \boldsymbol{z}_{ij}$ referred to as the MSA and pair representation. These go on to be heavily refined in a residual manner by the Evoformer described below. The pair representation $\boldsymbol{z}_{ij}$ is constructed with an outer product of latents from projecting $\boldsymbol{f}_i^{\text{amino}}$ and added to the positional encoding $\boldsymbol{p}_{ij}$. The positional encoding $\boldsymbol{p}_{ij}$ is constructed using 65 parameter vectors which correspond to a difference of 32 indices backward and 32 indices forward, which fills in a pairwise representation $[N_{\text{res}}, N_{\text{res}}, c_z]$, a bias vector is also added to this. The MSA representation $\boldsymbol{m}_{s_c,i}$ is constructed by summing projections of $\boldsymbol{f}_i^{\text{amino}}$ and $\boldsymbol{f}_{s,i}^{\text{msa}}$.

- **Extra_MSA_Embedder**$(\boldsymbol{f}_{s,i}^{\textbf{extra_msa}}, \boldsymbol{f}_{s,i}^{\textbf{angle}})$**:**

  This module uses the same input embeddings as above with the extra MSA data. With N_{templ} rows at the bottom to embed the angles $\boldsymbol{f}_{s,i}^{\text{angle}}$. It is processed by a 4 layer Evoformer, and the pair representation is extracted. This is then processed by the main Evoformer.

- **Template_Embedder**$(\boldsymbol{f}_{s,i}^{\textbf{templ_pair_feat}})$**:**

  This module computes separate pairwise representation for each template sequence, by passing $\boldsymbol{f}_{s,i}^{\text{templ_pair_feat}})$ through a linear layer. These are then processed though blocks comprised of triangle attention and triangle multiplication interchanged with pair transition blocks. The pairwise representation from the input embedding then acts as queries to attend to the embedded templates.

- **Recycling_Embedder**$(\boldsymbol{m}_{1i}, \boldsymbol{z}_{ij}, \boldsymbol{x}_i^{C^\beta})$**:**

  This module the distogram from the the previous backbone carbon atoms, places them into bins and embeds them using a linear layer, $\boldsymbol{d}_{ij}$. The previous $\boldsymbol{z}_{ij}$ passes through a layer-norm and is added to $\boldsymbol{d}_{ij}$ to create the residual pair-representation. The previous first row of the MSA representation $\boldsymbol{m}_{1i}$ is passed through a layer-norm and makes the residual for the next MSA representation.

Evoformer

The Evoformer is composed of the following block stacked 48 times, without weight sharing. **Evoformer_Iteration**$(\boldsymbol{m}_{s_c,i}, \boldsymbol{z}_{ij}) \rightarrow \boldsymbol{m}_{s_c,i}, \boldsymbol{z}_{ij}, \boldsymbol{s}_i$. Each Evoformer iteration performs the following operations. Each operation is represented in Figure 4.2.2.

- **MSA_Row_Attention_With_Pair_Bias**$(\boldsymbol{m}_{s_c,i}, \boldsymbol{z}_{ij}) \rightarrow \delta \boldsymbol{m}_{s_c,i}$

  This module combines the pair representation into the processing of the MSA, by acting as a bias on the attention weights. Attention is performed over the residue dimension of the MSA (to fit with the pairwise representation).

- **MSA_Column_Attention**$(\boldsymbol{m}_{s_c,i}) \rightarrow \delta \boldsymbol{m}_{s_c,i}$

  This module can integrate information from the different sequences in the MSA. By attending across the MSA it can provide context to other possible mutations at each position in the sequence.

- **MSA_Transition**$(\boldsymbol{m}_{s_c,i}) \rightarrow \delta \boldsymbol{m}_{s_c,i}$

  This is a layer normalisation operation, followed by a simple two layer MLP which has a intermediate hidden size that is four times the input/output hidden size, with ReLU non-linearity.

- **Outer_Product_Mean**$(\boldsymbol{m}_{s_c,i}) \rightarrow \delta \boldsymbol{z}_{ij}$

  This block summarises the MSA representation into an update for the pair representation. First $\boldsymbol{m}$ is transformed by two linear layers, for two embeddings. An outer-product is performed over the sequence dimension, then an average is taken over the MSA dimension.

- **Triangle_Multiplication**$(\boldsymbol{z}_{ij}) \rightarrow \delta \boldsymbol{z}_{ij}$

  Any three points in 3D space will satisfy the triangle inequality. In a distogram the elements do not necessarily satisfy the triangle inequality. Consider points indexed by $\{i, j, k\}$, they form line segments/edges $\{ij, jk, ik\}$. Holding two edges fixed, it is trivial to find the solution of the final edge which satisfies the triangle inequality. Instead, it makes more sense to perform subtle updates, iterative improving the global solution, as one update will usually violate other sets of three points. Figure 4.2.1 shows how a pairwise representation can make little sense describing points in the lower dimensional system. Note that the distances are not scalar in $\boldsymbol{z}_{ij}$, so the analogy is relatively loose. The idea behind these triangle operations is to reduce any bending of the arrows in Figure 4.2.1. Note that this is done over outgoing arrows first, then over incoming arrows for all nodes.

- **Triangle_Attention**$(\boldsymbol{z}_{ij}) \rightarrow \delta \boldsymbol{z}_{ij}$

  Similarly, this module has two variants, over the rows and columns of the pairwise representation and the incoming and outgoing arrows. Namely, starting node and ending node triangular attention. The attention operation includes addition of a scalar bias term for each edge ij, of which one of the edge axes will have a softmax operation over it. Before returning the update, it is gated by multiplication with a linear layer and sigmoid operation on $\boldsymbol{z}$, so each edge ij controls whether it will be updated.

- **Pair_Transition**$(\boldsymbol{z}_{ij}) \rightarrow \delta \boldsymbol{z}_{ij}$

  This module is identical to the MSA transition.

Finally, the single representation $\boldsymbol{s}_i$ is produced by projecting $\boldsymbol{m}_{1,i}$.

Structure Module

The structure module also resembles the architecture of a transformer. A layer of the module first performs a residual update on the single representation $\boldsymbol{s}_i$ with Invariant-Point-Attention (IPA). Then it has a transition block operating on $\boldsymbol{s}_i$. It then computes transformations to update the backbone frames T_i using backbone update, $T_i \leftarrow T_i \circ \delta T_i$. An MLP computes the torsion angles α_i from $\boldsymbol{s}_i$. The auxiliary loss is computed by summing the frame aligned point error and the torsion angle loss. The first iteration of the structure module uses 'black-hole initialisation' on the frames T_i, and the frames T_i are free to move, they are referred to as a residue gas.

- **Invariant_Point_Attention**$(\boldsymbol{s}_i, \boldsymbol{z}_{ij}, T_i) \rightarrow \delta \boldsymbol{s}_i$

  This module operates on the backbone frames, single representation and pair representation, to produce an update for the single representation. The action of the

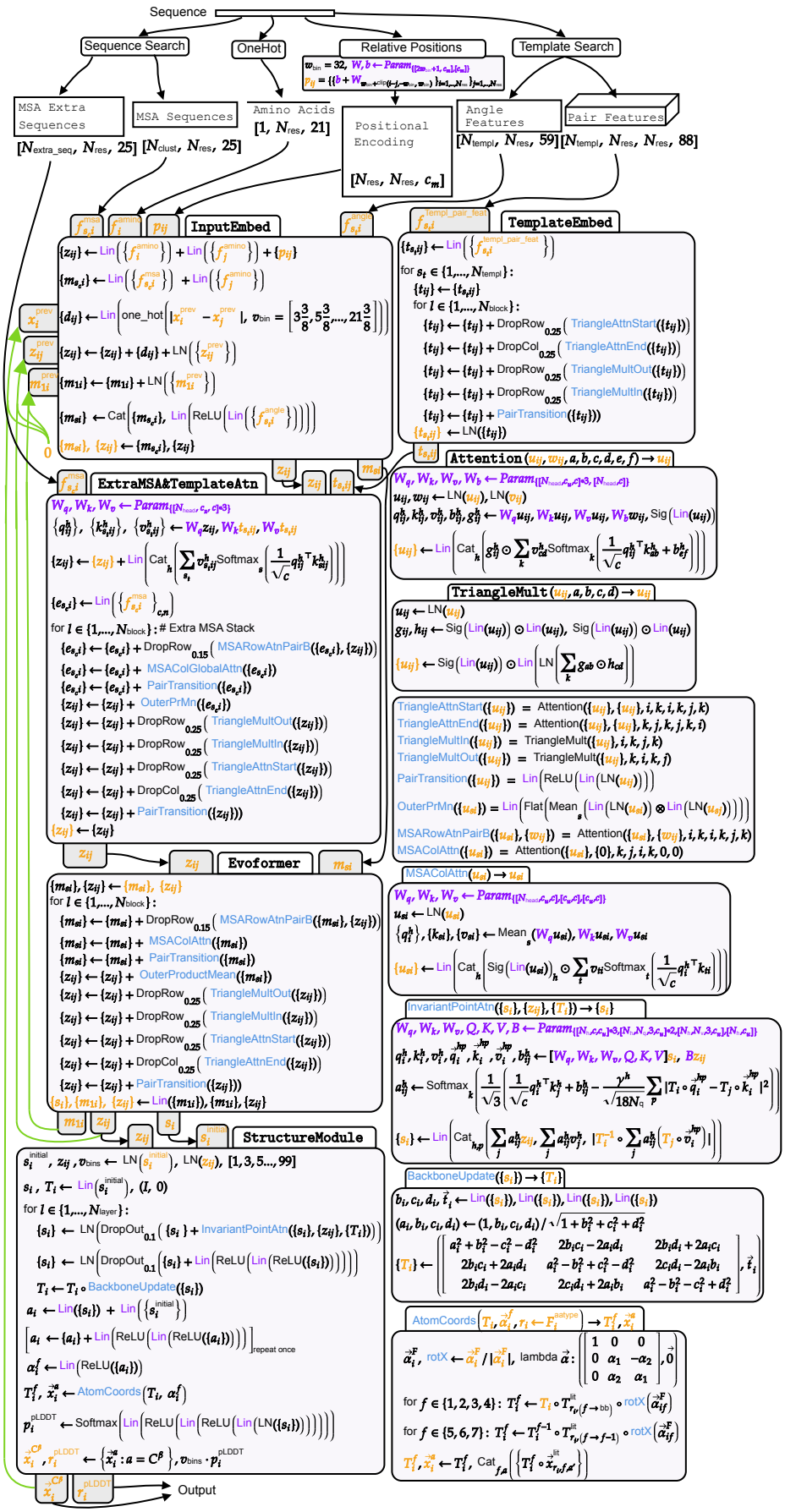


Figure 4.2.2: Alphafold function, this represents the forward pass of data through the model without relaxation. I created this diagram to summarise the full mathematical function AlphaFold represents. I used Mathcha [Nha, 2021] to create this diagram.

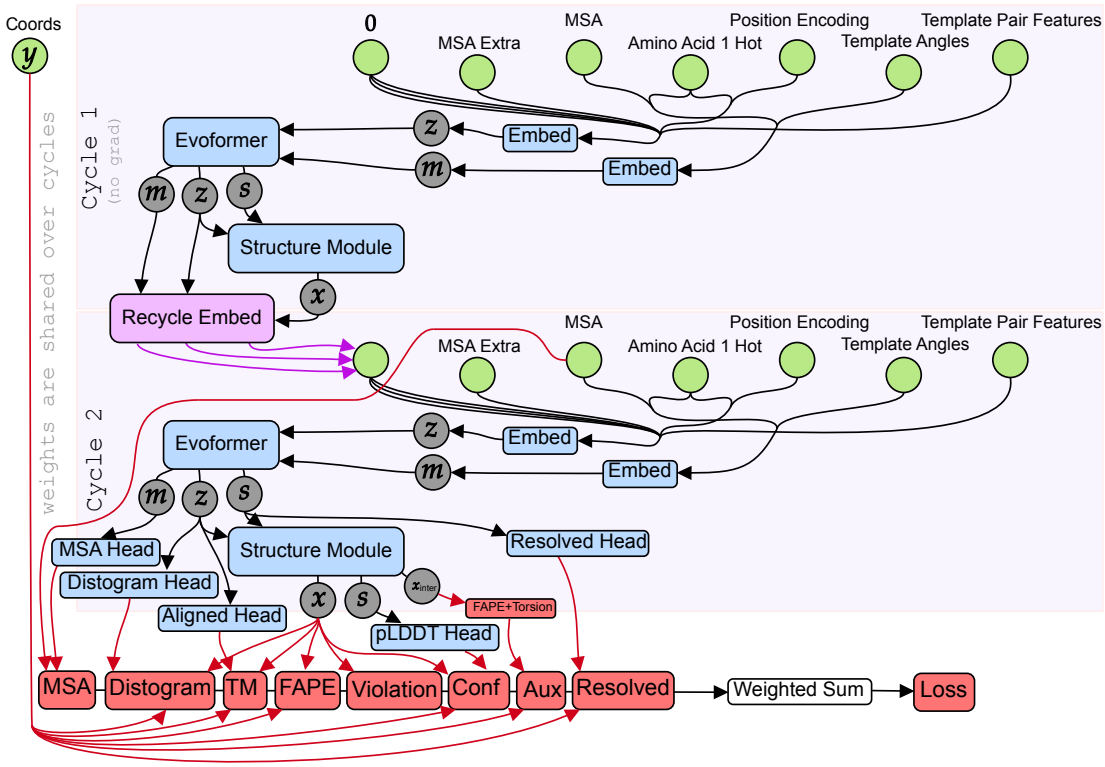


Figure 4.2.3: Alphafold training iteration. High level view of computational graph, modules with parameters are in blue/purple, loss functions in red and inputs in green. I used Mathcha [Nha, 2021] to create this diagram.

module is invariant to global transformations on the set of frames. The coordinates are represented in nanometers, the units affect the scaling of attention affinities. There are two sets of queries, keys and values. Both sets of queries and keys are used to compute the attention weights. Both sets are produced by projecting the single representation. However, one set also is acted upon by all backbone frame transformations. The similarity of this set is a negative sum of the squared vector differences. are all functions of just the single representation. The third term contributing to attention weights is bias, projected from the pairwise representation z_{ij}. The three sets of values are attended to, the third being z_{ij}. The outputs are concatenated and the passed through linear layer. This module can reason over sets of points in 3D.

- **Backbone_Update**$(s_i$

  This module computes translation vectors t_i and quaternions q_i for the backbone frame updates. Using a linear layer on the single representation, t_i and $q_{i,2:4}$ can be computed. Only three values need to be computed for each quaternion as the fourth is fixed by normalisation. Quaternions are a sensible parameterisation as they have just four parameters instead of nine for a rotation matrix. Additionally there are no unreasonable edge cases, for example Euler matrices have Gimbal lock discontinuities.

4.3. LOSS FUNCTIONS

AlphaFold is trained using many loss functions working in harmony,

$$\mathcal{L} = \begin{cases} 0.5\mathcal{L}_{\text{FAPE}} + 0.5\mathcal{L}_{\text{aux}} + 0.3\mathcal{L}_{\text{dist}} + 2.0\mathcal{L}_{\text{msa}} + 0.01\mathcal{L}_{\text{conf}} & \text{training} \\ 0.5\mathcal{L}_{\text{FAPE}} + 0.5\mathcal{L}_{\text{aux}} + 0.3\mathcal{L}_{\text{dist}} + 2.0\mathcal{L}_{\text{msa}} + 0.01\mathcal{L}_{\text{conf}} + 0.01\mathcal{L}_{\text{exp resolved}} + 1.0\mathcal{L}_{\text{viol}} & \text{fine-tuning} \end{cases}$$

the losses $\mathcal{L}_{\text{exp resolved}}$ and $\mathcal{L}_{\text{viol}}$ were not introduced at the start of training as they caused instability.

4.3.1. Frame Aligned Point Error, $\mathcal{L}_{\text{FAPE}}$ and $\mathcal{L}_{\text{aux}}$

The function requires the local frames for each residue $T_i^{\text{pred}} = (R_i^{\text{pred}}, \boldsymbol{t}_i^{\text{pred}})$ in the predicted chain, and the true frames $T_i^{\text{true}} = (R_i^{\text{true}}, \boldsymbol{t}_i^{\text{true}})$ and all of the atom coordinates $\boldsymbol{x}_a^{\text{true}}$ and their predictions $\boldsymbol{x}_a^{\text{pred}}$. The loss is

$$\mathcal{L}_{\text{FAPE}} = \frac{1}{Z \cdot N_{\text{res}} \sum_i N_{\text{atom}}^{(i)}} \sum_{i=1}^{N_{\text{res}}} \sum_{a=1}^{N_{\text{atom}}^{(i)}} \min\left(d_{\text{clamp}}, \sqrt{\epsilon + \left\| {T^{\text{true}}_i}^{-1} \circ \boldsymbol{x}_a^{\text{true}} - {T_i^{\text{pred}}}^{-1} \circ \boldsymbol{x}_a^{\text{pred}} \right\|}\right)$$

where $Z = 10$Å, $d_{\text{clamp}} = 10$Å, $\epsilon = 10^{-4}$Å^2. The operator $\circ$ does the following, $(R, \boldsymbol{t}) \circ \boldsymbol{x} = R\boldsymbol{x} + \boldsymbol{t}$. So the action of each ${T_i}^{-1}$ takes coordinates in local frame i and places them in the global frame. This loss performs efficient local and global penalisation. During intermediate folding iterations only the backbone $C\alpha$ atoms are used to reduce computation, this contributes to the auxiliary loss.

4.3.2. Torsion Angle Loss, $\mathcal{L}_{\text{aux}}$

This loss requires the ground truth backbone torsion angles and the predicted torsion angles represented as vectors in $\mathbb{R}^2$. The alternative ground truth angles due to symmetries of certain configurations can be calculated. Parameterising the angles as vectors avoids discontinuities with angles wrapping around. To keep them roughly close to the unit circle, an extra loss term calculates the difference of the norm of the vector with 1.

4.3.3. Distogram Loss, $\mathcal{L}_{\text{dist}}$

Given the Evoformer pairwise output $\boldsymbol{z}_{ij}$ a probability distribution over 64 distance bins is predicted. Specifically, the range 2Å to 22Å is split into 64 equal sized bins. Then $\boldsymbol{z}_{ij}$ passes through a linear layer, $\hat{\boldsymbol{z}}_{ij}$ and added to its transpose of the residue dimensions $\boldsymbol{g}_{ij} \leftarrow \hat{\boldsymbol{z}}_{ij} + \hat{\boldsymbol{z}}_{ji}$, to ensure the symmetry requirement of a distogram. The tensor $\boldsymbol{g}_{ij} \in \mathbb{R}^{64}$ acts as the logits, log probabilities, over the distance bins. The loss is the cross-entropy loss of $\boldsymbol{g}_{ij}$ with the true bins.

4.3.4. BERT Loss, $\mathcal{L}_{\text{msa}}$

The MSA representation is corrupted by masking and randomly changing different elements in the MSA. This is the objective used in the BERT paper [Devlin et al., 2019], a method from unsupervised language pre-training used in a supervised setting. This supervises the Evoformer stack to learn about evolutionary relationships in the MSA. This uses a cross-entropy loss.

4.3.5. Per-Residue Predicted Local Different Distance Test Loss, $\mathcal{L}_{\text{conf}}$

The Local Different Distance Test (lDDT) Mariani et al. [2013] is a metric used to compare the local differences of two structures, it is superposition-free, and can evaluate the difference in all atoms of the two structures. It can also validate stereochemical plausibility. In AlphaFold the per-residue lDDT-Cα scores are predicted, pLDDT. This is done by projecting the single representation from the structure module and projecting it into 50 bins, then computing the cross-entropy loss with a target value.

Using this as an objective allows the neural network to predict its own errors from the predicted structure, this gives an estimate of how confident the prediction is. This allows for the student self-distillation procedure which is based on [Xie et al., 2020]. This process involves expanding the size of the dataset by adding in its own predictions that it is sufficiently confident on.

4.3.6. Experimentally Resolved Loss, $\mathcal{L}_{\text{exp resolved}}$

This loss requires the all atom mask and the single representation from the structure module. The single representation is projected into logits for all atoms, then the cross-entropy loss is computed with the mask.

4.3.7. Violation Loss, $\mathcal{L}_{\text{viol}}$

This requires pre-calculated mean and standard deviations of the bond lengths and angles for all pairwise combinations of consecutive residues. There is a loss term to penalise the bonds that are too long, above a threshold. Another term computes the difference in cosine of the 'true' and predicted angles. A final term penalises bonds that are too short, below a threshold.

4.4. TRAINING PROCEDURE

The network is used in a recurrent manner to make predictions, by embedding the previous outputs as inputs. This is termed recycling and is shown to be particularly useful in the ablation studies in [Jumper et al., 2021]. On a given training example, the number of recycling iterations is uniformly sampled over the set $\{0, 1, 2, 3\}$. During training only the gradient for the final iteration of recycling is calculated see Figure 4.2.3. This allows for huge savings in time and memory, as the backward pass only has to travel back for one iteration, so only the activations and gradients for the final iteration need to be stored. Additionally, gradient check-pointing is done over the 48 evoformer blocks. So, the gradients and activations within each Evoformer block are recomputed, before being updated and freed from memory. This allows for larger crop sizes in the sequence dimension, as this is the main bottleneck, with cubic complexity from the outer-product mean operation.

Training takes place in two parts, initial training on 10^7 samples, and fine-tuning on $15 \cdot 10^5$ samples. The learning rate during initial training is 10^{-3} using the Adam optimiser with a linear warm up over $128,000$ samples, then decreasing by a factor of 0.95 every $6.4 \cdot 10^6$ samples. Fine tuning has a learning rate of $5 \cdot 10^{-4}$ and no scheduler. Training takes place with a batch size of 128 and one example per TPU core. The gradient norm is clipped to 0.1. Each recycling iteration uses new MSA data. The output representations from the Evoformer are ensembled three times using different augmentations new MSA for the inputs and averaged before passing into the model heads. Linear layers are initialised using LeCun initialisation [LeCun et al., 1998], biases with 0, linear layers followed by ReLU are initialised with He initialisation [He et al., 2015], attention projections use Glorot uniform initialisation [Glorot and Bengio, 2010]. Layer normalisation layers have gain set to 1 and biases set to 0, gating linear layers have 0 weight and 1 bias, to make them mostly open. The IPA weights after the softplus are initialised to 1.

During inference, after the all atom predictions have been generated, physics simulation takes place using the Amber force-field [D et al., 2021]. This usually makes subtle changes to the structure and is referred to as relaxation.

4.5. ABLATION STUDIES

In the final section of the supplementary information for the AlphaFold paper they show that many components of the final architecture are not crucial in order to achieve high performance. Templates were shown not to improve the model significantly. Student self-distillation was shown to significantly boost performance. Similarly recycling was shown to be very important. Removing the MSA did not cause a significant drop in performance, a surprising result as it has been crucial for previous state-of-the-art results, however, there were still MSA summary statistics provided. Another interesting result showed that the invariant point attention was not crucial. The authors note that when removing components one at a time there may be only a small dip in performance, but removing multiple components simultaneously can cause a significant decrease in performance.

4.6. APPLICATIONS OF ALPHAFOLD

Soon after the CASP14 many organisations began to adopt AlphaFold into their research. AlphaFold's predictions have helped accelerate the research of many scientists which rely on experimental protein structure determination. Hassabis [Rubiera, 2020] in a DeepMind blog post outlines the following examples of organisations that have utilised AlphaFold. The Drugs for Neglected Diseases Initiative (DNDi) now use AlphaFold to help advance their research into life-saving cures for diseases disproportionately affecting poorer parts of the world. The

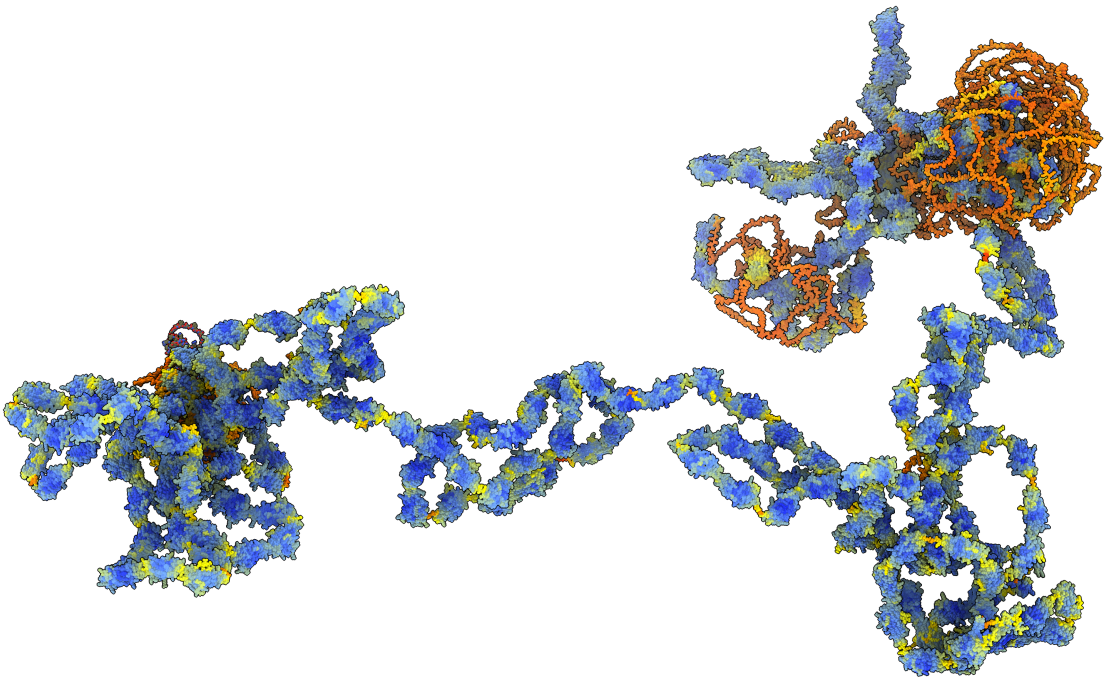


Figure 4.6.1: A prediction of titin, stitching together predictions from a 1400 amino acid crop size every 1200 residues and super-imposing then deleting the overlapping 200 residues [Goddard, 2021].

Centre for Enzyme Innovation at the University of Portsmouth (CEI) is using AlphaFold to help engineer faster enzymes for recycling some of our most polluting single-use plastics. A team at the University of Colorado Boulder is finding promise in using AlphaFold predictions to study antibiotic resistance, while a group at the University of California San Francisco has used them to increase their understanding of SARS-CoV-2 biology.

A success story from AlphaFold is from the group of Henning Tidow had been trying to solve the crystal structure the reductase FoxB for roughly two years. Upon seeing DeepMind's prediction they quickly managed to solve the problem by molecular replacement. Another interesting example from the group of Osnat Herzberg studying a phage tail protein. They noted close agreement of DeepMind's model with their structure, however, they had a different assignment for a cis-proline. They reviewed the analysis and realised they had made a mistake in the interpretation and corrected it. The Oxford Protein Informatics Group provided good summaries of the impact of AlphaFold [Rubiera, 2020].

Drug companies are able to massively scale their previous algorithms, for example within one week Cyclica [Kurji and Windemuth, 2021] managed to identified $\sim 200,000$ potential binders from $17,712$ proteins, roughly doubling the coverage in their database. Representing the largest AI-augmented proteome-wide screening capability in the pharma industry.

The muscle protein, titin, the largest human protein, $34,350$ residues long, has not yet been structurally determined. Yet, AlphaFold has produced a prediction of what it may look like, see Figure 4.6.1.

4.6.1. Response

Upon the release of AlphaFold the structure prediction community have probed AlphaFold to learn a bit more about what it can do. Many have questioned the importance of experimental structure prediction. Particularly interesting examples have shown that experimental structure prediction is still relevant, but may become used more for structures that AlphaFold is not confident on. One such example is of two different conformations of a protein, resolved using cryo-EM, interestingly, AlphaFold produces an intermediate conformation [Amunts, 2021]. This is expected, due to the nature of training a model on a dataset containing different y data for the same x data, it converges to an average of the y over distribution $\mathbb{P}(y|x)$ (given a sufficiently flexible approximating function). Another interesting observation is that many AlphaFold predicted proteins have regions of high

disorder, or low confidence in globular domains. However, these may become ordered in a complex [Cooper, 2021]. This points to a significant limitation of the AlphaFold model, the individual modelling of chains which are parts of a larger protein complex. It has no way of knowing what the other chains in the complex look like, it can only infer from other examples it has seen during training.

Many structures in the PDB are proteins that only fold upon binding to other proteins, proteins that fold upon binding to substrates or metal ions, proteins that only fold when they are chemically modified, and proteins that fold directly into large complexes, such as the ribosome. When using known structures, the context is directly available, both in the structures themselves and in the literature associated with them, and many structures are published with explicit descriptions of why a protein will not adopt that structure on its own. When structures are not known, AlphaFold allows you to obtain them at unprecedented accuracy, but the trade-off relative to earlier methods is that they come stripped of context [Vernon, 2021].

Final predictions from AlphaFold are produced using different versions of the model and ranking the predictions. There have been cases where some of the models will predict incorrect folds but the correct fold is picked from the selection of predictions. One such example is GA98 and GB98 which differ by one amino acid, yet fold very differently. AlphaFold model 5 gets the fold incorrect, yet model 1 gets it correct and it selected [Wei, 2021]. Methods to improve the performance of AlphaFold are being developed, some tricks on the alignment, reranking models, and using an older model make a significant improvement [Bryant et al., 2021]. Other tricks include changing the random seed, or running for more recycling iterations to predict more challenging examples. Efforts to boost the performance of AlphaFold predictions for species with lower confidence, such as Trypanosoma cruzi and Leishmania infantum (important unicellular eukaryotic human parasites) have been made [Wheeler, 2021]. The low confidence appears to be caused by poor sampling of this branch of life in the databases AlphaFold was trained on. Other ways to improve predictions is by combining AlphaFold with physics based docking software [Ghani et al., 2021]. An extension of AlphaFold manages to model the structure of peptide-protein interactions, enabled by linking the peptide sequence to the protein c-terminus via a poly glycine linker [Tsaban et al., 2021].

5 | MSA and Structure Data for RNA

For protein structure prediction, ProteinNet [AlQuraishi, 2019b] is a carefully created dataset, organised by what was available at each CASP competition. This makes it easy for machine learning researchers to use the dataset and compare results in a standardised way. Unfortunately, there is no equivalent for RNA data. This chapter outlines the dataset preparation process used for the later experiments.

5.1. COLLECTING EXPERIMENTALLY RESOLVED RNA STRUCTURES

The initial step I took when creating the dataset was collecting a list of RNA chains that had been experimentally resolved. The Bowling Green State University, BGSU RNA Group [RNA Group, 2021] have compiled lists of the PDB codes for the RNA chains with resolved structures organised into sets based on the resolution of the resolved structure. They give lists of all structures resolved to an accuracy less than the distances: $\{1.5\text{Å}, 2\text{Å}, 2.5\text{Å}, 3\text{Å}, 3.5\text{Å}, 4\text{Å}, 20\text{Å}, \text{other}\}$. However, some items in the higher resolution lists are not always placed into the lower resolution lists. To account for this I create sets $S_{1.5}, S_2, ..., S_{\text{other}}$ such that $S_{1.5} \subseteq S_2 \subseteq ... \subseteq S_{\text{other}}$. Then I construct a partition of their union, $T_{1.5} = S_{1.5}, T_2 = S_2 \setminus T_{1.5}, T_{2.5} = S_{2.5} \setminus T_2, ...$ This forms sets of RNA codes based on resolution bins. Given that there are very few experimentally resolved RNA structures, these lower resolution bins are useful to increase the size of the dataset. We can then train on the low resolution structures initially and increase the resolution as training goes on. That way we extract some information from the low resolution structures, instead of just throwing them away.

The next step involves collecting all of the PDB codes required to get all of the chains. These raw PDB files are then pulled from the Protein Data Bank [Berman et al., 2000]. Then for each chain, the relevant PDB file is instantiated in BioPython [Cock et al., 2009], the individual PDB and fasta files for the sequence are written for each required chain.

5.2. MSA PIPELINE

To construct multiple sequence alignments we used the RFAM families [Kalvari et al., 2020]. The first step in constructing the MSA was to pull the fasta files for each family from RFAM.

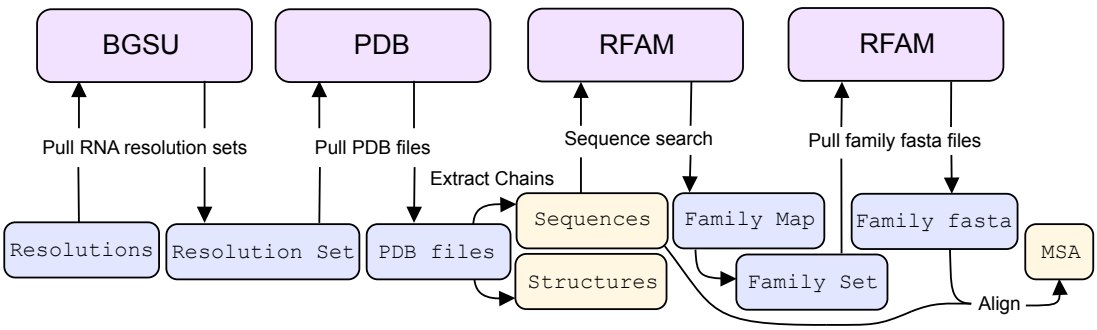


Figure 5.1.1: A flowchart showing the entire data pipeline. Online resources are in purple, local files are in blue and orange, orange boxes make up the raw file of the final dataset. I used Mathcha [Nha, 2021] to create this diagram.

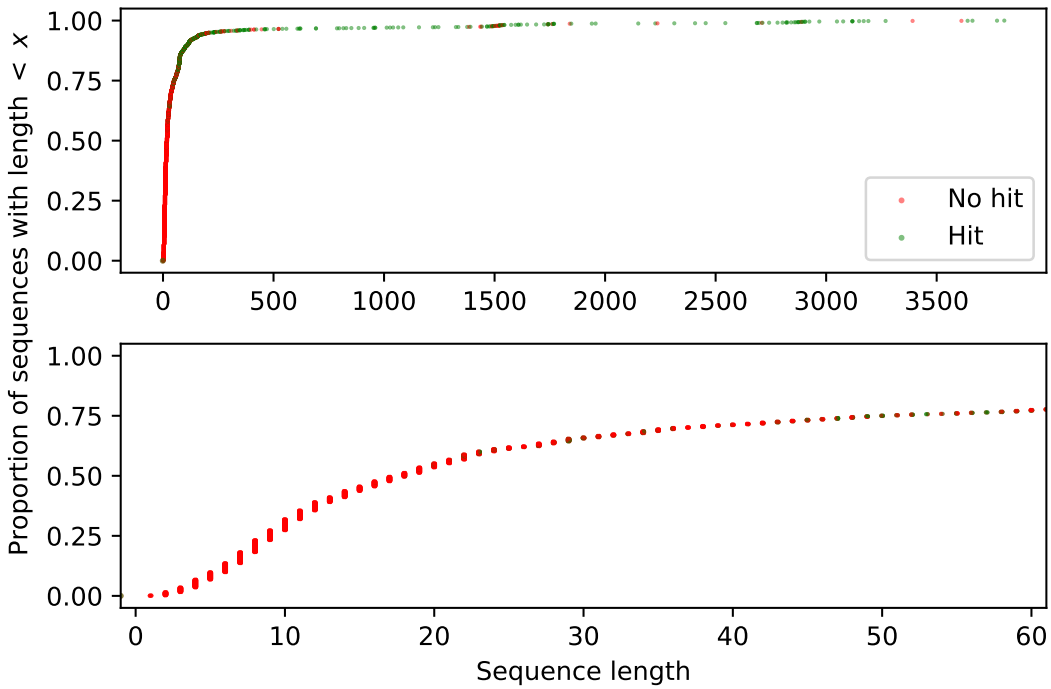


Figure 5.1.2: The proportion of the examples in the full dataset (y-axis) shorter than length (x-axis). If sequence search gave a hit to an RFAM family then it is coloured green, else red. The bottom plot is as above, zoomed in on the x-axis.

5.2.1. RFAM Sequence Search

The RFAM sequence search uses the RNAcentral server [Consortium, 2020], which performs local alignment using nhmmer [Potter et al., 2018]. I wrote a python program to communicate with the RFAM API and search all of the fasta sequences previously collected. This returns a ranked list of hits against RFAM families for each sequence. This returned strong hits, with low e-values, for most of the longer sequences. However, many of the shorter sequences did not return hits as the e-value was too high. The sequence could be completely unrelated and happen to be close to identical to another by random chance. Other sequences did not get hits because they are too far from the family. In total, this returned 677 hits to RFAM families with low e-values.

5.2.2. HMMER Sequence Alignment

This subsection was carried out by my lab partner. The first step involved preparing all of the data in the format used for HMMER. All of the sequence fasta files are concatenated into a single text file.

```
cat FASTA_PATH*.fasta > OUT_PATH
```

Listing 5.1: Concatenate raw fasta BGSU RNA sequences.

Similarly, all 4070 families from RFAM stockholm family alignments are concatenated into one text document.

```
cat MSA_PATH* > OUT_PATH
```

Listing 5.2: Concatenate raw MSAs of existing RFAM families.

A HMMER profile is constructed from the text document containing all the family alignments. This uses the command hmmbuild to make the profile and hmmpress to prepare the profile for use.

```
hmmbuild PROFILE_PATH FAMILY_PATH
hmmpress PROFILE_PATH
```

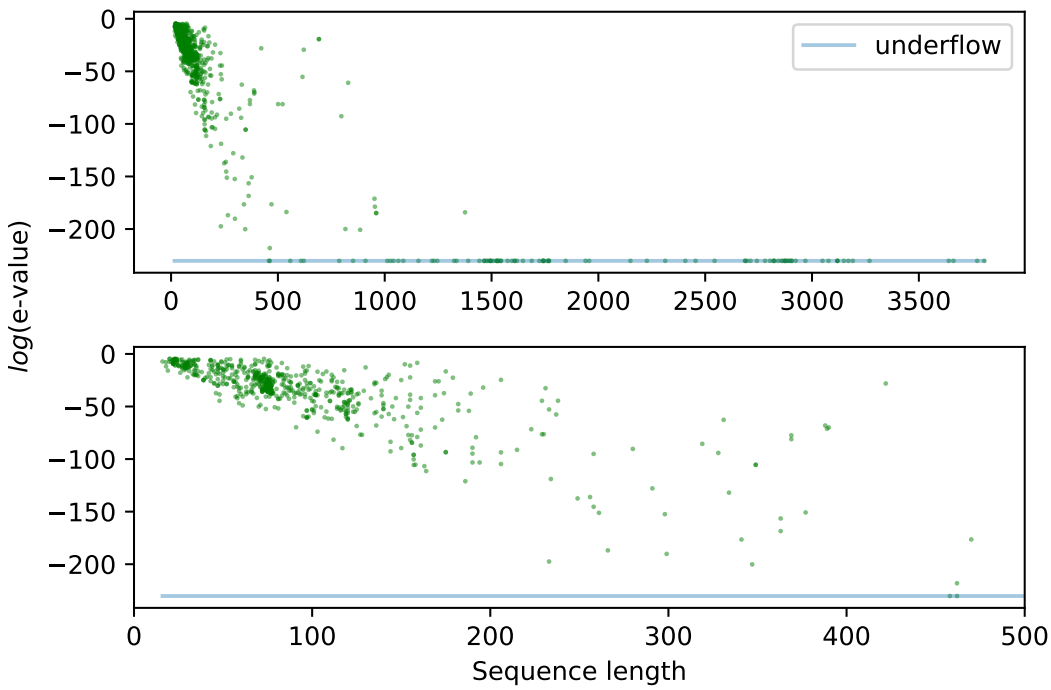


Figure 5.2.1: A plot of the lengths of the hits and the logarithm of the e-value to the family. The bottom plot is as above, zoomed in on the x-axis.

Listing 5.3: Build a profile using concatenated txt file with MSAs from RFAM families.

Then, for each RNA sequence in the concatenated fasta file scan the profile. The e-value is selected as being 10^{-2}, this picks up very weak correlations. This will generate some poor quality alignments, however, the e-values are recorded and filtered later. Collecting as many hits as possible allows for investigation of the quality of the alignments. The HMMER nucleotide scan command 'nhmmscan' is used. The flag '–tblout' saves a table with hits, the '–max' flag turns off all filters, increasing the sensitivity. However, this has substantial cost in speed, this routine took roughly 12 hours to complete.

```
nhmmscan -o OUT_PATH --tblout TAB_PATH -E E_VAL --max PROFILE_PATH INPUT_PATH
```

Listing 5.4: Scanning the profile with the fasta files.

5.2.3. Computing the MSA

This subsection was carried out by my lab partner. The resulting hits from the RFAM sequence search API and the HMMER sequence alignment routines are combined. Specifically, the family associated with each sequence that gave the lowest e-value is recorded. From this, the set of unique families can be computed. Then the RNA sequence fasta files for each family are pulled from RFAM.

The target RNA sequence and its corresponding family fasta file can then be joined and saved. A HMMER database can be created for the resultant fasta file. This is done using the 'makehmmerdb' command.

```
makehmmerdb TARGET_FAMILY_PATH BINARY_DATABASE_PATH
```

Listing 5.5: Build a HMMER database using the concatenated target fasta file and RFAM family.

Next, a stockholm[1] MSA was generated using the 'nhmmer' command with '–rna' and '-A' flags, '-A' indicates that the result should be given in stockholm format.

[1] Stockholm format is text representation of a multiple sequence alignment.

```
nhmmer --rna -A MSA_PATH TARGET_PATH DATABASE_PATH
```

Listing 5.6: Construct the MSA.

This was performed successfully in the majority of cases. Unfortunately, a small number of very close sequence-family hits with e-values less than 10^{-60} caused problems with HMMER. The routine would freeze and the alignment could not be generated. This would also occur with very long sequences. See Figure 5.2.1 for an indication of the distribution of sequences over e-values and length.

These can then be converted to MSA to a3m format using HH-SUITE software 'reformat.pl' function.

```
PATH_TO_HHSUITE_REFORMATPL sto a3m STOCKHOLM_PATH A3M_PATH
```

Listing 5.7: Convert to a3m format.

The a3m files are tidied so that the nucleotides from each sequence in the alignment are on the same line. The lower case letter in the a3m files indicate deletions have taken place, removing these produces the correct size MSA data. These are kept to generate the deletion matrices later.

5.2.4. Cleaning MSA Note

Roughly 46% of the final MSA files had one or two less bases than the fasta sequence, missing at the ends of the sequence. To account for these, the corresponding bases from the fasta sequence were inserted back, unless that row of the alignment had a gap at the end, in this case gaps were inserted. This produced sensible looking alignments, with very minor effects to the raw MSA output.

5.3. PARTITIONING THE DATA

In total we collected $3,825$ sequences, 722 of these has an MSA generated. To ensure that the dataset could be split in a meaningful way, into training, validation and hold out sets, it is important that there is as little overlap of evolutionary information/similar structures as possible.

For the sequences with an MSA, and hence associated with a family, we can group the examples by family. We can then add these groups one by one to a hold out set, until it is a certain size. Specifically I check when it is larger than $0.3 \cdot N_{\text{msa}}$, where N_{msa} is the number of examples with an MSA. Similarly for the validation set when it is larger than $0.2 \cdot N_{\text{msa}}$. The remainder of examples with an MSA are placed in a training set.

The remaining 3103 sequences without an MSA (single sequences) are not associated with a family. Partitioning the single sequences independently at random would potentially contaminate the validation/hold-out sets with very similar examples to those in the training set. To avoid this, I compute the pairwise global alignment of each fasta sequence using the pairwise2 function in the BioPython package [Cock et al., 2009]. I use a simple scoring matrix: -1 if the comparison is different, else 0, and -10 for an opening gap and -0.5 for an extending gap. This then returns the score of the maximum scoring alignment s_{ij}, 0 if they are identical, else negative. From these scores I compute a 'sequence distance' as follows

$$d(s_{ij}, l_i, l_j) = \frac{0.01 - s_{ij}}{1 + \min(l_i + l_j)}$$

where s_{ij} is the score for sequences i and j and l_i is the length of sequence i. The motivation for this 'sequence distance' function is that we require a strictly positive distance $d(s_{ij}, l_i, l_j) > 0$, note $s_{ij} \leq 0$. Sequences that differ greatly in length will be very different structurally, normalising by the average or maximum length sequence would give it an unreasonable low distance. However, normalising by the minimum length of the two sequences will penalise sequences that differ greatly in length, a desirable behaviour.

The resulting distance matrix can be used to cluster the sequences. I used an agglomerative [1] hierarchical clustering method with average linkage. Average linkage computes the average of the

[1] Agglomerative clustering methods start with singleton clusters and successively merges them forming a tree.

distances of each observation of the two sets. To do this I used the scikit-learn [Pedregosa et al., 2011] implementation with 100 clusters. Referring to the clusters of sequences as 'artificial families' I then add these artificial families to the training, validation and hold-out sets as done before for the 'real' families. These sets are saved and kept fixed throughout all experiments. The final proportions of MSA data-points is in the training, validation and hold-out set are $[0.27, 0.16, 0.57]$, and for single data-points is it $[0.21, 0.19, 0.6]$.

Not partitioning the single sequences in this way would require us to place only examples with MSA in the validation/hold-out sets and remove a random subset of their MSAs. This is not ideal in the limited data setting as we will have to remove valuable MSA training examples to place in validation/hold-out sets. However, because we have partitioning single sequences by how different they are, we can place a higher proportion of MSA examples in the training set than we otherwise would have.

5.4. BUILDING THE DATASETS

Once the training, validation and hold-out sets have been split, each dataset can be constructed by reading in the single chain PDB file and the corresponding MSA file, if it exists. The PDB file is parsed and instantiated using BioPython, then an RNA struct is created. See Listing 7.3 for the different fields. I provide more details about how these fields are extracted below.

```
@dataclasses.dataclass(frozen=True)
class RNA:
  code: str
  num_bases: int
  num_msa: int
  atom_positions: list  # [num_bases, num_atom_type, 3]
  sequence: str  # [num_bases]
  atom_mask: list  # [num_bases, num_atom_type]
  base_index: list  # [num_bases]
  b_factors: list  # [num_bases, num_atom_type]
  msa: list  # [num_msa, num_bases]
```

Listing 5.8: RNA data-structure.

Each base has the shares the same backbone sub-unit, and hence all of the atoms in it. The convention in PDB [PDB, 1992] is to store them in the order given in Listing 5.9.

```
backbone = ["P","OP1","OP2","O5'","C5'","C4'","O4'","C3'","O3'","C2'","O2'","C1'"]
```

Listing 5.9: Backbone Atoms.

Each base is composed of different atoms, making up four unique sub-units. See Listing 5.10 for the ordering in the PDB convention.

```
bases = {
  'G':["N9","C8","N7","C5","C6","O6","N1","C2","N2","N3","C4"],
  'A':["N9","C8","N7","C5","C6","N6","N1","C2","N3","C4"],
  'C':["N1","C2","O2","N3","C4","N4","C5","C6"],
  'U':["N1","C2","O2","N3","C4","O4","C5","C6"],
}
```

Listing 5.10: Base Atom Order.

In a PDB file each base has a number of rows to specify the coordinates of each atom in the base. For example 'G' has 23 atoms, 12 in the backbone (`backbone`), 11 in the nucleobase (`bases['G']`). Sequentially, the bases in the RNA are iterated over. For each of these, a mask is created corresponding to each atom present for that base. For example the mask for 'G' is an array of zeros of size 23, with ones in the indices where the atom is present in the PDB file. Similarly the coordinates for 'G' is an array of zeros of size (23,3), with the coordinates written into the appropriate index along the first axis. The temperature factor is also recorded. This is otherwise know as the isotropic B-value.

The raw data is parsed into the RNA structs in Listing 7.3. Then made into a python dictionary using the '__dict__' attribute. These are placed in a dictionary mapping from there codes, this is parsed into a json string and compressed using gzip. The reason for doing it in this way is that the Google Colab IO is faster at loading a single file and decompression and re-instantiation is relatively fast in python.

6 | Adapting AlphaFold to RNA

6.1. IMPLEMENTATION

Deepmind released their implementation of AlphaFold [Jumper et al., 2021]. The implementation is in python, and the autograd library used is JAX [Bradbury et al., 2018]. It also makes use of Haiku [Hennigan et al., 2020], a higher level library for neural network modules. I am more familiar with the PyTorch library [Paszke et al., 2019] and have reimplemented the majority of AlphaFold in PyTorch-Lightning [Falcon et al., 2019]. I describe the architectural changes below. Due to time constraints I have implemented the necessary components for backbone prediction and not all atom prediction. It has been shown that AlphaFold can still perform well when augmented in this way [Moffat et al., 2021].

6.2. TESTING THE IMPLEMENTATION

Since there are numerous modules it is easy to make bugs. To avoid this I tested each module individually against the official JAX implementation. The challenging part about writing the testing code, is copying parameters between Haiku and PyTorch code. Even though I structured the modules in the same way, when creating sequential layers of a given neural network module, Haiku stacks all parameters into a single tensor. Whereas PyTorch just appends an index to the parameter path in the state dictionary. To correctly transfer parameters between multiple levels of sequential stacks of modules, the path indices and parameters axes must organise the rearrangement recursively. After doing this it is fairly trivial porting all of code from Jax-Haiku to PyTorch. The tests consist of initialising one implementation of the module, transferring its parameters to the other implementation. This is followed by passing in the same randomly generated input tensors into each module and computing the absolute error of their difference and asserting that it is below a threshold set to 10^{-6}. The only components I did not test were dropout layers, as manually keeping track of the random number generator keys is more challenging in PyTorch as it does this automatically.

6.3. ARCHITECTURE CHANGES

The majority of the parameters in the AlphaFold architecture are found in the main trunk, the 48 block Evoformer. There are embedding modules that feed into the Evoformer, converting the input tensors into latent representations which the Evoformer operates on. Finally, there are a number of neural network heads which output a prediction which is compared using a loss function to ground truth data.

6.3.1. Changes to Embedding Modules

I have removed the template embedding modules and the corresponding template data pipeline: template search, template pair features and template angle features. This corresponds to the TemplateEmbed block in Figure 4.2.2. Additionally, the linear layers corresponding to left and right amino acid embeddings are replaced with linear layer embeddings for RNA bases. Similarly, the linear layers for MSA and extra MSA embeddings were adapted for the number of RNA tokens.

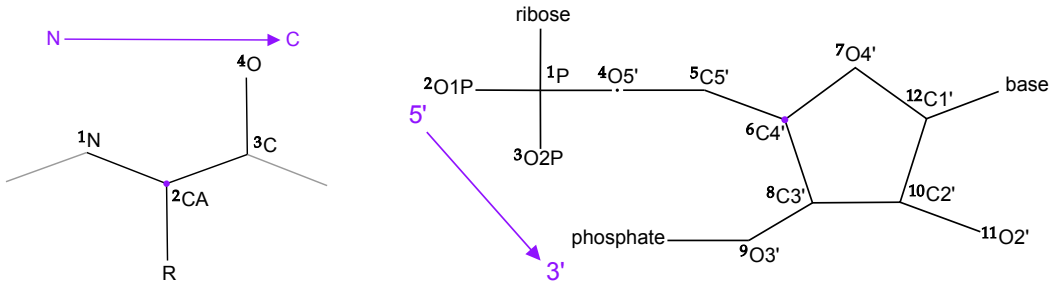


Figure 6.3.1: The backbone of a protein is shown on the left, the backbone of RNA is shown on the right. The relevant atoms are labelled using the PDB codes with a preceding superscript indicating the order they are listed in the PDB file for a given structure. I have recreated the PDB figures from [PDB, 1992] in Mathcha [Nha, 2021].

6.3.2. Changes to the Evoformer

Architecturally, the Evoformer remains identical to the original AlphaFold. However, I do change the hyperparameters. Specifically, I start off training with 0 dropout in the attention modules, this was previously 0.15. I also decrease the number of blocks from 48 to 8, but increase this to 12 later in training.

6.3.3. Changes to Network Heads

I removed two of the network heads and corresponding losses. Namely, the predicted LDDT head with its 'conf' loss and the experimentally resolved head with its 'exp resolved' loss. These correspond to 'Conf' and 'Resolved' losses in Figure 4.2.3. The masked MSA head and loss remains identical to AlphaFold, apart from changing the number of output logits to correspond with the number of RNA tokens (plus the true mask, bert mask and gap tokens as done in AlphaFold).

Distogram Head

I performed overfitting experiments on a small dataset of five RNA examples, with a simplified AlphaFold architecture. I found that the original binned distogram prediction and corresponding cross-entropy loss would take a large number of gradient steps, ~ 2000, to fit a single example. More details about my experiments are given in the next chapter. During training the predictions would show no resemblance for a large number of steps, then suddenly become very accurate. In my limited compute setting I created a faster, by possibly less accurate loss. This new loss computes two logits for the minimum bin distance and the maximum bin distance, and interpolates between them to give the distance at that position. More details are given in the following chapter.

Structure Module

In the structure module I removed the torsion angle prediction network, therefore there is no torsion angle loss term in the auxiliary loss. Additionally, I removed the side-chain prediction with corresponding violation loss and side-chain FAPE loss. The only remaining components are the backbone affine frames head and the backbone FAPE loss. These changes affect the 'Aux' loss in Figure 4.2.3.

I will now describe the process of creating the backbone frames. I mimic the process used in AlphaFold to create the protein backbone frames. Unfortunately the backbone atoms for RNA have many more degrees of freedom than for proteins. This makes it unclear how to construct basis frames for each nucleotide from the RNA backbone atoms. A reasonable basis is shown in Figure 6.3.1. I have decided to test out the coordinate frames $\mathcal{B}_1, \mathcal{B}_2$,

$$\mathcal{B}_1 = \mathrm{Basis}_i(\mathsf{C4'}_i, \mathsf{O3'}_i, \mathsf{O5'}_i),$$
$$\mathcal{B}_2 = \mathrm{Basis}_i(\mathsf{C4'}_i, \mathsf{P}_{i+1}, \mathsf{P}_i).$$

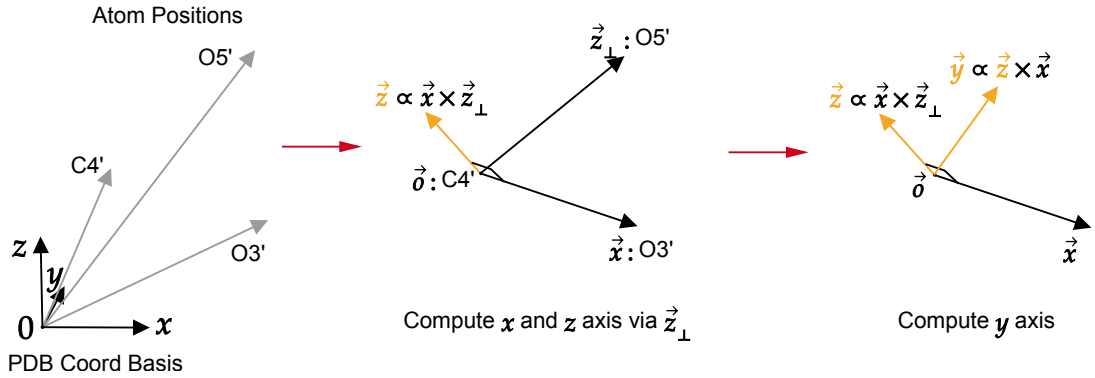


Figure 6.3.2: Constructing the frame for a base in the RNA. This is the same as in AlphaFold [Jumper et al., 2021] but with different atoms of course. Note that this gives a transformation (translation and rotation) from the PDB basis to the base-basis. The atoms correspond to basis $\mathcal{B}_1$, the process is the same for $\mathcal{B}_2$. I used Mathcha [Nha, 2021] to create these diagrams.

The operator '$\mathrm{Basis}(\cdot,\cdot,\cdot)$' accepts three arguments. The first argument is the atom coordinate which will be used as the origin for that nucleotide. The second argument is the atom coordinate for which the x-axis will be constructed. The third argument is the coordinate of the atom which lies in the xy-plane of the frame. Computing the coordinate frame is fairly trivial geometry and is omitted, I provide a visualisation for the basis construction in Figure 6.3.2.

7 | Experiments

Since this is a supervised learning problem, I will describe the training procedure with a supervised learning taxonomy in mind. If the test set is given in advance it is possible to perform transductive learning, where the goal is to achieve good performance on the test points. This usually does not generalise well as it does not consider the marginal data distribution $\mathbb{P}(X)$. It minimises

$$\frac{1}{n_{\text{test}}} \sum_{j=1}^{n_{\text{test}}} \int_{\mathcal{Y}} \ell(f(x_j), y) p(y|x_j) dy.$$

Similarly, fixed design experiments attempt to get good estimates on a set of training points ignoring $\mathbb{P}(X)$. By comparison inductive learning is agnostic about what the learned model is tested on. The goal is to predict well with respect to $\mathbb{P}(X, Y)$. Similarly, random design experiments aim to make good predictions with respect to $\mathbb{P}(X, Y)$.

The experimental design I use is neither fixed, nor random design, but somewhere in between. With a distribution weighting more difficult, longer RNA chains as well as higher resolution chains with larger probability. The learning is performed in an [1]offline manner. However, the dataset is not drawn independently from a uniform distribution over all possible RNA examples. This is because the data is a function of the evolution that produced it, efforts to ensure that the hold-out and validation sets are significantly different from the training data are discussed in chapter 5.

Due to limited computational resources I have not performed cross-validation or hyperparameter tuning. There are many hundred discrete and continuous hyperparameters and even performing random hyperparameter search would be too slow seeing as only a few training procedures can be performed. Before running experiments I confirmed the neural network modules computed the same function as the original jax counterpart in AlphaFold, details are in chapter 6. I then created lists of a random selection of the chain codes for ~ 200 high resolution structures from the training set to record metrics for. I also create another subset of ~ 50 training example chain codes to record the Evoformer distograms and the final quaternions and positions for each frame. Additionally, the loss value and code are recorded for all examples. The reason for only using subsets for metrics, frames and distograms is the large memory cost. Furthermore, I record all metrics for the validation and hold out sets. I also record the frames for a subset of the validation codes. All models are implemented in PyTorch-Lightning for faster training on the GPU. I trained on Colab Notebooks, usually on the Telsa K8 or P100. Hyper-parameters match AlphaFold closely, the biggest changes are removing dropout and decreasing the number of Evoformer blocks to 8 instead of 48.

7.1. SUB-MODEL ARCHITECTURES

Over the course of the final section of the project I attempted to train the parameters of an AlphaFold based neural network architecture. Only having access to single Google Colab GPUs limited the pace of running experiments and the length of training time. While I initially performed experiments with a complete model, see Architecture-1 in Figure 7.1.1, running on the full training dataset took roughly an hour to train for just a single epoch. The loss would decrease on the training and validation set, plateauing with diminishing returns in rate of decrease in loss. Given the length of experiment time it was very difficult to tune crucial hyperparameters such as learning rate, schedulers and loss weights. In the end I decided to focus on the crucial first step

[1]Offline learning assumes a finite dataset, online learning corresponds to minimising the regret as more examples are observed.

```
alphafold_iteration (r) <-- unif{1,2,3,4}:
  embeddings_and_evoformer:
    relpos
    seq_emb
    msa_emb
    extra_msa_emb
    prev_emb
    evoformer_iteration (48): {
      msa_row_attn_p_b
      col_glob_attn
      msa_transition
      outer_prod_mean
      tri_mult_out
      tri_mult_in
      tri_attn_str
      tri_attn_end
      pair_transition
    }
  distogram_head
  msa_head
  folding_iteration (12):
    invariant_point_attn
    transition
    backbone_update
  structure_head
```

Listing 7.1: Architecture-1 the full RNA-AlphaFold model.

```
embeddings_and_evoformer:
  relpos
  seq_emb
distogram_head
```

Listing 7.2: Architecture-2.

```
trunk (1):
  embeddings_and_evoformer:
    relpos
    seq_emb
    msa_emb
    extra_msa_emb
    evoformer_iteration (8): {
      msa_row_attn_p_b
      col_glob_attn
      msa_transition
      outer_prod_mean
      tri_mult_out
      tri_mult_in
      tri_attn_str
      tri_attn_end
      pair_transition
    }
  distogram_head
```

Listing 7.3: Architecture-3.

```
new_af_iter (r) <-- unif{1,..,R}:
  embeddings_and_evoformer:
    relpos
    seq_emb
    msa_emb
    extra_msa_emb
    prev_emb
    evoformer_iteration (E): {
      msa_row_attn_p_b
      col_glob_attn
      msa_transition
      outer_prod_mean
      tri_mult_out
      tri_mult_in
      tri_attn_str
      tri_attn_end
      pair_transition
    }
  folding_iteration (8):
    invariant_point_attn
    transition
    backbone_update
  structure_head
```

Listing 7.4: Architecture-4.

Figure 7.1.1: A module is shown in red if there is no weight sharing of its submodules, if there is weight sharing it is shown in purple. I have coloured the distogram head in orange as it is interchanged for other variations of distogram heads.

of designing a machine learning model, overfitting a single example perfectly. To do this I first trained Architecture-1 on a single example dataset. However, this still could not perfectly overfit, even after thousands of iterations. The explanation for this could be one of three things (i) there is a bug in the training or model code, (ii) the hyperparameters are not allowing the model to overfit, either it will take to long to converge or it is taking too large steps, (iii) the architecture does not have the capacity to fit the data. Fortunately the reason turned out to be (ii). I determined this by creating small sub-architectures and training on a single example. These small networks were fast to train, allowing me to experiment with many different hyperparameter settings. After performing rapid prototyping I found a number of techniques that allowed the model to learn progressively but increasing the difficulty of the task while increasing the model capacity. I discuss these methods in the following sections.

7.2. INITIAL TRAINING EXPERIMENTS

Although much of the work in this section did not bare fruit, it did show that transferring previously trained AlphaFold parameters hindered performance. I began by constructing experiments based on a full training procedure and comparing hyperparameters settings, variations on defining coordinate frames and with or without MSA data.

Since we are in the low data setting it is necessary to perform a lot more training on the same examples, making it particularly challenging to get good validation performance without overfitting. This is especially relevant when attempting to learn highly complex function, its similar to having two data-points from a one dimensional sin wave function and fitting them with a straight line. To minimise the effects of reusing the same examples I introduces a randomised dataloader which supplied crops of longer sequences with higher probability than unseen sequences. So that for sequences greater than the crop size it serves an subset of these sequences with uniform probability over the total length of RNA in the dataset. I would also control the probability based on resolution, initially serving low resolution structures, then sending the probability of serving them to 0.

To train the neural network I designed a probability distribution over the training set. The probability mass function (pmf) over the data depends on the length of the training example, the resolution of the resolved structure, and the current epoch. The motivation for probabilities being based on these variables is the following. Early in training is best to use the low resolution structures, to extract some information from them. Whereas later in training is best for the high resolution structures, to give accurate target values to converge to. To train the model I serve training examples with probability weighted according to their resolution and the current epoch. I use the term 'epoch' very loosely to mean giving the model N random examples, where N is the number of examples in the training set. So many examples will likely not be seen in a 'epoch' and many examples may be sampled multiple times.

The probability mass functions I use are from the exponential distribution, over the set $X = \{0, \frac{1}{N_{\text{bin}}+1}, \frac{2}{N_{\text{bin}}+1}, ..., 1\}$, where N_{bin} is the number of resolution bins. Defining $t \in \{0, 1, 2, ...\}$ to be the epoch and c to be the rate of decay, then the equation is

$$p_1(x, t; c) = \frac{\exp(-cxt)}{\sum_{z \in X} \exp(-czt)}. \tag{7.1}$$

Similarly, let π be the vector $[0, 0, 0, 0, 0, 0.2, 0.6, 0.8]^\top$, and x is the index of the resolution bin where 1.5Å is index 1 and 'other' is index 8,

$$p_2(x, t; c) = \frac{\exp(-c\pi_x t)}{\sum_{z \in X} \exp(-czt)}. \tag{7.2}$$

The second probability mass function $p_2(x, t; c)$ is more useful since it clips the higher resolution bins to have a fixed, equal probability so that they do not all decay to 0 apart from the 1.5Å bin.

Due to the quadratic scaling of the pairwise representation and a cubic operation (outer-product mean) with respect to sequence length we must crop the sequences to a fixed length, L_{crop}. To crop the longer sequences I use the following procedure:

- Compute $l_{\text{extra}} = l_{\text{seq}} - L_{\text{crop}}$.

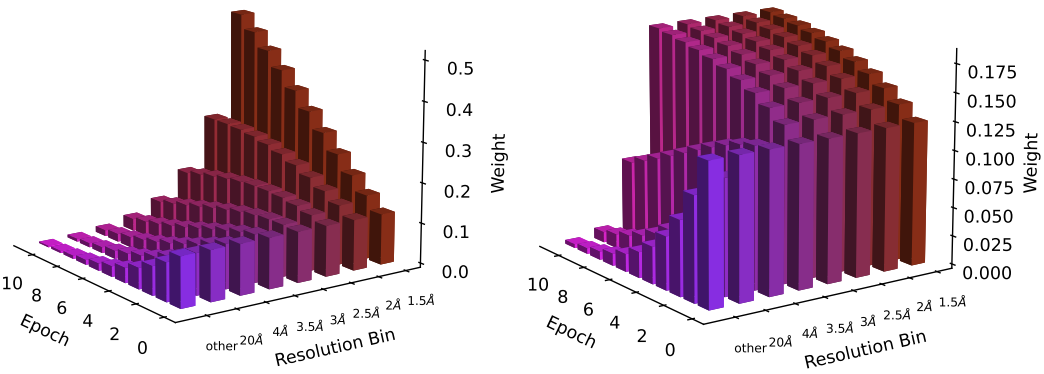


Figure 7.2.1: The probability mass functions p_1 (left) p_2 (right) over resolution bins at each epoch. The function generating these bars is given in equation (7.1) (left) and equation (7.2) (right) with $c = \frac{1}{2}$. The plot on the right is the same function but clipped to onefor resolutions above 3.5Å.

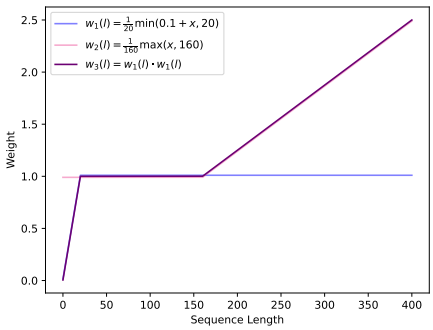


Figure 7.2.2: Weighting function over RNA length. The pmf is proportional to the weights.

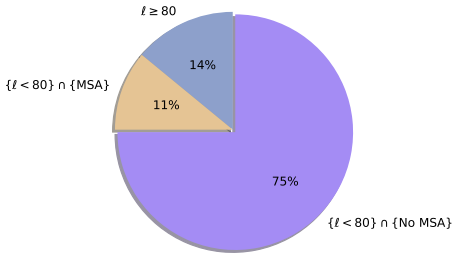


Figure 7.2.3: Full dataset partitioned by length (less than 80) and whether an MSA exists.

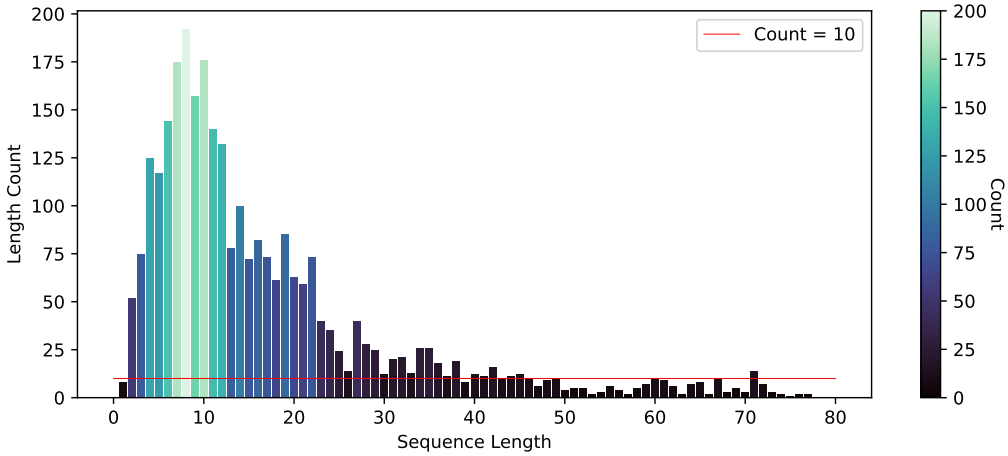


Figure 7.2.4: A histogram with bin size equal to the integers of the number of sequences in the dataset with lengths 1 to 80.

- Sample $i \sim U\{1, ..., l_{\text{extra}}\}$.
- Compute the midpoint, $m = i + \text{floor}((L_{\text{crop}} + 1)/2)$.
- Select the set $J = \{i - m, i - m + 1, ..., i + m - 1\}$, which is used to index all inputs along the sequence dimension.

This procedure only samples a $\frac{L_{\text{crop}}}{l_{\text{seq}}} \in [0, 1]$ size region of the chain. This means that for a given resolution bin, regions of chains are not sampled uniformly. A shorter sequence will be seen more on average than all regions of a longer sequence, as the longer sequence is cropped. This does not make the most efficient use of the limited structure data. To account for this, I sample the sequences in a given resolution bin with probability proportional to $\max(L_{\text{crop}}, l_{\text{seq}})$, this is shown in Figure 7.2.2, note there is also a penalty term for very short sequences.

A major bottleneck in training the AlphaFold neural network is the large memory of the activations, scaling $\mathcal{O}(l_{\text{seq}}^3)$. Since sequences significantly shorter than the crop size make up most of the dataset, multiple batches of these can fit into memory amortising more of the forward and backward pass computations. However, these can only be batched if they are exactly the same length, and have the same number of MSA sequences. Note in Figure 5.1.2 that $\sim 70\%$ of sequences are shorter than 60 bases. It is possible to sample an epoch and compute the lengths of the sequences in the epoch, then cluster nearby sequence which are the same length. Note that most of the data will be able to be batched provided it has the same length, see Figure 7.2.3, the total number of sequences is $3,825$. The counts of sequences of a given length are shown in Figure 7.2.4. However, when clustering batches by sequence length over large regions it seemed to result in poor performance. I hypothesise that it would create biased gradient steps which did not give a robust path through parameter space. The gradient step was produced by an average over the batch as the batch sizes differed, computing a sum over the gradients of the batch would drastically change the effective learning rate and corrupt training. However, taking an average meant that the shorter examples would be placed in a larger batch and each individual example in that batch would contribute less in the gradient step. Due to difficulties in training with variable batch sizes, I set the batch size to one for all future experiments.

To increase the crop size, to train on longer examples I found it useful to use gradient/activation checkpointing on each block of the Evoformer. With this change it is possible to scale the length of sequence crops to over 160.

7.2.1. Metrics

I decided to evaluate the training, validation and hold-out performance on a number of metrics. The root-mean-square deviation (RMSD) [Coutsias et al., 2004] is the distance between each point squared, then averaged, then the square root is taken. The global distance test (GDT) [Zemla, 2003] is intended as a more accurate measurement than the common RMSD metric. RMSD is sensitive to outliers, e.g. if there is poor modeling of individual loop regions in a structure that is otherwise reasonably accurate. I report GDT as a proportion from 0 to 1 over the origins of each backbone frame. GDT is based on rigid-body superposition and will heavily penalise a small error if it displaces the rest of the chain. The local Distance Difference Test (lDDT) score [Mariani et al., 2013] is superposition-free. The lDDT score lends itself well to comparing functionally relevant regions of the structure locally. The TM-score [Zhang and Skolnick, 2004] is intended as an accurate measure of global similarity of structures than RMSD. However, the TM-score requires global optimising over a transformation for each residue in the chain. A lower bound of the TM-score is shown in [Jumper et al., 2021] which I will report on.

7.2.2. Experiments

I keep many hyperparameters the same as that in AlphaFold. I also experiment with loading the original trained weights for model_5_ptm, denote this by AF_init. Alternatively, the parameters are initialised from scratch, denote this by new_init, otherwise from a previous checkpoint: v5_init which means the final parameters from training version 5 were used as initialisation. I also explored the different probability mass functions in Equations 7.1 and 7.2. I used a number of optimiser settings:

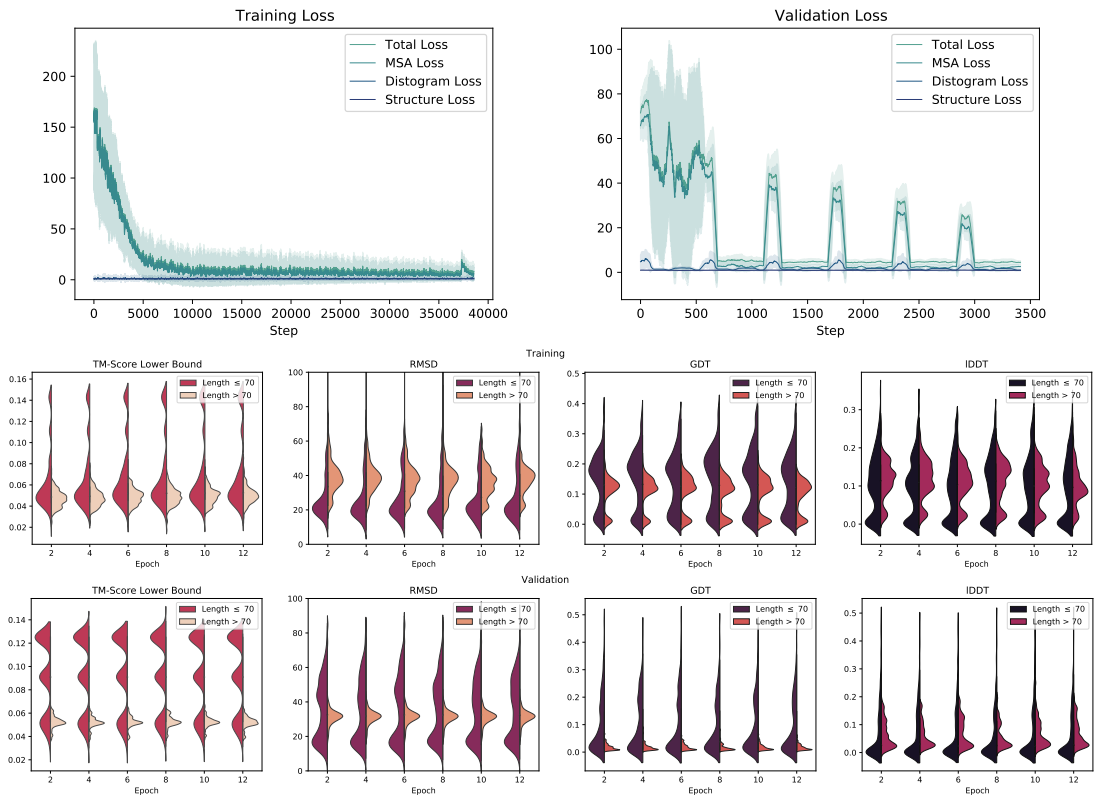


Figure **7.2.5.a:** This experiment corresponds to version 6, a model initialised with previously trained AlphaFold parameters.

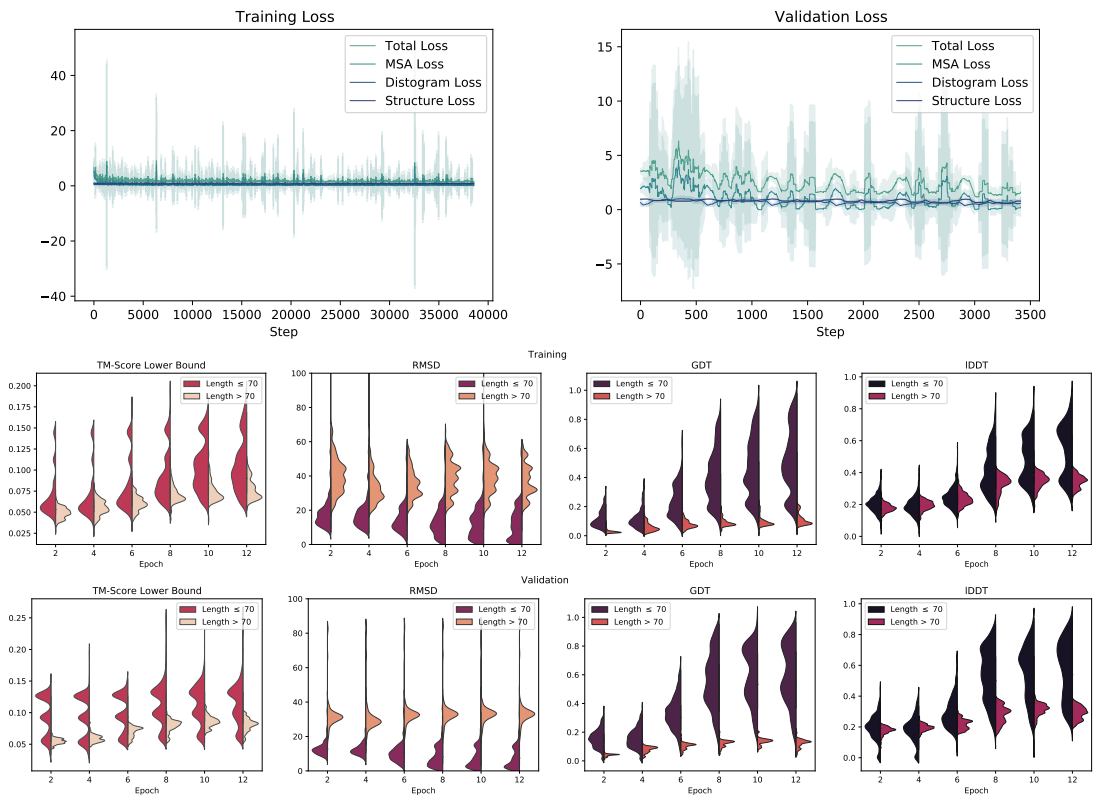


Figure **7.2.5.b:** This experiment corresponds to version 7, a model initialised with new parameters.

Figure 7.2.5: In the above figures the top row of plots are the loss components during training (left) and validation (right). The middle row of plots are the metrics for a subset of training examples (the subset is randomly picked before experiments and kept constant). The final row is of validation metrics. The violin plots are normalised by maximum mode (width), not by area or counts.

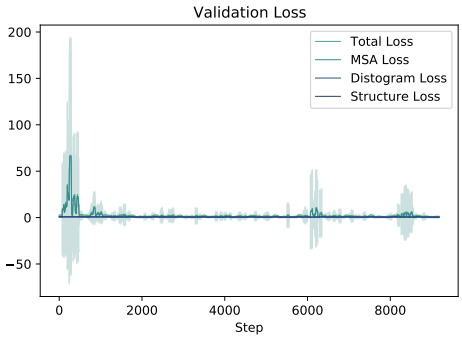


Figure **7.2.6.a:** Validation loss components over epochs.

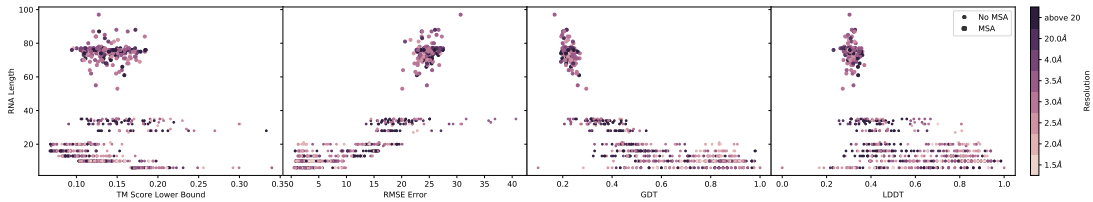


Figure **7.2.6.b:** Test metrics after training.

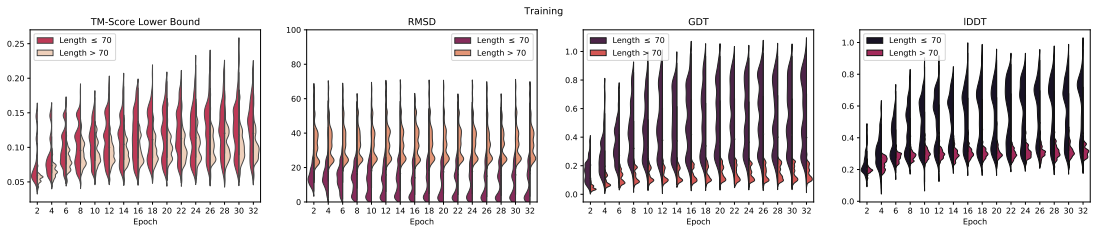


Figure **7.2.6.c:** Training metrics on a fixed subset of training examples.

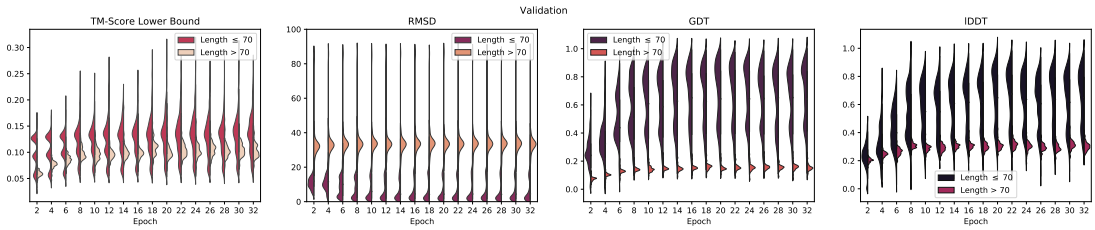


Figure **7.2.6.d:** Validation metrics.

Figure 7.2.6: This experiment was initialised with new parameters. The violin plots are normalised by maximum mode (width), not by area or counts.

- **close_lr**: This setting is tailored to trained AlphaFold parameter initialisation and contains three parameters groups. The first parameter group corresponds to embedding layers with a large learning rate of 10^{-3}, the second to the structure module with a learning rate of $4 \cdot 10^{-4}$ else $3 \cdot 10^{-4}$. All groups are optimised using SGD.
- **far_lr**: This setting identical to above but with different learning rates, the embedding group is set to 10^{-3}, structure module to $2 \cdot 10^{-4}$ else 10^{-4}.
- **default**: The setting is optimised using Adam and has learning rate $3 \cdot 10^{-4}$.
- **original**: The setting is optimised using Adam and has learning rate $3 \cdot 10^{-3}$.
- **decay**: The setting is optimised using Adam and has learning rate of 10^{-3}, with a linear decay scheduler down to 0.1 of the learning rate over the total number of epochs.
- **cosine**: It is common for transformers to be optimised with a cosine annealing learning rate scheduler. The setting is optimised using Adam and has learning rate of 10^{-3}, but with a scheduler which multiplies the learning rate by $(11 + 9\cos(40t))/20$ where t is the time in epochs and proportion through the epoch.

The table of experiments is given in Table 7.2.1.

Version	Basis	Optim Setting	Init.	Evo Bl.	Fold Ly.	Filter MSA	PMF
1	$\mathcal{B}_1$	close_lr	AF_init	-	-	-	p_1
2	$\mathcal{B}_2$	close_lr	AF_init	-	-	-	p_1
3	$\mathcal{B}_1$	close_lr	new_init	-	-	-	p_1
5	$\mathcal{B}_1$	default	v4_init	-	-	-	p_1
6	$\mathcal{B}_2$	far_lr	AF_init	-	-	-	p_1
7	$\mathcal{B}_2$	decay	new_init	-	12	-	p_1
8	$\mathcal{B}_2$	decay	new_init	-	-	-	p_1
9	$\mathcal{B}_2$	decay	new_init	8	-	-	p_1
10	$\mathcal{B}_2$	decay	new_init	-	-	10^{-10}	p_1
11	$\mathcal{B}_2$	decay	new_init	8	-	10^{-10}	p_1
12	$\mathcal{B}_2$	cosine	new_init	8	-	10^{-10}	p_1
13	$\mathcal{B}_2$	close_lr	AF_init	8	-	10^{-10}	p_1
14	$\mathcal{B}_2$	original	new_init	8	-	10^{-6}	p_2

Table 7.2.1: Initial experiments, each is run for 12 epochs apart from version 14 which is run for 32 epochs.

A selection of the results are shown in Figures 7.2.5 and 7.2.6. In Figure 7.2.5.a. the validation loss shows a repeating pattern, this is because it is deterministic and there is cluster of long sequences from a certain RNA family. While overall performance is poor, interestingly the AlphaFold initialised model seemed to perform better on the longer sequences, see the violin plot. The performance of the newly initialised model in Figure 7.2.5.b shows promising learning in the training and validation violin plots, particularly for the shorter sequences.

Overall the performance was rather poor, the optimiser changes in settings listed in Table 7.2.1 had relatively little effect. Changes that seemed to be important are: initialising with new parameters, not the previously trained AlphaFold parameters trained on proteins; using the basis $\mathcal{B}_2$. There was relatively little change when decreasing the number of Evoformer layers down from 48 to 8. However, these deductions are not conclusive as there was no cross-validation performed or even re-running each model, and perhaps the biggest issue is short training time.

In summary, this procedure managed to quickly learn the short sequences but seemed to hit a wall with longer sequences. I repeatedly tried to train with different settings to break past this, but did not manage to achieve it this way. To analyse where the model was going wrong I decided to select a subset of five long RNA sequences and train the full model just on those examples and monitored the overfitting properties. This is discussed in the next section.

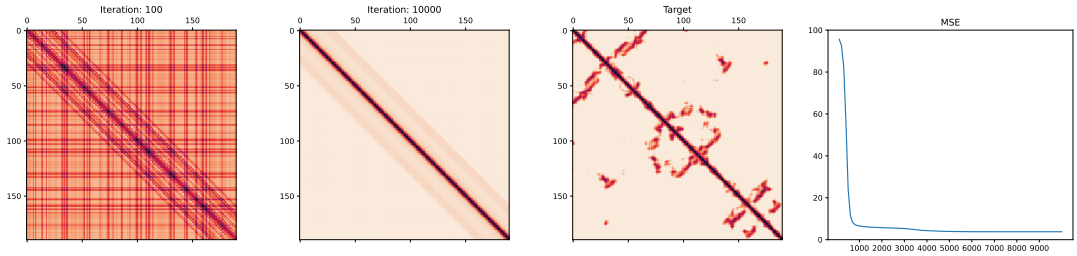


Figure 7.3.1: Distogram predictions of the chain 6D9J-4 using just relpos, left-right-outer embedding and a distogram head with the original distogram loss. The leftmost plot is early in training, iteration 100, the following plot is iteration 10000, then the target distogram and the MSE loss.

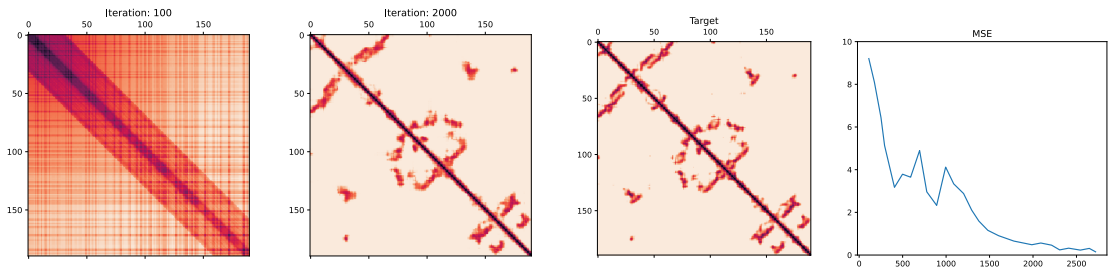


Figure 7.3.2: Distogram predictions of the chain 6D9J-4. The model consists of relpos, left-right-outer embedding, triangle attention, triangle multiplication, pair representation transition block and a distogram head with the original distogram loss. The leftmost plot is early in training, iteration 100, the following plot is iteration 2000, next is the target distogram, and the MSE over iterations of gradient decent.

7.3. DISTOGRAM LOSS EXPERIMENTS

The model managed to get close to the structure after thousands of iterations but could not overfit. This is big issue, in attempt to rectify this, I striped down the model to contain the minimal number of modules and overfit on a single example. This worked. From there, I progressively rebuilt the Evoformer block trained on just the distogram loss. However, the original distogram loss would take relatively long to learn so I adapted it to a slightly different distogram loss which managed to learn faster, but seemingly less accurately. As this is an auxiliary loss I decided that this was a reasonable trade off and proceeded to build up the model.

The most simplified architecture consisting of just the relative position encoding, the sequence embedding and the distogram loss, this corresponds to Architecture-1 in Listing 7.2. The sequence embedding is done twice (with different parameters) and then an outer product is computed, this pairwise embedding and the relative position embedding are added, to produce the input to the distogram head. As in the original distogram head, it projects into 64 distance bins over the range 2Å to 22Å, with a transpose and sum to make it symmetric. The distogram loss computes the true bin for each position in the true distance matrix and computes the softmax loss using the output logit from the distogram head. This model managed to learn correctly, as best it can, see Figure 7.3.1. Note that the main diagonal learns the best values to fit the true distogram for 32 positions (this is of course because the relative-positional encoding is 32 bins wide). Notice that early in training it has vertical and horizontal streaks, these correspond to the different pairwise combinations of bases, from projecting through the left and right embeddings (followed by an outer product). This experiment shows that the original distogram loss is working correctly.

To allow the model to reason over the context of the sequence I added attention modules. In theory this model can overfit to produce an arbitrary distogram matrix (y) even for an input sequence of all the same bases (x). This is because the relative position encoding gives different projections for a region around the main diagonal (a vector space defined by 65 vectors, 32 bins up and down). Attention is performed over rows i. Linearly combining projections of the set of different vectors around the diagonal, the linear combination is a function of the other sequence

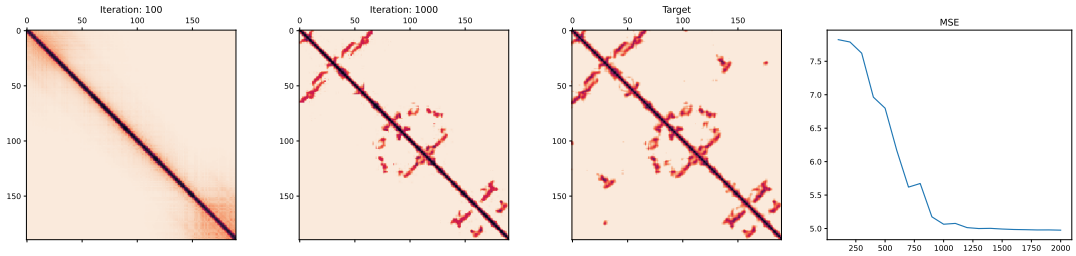


Figure 7.3.3: Distogram predictions of the chain 6D9J-4. The model consists of relpos, left-right-outer embedding, triangle attention, triangle multiplication, pair representation transition block and a distogram head with the delta-distogram loss. The leftmost plot is early in training, iteration 100, the following plot is iteration 1000, next is the target distogram, and the MSE over iterations of gradient decent.

axis j. Therefore each element i, j can produce a an independent set of vectors over i and j. The pair transition acts as a small MLP over all i, j, so it can ease the prediction of arbitrary data over i, j. Specifically, this model consists of a simplified evoformer block with triangle multiplication out, then in, triangle attention start, then end, and a pair transition block. The performance of this model is shown in Figure 7.3.2. This shows that this model can fit a data-point arbitrarily well. However, it takes ~ 2000 gradient steps to fit a single example. I hypothesise that the projection to 64 distance bins trained with cross-entropy loss does not allow for the most efficient possible learning as it does not encode the spacial relationship of the bins. I attempted to create a loss function that encode the spacial relationship better. I decided to investigate a distogram head which uses a linear layer on the pairwise representation to directly predict the distance at that position. Using the predicted distogram I then compute the MSE loss with the true distogram. This method was not robust, and did not seem to fit the data well. I replaced the linear layer with a two layer MLP, however this did not help much. It may have been challenging to learn a large distance range compared with the magnitude of the parameters. One method that may have helped is using a $\sin$ or $\cos$ basis to encode the distance.

I then investigated computing a loss term by multiplying the predicted probabilities over bins (exponentiated and normalised logits) by the bin centres to act as an expectation over the distance categories. This seemed to overfit to the example fairly well. However, upon closer inspection it was not learning as intended. When looking at the predicted probabilities, it would not cluster high probability around the correct distance bin as intended. Instead it would place probability over a range of bins that produced a good averaged prediction. To stop this I added an entropy term to the loss. However, this caused to model to place probability mass on the first and last distance bin, pulling the predicted distance from each side. I finally tried to replace the entropy loss term with a term which summed all probabilities but the maximising one, however, this was unnoticeably different to the entropy version.

Finally, I created the following method, inspired by the modalities in the first and last bin of the entropy bin-centre method, to consist of a head which predicted two logits instead of 64 for each distance. Then instead of only selecting a single bin centre over all logits, or computing the expectation over bin centres, it linearly interpolates the first and last break (minimum and maximum length). This was significantly faster at overfitting to a small experimental dataset of five examples than the original loss. However, it would get stuck on certain regions and take a long time to correct them, see Figure 7.3.3. Explicitly the distogram head does the following,

$$\begin{aligned}
\hat{\boldsymbol{l}}_{ij} &\leftarrow W\boldsymbol{z}_{ij} + \boldsymbol{b}, \\
\boldsymbol{p}_{ij} &\leftarrow \exp\left(\hat{\boldsymbol{l}}_{ij} + \hat{\boldsymbol{l}}_{ji}\right), \\
\hat{d}_{ij} &\leftarrow \frac{b_0 p_{1ij} + b_{-1} p_{2ij}}{p_{1ij} + p_{2ij}}
\end{aligned}$$

where b_0, b_{-1} are the first and last breaks (min and max distances) and $W \in \mathbb{R}^{2\times c_m}, \boldsymbol{b} \in \mathbb{R}^2$ are the parameters of a linear layer. I call this the delta-distogram head, since interpolation of two distances mimics the arms of a delta-robot. It may be possible to improve this method using a

sin or cos basis over the distance range.

To compute the error I chose to use the MSE loss between distograms, computing the square error as the difference in distograms and averaging over over the pairwise sequence dimensions. Prior to computing this loss I clamp the true distance matrix to be within the first and last break and compute a weighting w_{ij} over i, j as follows,

$$
\begin{aligned}
D_{ij} &= ||\boldsymbol{x}_i - \boldsymbol{x}_j||, \\
n_{\text{over}} &= \sum_{i,j} \mathbb{I}(D_{ij} > b_{-1}), \\
w_{ij} &= \begin{cases} 1 - \frac{n_{\text{over}}}{N^2}, & \text{if } D_{ij} > b_{-1} \\ \max(0.01, \frac{n_{\text{over}}}{N^2}), & \text{otherwise} \end{cases} \\
d_{ij} &= \min\big(b_0, \max\big(b_{-1}, D_{ij}\big)\big), \\
L &= \frac{1}{N^2} \sum_{i,j=1}^{N} m_i m_j w_{ij} (d_{ij} - \hat{d}_{ij})^2
\end{aligned}
$$

where m_i is a mask for that coordinate, 1 if it is present, else 0. Similarly the $\mathbb{I}$ is the indicator function which gives 1 if the predicate in the argument is true else 0. The w_{ij} term above is a re-weighting term which is motivated by the observation that the proportion of distogram elements which are clamped from above increases roughly quadratically with sequence length. This means that the network can easily achieve low loss on long sequences by predicting all at the maximum break. Decreasing the weight of the ground truth elements at the maximum break forces the network to work on the more important close distances.

7.3.1. Progressive Training

Using only the delta-distogram loss in the previous section I train the a model consisting of the standard Evoformer with 8 blocks and embedding modules for MSA 2 blocks, sequence bases and relative positions, this corresponds to Architecture-3 in Listing 7.3. The sequence bases and MSA embedding layers are adapted to have the appropriate number of tokens for RNA. I also changed the first and last break from 2Å and 22Å to 3Å and 35Å to account for the larger backbone in RNA than in proteins. I also set all dropouts to 0, as the first important task is to overfit training examples.

I perform the initial training as follows, where training examples are drawn uniformly at random.

- Train for 10^5 steps on sequences longer than 8 and a crop size of 40.
- Continue training for 3.6×10^5 steps on sequences longer than 16 and a crop size of 80.
- Continue training for 1.2×10^5 steps on sequences longer than 24 and a crop size of 160.
- Load up parameter from the previous step and augment architecture by adding 4 Evoformer blocks. Train only the new parameters for 6×10^4 steps on sequences longer than 24 and a crop size of 160. I refer to this model as 'disto-pretrained'.

The delta distogram loss quickly trained the model to be roughly correct. A selection of predicted distograms for the disto-pretrained model are shown in Figure 7.3.5.

I then attempted to reintroduce the original distogram loss by training only the original distogram head for 2×10^4 steps however this did not seem to work so I reverted back to the delta distogram loss.

Next I add in the structure module, and this makes the final architecture for the remainder of training, see Listing 7.4. I also set the all dropout modules in the structure module to 0. I then resumed training, still with 0 recycling iterations. I set the loss weights to 1 on the delta-distogram loss and 1.5 on the structure module backbone loss.

- Train for 1.2×10^5 steps on sequences longer than 16 and a crop size of 120.
- Train for 10^5 steps on sequences longer than 24 and a crop size of 160.
- Add in recycling, randomly selecting 0 or 1 recycling iteration. Train for 10^5 steps on sequences longer than 24 and a crop size of 160.

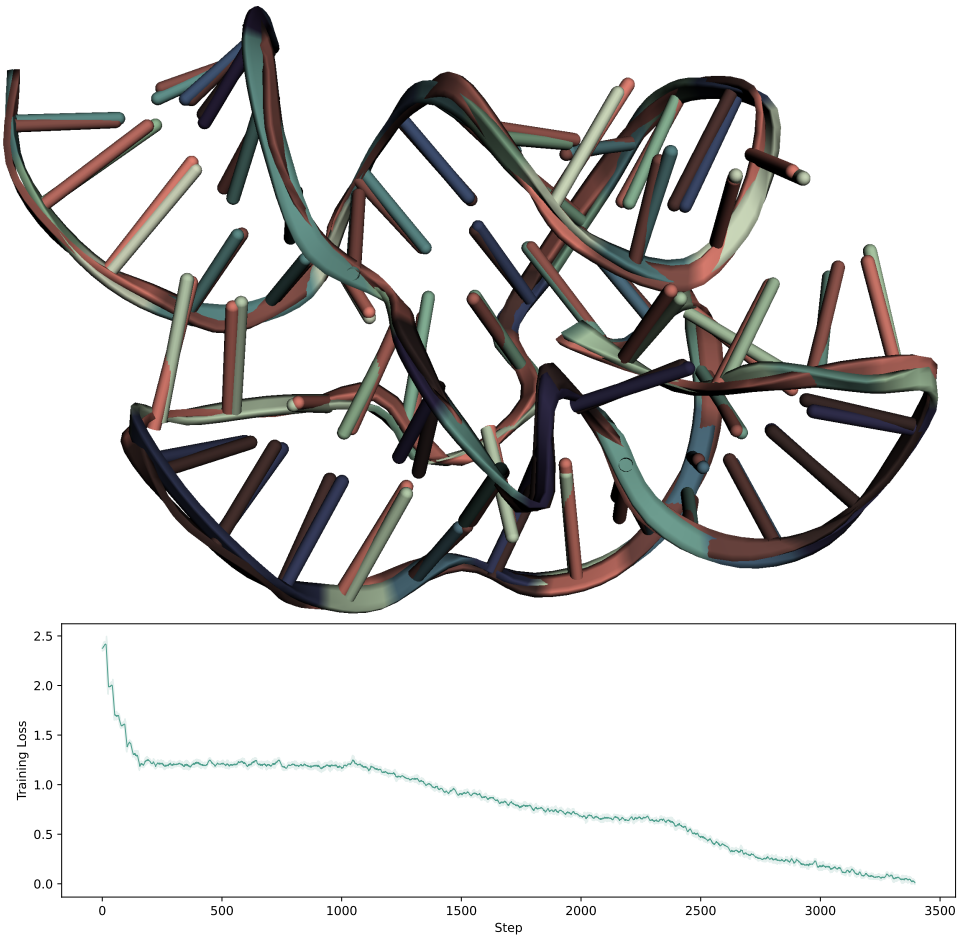


Figure 7.3.4: This is the RNA chain 1MMS-C, visualised in green is the true structure, in red is the prediction of model trained after setting the distogram loss to 0. This model is trained only on this example for 3000 gradient steps using a learning rate of 10^{-3}. This experiment was done to test that it could overfit successfully. The structure model loss curve is shown on the bottom, it has been smoothed with a window of 100 iterations and standard deviations calculated for each window.

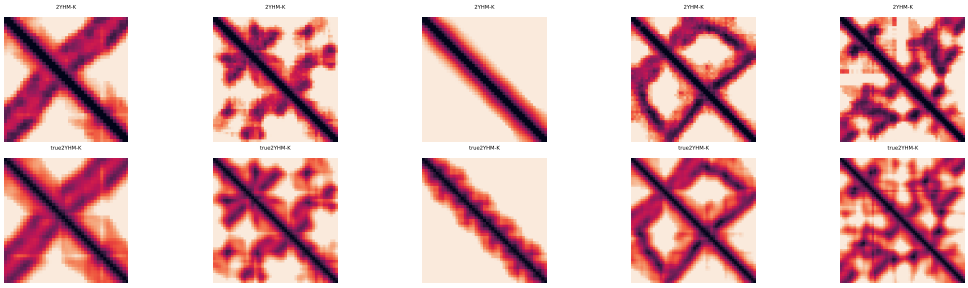


Figure **7.3.5.a:** These examples are from the training set.

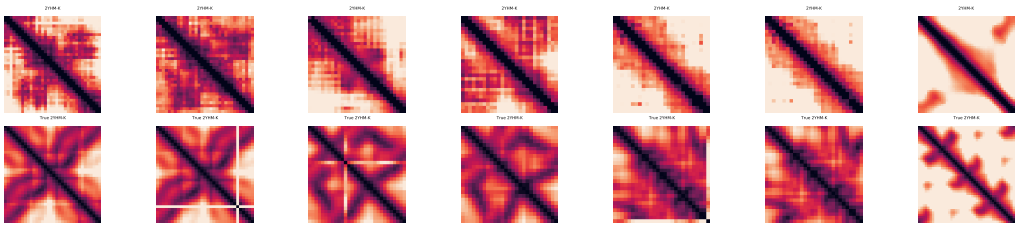


Figure **7.3.5.b:** These examples are from the validation set.

Figure 7.3.5: A selection of the Evoformer output distograms from the disto-pretrained model, the predictions are on the top row and the true clipped distograms are on the bottom row.

- Recycle with 0,1,2 or 3 iterations. Train for 10^5 steps on sequences longer than 24 and a crop size of 160. I refer to the model as 'part-trained'

After suffering from slow learning again I decided to perform more single example overfitting experiments. I took a single chain, 1MMS-C, and ran numerous loss weightings and different learning rates. Interestingly performance was very poor when the distogram loss had non-zero weight, I hypothesise that because it it much closer to the majority of the parameters in the computational graph it would produce the greatest change in the parameters corrupting the learning from the structure module loss. However, I did eventually set this to 0, the model then managed to perfectly overfit to a single example, see Figure 7.3.4. The model was trained by randomly selecting the number of recycling iterations uniformly from $\{0, 1, 2\}$ for each training iteration.

The final experiment I ran was initialised from the disto-pretrained model, with the structure module. The loss for the delta-distogram is set to 0 and the loss for the structure module is set to 1.5, the learning rate is set to 10^{-3} and II reduce the crop size to 60. Training examples are served for a random number of recycling iterations sampled uniformly from $\{0, 1, 2\}$ for each training iteration. This experiment had to be cut short due as I ran out of time, I refer to this model as 'final-model'. The model had not managed to overfit the training set yet. I present some of the training predictions for the final-model in Figure 7.3.6.a and validation predictions in Figure 7.3.6.b. The model is clearly overfitting the training set very well in most cases. The example where it performs poorly are the sequences longer than the final crop size of 60, this is because it will see each possible crop less often than an un-cropped example. The metrics are shown in Figure 7.3.7.

7.4. RECOMMENDED TRAINING PROCEDURE

To summarise my experiments, I would recommend initially training a structure prediction neural network to overfit a single example, then the whole training set, then focus on reducing overfitting to maximise validation performance. From my experiments I would suggest the following:

- Perform hyperparameter tuning to get the network to fit a single example as quickly as possible. Things to tune would be learning rate, schedulers, batch sizes, recycling or any other model hyperparameters such as embedding/latent dimensions and number of layers.

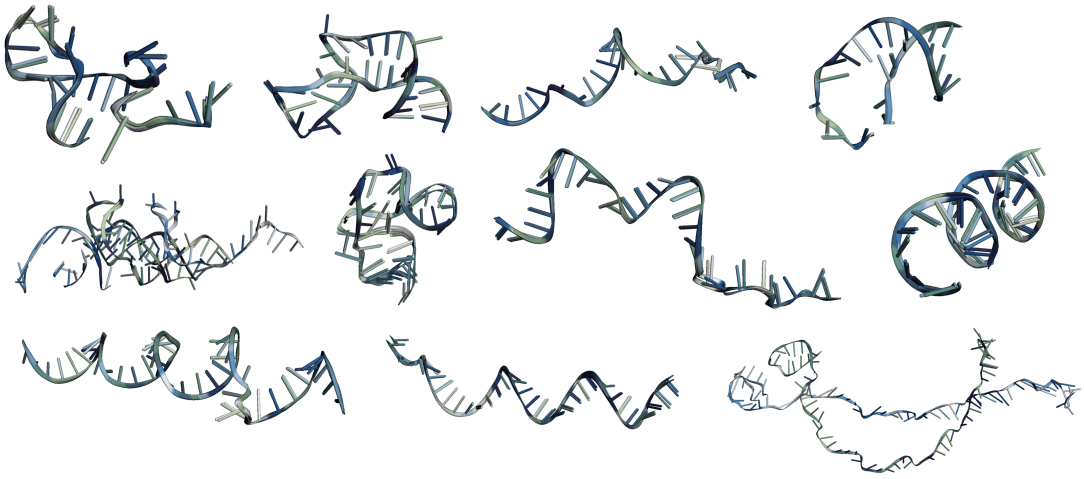


Figure **7.3.6.a:** These examples are from the training-set. The predictions are shown in blue and the true structure is shown in green. Note that most predictions are nearly indistinguishable from the ground truth. From left-to-right, top-to-bottom the codes are 2BH2-D, 2OZB-C, 3R9W-B, 6F4G-C, 2QWY-C, 6SY6-D, 3R9W-B, 6VFF-C, 5UNE-B, 7KRN-P, 5H9F-L.

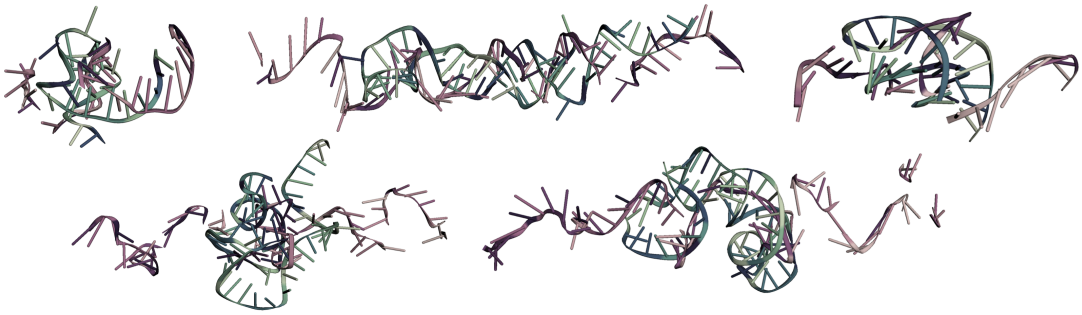


Figure **7.3.6.b:** These examples are from the validation-set. The predictions are shown in pink and the true structure is shown in green. From left-to-right, top-to-bottom the codes are 6V9B-B, 5AOX-C, 3K1V-A, 7OO7Y-B7, 4Y40-1B.

Figure 7.3.6: Visualisations of the many of the model predictions from the final-model, the predictions are superimposed onto the true structure using the Kabsch algorithm [Kabsch, 1976] which minimises RMSD over a rigid transformation, the per-base origin to atom vectors are re-inserted from the ground truth.

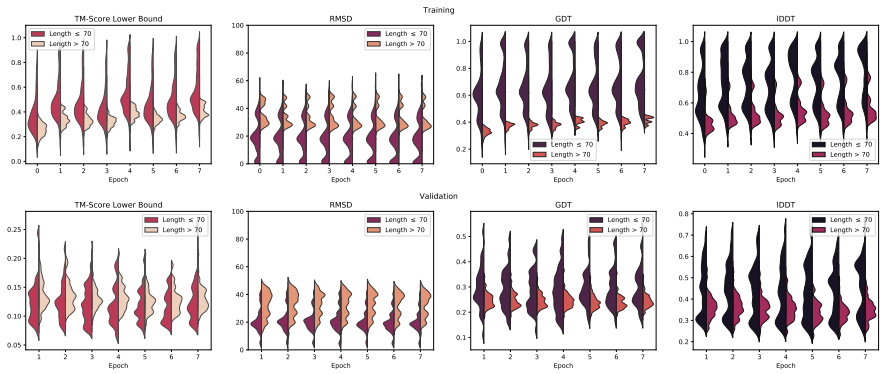


Figure 7.3.7: This experiment corresponds to the final-model. The top row of plots are the metrics for a subset of training examples (the subset is randomly picked before experiments and kept constant). The final row is of validation metrics. The violin plots are normalised by maximum mode (width), not by area or counts.

It is necessary to not move on until this step is completed, a likely cause could be lack of network capacity or a badly designed architecture. Perform variations of different loss weights if there are auxiliary losses. Pre-train with certain losses, then turn them off and transfer the learnt parameters. Make sure to update a display of the predictions as the model trains so that it is clear how training is going.

- Work on overfitting the whole dataset. A powerful technique is to progressively increase the difficulty of the problem with the network capacity if possible. In this case starting with short sequences made for simpler problems to solve, and increasing the number of layers gave the network more freedom to overfit the increasingly complex data.

I did not have time to experiment further with fully overfitting the training data, and then maximising validation performance. I will outline my plan of how I would have proceeded to maximise the validation performance.

- Ensure the training procedure automatically saves the parameters that achieve lowest validation accuracy.
- Use the hyperparameter settings and model architecture that produced the fastest overfitting of the training set.
- Add in dropout to linear layers of the network possibly starting with a small dropout probability, then increasing it to a maximum value.
- Use weight decay to bias to model to simpler functions.
- Perform more aggressive data augmentation, for example greater MSA corruption probability for masking and replacing.
- Ensemble.

A powerful neural network training checklist is given in Andrej Karpathy's blog [Karpathy, 2019].

8 | Applications and Future Work

8.1. LIMITATIONS AND IMPROVEMENTS FOR MY MODEL

The model developed in this project is a minimal implementation of AlphaFold, the most natural next addition would be the RNA equivalent for all atom prediction. This would then allow for easily adding in the violation loss. A slightly more challenging addition would be the torsion angle heads as RNA has more degrees of freedom than proteins. Additionally, the Amber force field would likely boost performance significantly.

A limitation of the model is the relatively short crop size, or even cropping sequences at all. Cropping sequences is not an ideal way to frame the problem of structure prediction, increasing the crop size would help with this issue. Alternatively developing sparse alternative to the pairwise representations could scale the crop size significantly. This would be extremely challenging though as the pairwise latents are updated throughout the whole model through residual connections. Improving the complexity of attention could involve subquatratic clustering over the set of keys and/or queries to then perform approximate attention.

Another limitation is that passing in multiple sequences in the same complex is not representative of the problem folding problem, and it is surprising that AlphaFold performs so well without this crucial context. AlphaFold is essentially an approximate kernel machine over the structures in the Protein Data Bank.

8.2. POSSIBLE RESEARCH AREAS TO IMPROVE ALPHAFOLD

In the case of multiple chains in the same complex, it may make sense to pass multiple chains through at once. Where only one chain is predicted, but the atom positions of the other chain are given as features. Alternatively, it could pass multiple examples through the Evoformer separately, then into the Structure Modules at the same time. Otherwise, upon recycling the next iteration it could reason about the other chains in the complex.

Given a related protein task it may be useful to use the embedding layers of AlphaFold to as input features. For example, embed two proteins then design a model to see how they interact. Another example could be a generative algorithm to sample possible molecular dynamics trajectories.

The properties of noisy student self-distillation may cause a bias in the predictions. Future work could investigate how predictions may drift and how well the model can generalise. For example if the process of expanding the dataset using high confidence examples kept repeating, investigating how the confidence changes over these iterations and what effect this has on the predictions vs the ground truth.

A way to scale to greater chain lengths, or multiple chains could be to develop local attention for point cloud data. So for each point, randomly sample other points and keep with probability proportional to their closeness. A neural network architecture which may lend itself well to processing information in a similar way to the structure module is an SE3 transformer [Fuchs et al., 2020].

Including physics-based simulation within the forward pass could be an option. For example between recycling iterations a structure could be relaxed. Otherwise, as the structure module iteratively refines the position of the frames in the residue gas, a physics simulator could compute the forces on each atom and use that as features to embeddings to inform the intrinsic folding process.

To produce a diverse MSA it may be possible to define and sample [1]Determinantal Point Process (DPP) [Kulesza, 2012] defined on phylogenetic tree over MSA alignment sequences, this would produce a rich set of sequences to be used as features.

8.3. PROTEIN-PROTEIN COMPLEX MODELING

Cellular processes rely on protein–protein interactions. Determining the structure of protein–protein complexes is a key step to understand their molecular mechanisms. Given an understanding of the mechanisms it may be possible to develop drugs targeting the protein–protein interactions. To approach this problem in the same way as AlphaFold requires joint multiple sequence alignments of two interacting proteins. Inter-protein contact prediction is limited due to the fact that the MSA must search over homologous pairs of sequence involved in that interaction .

The model DeepHomo is a deep learning model to predict inter-protein residue–residue contacts across homo-oligomeric protein interfaces [Yan and Huang, 2021]. This integrates the intra-protein distance map and inter-protein docking pattern, in addition to evolutionary coupling, sequence conservation, and physico-chemical information of monomers. DeepHomo outperformed state-of-the-art direct-coupling analysis and machine learning-based approaches at CASP14.

8.4. PROTEIN MOTION

Recovering the continuous motion and detailed 3D structure of flexible proteins from cryo-EM data is a difficult challenge. 3DFlex [Punjani and Fleet, 2021] is a motion-based deep neural network model which jointly learns a single canonical 3D map, latent vectors in the protein's conformational landscape, and a flow generator that, given a latent position as input, outputs a 3D deformation field, all from 2D image data. This deformation field convects the canonical map into appropriate conformations to explain experimental images. The model learns non-rigid motion over several orders of magnitude preserving details of secondary structure elements.

8.5. PROTEIN AND RNA DESIGN

One of the ultimate applications of advancements in protein structure prediction is to be able to be able to design a biomolecule capable of a certain function [Kuhlman and Bradley, 2019]. Reinforcement learning has been used to design molecules with reward functions based on fundamental physical properties such as the energy, approximated with fast quantum-chemical methods [Simm et al., 2020]. A framework for protein design using a biologically-grounded reward function, sequential action-space formulation, and policy gradient reinforcement learning algorithms is shown in [Skwark et al., 2020]. The authors show how this can be done with the example of a novel variant of the human angiotensin-converting enzyme 2 (ACE2). This enzyme binds to the SARS-CoV-2 spike protein and diverts it from human cells.

The state-of-the-art model for reaction prediction is the Molecular Transformer [Schwaller et al., 2019] which uses a text-based representation, simplified molecular-input line-entry system (SMILES) strings of reactants, reagents, and the products. In [Kovács et al., 2021] they interpret the model as it is unclear if the correct predictions are made because the model inferred the salient chemistry, or which training example they are relying on to reach a prediction. They develop a framework to attribute predicted reaction outcomes both to specific parts of reactants, and to reactions in the training set.

8.5.1. Drug Design

An example of how geometric deep learning can be used to improve structure-based drug design is shown in [Méndez-Lucio et al., 2021] predicting the binding conformations of ligands to protein targets. A potential based on distance likelihood for ligand-target pairs is shown to perform

[1]A DPP is a probability distribution defined over the superset of a set of items. It samples a subset of items accounting for all pairwise similarities, such that it samples a diverse set of items

similar or better than well-established scoring functions for docking and screening tasks. Global optimisation algorithms can then reproduce experimental binding conformations of ligands.

In a blog post Mohammed AlQuraishi [AlQuraishi, 2020] addresses the topic of how AlphaFold will affect drug discovery. He notes that if crystallography were to become fast and routine, it would still not fundamentally alter the dynamics of drug discovery as it is practiced today, since most of the cost is in the later stages of drug development beyond medicinal chemistry and well into biology and physiology. AlphaFold can help is in determining structures of protein targets that can be modulated for therapeutic purposes. AlphaFold is trained to predict apo (unbound) protein structures, however most medicinal chemistry applications require complexes of the protein bound to a small molecule. Often sub-angstrom resolution is necessary, which AlphaFold is not trained at that resolution, partly because training data is not that accurate. An AlphaFold based system may be capable of designing protein-based therapeutics (antibodies and peptides). AlphaFold may be able to play a role in a system where drugs are designed for their polypharmacology, intentionally modulate multiple protein targets. Drugs with designed polypharmacology could be capable of modulating entire signaling pathways instead of acting on one protein at a time.

8.5.2. Catalyst Design

The OC20 dataset [Chanussot et al., 2021] has been created to benchmark machine learning algorithms to develop algorithms for catalyst discovery and optimisation. They tested state of the art graph neural network algorithms CGCNN, SchNet, DimeNet++. The dataset contains a million Density Functional Theory (DFT) relaxations (200 million single point evaluations) across many materials, surfaces, and adsorbates. Additionally, randomly perturbed structures, short timescale molecular dynamics, and electronic structure analyses are included. One example of a problem that falls into this category is scalable and cost-effective solutions to renewable energy storage. Increasing our reliance on renewable energy sources such as wind and solar, which produce intermittent power requires storage to transfer power from times of peak generation to peak demand. One solution is converting renewable energy to other fuels, e.g. hydrogen or methane. In [Zitnick et al., 2020] they use the OC20 dataset to train machine learning models for this task.

9 | Conclusion

In this project I have explored the state of the art machine models for protein and RNA structure prediction. The main focus has been on the recently open-sourced AlphaFold [Jumper et al., 2021] and how it is changing the field. Unfortunately I could not complete my full plan for this project. The reasons for this is partly because AlphaFold was released half way through my project, not giving much time to act in response to it. I converted the majority of the AlphaFold JAX-Haiku repository to PyTorch code, as this is a large repository this took a substantial amount of time to test and convert the code. The prototyping and training of models was time consuming as it is a large model to run, even on a GPU. Nevertheless, I did manage to make an important step in training and overfit a single example perfectly, a proof-of-concept for overfitting the whole dataset. Additionally, I made progress on overfitting the full dataset, and this would likely have occurred given more training time. In the process I developed methods to train large models on the task of structure prediction. In particular, alternating auxiliary losses when training and progressively increasing the difficulty of training examples while simultaneously increasing model capacity. However, I did not make any developments on the significantly more challenging final step of getting strong validation performance, while only training on a small dataset. This would most likely have required magnitudes more compute power and time to account for dropout in particular.

The project also provided background of why the structure prediction task is useful, by describing the history of the challenging field of experimentally resolving protein and RNA structures. I discussed other methods for structure prediction, as well as focusing on adjacent fields such as physics-based simulation for the protein and RNA folding problems. Given the exciting recent progress from AlphaFold, these fields will likely surge forward at unprecedented speeds. I covered the machine learning fundamental including statistical learning theory and more recent research in overfitting in deep learning. I covered AlphaFold in depth and how to adapt it to RNA. My lab partner and I contributed a clear dataset pipeline to construct an RNA dataset, including performing multiple sequence alignment search and parsing almost all fields recorded in PDB files. I created a training framework for structure prediction and performed a number of revealing experiments. The future work section touched on a number of exciting fields that are becoming more popular, coming into the limelight as the structure prediction community moves on to the next challenge.

Bibliography

B. Adhikari and J. Cheng. Confold2: improved contact-driven ab initio protein structure modeling. *BMC bioinformatics*, 19(1):1–5, 2018.

M. Akiyama, K. Sato, and Y. Sakakibara. A max-margin training of rna secondary structure prediction integrated with the thermodynamic model. *bioRxiv*, 2017. doi: 10.1101/205047. URL https://www.biorxiv.org/content/early/2017/10/18/205047.

A. S. A. Almakarem, A. I. Petrov, J. Stombaugh, C. L. Zirbel, and N. B. Leontis. Comprehensive survey and geometric classification of base triples in RNA structures. *Nucleic Acids Research*, 40(4):1407–1423, Nov. 2011. doi: 10.1093/nar/gkr810. URL https://doi.org/10.1093/nar/gkr810.

M. AlQuraishi. End-to-end differentiable learning of protein structure. *Cell Systems*, 8(4): 292–301.e3, 2019a. ISSN 2405-4712. doi: https://doi.org/10.1016/j.cels.2019.03.006. URL https://www.sciencedirect.com/science/article/pii/S2405471219300766.

M. AlQuraishi. Proteinnet: a standardized data set for machine learning of protein structure. *BMC Bioinformatics*, 20(1):311, Jun 2019b. ISSN 1471-2105. doi: 10.1186/s12859-019-2932-0. URL https://doi.org/10.1186/s12859-019-2932-0.

M. AlQuraishi. Alphafold2 @ casp14: "it feels like one's child has left home.", 2020. URL https://moalquraishi.wordpress.com/2020/12/08/alphafold2-casp14-it-feels-like-ones-child-has-left-home/#s4.4.

A. Amunts, July 2021. URL https://twitter.com/A_Amunts/status/1418452561779968.

M. Andronescu, A. Condon, H. Hoos, D. Mathews, and K. Murphy. Efficient parameter estimation for rna secondary structure prediction. *Bioinformatics (Oxford, England)*, 23:i19–28, 08 2007. doi: 10.1093/bioinformatics/btm223.

M. Andronescu, A. Condon, H. H. Hoos, D. H. Mathews, and K. P. Murphy. Computational approaches for RNA energy parameter estimation. *RNA*, 16(12):2304–2318, Oct. 2010. doi: 10.1261/rna.1950510. URL https://doi.org/10.1261/rna.1950510.

B. Apolloni, L. Lotorto, A. Morpurgo, and A. Zanaboni. Rna secondary structure prediction by mft neural networks.

P. R. Arantes, M. Depólo Polêto, C. Pedebos, and R. Ligabue-Braun. Making it rain: cloud-based molecular simulations for everyone. *ChemRxiv*, 2021. doi: 10.33774/chemrxiv-2021-9f2m5.

W. T. Astbury and F. O. Bell. Some recent developments in the x-ray study of proteins and related structures. In *Cold Spring Harbor Symposia on Quantitative Biology*, volume 6, pages 109–121. Cold Spring Harbor Laboratory Press, 1938.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.

A. Bacle, P. Buslaev, R. Garcia-Fandino, F. Favela-Rosales, T. Mendes Ferreira, P. F. J. Fuchs, I. Gushchin, M. Javanainen, A. M. Kiirikki, J. J. Madsen, J. Melcr, P. Milán Rodríguez, M. S. Miettinen, O. H. S. Ollila, C. G. Papadopoulos, A. Peón, T. J. Piggot, Á. Piñeiro, and S. I. Virtanen. Inverse conformational selection in lipid–protein binding. *Journal of the American Chemical Society*, 143(34):13701–13709, Sep 2021. ISSN 0002-7863. doi: 10.1021/jacs.1c05549. URL https://doi.org/10.1021/jacs.1c05549.

M. Baek, F. DiMaio, I. Anishchenko, J. Dauparas, S. Ovchinnikov, G. R. Lee, J. Wang, Q. Cong, L. N. Kinch, R. D. Schaeffer, C. Millán, H. Park, C. Adams, C. R. Glassman, A. DeGiovanni, J. H. Pereira, A. V. Rodrigues, A. A. van Dijk, A. C. Ebrecht, D. J. Opperman, T. Sagmeister, C. Buhlheller, T. Pavkov-Keller, M. K. Rathinaswamy, U. Dalwadi, C. K. Yip, J. E. Burke, K. C. Garcia, N. V. Grishin, P. D. Adams, R. J. Read, and D. Baker. Accurate prediction of protein structures and interactions using a three-track neural network. *Science*, 2021. URL http://science.sciencemag.org/content/early/2021/07/14/science.abj8754https://www.ipd.uw.edu/wp-content/uploads/2021/07/Baek_etal_Science2021_RoseTTAFold.pdf.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2016.

M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine learning practice and the bias-variance trade-off, 2019.

S. Bellaousov and D. H. Mathews. ProbKnot: Fast prediction of RNA secondary structure including pseudoknots. *RNA*, 16(10):1870–1880, Aug. 2010. doi: 10.1261/rna.2125310. URL https://doi.org/10.1261/rna.2125310.

I. Bello. Lambdanetworks: Modeling long-range interactions without attention, 2021.

B. J. Bender, A. Cisneros, A. M. Duran, J. A. Finn, D. Fu, A. D. Lokits, B. K. Mueller, A. K. Sangha, M. F. Sauer, A. M. Sevy, G. Sliwoski, J. H. Sheehan, F. DiMaio, J. Meiler, and R. Moretti. Protocols for molecular modeling with rosetta3 and rosettascripts. *Biochemistry*, 55(34):4748–4763, Aug 2016. ISSN 0006-2960. doi: 10.1021/acs.biochem.6b00444. URL https://doi.org/10.1021/acs.biochem.6b00444.

T. Bepler and B. Berger. Learning the protein language: Evolution, structure, and function. *Cell Systems*, 12(6):654–669.e3, 2021. ISSN 2405-4712. doi: https://doi.org/10.1016/j.cels.2021.05.017. URL https://www.sciencedirect.com/science/article/pii/S2405471221002039.

H. Berendsen, D. van der Spoel, and R. van Drunen. Gromacs: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 91(1):43–56, 1995. ISSN 0010-4655. doi: https://doi.org/10.1016/0010-4655(95)00042-E. URL https://www.sciencedirect.com/science/article/pii/001046559500042E.

H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank, nucleic acids research. *Nucleic Acids Res.*, 2000. URL https://www.rcsb.org/.

J. D. Bernal and D. Crowfoot. X-ray photographs of crystalline pepsin. *Nature*, 133(3369):794–795, 1934.

J. Bijvoet. Structure of optically active compounds in the solid state. *Nature*, 173(4411):888–891, 1954.

E. BINDEWALD. RNA secondary structure prediction from sequence alignments using a network of k-nearest neighbor classifiers. *RNA*, 12(3):342–352, Mar. 2006. doi: 10.1261/rna.2164906. URL https://doi.org/10.1261/rna.2164906.

L. C. Blum and J.-L. Reymond. 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *J. Am. Chem. Soc.*, 131:8732, 2009.

L. V. Bock, M. H. Kolář, and H. Grubmüller. Molecular simulations of the ribosome and associated translation factors. *Current Opinion in Structural Biology*, 49:27–35, 2018. ISSN 0959-440X. doi: https://doi.org/10.1016/j.sbi.2017.11.003. URL https://www.sciencedirect.com/science/article/pii/S0959440X1730132X. Theory and simulation • Macromolecular assemblies.

M. J. Boniecki, G. Lach, W. K. Dawson, K. Tomala, P. Lukasz, T. Soltysinski, K. M. Rother, and J. M. Bujnicki. SimRNA: a coarse-grained method for RNA folding simulations and 3d structure prediction. *Nucleic Acids Research*, 44(7):e63–e63, Dec. 2015. doi: 10.1093/nar/gkv1479. URL https://doi.org/10.1093/nar/gkv1479.

K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, and D. E. Shaw. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, page 84–es, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 0769527000. doi: 10.1145/1188455.1188544. URL https://doi.org/10.1145/1188455.1188544.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

W. H. Bragg. Xcvii. the intensity of reflexion of x rays by crystals. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 27(161):881–899, 1914.

W. H. Bragg. Ix. bakerian lecture.—x-rays and crystal structure. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 215(523-537):253–274, 1915.

J. C. Brooks-Bartlett and E. F. Garman. The nobel science: One hundred years of crystallography. *Interdisciplinary Science Reviews*, 40(3):244–264, 2015a. doi: 10.1179/0308018815Z.000000000116. URL https://doi.org/10.1179/0308018815Z.000000000116.

J. C. Brooks-Bartlett and E. F. Garman. The nobel science: one hundred years of crystallography. *Interdisciplinary Science Reviews*, 40(3):244–264, 2015b.

S. Brown and T. Head-Gordon. Intermediates and the folding of proteins l and g. *Protein science : a publication of the Protein Society*, 13(4):958–970, Apr 2004. ISSN 0961-8368. doi: 10.1110/ps.03316004. URL https://pubmed.ncbi.nlm.nih.gov/15044729. 15044729[pmid].

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

A. T. Brunger. Version 1.2 of the crystallography and nmr system. *Nature Protocols*, 2(11): 2728–2733, Nov 2007. ISSN 1750-2799. doi: 10.1038/nprot.2007.406. URL https://doi.org/10.1038/nprot.2007.406.

P. Bryant, G. Pozzati, and A. Elofsson. Improved prediction of protein-protein interactions using alphafold2 and extended multiple-sequence alignments. *bioRxiv*, 2021. doi: 10.1101/2021.09.15.460468. URL https://www.biorxiv.org/content/early/2021/09/15/2021.09.15.460468.

N. Calonaci, A. Jones, F. Cuturello, M. Sattler, and G. Bussi. Machine learning a model for RNA structure prediction. *NAR Genomics and Bioinformatics*, 2(4), Nov. 2020. doi: 10.1093/nargab/lqaa090. URL https://doi.org/10.1093/nargab/lqaa090.

E. P. Carpenter, K. Beis, A. D. Cameron, and S. Iwata. Overcoming the challenges of membrane protein crystallography. *Current opinion in structural biology*, 18(5):581–586, 2008.

N. Carranza and M. Skwark. Deepchain, 2020. URL https://deepchain.bio/.

D. A. Case, T. E. Cheatham III, T. Darden, H. Gohlke, R. Luo, K. M. Merz Jr., A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods. The amber biomolecular simulation programs. *Journal of Computational Chemistry*, 26(16):1668–1688, 2005. doi: https://doi.org/10.1002/jcc.20290. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.20290.

L. Chanussot, A. Das, S. Goyal, T. Lavril, M. Shuaibi, M. Riviere, K. Tran, J. Heras-Domingo, C. Ho, W. Hu, and et al. Open catalyst 2020 (oc20) dataset and community challenges. *ACS Catalysis*, 11(10):6059–6072, May 2021. ISSN 2155-5435. doi: 10.1021/acscatal.0c04525. URL http://dx.doi.org/10.1021/acscatal.0c04525.

X. Chen, Y. Li, R. Umarov, X. Gao, and L. Song. Rna secondary structure prediction by learning unrolled algorithms, 2020.

P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and M. J. L. de Hoon. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 03 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp163. URL https://doi.org/10.1093/bioinformatics/btp163.

T. S. Cohen and M. Welling. Steerable cnns, 2016.

R. Consortium. RNAcentral 2021: secondary structure integration, improved sequence search and new member databases. *Nucleic Acids Research*, 49(D1):D212–D220, 10 2020. ISSN 0305-1048. doi: 10.1093/nar/gkaa921. URL https://doi.org/10.1093/nar/gkaa921.

C. Cooper, July 2021. URL https://twitter.com/huddscooperlab/status/1418258246507315206.

E. Coutsias, C. Seok, and K. Dill. Using quaternions to calculate rmsd. *J Comput Chem.*, 2004. URL https://pubmed.ncbi.nlm.nih.gov/15376254/.

R. Crowther and A. Klug. Structural analysis of macromolecular assemblies by image reconstruction from electron micrographs. *Annual review of biochemistry*, 44(1):161–182, 1975.

C. D, A. H, B. K, B.-S. I, B. S, C. D, I. T, Cheatham, C. G, C. V, D. T, D. R, G. Giambasu, G. M, H. Gohlke, G. A, H. R, I. S, I. S, J. C, K. K, K. M, K. E, K. A, K. T, L. T, S. LeGrand, L. P, L. C, L. J, L. T, L. R, M. M, M. V, M. M, M. K, M. Y, M. O, M. G, N. H, O. K, O. A, P. F, P. S, Q. R, R. A, R. D, R. A, S. C, S.-V. S, S. J, S. C, S. N, S. J, S. J, W. R, W. J, W. H, W. R, W. X, X. Y, Y. D, Z. S, and P. Kollman. Amber, 2021. URL https://ambermd.org/index.php.

C. G. Darwin. Lxxviii. the theory of x-ray reflexion. part ii. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 27(160):675–690, 1914a.

C. G. Darwin. Xxxiv. the theory of x-ray reflexion. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 27(158):315–333, 1914b.

R. Das and D. Baker. Automated de novo prediction of native-like rna tertiary structures. *Proceedings of the National Academy of Sciences*, 104(37):14664–14669, 2007. ISSN 0027-8424. doi: 10.1073/pnas.0703836104. URL https://www.pnas.org/content/104/37/14664.

M. Davies. Peter debye (1884-1966): a centenary appreciation. *The Journal of Physical Chemistry*, 88(26):6461–6462, 1984.

J. Deisenhofer, O. Epp, K. Miki, R. Huber, and H. Michel. Structure of the protein subunits in the photosynthetic reaction centre of rhodopseudomonas viridis at 3å resolution. *Nature*, 318 (6047):618–624, 1985.

R. Delli Ponti, S. Marti, A. Armaos, and G. G. Tartaglia. A high-throughput approach to profile rna structure. *Nucleic acids research*, 45(5):e35–e35, 2017.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

C. B. Do, D. A. Woods, and S. Batzoglou. CONTRAfold: RNA secondary structure prediction without physics-based models. *Bioinformatics*, 22(14):e90–e98, 07 2006. ISSN 1367-4803. doi: 10.1093/bioinformatics/btl246. URL https://doi.org/10.1093/bioinformatics/btl246.

P. Domingos. Every model learned by gradient descent is approximately a kernel machine, 2020.

M. Eckert. Disputed discovery: the beginnings of x-ray diffraction in crystals in 1912 and its repercussions. *Acta Crystallographica Section A: Foundations of Crystallography*, 68(1):30–39, 2012.

A. Elnaggar, M. Heinzinger, C. Dallago, G. Rihawi, Y. Wang, L. Jones, T. Gibbs, T. Feher, C. Angerer, M. Steinegger, D. Bhowmik, and B. Rost. Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing, 2021.

P. Ewald. Zur theorie der interferenzen der röntgenstrahlen in kristallen. *Physikalische Zeitschrift*, 14(11):465–472, 1913.

W. Falcon et al. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

C. Fang, J. Ding, Q. Huang, T. Tong, and Y. Sun. The overfitting iceberg, 2020. URL https://blog.ml.cmu.edu/2020/08/31/4-overfitting/.

D. Filmore. It's a gpcr world. *Modern Drug Discovery*, 7:24–28, 2004.

F. Fuchs. Alphafold 2 & equivariance, 2020. URL https://fabianfuchsml.github.io/alphafold2/.

F. B. Fuchs, D. E. Worrall, V. Fischer, and M. Welling. Se(3)-transformers: 3d roto-translation equivariant attention networks, 2020.

E. F. Garman. Developments in x-ray crystallographic structure determination of biological macromolecules. *Science*, 343(6175):1102–1108, 2014.

E. F. Garman and T. R. Schneider. Macromolecular cryocrystallography. *Journal of Applied Crystallography*, 30(3):211–237, 1997.

U. Ghani, I. Desta, A. Jindal, O. Khan, G. Jones, S. Kotelnikov, D. Padhorny, S. Vajda, and D. Kozakov. Improved docking of protein models by a combination of alphafold2 and cluspro. *bioRxiv*, 2021. doi: 10.1101/2021.09.07.459290. URL https://www.biorxiv.org/content/early/2021/09/07/2021.09.07.459290.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

T. Goddard, September 2021. URL https://rbvi.github.io/chimerax-recipes/big_alphafold/bigalpha.html.

J. G. Greener and D. T. Jones. Differentiable molecular simulation can learn all the parameters in a coarse-grained force field for proteins. *bioRxiv*, 2021. doi: 10.1101/2021.02.05.429941. URL https://www.biorxiv.org/content/early/2021/02/07/2021.02.05.429941.

J. G. Greener, S. M. Kandathil, and D. T. Jones. Deep learning extends de novo protein modelling coverage of genomes using iteratively predicted structural constraints. *Nature Communications*, 10(1):3977, Sep 2019. ISSN 2041-1723. doi: 10.1038/s41467-019-11994-0. URL https://doi.org/10.1038/s41467-019-11994-0.

P. Güntert, C. Mumenthaler, and K. Wüthrich. Torsion angle dynamics for nmr structure calculation with the new program dyana11edited by p. e. wright. *Journal of Molecular Biology*, 273 (1):283–298, 1997. ISSN 0022-2836. doi: https://doi.org/10.1006/jmbi.1997.1284. URL https://www.sciencedirect.com/science/article/pii/S0022283697912845.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. doi: 10.1109/ICCV.2015.123.

R. Henderson and P. Unwin. Molecular structure determination by electron microscopy of unstained crystalline samples. *J Mol Biol*, 94:425–440, 1975.

T. Hennigan, T. Cai, T. Norman, and I. Babuschkin. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

L. Heo and M. Feig. High-accuracy protein structures by combining machine-learning with physics-based refinement. *Proteins: Structure, Function, and Bioinformatics*, 88(5):637–642, 2020. doi: https://doi.org/10.1002/prot.25847. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/prot.25847.

P. W. Hildebrand, A. S. Rose, and J. K. Tiemann. Bringing molecular dynamics simulation data into view. *Trends in Biochemical Sciences*, 44(11):902–913, 2019. ISSN 0968-0004. doi: https://doi.org/10.1016/j.tibs.2019.06.004. URL https://www.sciencedirect.com/science/article/pii/S0968000419301379.

N. Hiranuma, H. Park, M. Baek, I. Anishchenko, J. Dauparas, and D. Baker. Improved protein structure refinement guided by deep learning based accuracy estimation. *Nature Communications*, 12(1):1340, Feb 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-21511-x. URL https://doi.org/10.1038/s41467-021-21511-x.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

C.-Y. Hor, C.-B. Yang, C.-H. Chang, C.-T. Tseng, and H.-H. Chen. A tool preference choice method for RNA secondary structure prediction by SVM with statistical tests. *Evolutionary Bioinformatics*, 9:EBO.S10580, Jan. 2013. doi: 10.4137/ebo.s10580. URL https://doi.org/10.4137/ebo.s10580.

A. Hospital, P. Andrio, C. Cugnasco, L. Codo, Y. Becerra, P. D. Dans, F. Battistini, J. Torres, R. Goñi, M. Orozco, and J. L. Gelpí. BIGNASim: a NoSQL database structure and analysis portal for nucleic acids simulation data. *Nucleic Acids Research*, 44(D1):D272–D278, 11 2015. ISSN 0305-1048. doi: 10.1093/nar/gkv1301. URL https://doi.org/10.1093/nar/gkv1301.

M. Hutchinson, C. L. Lan, S. Zaidi, E. Dupont, Y. W. Teh, and H. Kim. Lietransformer: Equivariant self-attention for lie groups, 2021.

A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira. Perceiver: General perception with iterative attention, 2021.

Y. Jian, X. Wang, J. Qiu, H. Wang, Z. Liu, Y. Zhao, and C. Zeng. Direct: Rna contact predictions by integrating structural patterns. *BMC Bioinformatics*, 20(1):497, Oct 2019. ISSN 1471-2105. doi: 10.1186/s12859-019-3099-4. URL https://doi.org/10.1186/s12859-019-3099-4.

D. T. Jones, D. W. A. Buchan, D. Cozzetto, and M. Pontil. PSICOV: precise structural contact prediction using sparse inverse covariance estimation on large multiple sequence alignments. *Bioinformatics*, 28(2):184–190, 11 2011. ISSN 1367-4803. doi: 10.1093/bioinformatics/btr638. URL https://doi.org/10.1093/bioinformatics/btr638.

J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 2021. doi: 10.1038/s41586-021-03819-2. (Accelerated article preview).

J. M. Jumper, K. F. Freed, and T. R. Sosnick. Trajectory-based parameterization of a coarse-grained forcefield for high-thoughput protein simulation. *bioRxiv*, 2017. doi: 10.1101/169326. URL https://www.biorxiv.org/content/early/2017/07/27/169326.

J. M. Jumper, N. F. Faruk, K. F. Freed, and T. R. Sosnick. Accurate calculation of side chain packing and free energy with applications to protein molecular dynamics. *PLOS Computational Biology*, 14(12):1–25, 12 2018. doi: 10.1371/journal.pcbi.1006342. URL https://doi.org/10.1371/journal.pcbi.1006342.

W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, 32(5):922–923, Sep 1976. doi: 10.1107/S0567739476001873. URL https://doi.org/10.1107/S0567739476001873.

I. Kalvari, E. P. Nawrocki, N. Ontiveros-Palacios, J. Argasinska, K. Lamkiewicz, M. Marz, S. Griffiths-Jones, C. Toffano-Nioche, D. Gautheret, Z. Weinberg, E. Rivas, S. R. Eddy, R. D. Finn, A. Bateman, and A. I. Petrov. Rfam 14: expanded coverage of metagenomic, viral and microrna families. *Nucleic Acids Research*, 49(D1):D192–D200, 11 2020. ISSN 0305-1048. doi: 10.1093/nar/gkaa1047. URL https://doi.org/10.1093/nar/gkaa1047.

S. M. Kandathil, J. G. Greener, and D. T. Jones. Prediction of interresidue contacts with deepmetapsicov in casp13. *Proteins: Structure, Function, and Bioinformatics*, 87(12):1092–1099, 2019. doi: https://doi.org/10.1002/prot.25779. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/prot.25779.

A. Karpathy. A recipe for training neural networks, 2019. URL https://karpathy.github.io/2019/04/25/recipe/.

K. S. Keating, E. L. Humphris, and A. M. Pyle. A new way to see RNA. *Quarterly Reviews of Biophysics*, 44(4):433–466, May 2011. doi: 10.1017/s0033583511000059. URL https://doi.org/10.1017/s0033583511000059.

D. E. Kim, D. Chivian, L. Malmström, and D. Baker. Automated prediction of domain boundaries in casp6 targets using ginzu and rosettadom. *Proteins: Structure, Function, and Bioinformatics*, 61 (S7):193–200, 2005. doi: https://doi.org/10.1002/prot.20737. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/prot.20737.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

J. Klicpera, J. Groß, and S. Günnemann. Directional message passing for molecular graphs. *CoRR*, abs/2003.03123, 2020. URL https://arxiv.org/abs/2003.03123.

A. Klug. From macromolecules to biological assemblies (nobel lecture). *Angewandte Chemie International Edition in English*, 22:565–582, 1983.

B. Knudsen and J. Hein. Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Research*, 31(13):3423–3428, July 2003. doi: 10.1093/nar/gkg614. URL https://doi.org/10.1093/nar/gkg614.

D. R. Koessler, D. J. Knisley, J. Knisley, and T. Haynes. A predictive model for secondary RNA structure using graph theory and a neural network. *BMC Bioinformatics*, 11(S6), Oct. 2010. doi: 10.1186/1471-2105-11-s6-s21. URL https://doi.org/10.1186/1471-2105-11-s6-s21.

D. P. Kovács, W. McCorkindale, and A. A. Lee. Quantitative interpretation explains machine learning models for chemical reaction prediction and uncovers bias. *Nature Communications*, 12(1):1695, Mar 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-21895-w. URL https://doi.org/10.1038/s41467-021-21895-w.

B. Kuhlman and P. Bradley. Advances in protein structure prediction and design. *Nature Reviews Molecular Cell Biology*, 2019. URL https://www.nature.com/articles/s41580-019-0163-x.

A. Kulesza. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2-3):123–286, 2012. ISSN 1935-8245. doi: 10.1561/2200000044. URL http://dx.doi.org/10.1561/2200000044.

N. Kurji and A. Windemuth. Cyclica successfully integrates deepmind's alphafold 2, August 2021. URL https://blog.cyclicarx.com/cyclica-successfully-integrates-alphafold-2.

KZVK3. Msc project repository, 2021. URL https://github.com/KZVK3/MSc-Project.

B. W. L. The development of x-ray analysis. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 262:145–158, 1961.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4): 541–551, 12 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541. URL https://doi.org/10.1162/neco.1989.1.4.541.

Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-49430-0. doi: 10.1007/3-540-49430-8_2. URL https://doi.org/10.1007/3-540-49430-8_2.

L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search, 2019.

P. F. Lindley. The use of synchrotron radiation in protein crystallography. *Radiation Physics and Chemistry*, 45(3):367–383, 1995.

Q. Liu, X. Ye, and Y. Zhang. A hopfield neural network based algorithm for rna secondary structure prediction. volume 1, pages 10–16, 06 2006. doi: 10.1109/IMSCCS.2006.9.

W. Lu, Y. Tang, H. Wu, H. Huang, Q. Fu, J. Qiu, and H. Li. Predicting RNA secondary structure via adaptive deep recurrent neural networks with energy-based filter. *BMC Bioinformatics*, 20(S25), Dec. 2019. doi: 10.1186/s12859-019-3258-7. URL https://doi.org/10.1186/s12859-019-3258-7.

M. Magnus, M. Antczak, T. Zok, J. Wiedemann, P. Lukasiak, Y. Cao, J. M. Bujnicki, E. Westhof, M. Szachniuk, and Z. Miao. RNA-Puzzles toolkit: a computational resource of RNA 3D structure benchmark datasets, structure manipulation, and evaluation tools. *Nucleic Acids Research*, 48(2):576–588, 12 2019. ISSN 0305-1048. doi: 10.1093/nar/gkz1108. URL https://doi.org/10.1093/nar/gkz1108.

V. Mariani, M. Biasini, A. Barbato, and T. Schwede. lddt: a local superposition-free score for comparing protein structures and models using distance difference tests. *Bioinformatics*, 2013. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3799472/.

T. Meyer, M. D'Abramo, A. Hospital, M. Rueda, C. Ferrer-Costa, A. Pérez, O. Carrillo, J. Camps, C. Fenollosa, D. Repchevsky, J. L. Gelpí, and M. Orozco. Model (molecular dynamics extended library): A database of atomistic molecular dynamics trajectories. *Structure*, 18(11):1399–1409, 2010. ISSN 0969-2126. doi: https://doi.org/10.1016/j.str.2010.07.013. URL https://www.sciencedirect.com/science/article/pii/S0969212610003539.

Z. Miao and E. Westhof. Rna structure: Advances and assessment of 3d structure prediction., 2017.

M. Mirdita, L. von den Driesch, C. Galiez, M. J. Martin, J. Söding, and M. Steinegger. Uniclust databases of clustered and deeply annotated protein sequences and alignments. *Nucleic Acids Research*, 45(D1):D170–D176, Nov. 2016. doi: 10.1093/nar/gkw1081. URL https://doi.org/10.1093/nar/gkw1081.

M. Mirdita, S. Ovchinnikov, and M. Steinegger. Colabfold - making protein folding accessible to all. *bioRxiv*, 2021. doi: 10.1101/2021.08.15.456425. URL https://www.biorxiv.org/content/early/2021/08/15/2021.08.15.456425.

A. L. Mitchell, A. Almeida, M. Beracochea, M. Boland, J. Burgin, G. Cochrane, M. R. Crusoe, V. Kale, S. C. Potter, L. J. Richardson, E. Sakharova, M. Scheremetjew, A. Korobeynikov, A. Shlemov, O. Kunyavskaya, A. Lapidus, and R. D. Finn. MGnify: the microbiome analysis resource in 2020. *Nucleic Acids Research*, 48(D1):D570–D578, 11 2019. ISSN 0305-1048. doi: 10.1093/nar/gkz1035. URL https://doi.org/10.1093/nar/gkz1035.

E. Mitchell, P. Kuhn, and E. Garman. Demystifying the synchrotron trip: a first time user's guide. *STRUCTURE-LONDON-CURRENT BIOLOGY LIMITED-*, 7:R111–R121, 1999.

E. Mixcoha, R. Rosende, R. Garcia-Fandino, and A. Piñeiro. Cyclo-lib: a database of computational molecular dynamics simulations of cyclodextrins. *Bioinformatics*, 32(21): 3371–3373, 06 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw289. URL https://doi.org/10.1093/bioinformatics/btw289.

L. Moffat, J. G. Greener, and D. T. Jones. Using alphafold for rapid and accurate fixed backbone protein design. *bioRxiv*, 2021. doi: 10.1101/2021.08.24.457549. URL https://www.biorxiv.org/content/early/2021/08/26/2021.08.24.457549.

O. Méndez-Lucio, M. Ahmad, E. A. del Rio-Chanona, and J. K. Wegner. A geometric deep learning approach to predict binding conformations of bioactive molecules. *ChemRxiv*, 2021. doi: 10.26434/chemrxiv.14453106.v1.

T. D. Newport, M. S. Sansom, and P. J. Stansfeld. The MemProtMD database: a resource for membrane-embedded protein structures and their lipid interactions. *Nucleic Acids Research*, 47(D1):D390–D397, 11 2018. ISSN 0305-1048. doi: 10.1093/nar/gky1047. URL https://doi.org/10.1093/nar/gky1047.

B. D. Nha. mathcha, 2021. URL https://www.mathcha.io/.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

A. L. Patterson. A fourier series method for the determination of the components of interatomic distances in crystals. *Physical Review*, 46(5):372, 1934.

PDB. Atomic coordinate and bibliographic entry format description, 1992. URL https://cdn.rcsb.org/wwpdb/docs/documentation/file-format/PDB_format_1992.pdf.

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

J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten. Scalable molecular dynamics with namd. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005. doi: https://doi.org/10.1002/jcc.20289. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.20289.

P. Piatkowski, J. M. Kasprzak, D. Kumar, M. Magnus, G. Chojnowski, and J. M. Bujnicki. Rna 3d structure modeling by combination of template-based method moderna, template-free folding with simrna, and refinement with qrnas. *Methods in molecular biology (Clifton, N.J.)*, 1490:217—235, 2016. ISSN 1064-3745. doi: 10.1007/978-1-4939-6433-8_14. URL https://doi.org/10.1007/978-1-4939-6433-8_14.

M. Popenda, M. Blazewicz, M. Szachniuk, and R. Adamiak. Rna frabase version 1.0: An engine with a database to search for the three-dimensional fragments within rna structures. *Nucleic acids research*, 36:D386–91, 02 2008. doi: 10.1093/nar/gkm786.

M. Popenda, M. Szachniuk, M. Antczak, K. J. Purzycka, P. Lukasiak, N. Bartol, J. Blazewicz, and R. W. Adamiak. Automated 3d structure composition for large RNAs. *Nucleic Acids Research*, 40(14):e112–e112, Apr. 2012. doi: 10.1093/nar/gks339. URL https://doi.org/10.1093/nar/gks339.

S. C. Potter, A. Luciani, S. R. Eddy, Y. Park, R. Lopez, and R. D. Finn. Hmmer web server: 2018 update. *Nucleic acids research*, 46(W1):W200–W204, Jul 2018. ISSN 1362-4962. doi: 10.1093/nar/gky448. URL https://pubmed.ncbi.nlm.nih.gov/29905871. 29905871[pmid].

A. Punjani and D. J. Fleet. 3d flexible refinement: Structure and motion of flexible proteins from cryo-em. *bioRxiv*, 2021. doi: 10.1101/2021.04.22.440893. URL https://www.biorxiv.org/content/early/2021/04/22/2021.04.22.440893.

R. Rao, J. Meier, T. Sercu, S. Ovchinnikov, and A. Rives. Transformer protein language models are unsupervised structure learners. *bioRxiv*, 2020. doi: 10.1101/2020.12.15.422761. URL https://www.biorxiv.org/content/early/2020/12/15/2020.12.15.422761.

M. Remmert, A. Biegert, A. Hauser, and J. Söding. Hhblits: lightning-fast iterative protein sequence searching by hmm-hmm alignment. *Nature Methods*, 9(2):173–175, Feb 2012. ISSN 1548-7105. doi: 10.1038/nmeth.1818. URL https://doi.org/10.1038/nmeth.1818.

A. Rives, J. Meier, T. Sercu, S. Goyal, Z. Lin, J. Liu, D. Guo, M. Ott, C. L. Zitnick, J. Ma, and R. Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15), 2021. ISSN 0027-8424. doi: 10.1073/pnas.2016239118. URL https://www.pnas.org/content/118/15/e2016239118.

B. G. S. U. RNA Group. Representative set. 2021. URL http://rna.bgsu.edu/rna3dhub/nrlist/release/3.183/3.5A.

I. Rodríguez-Espigares, M. Torrens-Fontanals, J. Tiemann, D. Aranda-García, J. Ramírez-Anguita, T. Stepniewski, N. Worp, A. Varela-Rial, A. Morales-Pastor, B. Medel-Lacruz, G. Pándy-Szekeres, E. Mayol, T. Giorgino, J. Carlsson, X. Deupi, S. Filipek, M. Filizola, J. Gómez-Tamayo, A. Gonzalez, H. G. de Terán, M. Jiménez-Rosés, W. Jespers, J. Kapla, G. Khelashvili, P. Kolb, D. Latek, M. Marti-Solano, P. Matricon, M. Matsoukas, P. Miszta, M. Olivella, L. Perez-Benito, D. Provasi, S. Ríos, I. Torrecillas, J. Sallander, A. Sztyler, S. Vasile, H. Weinstein, U. Zachariae, P. Hildebrand, G. De Fabritiis, F. Sanz, D. Gloriam, A. Cordomi, R. Guixà-González, and J. Selent. Gpcrmd uncovers the dynamics of the 3d-gpcrome. *Nature Methods*, 17(8):777–787, Aug. 2020. ISSN 1548-7091. doi: 10.1038/s41592-020-0884-y.

M. Rossmann and D. Blow. A method of positioning a known molecule in an unknown crystal structure. *Acta Crystallogr*, 15:24–31, 1962.

C. O. Rubiera. Casp14: what google deepmind's alphafold 2 really achieved, and what it means for protein folding, biology and bioinformatics, 2020. URL https://www.blopig.com/blog/2020/12/casp14-what-google-deepminds-alphafold-2-really-achieved-and-what-it-means-for-protein-folding

Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjölander, R. C. Underwood, and D. Haussler. Stochastic context-free grammers for tRNA modeling. *Nucleic Acids Research*, 22(23):5112–5120, 1994. doi: 10.1093/nar/22.23.5112. URL https://doi.org/10.1093/nar/22.23.5112.

K. Sato, M. Akiyama, and Y. Sakakibara. RNA secondary structure prediction using deep learning with thermodynamic integration. *Nature Communications*, 12(1), Feb. 2021. doi: 10.1038/s41467-021-21194-4. URL https://doi.org/10.1038/s41467-021-21194-4.

F. Schuller. Lectures on the geometric anatomy of theoretical physics, 2013. URL https://www.fau.tv/course/id/242.html.

G. E. Schulz and G. Rosenbaum. The multi-wire proportional chamber as an area detector for protein crystallography in comparison with photographic film; guidelines for future development of area detectors. *Nuclear Instruments and Methods*, 152(1):205–208, 1978.

P. Schwaller, T. Laino, T. Gaudin, P. Bolgar, C. A. Hunter, C. Bekas, and A. A. Lee. Molecular transformer: A model for uncertainty-calibrated chemical reaction prediction. *ACS central science*, 5(9):1572–1583, Sep 2019. ISSN 2374-7943. doi: 10.1021/acscentsci.9b00576. URL https://pubmed.ncbi.nlm.nih.gov/31572784. 31572784[pmid].

C. D. Schwieters, J. J. Kuszewski, N. Tjandra, and G. M. Clore. The xplor-nih nmr molecular structure determination package. *Journal of magnetic resonance*, 160(1):65–73, 2003.

R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units, 2016.

D. E. Shaw, P. Maragakis, K. Lindorff-Larsen, S. Piana, R. O. Dror, M. P. Eastwood, J. A. Bank, J. M. Jumper, J. K. Salmon, Y. Shan, and W. Wriggers. Atomic-level characterization of the structural dynamics of proteins. *Science*, 330(6002):341–346, 2010. doi: 10.1126/science.1187409. URL https://www.science.org/doi/abs/10.1126/science.1187409.

Z. Shui and G. Karypis. Heterogeneous molecular graph neural networks for predicting molecule properties. *CoRR*, abs/2009.12710, 2020. URL https://arxiv.org/abs/2009.12710.

G. N. C. Simm, R. Pinsler, and J. M. Hernández-Lobato. Reinforcement learning for molecular design guided by quantum mechanics, 2020.

M. J. Skwark, N. L. Carranza, T. Pierrot, J. Phillips, S. Said, A. Laterre, A. Kerkeni, U. Şahin, and K. Beguir. Designing a prospective covid-19 therapeutic with reinforcement learning, 2020.

E. W. Steeg. Neural networks, adaptive optimization, and rna secondary structure prediction, 1993.

B. E. Suzek, Y. Wang, H. Huang, P. B. McGarvey, C. H. Wu, and the UniProt Consortium. UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics*, 31(6):926–932, 11 2014. ISSN 1367-4803. doi: 10.1093/bioinformatics/btu739. URL https://doi.org/10.1093/bioinformatics/btu739.

J. Söding. Protein homology detection by hmm-hmm comparison., 2005. URL https://pubmed.ncbi.nlm.nih.gov/15531603/.

Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient transformers: A survey, 2020.

T.-Y. Teng. Mounting of crystals for macromolecular crystallography in a free-standing thin film. *Journal of Applied Crystallography*, 23(5):387–391, 1990.

I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision, 2021.

R. J. L. Townshend, S. Eismann, A. M. Watkins, R. Rangan, M. Karelina, R. Das, and R. O. Dror. Geometric deep learning of rna structure. *Science*, 373(6558):1047–1051, 2021. doi: 10.1126/science.abe5650. URL https://www.science.org/doi/abs/10.1126/science.abe5650.

T. Tsaban, J. Varga, O. Avraham, Z. Ben-Aharon, A. Khramushin, and O. Schueler-Furman. Harnessing protein folding neural networks for peptide-protein docking. *bioRxiv*, 2021. doi: 10.1101/2021.08.01.454656. URL https://www.biorxiv.org/content/early/2021/08/02/2021.08.01.454656.

F. H. D. van Batenburg. PseudoBase: structural information on RNA pseudoknots. *Nucleic Acids Research*, 29(1):194–195, Jan. 2001. doi: 10.1093/nar/29.1.194. URL https://doi.org/10.1093/nar/29.1.194.

M. W. van der Kamp, R. D. Schaeffer, A. L. Jonsson, A. D. Scouras, A. M. Simms, R. D. Toofanny, N. C. Benson, P. C. Anderson, E. D. Merkley, S. Rysavy, D. Bromley, D. A. C. Beck, and V. Daggett. Dynameomics: a comprehensive database of protein dynamics. *Structure (London, England : 1993)*, 18(4):423–435, Mar 2010. ISSN 1878-4186. doi: 10.1016/j.str.2010.01.012. URL https://pubmed.ncbi.nlm.nih.gov/20399180. 20399180[pmid].

V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

R. Vernon. Alphafold 2: What no one is talking about, September 2021. URL https://blog.cyclicarx.com/limitations-of-alphafold.

A. Villegas-Morcillo, S. Makrodimitris, R. C. H. J. van Ham, A. M. Gomez, V. Sanchez, and M. J. T. Reinders. Unsupervised protein embeddings outperform hand-crafted sequence and structure features at predicting molecular function. *Bioinformatics*, 37(2):162–170, 08 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btaa701. URL https://doi.org/10.1093/bioinformatics/btaa701.

V. A. Voelz, G. R. Bowman, K. Beauchamp, and V. S. Pande. Molecular simulation of ab initio protein folding for a millisecond folder ntl9(1-39). *Journal of the American Chemical Society*, 132(5):1526–1528, Feb 2010. ISSN 0002-7863. doi: 10.1021/ja9090353. URL https://doi.org/10.1021/ja9090353.

A. Vyas, A. Katharopoulos, and F. Fleuret. Fast transformers with clustered attention, 2020.

E. Wagstaff, F. B. Fuchs, M. Engelcke, M. A. Osborne, and I. Posner. Universal approximation of functions on sets, 2021.

L. Wang, Y. Liu, X. Zhong, H. Liu, C. Lu, C. Li, and H. Zhang. DMfold: A novel method to predict RNA secondary structure with pseudoknots based on deep learning and improved base pair maximization principle. *Frontiers in Genetics*, 10, Mar. 2019a. doi: 10.3389/fgene.2019.00143. URL https://doi.org/10.3389/fgene.2019.00143.

L. Wang, X. Zhong, S. Wang, H. Zhang, and Y. Liu. A novel end-to-end method to predict rna secondary structure profile based on bidirectional lstm and residual neural network. *BMC Bioinformatics*, 22(1):169, Mar 2021. ISSN 1471-2105. doi: 10.1186/s12859-021-04102-x. URL https://doi.org/10.1186/s12859-021-04102-x.

Z. Wang, J. M. Jumper, K. F. Freed, and T. R. Sosnick. On the interpretation of force-induced unfolding studies of membrane proteins using fast simulations. *Biophysical Journal*, 117(8): 1429–1441, 2019b. ISSN 0006-3495. doi: https://doi.org/10.1016/j.bpj.2019.09.011. URL https://www.sciencedirect.com/science/article/pii/S0006349519307854.

J. D. Watson and F. H. Crick. The structure of dna. In *Cold Spring Harbor symposia on quantitative biology*, volume 18, pages 123–131. Cold Spring Harbor Laboratory Press, 1953.

K. Wei, August 2021. URL https://twitter.com/KathyYWei1/status/1426646353812221954.

Y. Wei, F. Yang, and M. J. Wainwright. Early stopping for kernel boosting algorithms: A general analysis with localized complexities, 2018.

F. Weik, R. Weeber, K. Szuttor, K. Breitsprecher, J. de Graaf, J. L. M. Kuron, H. Menke, D. Sean, and C. Holm. Espresso 4.0 – an extensible software package for simulating soft matter systems. *The European Physical Journal Special Topics*, 14(227):1789–1816, 2019. doi: https://doi.org/10.1002/jcc.20289.

C. Weinreb, A. J. Riesselman, J. B. Ingraham, T. Gross, C. Sander, and D. S. Marks. 3d RNA and functional interactions from evolutionary couplings. *Cell*, 165(4):963–975, May 2016. doi: 10.1016/j.cell.2016.03.030. URL https://doi.org/10.1016/j.cell.2016.03.030.

K. Weissenberg. Ein neues röntgengoniometer. *Zeitschrift für Physik*, 23(1):229–238, 1924.

E. Westhof. Twenty years of RNA crystallography. *RNA*, 21(4):486–487, Mar. 2015. doi: 10.1261/rna.049726.115. URL https://doi.org/10.1261/rna.049726.115.

R. J. Wheeler. A resource for improved predictions of trypanosoma and leishmania protein three-dimensional structure. *bioRxiv*, 2021. doi: 10.1101/2021.09.02.458674. URL https://www.biorxiv.org/content/early/2021/09/02/2021.09.02.458674.

D. Willmott, D. Murrugarra, and Q. Ye. Improving rna secondary structure prediction via state inference with deep recurrent neural networks:. *Computational and Mathematical Biophysics*, 8(1):36–50, 2020. doi: doi:10.1515/cmb-2020-0002. URL https://doi.org/10.1515/cmb-2020-0002.

J. Wu, S. Luo, T. Shen, H. Lan, S. Wang, and J. Huang. Ebm-fold: Fully-differentiable protein folding powered by energy-based models, 2021a.

J. Wu, T. Shen, H. Lan, Y. Bian, and J. Huang. Se(3)-equivariant energy-based models for end-to-end protein folding. *bioRxiv*, 2021b. doi: 10.1101/2021.06.06.447297. URL https://www.biorxiv.org/content/early/2021/06/07/2021.06.06.447297.

T. Xia, J. SantaLucia, M. E. Burkard, R. Kierzek, S. J. Schroeder, X. Jiao, C. Cox, and D. H. Turner. Thermodynamic parameters for an expanded nearest-neighbor model for formation of rna duplexes with watson-crick base pairs. *Biochemistry*, 37(42):14719–14735, Oct 1998. ISSN 0006-2960. doi: 10.1021/bi9809425. URL https://doi.org/10.1021/bi9809425.

Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le. Self-training with noisy student improves imagenet classification, 2020.

X. Xu and S.-J. Chen. A method to predict the 3d structure of an rna scaffold. *Methods in molecular biology (Clifton, N.J.)*, 1316:1–11, 2015. ISSN 1940-6029. doi: 10.1007/978-1-4939-2730-2_1. URL https://pubmed.ncbi.nlm.nih.gov/25967048. 25967048[pmid].

Y. Yan and S.-Y. Huang. Accurate prediction of inter-protein residue–residue contacts for homo-oligomeric protein complexes. *Briefings in Bioinformatics*, 22(5), 03 2021. ISSN 1477-4054. doi: 10.1093/bib/bbab038. URL https://doi.org/10.1093/bib/bbab038. bbab038.

J. Yang, R. Yan, A. Roy, D. Xu, J. Poisson, and Y. Zhang. The i-tasser suite: protein structure and function prediction. *Nature Methods*, 12(1):7–8, Jan 2015. ISSN 1548-7105. doi: 10.1038/nmeth.3213. URL https://doi.org/10.1038/nmeth.3213.

H. Yonemoto, K. Asai, and M. Hamada. A semi-supervised learning approach for rna secondary structure prediction. *Computational Biology and Chemistry*, 57, 02 2015. doi: 10.1016/j.compbiolchem.2015.02.002.

S. Zakov, Y. Goldberg, M. Elhadad, and M. Ziv-Ukelson. Rich parameterization improves rna structure prediction. volume 6577, 03 2011. ISBN 978-3-642-20035-9. doi: 10.1007/978-3-642-20036-6_48.

A. Zemla. Lga: A method for finding 3d similarities in protein structures. 2003. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC168977/.

H. Zhang, C. Zhang, Z. Li, C. Li, X. Wei, B. Zhang, and Y. Liu. A new method of RNA secondary structure prediction based on convolutional neural network and dynamic programming. *Frontiers in Genetics*, 10, May 2019. doi: 10.3389/fgene.2019.00467. URL https://doi.org/10.3389/fgene.2019.00467.

S. Zhang, Y. Liu, and L. Xie. Molecular mechanics-driven graph neural network with multiplex graph for molecular structures, 2020.

Y. Zhang and J. Skolnick. Scoring function for automated assessment of protein structure template quality. 2004. URL https://pubmed.ncbi.nlm.nih.gov/15476259/.

C. Zhao, X. Xu, and S.-J. Chen. Predicting rna structure with vfold. *Methods in molecular biology (Clifton, N.J.)*, 1654:3–15, 2017. ISSN 1940-6029. doi: 10.1007/978-1-4939-7231-9_1. URL https://pubmed.ncbi.nlm.nih.gov/28986779. 28986779[pmid].

Q. Zhao, Z. Zhao, X. Fan, Z. Yuan, Q. Mao, and Y. Yao. Review of machine learning methods for rna secondary structure prediction. *PLOS Computational Biology*, 17(8):1–22, 08 2021. doi: 10.1371/journal.pcbi.1009291. URL https://doi.org/10.1371/journal.pcbi.1009291.

Y. Zhao, Y. Huang, Z. Gong, Y. Wang, J. Man, and Y. Xiao. Automated and fast building of three-dimensional rna structures. *Scientific Reports*, 2(1):734, Oct 2012. ISSN 2045-2322. doi: 10.1038/srep00734. URL https://doi.org/10.1038/srep00734.

Y. Zhu, Z. Xie, Y. Li, M. Zhu, and Y.-P. P. Chen. Research on folding diversity in statistical learning methods for rna secondary structure prediction. *International journal of biological sciences*, 14(8):872–882, May 2018. ISSN 1449-2288. doi: 10.7150/ijbs.24595. URL https://pubmed.ncbi.nlm.nih.gov/29989089. 29989089[pmid].

C. L. Zitnick, L. Chanussot, A. Das, S. Goyal, J. Heras-Domingo, C. Ho, W. Hu, T. Lavril, A. Palizhati, M. Riviere, M. Shuaibi, A. Sriram, K. Tran, B. Wood, J. Yoon, D. Parikh, and Z. Ulissi. An introduction to electrocatalyst design using machine learning for renewable energy storage, 2020.