



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

PREDICTION OF THE EFFECT OF MUTATIONS ON PROTEIN STABILITY

PREDIKCE VLIVU MUTACÍ NA STABILITU PROTEINŮ

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Bc. MONIKA ROSINSKÁ

SUPERVISOR

VEDOUCÍ PRÁCE

Ing. MILOŠ MUSIL, Ph.D.

BRNO 2024

Master's Thesis Assignment



154360

Institut: Department of Information Systems (DIFS)
Student: **Rosinská Monika, Bc.**
Programme: Information Technology and Artificial Intelligence
Specialization: Bioinformatics and Biocomputing
Title: **Prediction of the effect of mutations on protein stability**
Category: Biocomputing
Academic year: 2023/24

Assignment:

1. Study the problematics of protein engineering and protein stability.
2. Review the current state-of-the-art for the prediction of the effect of mutations on protein stability.
3. Construct a training dataset from available sources.
4. Clean the acquired data and build datasets based on different parameters (position, type of mutation, etc.).
5. Test these datasets on new approaches like ProteinNPMM or RFDiffusion.
6. Select a set of sequence and structural features to train the predictive model.
7. Train the predictive model using appropriate machine-learning methods.
8. Evaluate the results.

Literature:

- Dauparas J, et al. Robust deep learning-based protein sequence design using ProteinMPNN. Science. 2022 Oct 7;378(6615):49-56.
- Stourac J, et al. FireProtDB: database of manually curated protein stability data. Nucleic Acids Res. 2021 Jan 8;49(D1):D319-D324.
- Musil M, et al. Computational design of stable and soluble biocatalysts. ACS Catalysis. 2019 Dec 18; 9(2):1033-1054.

Requirements for the semestral defence:
First five points of the assignment.

Detailed formal requirements can be found at <https://www.fit.vut.cz/study/theses/>

Supervisor: **Musil Miloš, Ing., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: 1.11.2023
Submission deadline: 17.5.2024
Approval date: 30.10.2023

Abstract

The aim of this thesis is to develop a classification model for protein mutations, distinguishing between stabilizing and destabilizing variants. The model was implemented using a convolutional neural network architecture, utilising the ResNet50V2 pre-trained neural network and a dataset sourced from FireProt^{DB}. With an accuracy of 0.7, the model effectively predicts the impact of mutations on protein stability. Furthermore, it can provide a confidence measure for its predictions.

Abstrakt

Cílem této práce bylo vytvořit klasifikační model, který dokáže rozlišovat mutace proteinu jako stabilizující či destabilizující. Model byl implementován pomocí konvolučních neuronových sítí s využitím předtrénované sítě ResNet50V2 a datové sady založené na FireProt^{DB}. Model dokáže predikovat vliv mutací na stabilitu proteinu s přesností 0.7 a zároveň poskytuje informaci o míře jistoty dané predikce.

Keywords

protein mutant stability prediction, protein stability, single-point mutation, protein engineering, convolutional neural network, machine learning

Klíčová slova

predikce stability mutovaného proteinu, stabilita proteinu, jednobodová mutace, proteinové inženýrství, konvoluční neuronové sítě, machine learning

Reference

ROSINSKÁ, Monika. *Prediction of the effect of mutations on protein stability*. Brno, 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Miloš Musil, Ph.D.

Rozšířený abstrakt

Proteiny mají důležitou roli v biomedicínském a biotechnologickém průmyslu. V těchto odvětvích je ale často třeba protein přizpůsobit tak, aby byl funkční i v nepříznivých a náročných podmínkách, jako vysoké teploty nebo nepříznivé Ph hodnoty. Přizpůsobení proteinů se provádí pomocí mutací jejich sekvence, přičemž tyto mutace ovlivňují různé vlastnosti proteinu, například stabilitu. Syntéza mutovaných proteinů v laboratoři je však nákladná a časově náročná, a proto se typicky využívá in silico návrh proteinů, který umožňuje předpovědět, jak daná mutace ovlivní vlastnosti proteinu. Tento přístup urychluje vývoj nových proteinů a značně snižuje náklady.

Existuje nespočet metod a nástrojů, které vyhodnocují a předpovídají vlastnosti proteinů. Tyto metody typicky využívají výpočet volné energie proteinu nebo jeho fyzikálně-chemické vlastnosti. V posledních letech se také stále více stávají populárními metody strojového učení. Tyto metody mohou snížit výpočetní čas a rychleji poskytovat výsledky a tím konkurovat ostatním přístupům. Na druhou stranu jejich výsledky jsou dány kvalitou a počtem dat, které je i třeba předzpracovat.

Cílem této práce bylo vytvořit klasifikační model, který dokáže rozlišovat mutace proteinu jako stabilizující a destabilizující. Byly implementovány dva přístupy. Prvním přístupem bylo využití existujícího modelu ProteinMPNN, ten byl však neúspěšný. Druhým přístupem byla implementace vlastní neuronové sítě, jejímž vstupem jsou grafy zobrazující vybrané vlastnosti proteinu a jeho mutované varianty. Model byl implementován pomocí konvolučních neuronových sítí s využitím předtrénované sítě ResNet50V2 a datové sady založené na FireProt^{DB}. Model dokáže predikovat vliv mutací na stabilitu proteinu s přesností 0.7 a zároveň poskytuje informaci o míře jistoty dané predikce. Tento přístup využívá ProMod3 a Clustal Omega nástroje.

Prediction of the effect of mutations on protein stability

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mr. Ing. Miloš Musil, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Monika Rosinská
May 16, 2024

Acknowledgements

I would like to express my sincere gratitude to my supervisor Ing. Miloš Musil, Ph.D., for his guidance and support during the development of this thesis. His feedback, expert advice and encouragement were instrumental for this work. I would also like to extend my appreciation to the Loschmidt Laboratories, for providing the necessary resources.

Contents

1	Introduction	5
2	Proteins and their Folding Process	6
2.1	What is Protein?	6
2.2	Molecular Forces Shaping Protein	7
2.3	Protein Structure	9
2.4	Protein Folding Process	10
3	Protein Stability and its Prediction	17
3.1	Key Contributors to Protein Stability	17
3.2	Principles of Protein Stability	18
3.3	Measurments of Protein Stability	20
4	Current Approaches for Protein Stability Prediction	22
4.1	Protein Stability Prediction using Force Field Methods	22
4.2	Protein Stability Prediction using Phylogenetic Analyses Methods	23
4.3	Protein Stability Prediction using Machine Learning	24
4.4	Protein Stability Prediction using Hybrid Approaches and Meta-Predictors	25
4.5	Current Methods for Protein Stability Prediction	25
4.6	Evaluation of Current Methods for Protein Stability Prediction	26
4.7	Problems of Current Methods for Protein Stability Prediciton	27
5	Basics of Machine Learning and Neural Networks	28
5.1	Understanding the Principle of Neural Networks	29
5.1.1	Activation Function of Neurons	30
5.2	Multi-layer Neural Networks and Deep Neural Networks	33
5.3	Basics of Neural Network Learning	34
5.4	Neural Network Generalisation and Overfitting: Techniques and Validation	38
5.4.1	Network Size Reduction	39
5.4.2	Early Stopping	39
5.4.3	Model Learning Regularisation	40
5.4.4	Data Augmentation: How to Extend Dataset without Collecting New Data	41
5.5	Data Preprocessing and Feature Engineering	41
5.6	Convolutional Neural Networks	42
6	Utilised Programs and Models	46
6.1	ProteinMPNN – Deep Learning-Based Protein Design Method	46

6.2	Clustal Omega	48
6.3	ProMod3	48
7	Design of the Predictor	50
7.1	Dataset	50
7.2	Design Utilising ProteinMPNN	51
7.3	Convolutional Neural Network using Protein Properties Graphs	53
7.3.1	Convolution Neural Network Design	54
8	Implementation, Training and Results	56
8.1	Implementation Utilising ProteinMPNN	56
8.2	Amino Acid Properties Chosen as Input Features	58
8.3	Data Preparation, Feature Extraction and Input Graphs	58
8.4	Data Loaders	60
8.5	Neural Network Implementation	63
8.6	Neural Network using Transfer Learning Implementation	64
8.7	Complications with Neural Network Training	66
8.8	Results	67
9	Conclusion	69
	Bibliography	70

List of Figures

2.1	Amino acid structure	6
2.2	Amino acid characteristics chart	7
2.3	Covalent bond between two atoms	8
2.4	Hydrogen bond	8
2.5	Ionic bond	8
2.6	Peptide bond in protein chain	9
2.7	Secondary structure bond	10
2.8	α -helix, antiparallel and parallel β -sheet structure	11
2.9	Comparison of protein folding speed	13
2.10	Ramachandran Diagram	14
2.11	Protein folding energy landscape and kinetic traps	15
3.1	Protein stability strengths and weaknesses oscillatory pattern	19
3.2	Folding free energy for protein classes	19
3.3	Free energy contributions	20
4.1	Methods $\Delta\Delta G$ classification accuracy	26
5.1	Artificial Neuron	30
5.2	Sigmoid activation function plot	30
5.3	Tanh activation function plot	31
5.4	ReLU activation function plot	31
5.5	Softplus activation function plot	32
5.6	Leaky ReLU activation function plot	32
5.7	Multilayer neural network	33
5.8	Gradient descent	35
5.9	Confusion matrix	37
5.10	ROC curve	39
5.11	Data augmentation example	41
5.12	Filters for edge detection	43
5.13	Convolution process in convolutional neural networks	43
5.14	Convolutional Neural Network Structure	45
6.1	ProteinMPNN sequence recovery rate	47
6.2	ProteinMPNN encoding/decoding.	47
6.3	ProteinMPNN network	47
6.4	Clustal Omega Performance	48
7.1	FireProt ^{DB} data collection scheme	50

7.2	Utilising ProteinMPNN Pipeline	52
7.3	Predictor Pipeline Design	53
7.4	Neural network input designs	55
8.1	Distribution of values in ProteinMPNN results	57
8.2	CSV record of input data for data loader	60
8.3	Output example of one-sequence-input data loader	61
8.4	RGB channels of a colourful image	62
8.5	General Neural Network Design	64
8.6	Neural Network Using Transfer Learning Design	65
8.7	The ROC curve of the model.	67
8.8	ROC curves of some of the other models	68

Chapter 1

Introduction

Proteins play a crucial role in the biomedical and biotechnological industries, where it is often needed to adapt the protein to stay functional in harsh environments. This protein adaptation is conceived by mutation of protein sequence, which can affect its properties, like stability. However, conducting physical experiments with protein mutants is expensive and time-consuming. The prioritisation of proteins for laboratory analyses can be performed by in silico protein design, which can be utilised to predict how the mutations affect protein features and hasten protein design and production at reduced cost.

There are plenty of methods for protein feature prediction, usually using free energy calculation or physical features. In recent years, the machine learning approach for protein feature prediction has become popular, and several algorithms have appeared. The reason for creating machine learning methods is that they can lower computational time and provide results faster, resolving the biggest problem of the force field methods. Jianwen Fang stated that most existing machine learning algorithms do not generalise¹ well, and each has several problems.

This work aims to combine two or more machine learning algorithms that complement each other, eventually improving success rate and generalisation using transfer learning. Creating a new machine learning model with high performance can replace computationally expensive physical-based approaches, reducing server computational time and cost. In this work, ProteinMPNN and its combination with FoldX are investigated. Also, a neural network utilising ProMod3 and Clustal Omega is designed.

The work is divided into nine chapters, including the introduction. The second chapter includes an introduction to proteins, forces holding the proteins together and protein folding. The third chapter brings insight into protein stability, its principles and measurements. Then, current approaches for protein stability prediction are analysed, including different force field methods, phylogenetic analyses, machine learning and hybrid approaches. Evaluation and problems of currently used methods are also discussed. The fifth chapter describes the basics of machine learning and neural networks, including important terms such as activation or loss function. This chapter also includes neural network performance analyses and techniques to improve the accuracy of the neural network. The next chapter describes utilised methods – ProteinMPNN, ProMod3 and Clustal Omega. The eighth chapter contains the dataset description and design of all the implemented programs, and in the next chapter, the implementation, model training and results are presented. The last chapter contains the conclusion for this work.

¹Generalisation means it is able to predict the correct outcome even for new, unseen data [4].

Chapter 2

Proteins and their Folding Process

Proteins are an essential component of life, facilitating regulated and faster metabolic reactions, transporting molecules, responding to stimuli and serving as building blocks of tissue. Those functions can be used for drug development and transportation, biomedical research, purification of industrial product streams, production of bio-fuels, catalysing food processing, cosmetics, and other biomedical and biotechnological industries [24]. For some of these practices, it is required to modify the target protein to enhance its activity or effectiveness, improve stability in specific working environments, or alter proteins' specificity [24]. These protein variants are created by introducing mutations, which can affect protein folding, stability, activity and other crucial properties.

2.1 What is Protein?

The basic building blocks of proteins are molecules called amino acids [4][41]. There are 20 naturally occurring amino acids, each with different properties [4][41]. Each amino acid comprises an amino group, alpha carbon and a carboxyl group constructing the amino acid backbone, and a variant group, called a side chain, as shown in Figure 2.1 [41]. The diverse characteristics of amino acids are caused by their side chains – the length of the side chain and the participating molecules. Amino acids can be electrically charged or uncharged, hydrophobic, hydrophilic, or acidic and have some unique characteristics. Amino acids and their traits can be seen in Figure 2.2.

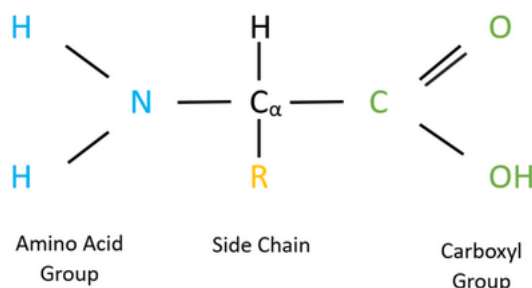


Figure 2.1: Structure of amino acid. The structure is the same for all amino acids except the side chain, represented by R, which differs for each amino acid.

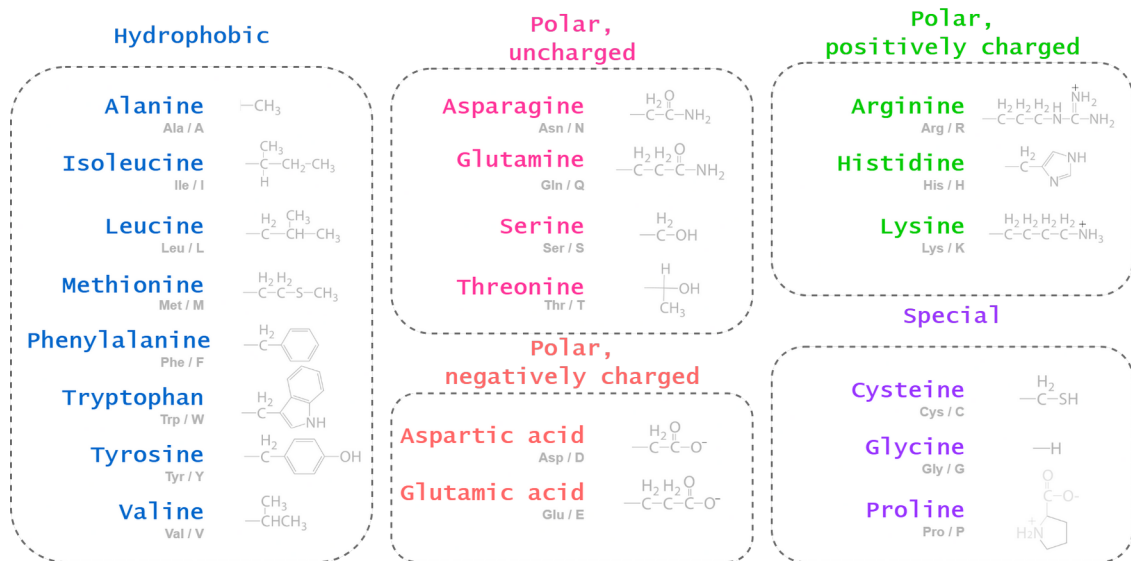


Figure 2.2: Chart of amino acids and their division based on characteristics. Each amino acid has its name, 3-letter code, 1-letter code, and side chain. This division is based on lectures from the course MITx 7.00x: Introduction to Biology – The Secret of Life by Eric Lander. The hydrophobic group can also be called nonpolar; all polar groups are hydrophilic, which is the opposite of hydrophobic [41]. Polar negatively charged amino acids are called acidic, polar positively charged amino acids are called basic [41].

2.2 Molecular Forces Shaping Protein

To comprehend the basics of protein structure, folding and stability, it is crucial to understand the forces between atoms and molecules and their influence on the whole protein. Among the main contributors are electronegativity, covalent bonds, hydrogen bonds, ionic bonds, hydrophobic forces and Van der Waals interactions.

Electronegativity tells how tightly an atom pulls close electrons; in other words, how much it attracts electrons that are nearby [5]. This force is essential for the creation of several bonds. The difference between two atoms' electronegativity determines how strong the bond will be [5]. The bond strength is also influenced by the distance between the atoms, known as Coulomb law.

A **covalent bond** is a strong and stable bond between two atoms. It involves sharing a pair of electrons between two atoms [5], as Figure 2.3 depicts. The number of covalent bonds an atom can participate in is given by the number of electrons in the atom. The length of the covalent bond can be range from $1.13 \text{ \AA}^1$, to $1.54 \text{ \AA}$ (for a typical carbon-carbon bond), with average value of $1.3 \text{ \AA}$ [5][3]. The length of the bond depends on the participating atoms and the strength of the bond (given by the number of electrons in the covalent bond) [5].

¹ $\text{\AA}$ or Ångström is unit of length, $1 \text{ \AA}$ is 0.1 nm .

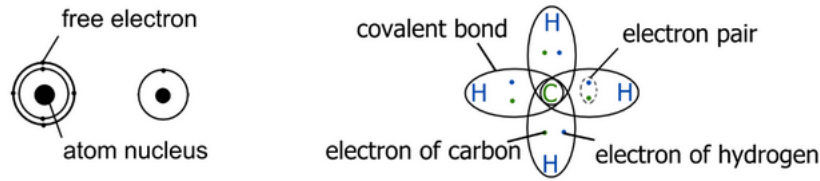


Figure 2.3: Covalent bond between carbon and hydrogen atoms. One electron of carbon makes a bond with an electron of hydrogen, creating an electron pair. On the right, the CH_4 molecule and its bonds are depicted.

A **hydrogen bond** is a bond between hydrogen in one molecule and an oxygen, nitrogen or phosphorus atom in another molecule [5][41]. Because of the electronegativity difference between hydrogen and a second atom, a charge is created between them [5]. The hydrogen carries a positive charge, and the other atom carries a negative charge, as shown in Figure 2.4. Hydrogen bond usually occurs at a distance of 1.5–2.6 Å [3].

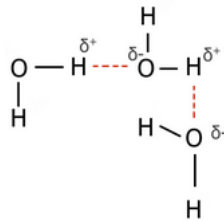


Figure 2.4: A visualisation of hydrogen bonds between water molecules (red lines). Oxygen pulls the electron of hydrogen, gaining a slightly negative charge, written δ^- , while hydrogen gains a slightly positive charge, written δ^+ .

An **ionic bond**, also called an **electrostatic bond**, is a bond between two oppositely charged molecules (ions) [5][41]. An electron from one molecule exits to another, creating a charge that connects the molecules. The donor of the electron gets a positive charge, while the acceptor receives a negative charge, as illustrated in Figure 2.5. This bond strongly depends on the environment and is influenced by the solvent's pH, salt concentration, and permittivity [28]. They are considered strong [5], although their strength in proteins is lower due to the partially charged aqueous environment of the cell [28] [41]. Electrostatic forces are significant even at a distance of 1.7 Å [3].

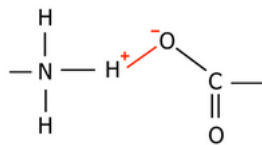


Figure 2.5: An illustration of an ionic bond between two amino acids (the rest of the molecules are depicted by one-sided bonds).

A **hydrophobic force** is caused by the fact that nonpolar molecules² are hydrophobic; they do not resolve or mix with water [41]. These molecules bond with each other in aqueous solutions, creating clusters that minimise contact with the solution and the number of

²Nonpolar molecules are molecules with a low difference of electronegativity between atoms.

hydrogen bonds [28]. The hydrophobicity of the amino acids is also one of the main driving forces folding the protein hydrophobic core [23] [28].

Van der Waals force, also called *dispersion force* is a very small omnipresent force [41] involving fluctuating partial charges inside uncharged molecules. Even if molecules are nonpolar, the charge moves a little bit with moving electrons, influencing other atoms and molecules and creating a force between them. Although small and easy to break, they are strong when many are put together. Van der Waals forces are negligible beyond 5 Å. [5]

Disulphide bonds are connections of two sulphides. In proteins, they are usually called **cystein briges** as the only amino acid containing sulphur is a Cysteine. Disulphide forces provide a significant stabilisation of protein structure [28] [23]. However, their presence in proteins is quite rare, and most proteins do not contain those bridges. The disulphide bond occurs in distance approximately 2.05 Å between sulphide atoms or 5.5 Å between carbon atoms.

2.3 Protein Structure

Proteins consist of amino acids, described in chapter 2.1. Amino acids can connect and create chains and more complex forms – primary, secondary, tertiary and quaternary structures [41]. These structures are formed utilising the forces described in previous chapter. Based on the current protein folding theories, super-secondary and domain structures can also be recognised. However, they serve only as a transition between secondary and tertiary structures.

A **primary structure** can be described as a chain of amino acids called a *polypeptide chain* or a *protein backbone*. The amino acids are connected into a chain by covalent bonds, called *peptide bonds*, between nitrogen and carbon atoms of two molecules, as illustrated in Figure 2.6. The amino acids linked into the polypeptide chain are often referred to as *residues*. [41]

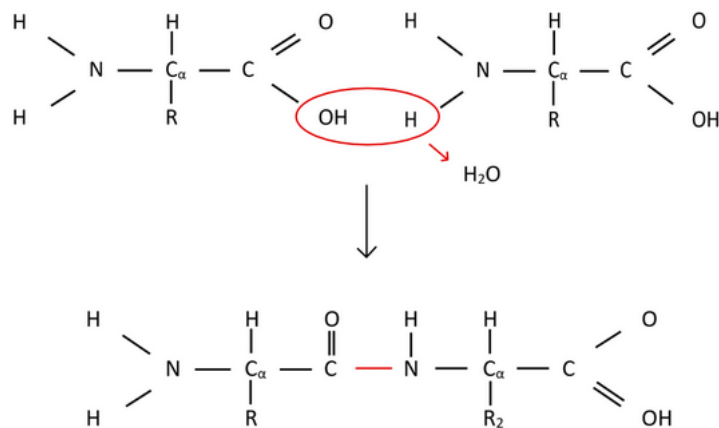


Figure 2.6: The carboxyl and amino acid groups of two amino acids bind; during the binding, two hydrogens and one oxygen are removed, creating a free water molecule. A peptide bond is created, holding the amino acid chain together.

A **secondary structure** is characterised by small arrangements of amino acids created by bonds between side chains of residues, usually between oxygen of the carboxyl group

and hydrogen of the nitrogen group, depicted in Figure 2.7. These structures are α -helix, β -sheet, β -turn, random coil (also called a loop), and their combinations [41]. They are stabilised by hydrogen and close peptide bonds [41].

The α -helix is depicted in Figure 2.8a. On average, one complete turn of the helix contains 3.6 residues and is approximately 0.54 nm (~ 10 amino acids) long [12]. The β -sheet consists of several β -strands, illustrated in Figure 2.8b for antiparallel and Figure 2.8c for parallel variant. They are stabilised mainly by hydrogen bonds, often involving the amino or carboxyl group and side chain polar groups [12].

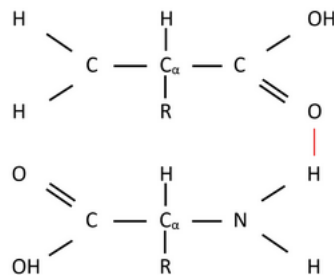


Figure 2.7: Bond between two residues, forming a secondary structure. The bond usually forms between oxygen and hydrogen atoms, meaning the residues must be shifted or somehow turned around.

A **tertiary structure** is a compact dimensional structure of protein created by the connection of secondary structures that represents a correctly folded and active state of monomers³. It is formed and held by hydrogen, hydrophobic, ionic, and disulphide bonds and Van der Waals forces between residues that are distant in the primary structure [41].

A **quarternary structure** is a connection of multiple polypeptide chains [41]. It is the folded state of multimers (proteins that need to link two or more polypeptide chains to be active) [41]. These chains can be the same (have the same sequence and tertiary structure) or differ, but the chains are specific for each multimer. It is stabilised by hydrogen, ionic, and disulphide bonds and Van der Waals forces.

The dimensional tertiary and quarternary structures are not entirely fixed and can slightly change their conformation when interacting with other molecules. This allows the protein to provide its function, which is often connected with these molecules.

2.4 Protein Folding Process

Protein folding is an essential process that transforms protein from primary to its tertiary or quarternary structure signified by the correct shape, which is needed for the proper biological function of the protein. Without its native shape, the protein may not be stable or active, or it may not have the right specificity⁴. Understanding the mechanism of protein folding is necessary to advance biotechnology and science. However, this process is not fully explained due to its complexity and the fact that the cell environment is likely an essential part of it. Mutations of proteins can cause changes in the folding process, inducing the wrong fold of the protein. Hence, protein folding principles must be kept in mind

³proteins consisting of one polypeptide chain

⁴Specificity means that protein binds only specific types of molecules, which it can interact with.

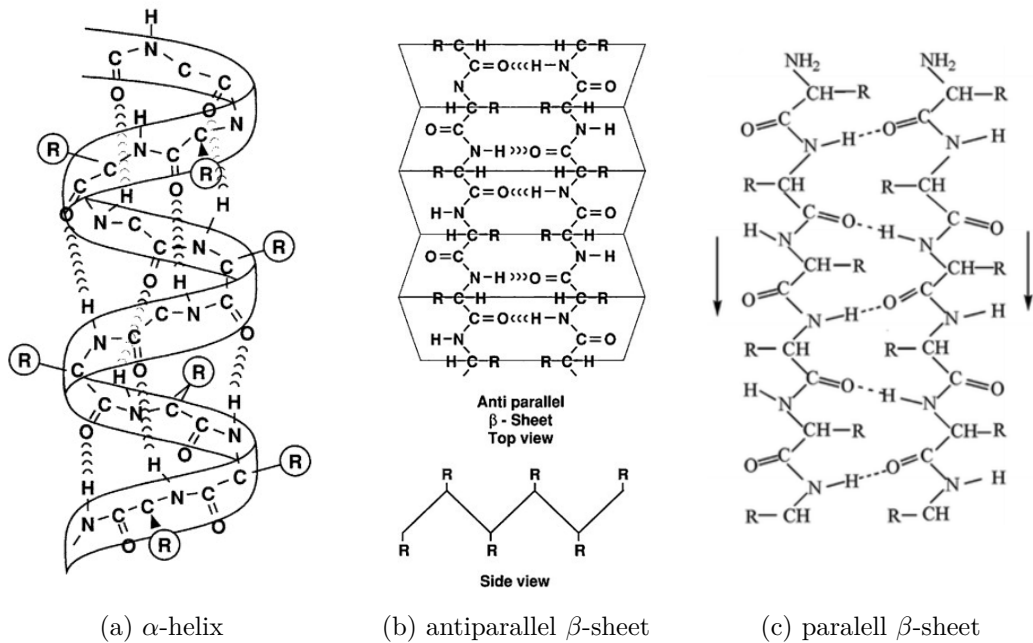


Figure 2.8: (a) An α -helix structure – the polypeptide chain is turned around the chain length axis of the molecule, with side chains pointing outwards. The helix is stabilised by hydrogen bonds described in picture 2.7, which are almost parallel with the axis of the helix. (b) An antiparallel β -sheet and (c) parallel β -sheet structures – several strands called β -strands are connected into a sheet. Between these strands are usually parts called β -turns, which change the direction of the polypeptide chain without causing a breakage of the chain.

Figures (a) and (b) were taken from DNA Structure and Function by Richard R. Sinden [40], and figure (c) was taken from Enzymatic processing of protein-based fibres by Fu Jiajia et al. in Applied Microbiology and Biotechnology [15].

when designing modified protein variants. Some parts of the folding process are sufficiently explained due to several independent studies and observations [1] [21] [26] [34] [35] [36] [41] [42] [46]; other parts are described by various theories [21] [26]. In this chapter, known facts and the most probable theories supported by several proofs are defined at first, and modern theories about protein folding are described at the end of the chapter.

The process of protein folding is complex and can vary between individual proteins. Small proteins (under 50 residues) usually fold quickly and independently; meanwhile, mid-size and large proteins (over 200 residues) often fold slowly, and some of them require the cellular apparatus to fold correctly and to be stable [1][21][42]. The speed of folding is likely not influenced only by the size of the protein but also by the length of the bonds in the protein. In fast-folding proteins, there are primarily short- and middle-range contacts⁵, while in slow-folding proteins prevail long-range contact influenced by hydrophobic residues pairs, which take long to form [12]. Some proteins tend to refold after release from ribosomes. They are called aggregation-prone proteins and usually have more transition states⁶. For that reason, they fold exceptionally slowly. They have an increased tendency to misfold and bind with other proteins, creating abnormal structures (aggregates) that are often insoluble.⁷ The reason for this behaviour of aggregation-prone proteins is likely that the folding rate is lower than the aggregation rate [21]. Figure 2.9 depicts differences in the folding speed of different sizes of proteins.

The machinery that ensures the folding of large proteins into their native structure⁸ is the cell apparatus. Parts of the cell system generate well-folded and stable proteins, avoid the generation of unwanted conformations of proteins and repair misfolded⁹ proteins. The most essential parts of cell machinery are ribosomes, chaperones and Hsp70 (heat shock protein).

Many proteins start to fold during synthesis, still bound to a ribosome. That way, ribosomes alter the folding energy and reduce the number of accessible conformations [41]. Accessible conformations are determined, besides others, by *torsion angles*¹⁰, especially ϕ and ψ angles. The expected conformations of α -helices and β -sheet structures are depicted in Figure 2.10 by *Ramachandran diagram*, and except random coils and β -turns the protein should contain predominantly these angle values. Chaperons prevent protein aggregation [41], and sometimes, they can revert it and unfold the aggregated proteins or make them soluble (up to 97% aggregation-prone proteins). They also catalyse folding and unfolding, bind proteins, and help them bury nonpolar amino acids¹¹. Hsp70, also known as heat shock protein, has a very similar function as chaperons, and in addition, it helps cells to withstand higher temperatures by stabilising proteins and prevent their aggregation. [21]

⁵Range of contacts was brought up by several scientists and slightly differs, but generally short-range contacts are between 1-2 closest residues in protein backbone, mid-range residues between 3-4 and long-range over five closest residues.

⁶Transition state is a somehow stable structure of a protein, which is not active and occurs only in the folding process.

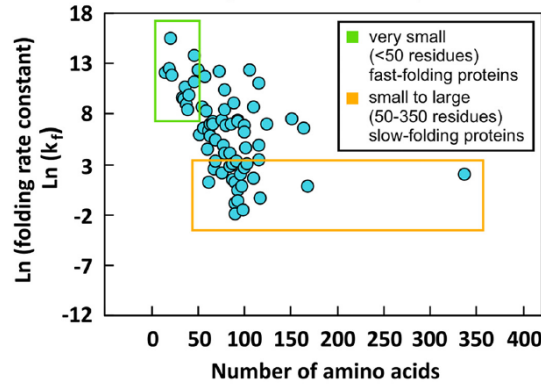
⁷Insoluble molecules do not dissolve in a particular environment; in this chapter, it is understood as in a cellular environment.

⁸Native structure is an adequately folded structure of a protein in which it can provide its function.

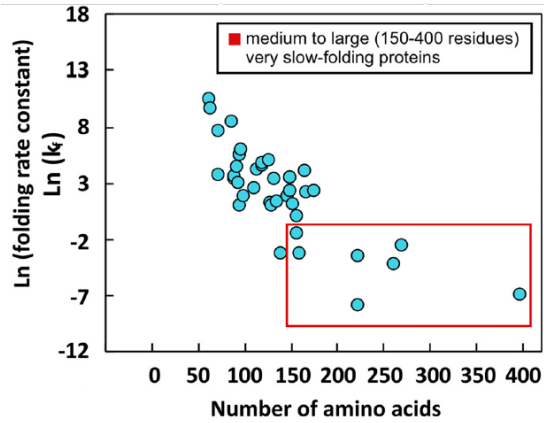
⁹Misfolded protein is a protein that has wrong 3D structure (different from the native one), usually cannot provide its biological activity.

¹⁰Torsion angle, also known as dihedral angle, is an angle between two planes defined by four consecutive atoms in a molecule. The angle can be divided to four: ϕ , ψ , ω and χ .

¹¹Nonpolar amino acids are hydrophobic and do not bind well with water. The cellular environment is aquatic, so nonpolar molecules tend to hide inside the protein to prevent contact with the environment.



(a) Folding rate constant for two-state folding proteins



(b) Folding rate constant for multi-state folding proteins

Figure 2.9: Speed of protein, expressed as folding rate constant, as a function of molecular size of protein, (a) for two-state folding proteins and (b) for multiple-state folding proteins. The higher the folding constant, the faster the protein folds. The folding rate constant is a measure of how fast protein folds. Figures taken from Protein folding in vitro and in the cell: From a solitary journey to a team effort [21].

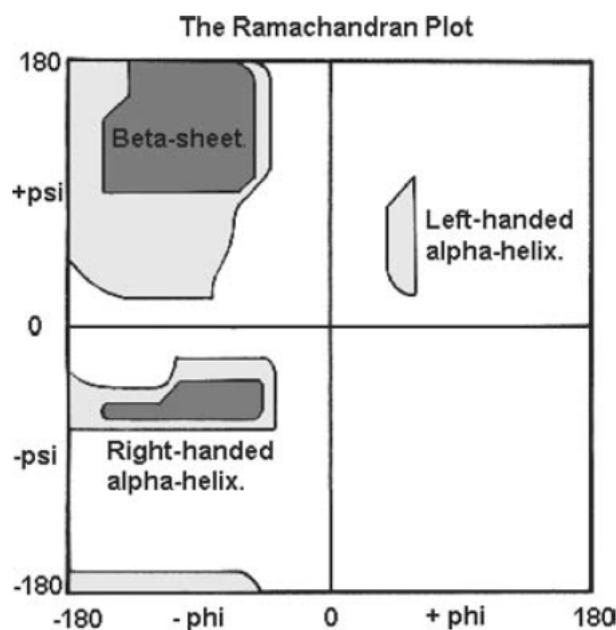


Figure 2.10: Ramachandran diagram showing possible conformations of ϕ and ψ angles. The image was taken from Protein Bioinformatics by Gromiha [12].

Even after proteins fold into their native state, they continue to investigate other conformations, and their shape changes slightly. This is essential for their biological activity, which often requires adjustment of the protein structure. However, some proteins have complex energy landscapes containing aggregates. For this reason, many native states are kinetically trapped from other states – aggregates, aggregation-prone intermediates, or unfold states, as shown in Figure 2.11 [42]. [21][46]

From a physicochemical point of view, small forces between atoms and polypeptide chains, listed in 2.2, are responsible for protein folding and its final structure, as they avoid disfavoured interactions. The only elements that can be both hydrogen-satisfied and extend indefinitely without causing a steric clash are α -helices and β -sheets, joined by random coils. The structure is also influenced by the chain's backbone, which limits possible folds¹² of the parts of chains, theoretically to circa 10 000 folds for protein domain [34]. Generally, possible protein folds are restricted by finite ways in which α -helices and β -sheets can assemble, as they have to respect the volume of molecules, need to hide hydrophobic groups or need for hydrogen bond satisfaction [34][35][36]. This is supported by the fact that the number of discovered protein sequences has increased exponentially while the number of protein structures increased only linearly [34]. This leads to a generally accepted theory that there is only a limited number of distinct folds.

Also, from a physical point of view, folding causes a loss of conformational entropy [28]. By physical laws, it must be compensated by other forces, generally by favourable interactions of side chains of amino acids, primarily by hydrogen bonds [28] [34]. This claim

¹²Protein fold is a 3D arrangement of part of the polypeptide chain, in other words, can be understood as part of the protein tertiary structure.

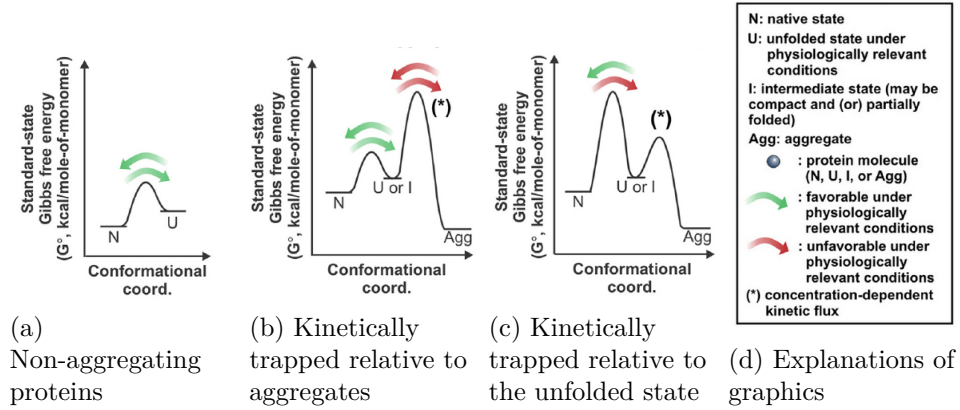


Figure 2.11: Representative protein folding energy landscape and kinetic traps for (a) non-aggregating proteins, (b) proteins kinetically trapped relative to the aggregation state and (c) proteins kinetically trapped relative to the unfolded state. Figures taken from Protein folding in vitro and in the cell: From a solitary journey to a team effort [21].

is supported by the fact that in experiments, many possible hydrogen bond participants were detached from the solvent. Thus, these molecules are probably satisfied by intramolecular partners because every theoretical hydrogen bond participant without a partner would be disfavoured by circa 5 kcal/mol (according to C. N. Pace et al. [27]) and thus destabilising [34][42]. A higher count of hydrogen bond acceptors and donors lacking a partner in a folded protein is not physically possible due to the free energy difference between folded and unfolded structures.

In this manner, protein folding can also be viewed as an elimination of disfavoured interactions, e.g. steric clashes and disrupted hydrogen bonds. Although these interactions are not specific, they can cause the formation of particular chain structures [34]. A *steric clash* is a situation in which two non-bonded atoms are impossibly close to each other, overlapping their Van der Waals radii, and can have a significant influence on protein structure. For example, because of steric clash, there has to be a random coil or turn between α -helix and β -sheet. As mentioned above, unsatisfied hydrogen bond acceptors and donors are destabilising, so they help to establish native interactions by looking for a partner to bind with.

However, some steps in protein folding are still unknown. It is generally difficult to observe the process of protein folding in its natural environment, i.e. in the cell. For that reason, only theories exist for some parts of the folding process. One of these unknown steps is how the folding begins and in which order structures are created. The scientists concur on the **Levinthal paradox**, which says that a random search of protein native structure among all the possible arrangements usually would take an unrealistic amount of time, while the natural proteins fold in a significantly lower time [18]. This leads to the assumption that proteins use some unknown mechanics that help them to fold correctly. There is also widely accepted **Afinsens thermodynamic hypothesis** stating that the native structure is determined only by the protein's amino acid sequence and protein folding is deterministic [2]. In other words, under the same environmental conditions, proteins always fold the same, and folding cannot be random.

One of the first, now dismissed theories was that the tertiary structure was formed around the initial nucleus consisting of the local secondary structure. A similar theory called **framework model** states that the protein starts by folding the secondary structures, which then fold into its three-dimensional formation. A theory of **hydrophobic collapse model** declares that secondary structures in regions with a high number of hydrophobic residues are formed first to avoid water in the environment. These structures are accumulated to a hydrophobic core, forming a growing tertiary structure. The theory of **nucleation-condensation model** says that secondary and tertiary structures are folded in parallel; however, it keeps the idea that tertiary structure folding is catalysed by the initial nucleus. [26]

The widely accepted theory is that some proteins fold via **intermediates**, i.e. several stages of partially-folded protein (that are not functional). Small single-domain proteins generally fold using zero or few stages, while larger proteins and multi-domain protein folding may require several intermediates. [21]

The folding of multi-domain proteins is even more complicated and misunderstood. The domains may either fold independently on each other or interdependent, relying on each other. Independent folding of domains (individual polypeptide chains) decreases the number of possible conformations of the backbone, which can prevent interdomain misfolding [21].

In conclusion, during the protein mutation design, we must keep in mind the rules of steric clash (i.e. needed coils, loops and turns between alpha-helices and beta-sheets, length of side chains of amino acids) and hydrogen bonding (mutation cannot cause creation of unsatisfied hydrogen donor/receiver). It is also important to remember that the C-terminal residue often carries important interaction parts and sometimes establishes contact with N-terminal regions of the protein [21]. Also, as a result of the Levinthal paradox, there is a need to ensure both that the protein folds in a particular structure and that the protein is less compatible with other structures [34]. In experiments with denatured or artificially synthesised proteins, the need for cell machinery to fold proteins successfully can be problematic. Considering the intermediates theory, one should take into account that mutations can influence the intermediates and the protein folding and can cause the protein to be unable to fold correctly.

Chapter 3

Protein Stability and its Prediction

Protein stability is an important property of protein, allowing it to withstand changes in the cell environment, with changing pH, temperature, and molecule concentrations, and it keeps the protein in its active folded state [24]. Increased stability allows the protein to survive and be active in harsh conditions, while lowered stability can increase protein activity and slightly change its specificity. Understanding protein stability is crucial to assess the impact of protein sequence mutation.

3.1 Key Contributors to Protein Stability

Protein stability is settled by a balance of several forces and bonds, primarily by forces described in chapter 2.2 – covalent, hydrogen, ionic and disulphide bond, hydrophobic and Van der Waals forces. These forces determine if the protein will fold and stay folded in the current environment and are dominant in the folded state of the protein. The contributions of these forces differ in diverse proteins; Table 3.1 shows average contributions according to Gromiha [12]. These forces are countered by *conformational entropy*¹, which represents the flexibility and freedom of motion of the polypeptide chain. The entropy pushes to be as high as possible, creating a force opposing the bonds mentioned earlier. When protein reaches the folded state, its flexibility decreases, decreasing conformational entropy. The balance between these forces determines the final stability of the protein. Thus, proteins with a higher number of secondary elements (α -helix and β -sheet) tend to be more stable compared to the proteins with higher concentrations of random coils as random coils have higher conformational entropy. This causes the protein to be less stable.

hydrophobic forces	50.8%
hydrogen bonding	27.1%
van der Waals forces	14.6%
electrostatic forces	6.4%
disulfide forces	1.1%

Table 3.1: Relative contributions of energies to protein stability, according to Gromiha [12].

¹Entropy states the amount of uncertainty or randomness in a system. Higher entropy means higher uncertainty.

The role of particular forces in protein stability can be determined by analyses of thermophilic proteins² [24]. Several methods declared that protein stability in these proteins is increased by the raised number of hydrogen bonds, salt bridges, electrostatic forces, ion-pairs, and van der Waals forces [12], including side chain to side chain hydrogen bonds and the fractional polar surface [24]. Thermophilic proteins often have a higher number of charged residues. However, it is important to remember that the position of mutated residue and the forces it creates are crucial, and a simple addition of bonds and forces cannot ensure protein stability; it can even disrupt it.

3.2 Principles of Protein Stability

As a result of protein feature research, various principles were discovered that can help to comprehend protein stability, even though these principles do not form the structure of the protein.

The **stability-activity trade-off** is a logical and well-known fact that an increase in protein activity often leads to a decrease in stability and vice versa [24]. In other words, to keep the activity of a protein, the protein cannot be as stable as it is physically possible – it would not be able to bind other molecules, and protein cannot be as active as is theoretically possible because it would not keep its folded state in most environments. This explains the failures of methods that blindly increase stability – these methods often increase stability in such a way that it blocks the catalytic site of protein or by rigidisation of the protein structure, limiting its conformational flexibility. However, it seems it is possible to improve both the thermostability and activity of the protein. [23] [38]

One of these principles is an **oscillatory stability compensation pattern** described in the article Enzyme Stability-Activity Trade-Off by Q. Hou, M. Rooman, and F. Pucci [14]. In this article, the authors have analysed patterns of stability strengths and weaknesses and the conservation of protein residues, depicted in Figure 3.1. They stated that there is an oscillatory stability compensation pattern centred on the catalytic centre. Catalytic residues are stability weaknesses because they are the functional part of the protein. In the first shell, residues are usually less weak or stabilising at an average of 1–2 Å from the catalytic region. Residues from the second shell, at about 2–3 Å, are stability weaknesses and residues at a distance greater than 4 Å are stabilising. Although residues in the last shell are mostly stabilising, they keep the oscillatory pattern and their stabilising force changes. The distances of shells vary between proteins and protein classes. However, the pattern is still distinguishable, as Figure 3.2 illustrates. [14]

Identification of residue as stability strength or weakness was performed by statistical potentials such as torsion angle domain, solvent accessibility, and distance between two residues. ΔG_i value is the average value of these potentials, as can be seen in Figure 3.3. They also observed anticorrelation with residue evolution conservation – the residues representing stability weakness are, on average, more conserved than residues producing stability strengths. However, this anticorrelation is not direct. [14]

²Thermophilic proteins are proteins which keep their active state even in higher temperatures.

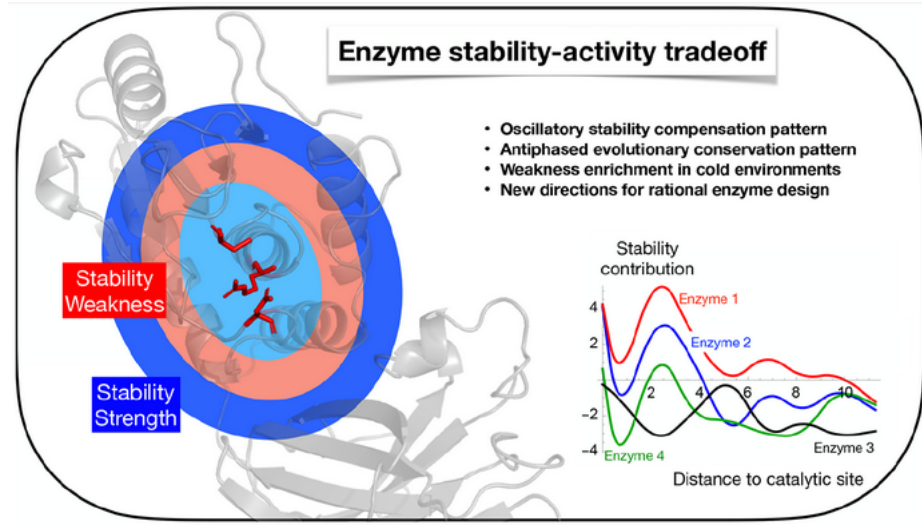


Figure 3.1: Visualisation of an oscillatory pattern of protein stability strengths and weaknesses. In the right corner, folding free energies of several enzymes are drawn to show the generality of the pattern. The figure was taken from the original article [14].

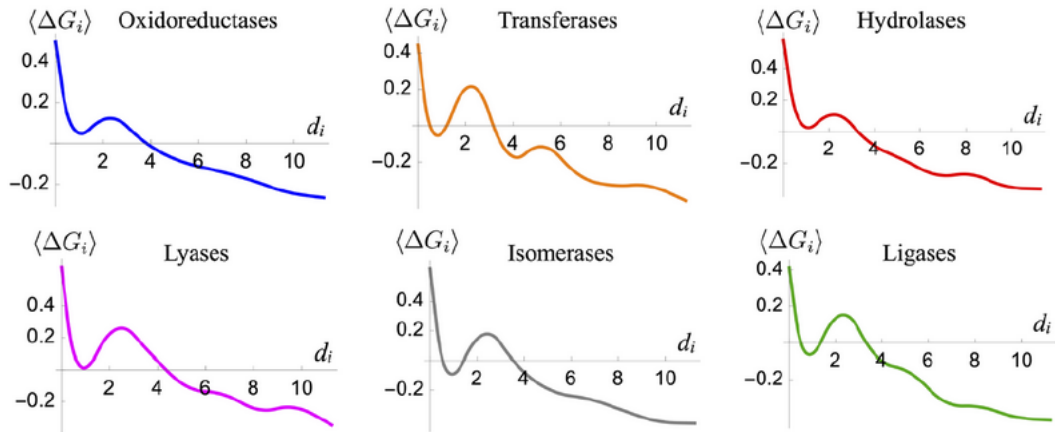


Figure 3.2: Folding free energy in kcal/mol for different classes of proteins. $\langle \Delta G_i \rangle$ is the free energy and d_i is distance of residue i from the catalytic residue. The figure was taken from the original article [14].

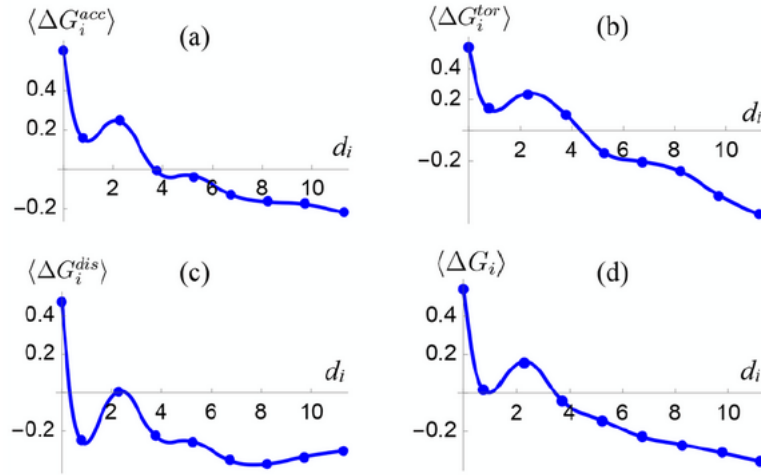


Figure 3.3: Free energy contribution in kcal/mol computed with (a) solvent accessibility, (b) backbone torsion angle, (c) inter-residue potential and (d) their combination as the average value. $\langle \Delta G_i \rangle$ is the free energy and d_i is distance of residue i from the catalytic residue. The figure was taken from the original article [14].

3.3 Measurements of Protein Stability

The stability of protein can be measured in several ways, although they are all estimations, e.g. calculation of free energy contributions, Gibbs free energy and melting temperature. The main contributors to total free energy are hydrophobic, electrostatic, hydrogen bonding and Van der Waals free energy. Calculations of these contributions, including some force field methods, are not part of this work; in case of interest, they are covered in Protein Bioinformatics by Gromiha [12].

The **Gibbs free energy change**, ΔG , also referred to as conformational stability of a protein, is defined in Equation 3.1. According to M. Gromiha, the free energy of the folded state can be computed using Equation 3.2, and the energy of unfolded protein can be calculated using Equation 3.3 [12]. However, the unfolded state energy can be described in other ways, e.g. Eric Lander stated in his course MITx 7.00x: Introduction to Biology – The Secret of Life that Gibbs free energy can be computed using Equation 3.4, where H represents enthalpy³, T is the temperature of the environment, and S stands for the entropy. When comparing protein and its mutant, the ΔG change is computed using Equation 3.5. Note that the lower free energy, the more stable the protein is, so the lower ΔG value, the more stable the protein is, and the lower the $\Delta \Delta G$ value, the more stabilising the mutation is, and vice versa.

$$\Delta G = G_{folded} - G_{unfolded} \quad (3.1)$$

$$G_{folded} = G_{hydrophobic} + G_{electrostatic} + G_{hydrogen} + G_{VanderWaals} + G_{disulphide} \quad (3.2)$$

$$G_{unfolded} = G_{entropic} + G_{nonentropic} \quad (3.3)$$

³Enthalpy is the sum of systems internal energy and the product of its pressure and volume. [5]

$$G = H - T * S \quad (3.4)$$

$$\Delta\Delta G = \Delta G_{wild-type} - \Delta G_{mutant} \quad (3.5)$$

The **melting temperature**, T_m , called **denaturation midpoint**, at which free energy of the unfolded and folded states is equal. Half of the population is unfolded, and the other half is folded. Note that the higher the melting temperature, the higher temperatures the protein can withstand and the more stable the protein is.

These two measurements of protein stability are the most common ones. Unfortunately, they are not entirely correlated, and they cannot be easily converted from one to the other. Some equations bring an estimation of melting temperature in Gibbs free energy and vice versa; however, they are not even close to being precise.

There are other measurements of protein stability, like enthalpy change (ΔH), entropy change (ΔS) or heat capacity change (ΔC_p); however, it is not possible to bring an exhaustive list and description of these methods, as each article can state a new one, and the Gibbs free energy change and melting temperature are the most used ones [24].

Chapter 4

Current Approaches for Protein Stability Prediction

Protein stability is a complex phenomenon influenced by several forces between atoms and molecules. New principles are discovered each year, bringing insight into how protein stability is connected to its sequence, structure, native environment, function and activity. Several well-performing algorithms use current knowledge to predict protein stability. Still, they are computationally expensive, and implementing new revelations into these algorithms is only sometimes possible, thus hindering their further improvement. Due to the high computational demands, creating new physical or free energy-based algorithms is not alluring. On the contrary, the machine learning approaches are nowadays much more prevalent. Besides enabling fast evaluation, they offer the potential for discovering new knowledge on protein stability. Although most current machine learning algorithms perform poorly, several biotechnological laboratories and companies research this method.

This chapter describes the problematics of protein engineering and current state-of-the-art algorithms for predicting the effect of mutation on protein stability – FoldX, Rosseta, SDM, Dynamut, Thermonet, or other tools.

4.1 Protein Stability Prediction using Force Field Methods

Force field methods are one of the approaches for the estimation of the stability of the protein. They calculate protein stability using free energy equations that describe the total energy of protein. This is done by computing the physicochemical effects of residues in the protein structure and by modelling molecular mechanics using known amino acid properties. They usually consider representations of protein energy, various bonds, torsion angles or bond angles and their lengths [12]. [23]

It can be viewed as a description of all interactions in protein [24], computed by energy field equations and $\Delta\Delta G$ value. The quality of particular algorithms differs, often balancing accuracy and time consumption. These methods are currently considered the most powerful, even though there is still space for improvement. There are certain limitations caused by protein sampling and relatively narrow datasets. In general, force fields tend to overestimate hydrophobicity, non-polarity and negative effect of unsatisfied bonds in the protein core.

Most precise are usually **free energy calculation methods** that use molecular dynamics [38] or Metropolis Monte Carlo simulations. The computational demands are extremely high, limiting their applicability only to small proteins. Conversely, **heuristic algorithms**

bring less precise results in exchange for lower computational difficulty. Some of these methods deliver highly useful predictions, like algorithms based on various effective energy functions.

Physical effective energy functions¹ are heuristic methods close to the clean force field algorithms. They simplify the physical and chemical laws but still compute the molecule interactions [16]. Therefore, they are still computationally expensive [16]. These methods are generally used in case of unusual environmental conditions, as they can predict molecule behaviour in different environments.

Statistical effective energy functions do not model molecular dynamics. Their equations are based on various stability descriptors estimated from datasets of experimental protein structures [16]. This approach is significantly faster compared to their physical counterparts (the stability of the protein is usually evaluated in the matter of minutes). They are also more robust and less sensitive to small displacements of atoms [16]. However, their accuracy is determined by the quality and diversity of protein structures used during energy field function determination. They can also be applied only in environments where the protein dataset was measured and are usually used for rapid analyses.

Empirical effective energy functions are approaches somewhere between physical and statistical functions, as they use both molecular dynamics calculations and statistical potentials. These methods are usually more precise than the statistical approach and less computationally demanding than physical energy functions. They also keep the disadvantage of statistical functions – they can be used for predictions in the same environmental conditions as proteins in the used dataset.

This section draws from the Computational Design of Stable and Soluble Biocatalysts article [26], Effective energy functions for protein structure prediction [17] and Energy estimation in protein design [22].

4.2 Protein Stability Prediction using Phylogenetic Analyses Methods

Phylogenetic analyses, also called evolutionary analyses, are methods that predict protein stability using the information contained in protein sequences and their homologs². They belong to sequence-based engineering methods, also called data-driven methods [24]. Their advantage over force field methods is that they use only a protein sequence; therefore, there is no need to know the tertiary structure of the protein. Databases of protein sequences are more extensive than databases of their structure [24]. Therefore, they can be applied to most of known proteins. On the other hand, they are problematic with poorly represented protein families with low amount of homologous sequences.

Ancestral sequence reconstruction is a method that analyses the shared evolutionary history of homologous proteins and reconstructs ancestral proteins using a phylogenetic tree. It analyses the evolutionary relationships between proteins given by the tree. The first step is aligning the sequences using the multi-sequence alignment method, from which the trees are (manually) constructed. After that, the information can be gathered from the tree and used for protein stability prediction. The main disadvantage is that it requires expert knowledge of the biological systems of interest, and various bioinformatics

¹Effective energy means the system’s free energy. [16]

²Homologous sequences are sequences that have a common ancestor. In other words, they are evolutionarily related.

tools. The algorithms are not utilised only for protein stability prediction; they are more often used for the prediction of extinct proteins, for thermostability, activity and specificity engineering and improvement. [13]

Consensus design examines the number of occurrences of amino acids in specific positions of the homologous sequences and, based on the frequency, detects segments of sequence that are conserved³. Conserved segments are considered somehow crucial for the shape or function of proteins and can indicate stabilising residues [24]. The first step is multi-sequence alignment, which aligns all the sequences [24], handles gaps or deletions in sequence and allows the computation of frequencies of the amino acids [23]. Then, segments with low amino acid variability are marked as conserved. To enhance protein stability, the conserved parts of the protein sequence are analysed. If any residue differs from the most common amino acid, it is mutated to match it [23]. The disadvantage is that the prediction can be biased when homologs that are too similar are used. The consensus design is more demanding on the count and dissimilarity of the given sequences than ancestral reconstruction. On the other hand, it can be extended by information about tertiary structure, which keeps important information about forces in proteins, secondary structures or active sites. [30]

Between other sequence-based methods belong analyses of residues' coupling or studying sequences with higher thermostability, described in other sources [24].

This chapter is informed by the Computational Design of Stable and Soluble Biocatalysts article [26].

4.3 Protein Stability Prediction using Machine Learning

Machine learning is becoming popular in various fields, including bioinformatics [7] and protein stability prediction. The main advantage is that there is no need to specify and program the precise algorithm to get the desired results. Only the structure of the neural network, the network's learning algorithm and its input are defined, and even these steps are simplified by Python libraries. The networks analyse the information in input data and detect patterns connected to the desired output. These patterns can be even different from patterns noticed by humans, and the model can bring new insight into the problem. Hence, expert knowledge about the problem is not usually required, as well as manual feature engineering. Another advantage is that after the model is trained, the results of the predictions are usually available in a matter of seconds. The main disadvantage is that the precision of the network is determined by the size and quality of the dataset, which is a major problem, especially in proteomics. Even proving the accuracy of these methods is complicated because of the limited amount of data. Thus, it is challenging to develop a model that yields remarkable results. Unfortunately, in most of the promising models with impressive results, models' accuracy drops when tested on unrelated datasets. Machine learning and machine learning models for protein stability prediction are covered later in this chapter and in chapter 5. [26]

³Conserved parts of sequences are the segments, where the amino acids are the same through almost all homologs, i.e. segments, that are not mutated.

4.4 Protein Stability Prediction using Hybrid Approaches and Meta-Predictors

Hybrid approaches combine several methods and approaches mentioned above, where one method balances the disadvantages of the other. Combining different approaches often leads to the discovery of stabilising mutations overlooked by individual approaches. These methods can often be used even for the design of multiple-point mutants. Meta-predictors combine the results of used approaches into one prediction, given by the most common result of multiple methods. The results are usually robust and more reliable. [22] [26]

4.5 Current Methods for Protein Stability Prediction

There are several methods that predict the stability of protein mutants. The most famous ones are FoldX, Rosetta, ThermoNet, DynaMut, I-Mutant Suite, and SDM.

FoldX is an empirical force field method using molecular dynamic modelling and statistical potentials. The influence of the mutation on protein stability is estimated by the change in free energy. The whole computation is based on a high-resolution 3D protein structure. The results are satisfying for proteins with well-defined 3D structures; otherwise, the accuracy decreases. [37]

Rosetta is a force field method widely used in structural bioinformatics and computational biology for modelling macromolecular structures, which can also be used for the prediction of mutation influence on protein stability. It utilises energy minimisation, molecular dynamics and even Monte Carlo simulations and models the forces between atoms using energy functions. Rosetta is an irreplaceable tool, advancing protein engineering and human knowledge about proteins. The results are remarkable; the only disadvantage is the enormous computational demands. [33]

SDM or **Side-Directed Mutator** is a statistical force field method utilising statistical potentials and free energy calculations. It uses environment-specific amino acid substitution frequencies within homologous protein families to calculate a stability score and a set of conformationally constrained environment-specific substitution tables. [47]

DynaMut is a consensus hybrid approach utilising molecular dynamics and machine learning and combining several tools. It also employs normal mode approaches with graph-based signatures. [32]

ThermoNet is a deep learning-based method utilising 3D convolutional neural networks constructed as multi-channel voxel grids. These grids are filled by biophysical properties derived from the protein structure, specifically atom coordinates. ThermoNet utilises Rosetta for constructing the mutant structure. [19]

Several other algorithms provide protein stability prediction, as mentioned in the following section. Also, the performance of the mentioned method is described in the next section. In case of interest in their implementation description, it is recommended to read the articles published with the specific algorithm.

4.6 Evaluation of Current Methods for Protein Stability Prediction

Based on the research of the University of Torino research team [29], most current tools are not sensitive to physiological conditions, allowing stability prediction of molecules measured in various conditions. As seen in Figure 4.1, most methods are biased toward destabilising mutations [29] [31]. This bias is presumably caused by unbalanced datasets, which generally contain more destabilising mutations than stabilising ones [31]. In the figure, it is apparent the most balanced methods are ThermoNet, Dynamut, Rosetta and SDM, though, for some datasets, their predictions are close to 50%, i.e. neutral. Nevertheless, some methods are not so balanced, with good results, such as DDGun3D, FoldX and MAESTRO. [29]

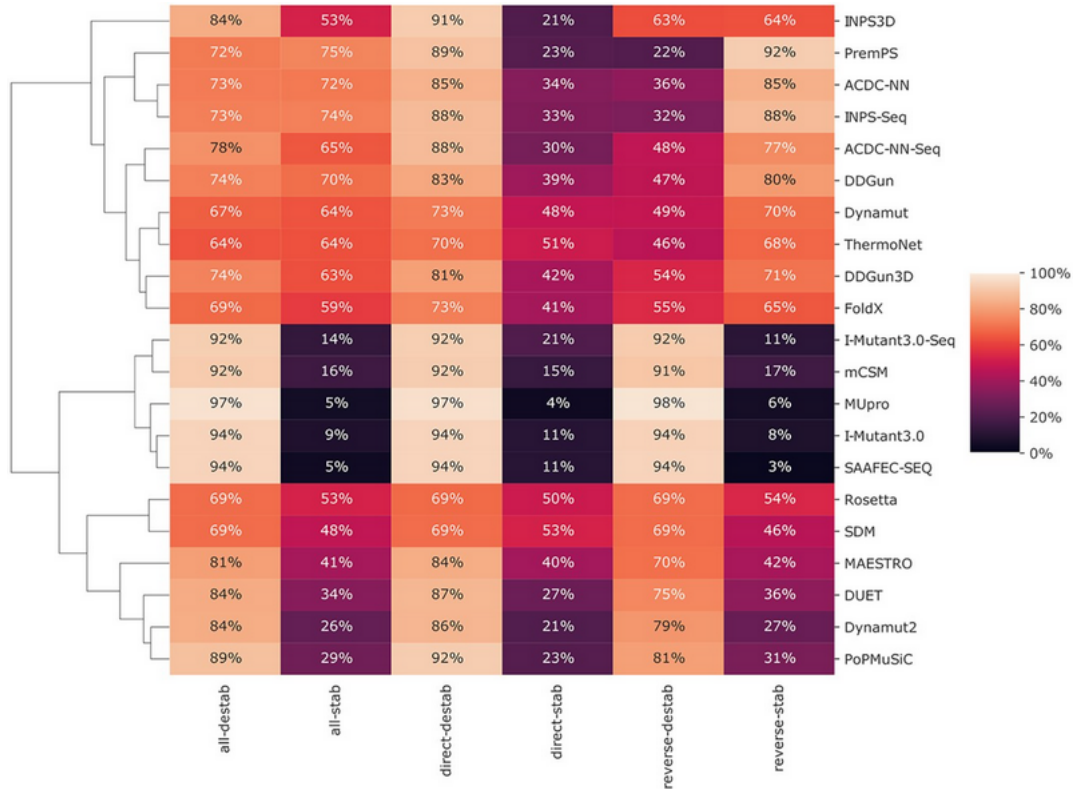


Figure 4.1: $\Delta\Delta G$ classification accuracy of methods, i.e. prediction of the direction of stability change – stabilising or destabilising. The experiment was done on several datasets containing direct mutations, reverse mutations⁴ and their combinations. The „destab“ columns contain predictions for destabilising mutations; the „stab“ columns contain predictions for stabilising mutations. Image taken from the article Predicting protein stability changes upon single-point mutation: a thorough comparison of the available tools on a new dataset [29], containing results of the mentioned research.

⁴Reverse mutations are mutations which consider mutant as wild-type protein, and the mutation „creates“ the actual wild-type protein.

4.7 Problems of Current Methods for Protein Stability Prediction

The best-performing algorithms are mostly force field methods with high computational demands. They provide accurate results, however, these results are available after days, depending on the protein length and complexity. They are very useful for accurate stability prediction. However, they are not effective in eliminating destabilising mutations in the case of protein design. For this, faster approaches like machine learning should be utilised. However, methods based on machine learning can also be problematic, e.g. time-consuming design and training. Performance of these models is also limited by the dataset, its quality, distribution and noise [25].

The biggest problem in machine learning approaches seems to be small and unbalanced datasets [23] [31]. Collecting data about protein structure and the effects of mutations is difficult and time-consuming, as the protein needs to be synthesised [31]. The synthesised protein must fold in a laboratory environment, which can affect the folding process. Additionally, the protein and the environment need to be pure, and there are limited options on how to measure the synthesised protein properties.

This partly causes the unbalanced dataset problem. As it is difficult to find stabilising mutations, most of the experiments produce destabilising mutations. Hence, the datasets contain mostly destabilising mutations, with a much lower count of stabilising ones. This causes unbalance and leads to problems, especially in machine learning. [25] [31]

Another problem is the noise in the data. Even when the protein is measured in an identical environment like pH or temperature, the resulting $\Delta\Delta G$ value (or other measurements) can differ. This can be caused by different experimental conditions or techniques. Additionally, the protein properties can be measured in different environmental conditions, producing diverse results. Missing information about the conditions increases the uncertainty and noise of the data. [25]

Summarised, to overcome these problems and improve the performance of machine learning methods, the dataset must be thoroughly created and cleaned, keeping in mind possible noise. Additionally, the dataset needs to be appropriately balanced, for example, by inverse mutations [25] [31].

Chapter 5

Basics of Machine Learning and Neural Networks

Machine learning is a fast-evolving field, popular due to its ability to extract hidden knowledge from data and process large amounts of data in a relatively short time [7]. It is a term that incorporates algorithms and methods that allow learning from input data [4] and storing and using created models in computers [7]. The idea is based on the human learning process, i.e. learning from example [4][6]. The main principle is the creation of a model and formulating instructions for modifying the model when the model makes an error [6]. Simplified, the model consists of parametrised mathematical functions, which can take some input, process it and generate results in the correct format. The model is programmed to minimise the error, and in this process, its parameter values are changed to give correct results for the input as it sees more examples [6]. Machine learning often allows to solve different problems using the same basic framework and does not require specialised techniques [4]. Machine learning can solve tasks as classification¹, regression², transcription, machine translation, anomaly detection, synthesis, sampling, denoising, and many others [11].

Based on the problem, machine learning can be divided into four main types [4][8][11][45]:

- **supervised learning**, where the network is trained using annotated examples, also called labelled examples (with known outputs)
- **self-supervised learning**, where correct answers for input data are automatically annotated
- **unsupervised learning**, often called clustering [7], where the result is generated without any labelled examples, and the results are typically some kind of groups; the usual objective of learning is to find a structural aspect of the data to analyse
- **reinforcement learning**, where the machine maximises reward for the previous action given by its creator (or minimises penalties)

While self-supervised learning is usually used for language models or predictors, unsupervised learning can help with data analyses, cleaning and compression of data or clustering [7]. Reinforcement learning is typical for playing games or making recommendations and

¹Classification is assigning an input into one or more given categories.

²Regression is predicting the value of the output based on the input.

has its use in various fields, e.g. robotics or healthcare [7]. Supervised learning is a classic approach to data classification. For that reason, only supervised learning and its algorithms will be elucidated in this thesis.

Thanks to its wide applicability, supervised learning is a common practice that can be used for classification, regression, object detection, image segmentation, and even sequence generation [8]. It is applicable everywhere, where the user can create a training set containing enough labelled data, divided into desired classes that the network learns to recognise. The network processes the inputs and tries to change its own parameters to produce the desired outputs.

The most common machine learning algorithms are linear and logistic regression, decision trees, naive Bayesian, support vector machines or neural networks [7]. Some of these approaches are more analytical, using simple mathematical calculations, and are not considered to be machine learning from a modern point of view. For their potential, neural networks are a popular choice that can bring interesting results. Another reason many people use neural networks is that neural networks do not demand expert knowledge of the domain, and even a beginner can train a good model with a set of high-quality training data. However, expert knowledge is a great advantage, speeding up the process and increasing the chance of a good model [4].

5.1 Understanding the Principle of Neural Networks

Neural networks can be viewed as dynamically learning algorithms consisting of mathematical models [7]. They are based on information processing in the human brain [4][7][45]. The central unit of the artificial neural network is a neuron, depicted in Figure 5.1. The neuron is similar to the biological neurons, though simplified. Biological neurons have some connections to nerves and other neurons, and the strength of the connection is determined by how often the connection is used. Just like these neurons, the artificial neurons have some inputs, and each of these inputs is multiplied by its weight, representing the strength of the connection [45]. If the received signal is strong enough, the biological neurons send the processed signal to other neurons. Synthetic neurons create a sum of received signals, add a constant called *bias*, representing the needed strength of the signal (i.e. threshold), and process it by its *activation function* [45]. This output can then be sent to other neurons or network output. [6]

The computation of the neuron output is described in Equation 5.1. In practice, the resulting sum with bias, also called *logit*, is computed using vector and matrix manipulations because this representation can drastically speed up the computation. For this reason, the principle of a neural network can be understood as a sequence of matrix and vector operations, and the equation can be alternated into Equation 5.2, where W^T is a transposed matrix of weights of individual neurons, x is a vector of inputs of neurons, and b is a vector of biases of neurons. This way, the output of several neurons connected to the network input can be computed, or several input examples for one neuron can be evaluated. [4][6]

$$y = f\left(\sum_{i=0}^n (w_i * x_i) + b\right) \quad (5.1)$$

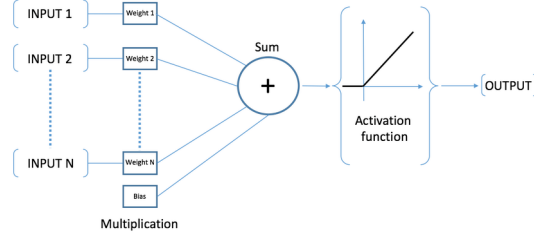


Figure 5.1: Schematics of Artificial Neuron: Neuron takes inputs, multiplies them by their specific weights, sums them, adds a bias and passes it through the activation function to produce the output. The picture was taken without modification from the Machine Learning Approach for Prediction of Hematic Parameters in Hemodialysis Patients [10] with CC BY 4.0 DEED license.

$$y = f(W^T * x + b) \quad (5.2)$$

5.1.1 Activation Function of Neurons

The mentioned above **activation function**, also called **gain function** [45], somehow similar to processing the signal, is extremely important for artificial neurons, as it determines the neuron output. Without the activation function, the neuron's output is linear, equal to the plain logit, which highly limits its usage [6]. To learn more complex and important features of input data, a non-linear activation function can be used. The non-linearity enlarges the searching space [8] and leads to more complex analytical properties, which in principle can solve most of the problems [4]. The most common activation functions are **Sigmoid**, **ReLU** or **Tanh** activation functions [6][7]. The purpose of the activation function is to set the neuron output based on the logit value and sometimes keep the output in a specific interval.

The **sigmoid activation function**, described by Equation 5.3 [4][45] and Figure 5.2, sets the output in $< 0; 1 >$ interval [11].

$$f(x) = \frac{1}{1 + e^{-x}} \quad (5.3)$$

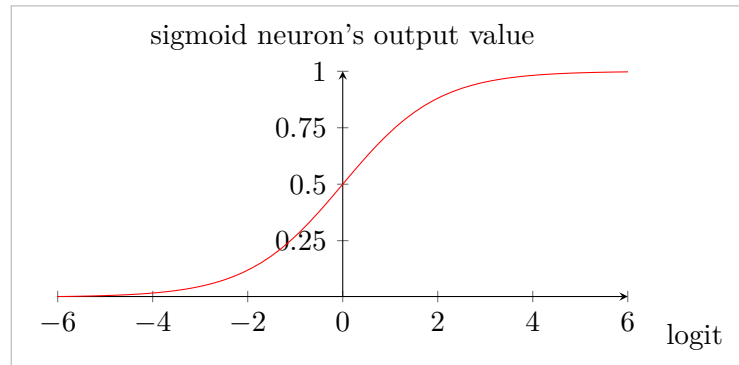


Figure 5.2: The output of a neuron with the sigmoid activation function is based on its logit. Logit near 0 leads to an output value of 0.5, small, negative logit results in an output value close to 0 and large logit yields an output value close to 1 [6].

The **tanh activation function** is expressed by Equation 5.4 and Figure 5.3 and places the output in $< -1; 1 >$ interval. Compared to the sigmoid function, the output value increases faster, reaching output close to 1 even with low logit values. The tanh function was popular in earlier years of machine learning; however, its popularity is dropping. [4][11]

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (5.4)$$

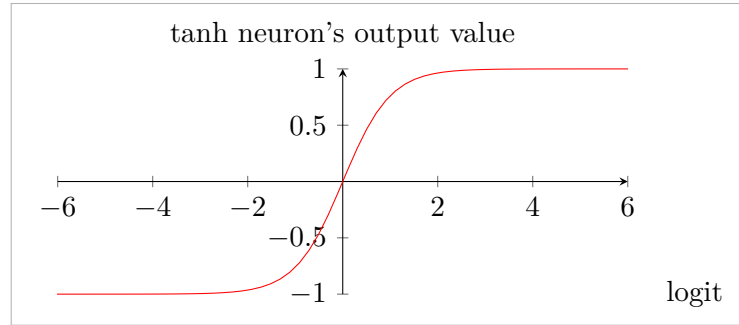


Figure 5.3: The output of a neuron with the tanh activation function is based on its logit. Logit near 0 leads to an output value of 0, small, negative logit results in an output value close to -1 and large logit yields an output value close to 1 [6].

The **ReLU activation function**, from Rectified Linear Unit, uses a function in Equation 5.5 [4] and is depicted in Figure 5.4. The main advantage against the sigmoid and tanh function is that the gradients do not vanish with large values, and it is less sensitive to the random initialisation of the parameters values. It is one of the best-performing activation functions [4]. However, it has problems with negative values, as it causes some neurons not to receive error signals if their value is very low. There are multiple alternations of the ReLU function, e.g. **softplus**, described by Equation 5.6 [4] and graph 5.5, or **leaky Relu**, with Equation 5.7 [4], where α is non-zero for inputs lower than 0; example for $\alpha = 0.5$ is plotted in Figure 5.6. [11]

$$f(x) = \max(0, x) \quad (5.5)$$

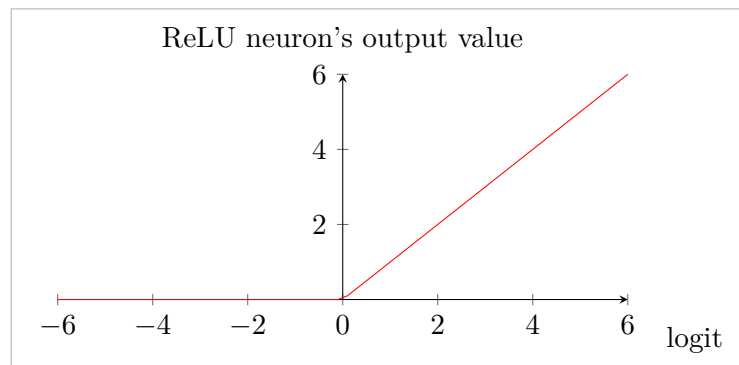


Figure 5.4: The output of a neuron with the ReLU activation function is based on its logit. Logit under 0 is reduced to 0 output value; other logit values are kept and sent to the output [6].

$$f(x) = \ln(1 + e^x) \quad (5.6)$$

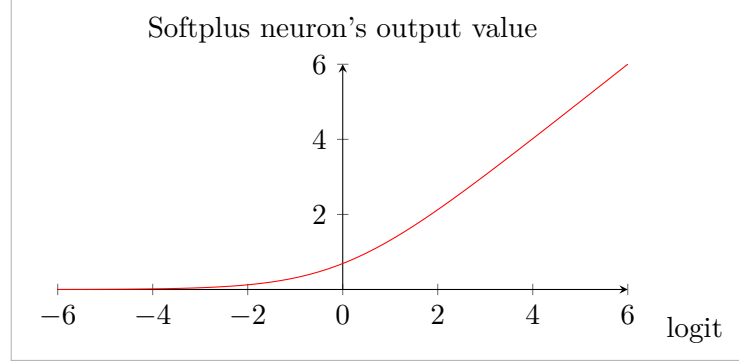


Figure 5.5: The neuron output with the softplus activation function. It can often be viewed as a smoother ReLU function, sometimes called smooth ReLU [4]. With large negative values, the output is close to 0; with increasing values, the output is closer to the input value.

$$f(x) = \max(0, x) + \alpha \min(0, x) \quad (5.7)$$

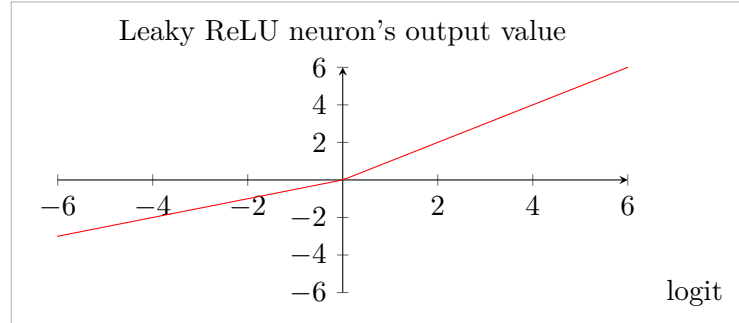


Figure 5.6: The neuron output with the leaky ReLU activation function with $\alpha = 0.5$. The output is equal to the input if the input is greater than zero. Otherwise, it has a small negative value.

A special activation function exists for network output that classifies inputs into two or more distinct classes. This activation function called a **softmax activation function**, is described by Equation 5.8 and converts the outputs to behave like probability – the sum of all outputs equals one and individual outputs express the probability of input belonging into the specific class [4][11]. The only exception is the two-class classification. There, the final number is between 0 and 1, where value 0 indicates the first class, value 1 indicates the second class, and the actual output tells how the network is sure it belongs to the second class, e.g. 0.01 means the network is sure it does not belong to the second class; therefore it belongs to the first one [4]. [6]

$$y_i = \frac{e^{x_i}}{\sum_j e^{z_j}} \quad (5.8)$$

This chapter is based on the books Deep Learning: Foundations and Concepts [4] and Fundamentals of Deep Learning [6].

5.2 Multi-layer Neural Networks and Deep Neural Networks

A network consisting of one or a few neurons is too simple to comprehend and analyse complex data. For this reason, networks with multiple, even tens or lower hundreds *layers* are used, illustrated in Figure 5.7. The consecutive layers can be understood as units performing data transformations, making it easier to solve the task [4]. Each layer contains one or more neurons. Depending on the network type, neurons can be connected in several ways, e.g. to the neurons in the previous layer, to the neurons in earlier layers, or even to the neurons further in the network [6]. A layer can be connected to all of the neurons in the previous layer, referred to as *dense layers* or *fully connected neurons* [4][7], or to only specific neurons in other layers.

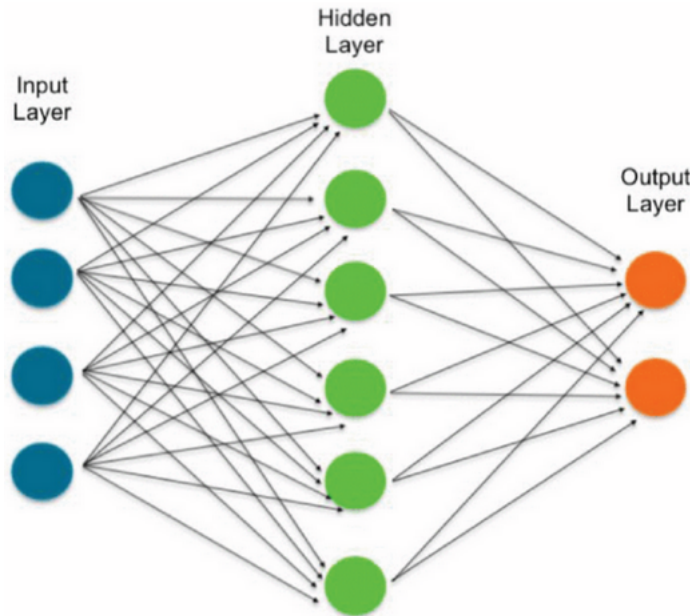


Figure 5.7: Multilayer neural network consisting of three layers: input layer, output layer and one hidden layer. The layers are fully connected, also called dense layers. The picture was taken without modification from the Machine Learning Approach for Prediction of Hematic Parameters in Hemodialysis Patients [10] with CC BY 4.0 DEED license.

Usually, the middle layers, i.e., other than input and output layers, are called *hidden layers*, as the programmer usually does not know what they do or represent in the calculation, and their values do not come from the training set (input or output) [4][7]. Neural networks with more hidden layers are called *deep neural networks* [4][7].

Each neuron in the network has its own inputs and parameters – weights and biases. The activation function is usually different in some network’s layers and can be even different for each neuron, although it is not commonly used. Thus, the larger networks usually require a larger training dataset, as there are more parameters to be trained and estimated [4].

The motivation behind deep networks is that larger and deeper models can perform more operations, and consecutive neurons can reduce the number of neurons in one layer that is required to represent the function. This leads to the reduction of a validation³ error. [11]

5.3 Basics of Neural Network Learning

The initialisation values of weights, biases and other parameters are usually random or somehow preselected [4][7]; however, they do not usually produce the correct result. The network adjusts the parameter's values in order to predict the correct outcome, referred to as neural network learning or network training [4]. During training, the neural network sees a large number of examples and changes its weights to minimise errors made on the examples [6][7]. In supervised learning, the network knows the correct answer and often even the difference between the correct and the predicted answer. Nevertheless, this information does not tell the network which parameters change and on which value. To gain and use this information, several principles can be utilised: loss function, gradient descent and backpropagation algorithm. In essence, the learning process can be summarised as follows:

1. The neural network receives input example(s), and generates output.
2. A loss function quantifies the size of the error between the network's output and expected output.
3. The back-propagation algorithm traverses through the network layers and computes gradients for the parameters and updates the parameter values.
4. These parameter gradients indicate how the parameter's value should be changed to reduce the error by going against the direction of the derivative.

Loss function, also called **cost**, **objective** or **error function**, computes and represents the difference between the correct answer and the network result [4][7][8][11][45]. In the learning process, the loss function is the function in which is minimised in order to get better results [4][8][11] and determines the values of parameters [45]. In other words, it is a measurement of how well the model is performing. The type of loss function depends on the task, e.g. [4][45]:

- cross-entropy – utilised in general classification problems; measures the difference between predicted probabilities and the actual labels
- categorical or binary cross-entropy – special cross-entropy for multi-class and binary classification
- logistic regression – another loss function for binary classification
- mean squared error (MSE) – typically used for regression; measures the average squared difference between the predicted values and the actual values
- least-squares error – used for regression with Gaussian distribution; minimises the sum of the squares of the differences between the observed and predicted values

³For term validation, see chapter 5.4.

and the choice of the loss function is incredibly important as it represents the goal of learning [45]. Choosing the wrong loss function often leads to a poor model.

To know if the parameter value should be lower or higher and how much to change it, **gradient descent** algorithm can be used, which minimises the loss function for given input data [45]. Gradient descent is based on the *derivative* of the loss function. This is possible because all operations in the network are differentiable, which means there is a derivative for all possible values of the function or operation. The derivative maps input values to the slope of the local approximation of the function [11]. In other words, it describes what happens when the parameter value is slightly increased – if the derivative is negative, a small change in the value leads to a lower output value and vice versa. In the context of machine learning, if the goal is minimising the loss function, the parameter values should be changed in the opposite direction from the derivative [4][11][45] – if the derivative is positive, we lower the parameter value; if the derivative is negative, we slightly increase the parameter value. To get deeper insight into derivative for machine learning, see Deep Learning v jazyku Python by F. Chollet on page 57 [8] or other resources. [8]

Gradient is a generalised derivative of a function with more parameters [11]. Since the goal is to minimise the loss function, i.e. finding the lowest point of the function, we want the gradient to be as low as possible and make step in the direction of steepest descent [6]. Hence, the method is called *gradient descent* [11]. In practice, the network process input computes the derivative for current network values and takes a step to lower the gradient, that is, moves the parameters in the direction of the greatest rate of decrease of the loss function [4]. This process is represented in Figure 5.8. [8]

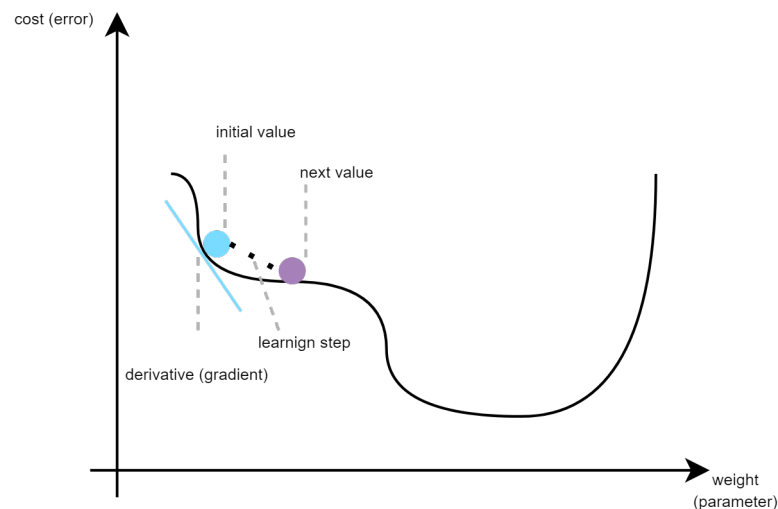


Figure 5.8: Gradient descent illustration for one parameter. The black line represents the loss function (which can be understood as the cost or value of the error). The blue dot represents the initial value of the parameter, the blue line is a derivative in this point, representing the gradient of the parameter. Note that if there were two parameters, there would be one common gradient determined by two derivatives and vice versa. Based on the gradient, the step towards the gradient descent is done, setting a new value of the parameter, depicted as a purple dot.

The problem is that the neural networks do not consist of one operation but of several subsequent operations. These operations can not be differentiated independently but in the correct order. For that, **backpropagation algorithm** is used. The backpropagation algorithm goes through the network [4], evaluating a derivative of the compound function based on *chain rule*⁴, and computes the gradient of all parameter values [4][11]. [8]

The gradient can be computed for the whole dataset, every single training input, or always for part of the dataset. The last two approaches are called **stochastic gradient descent**, as the examples are randomly shuffled [6]. This shuffling is usually done for every **epoch**, i.e. for every complete pass through the whole dataset [4][11]. The most common is the use of a part of the dataset, which is called *mini-batch stochastic gradient descent* [4][45][11], as *batch* is the word for the whole training dataset. The main reason for using mini-batches is that batch gradient descent is highly inefficient and computationally demanding for large networks and datasets [4][11].

The gradient determines if the parameter value should be increased or decreased, but how much the value changed is given by **learning rate**. Learning rate says how big step is taken in the direction of a negative gradient. A lower learning rate causes smaller steps and slower learning; a higher learning rate leads to faster learning, on the other hand, can cause overstepping the minimum and oscillating around the minimum. There are several methods for learning rate adaptation during training, like *AdaGrad* or *RMSProp*, but they are pushed out by more complex optimising algorithms, like *Adam*, which are covered later in this chapter [11]. [4][6]

The learning rate is one of the hyperparameters of the network. The *parameters* are values in a network, which are changed by training the network, like weights or biases. The *hyperparameters* are values used to control the model's behaviour that are not changed by the training process and often are even constant [11]. An example of a hyperparameter is the learning rate, number of layers or number of neurons in the layers. [6]

The best value of the hyperparameters is different for each problem, and the only way to find the finest values is by validating model settings, i.e. setting some values of hyperparameters, training the network and validating its performance on the validation dataset [6]. Then, the hyperparameters which lead to the best-performing model are kept [4][45]. For the validation dataset, we need other data that must differ from training and even from test data [11]. This is done by splitting the dataset into three parts: training, validation and test set [4][7]. **The training set** is used for the model training; it is data on which the network learns. **The validation set** is used to evaluate which hyperparameter values give the best result. That means the model is trained several times with different hyperparameter settings. Put simply, the model with the best accuracy on the validation dataset is chosen. However, with each validation round and tuning, there are some information leaks, and one should be careful and use the validation data as little as possible as it can lead to overfitting to the validation set (see chapter 5.4) [4]. **The test dataset** is used for the final model performance measurement of the chosen model, i.e. for model validation [11]. The model should not see the data in any phase of the training or tuning, and the model should not be changed based on these data. The model's performance on this dataset gives insight into how the model will generalise⁵ the previously unseen data. The parts of the whole dataset should be generated so that all three sets contain representative samples of the data, i.e. all

⁴Chain rule is utilised for computing the derivative of compound functions whose derivatives are known. The chain rule states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$. [11]

⁵Generalisation means it is able to predict the correct outcome even for new, unseen data [4].

possible classes/outputs in similar ratios. The intersection between every two sets should be empty, and the data should not be very similar. [7][8]

The one disadvantage of gradient descent is that it can get stuck in local minimum, saddle point or shallow areas. To overcome this, the **momentum** can be added to the gradient [45]. Momentum is based on the physical laws, and adds some velocity to the parameter, which depends on the slope of the derivative [11]. Thanks to this velocity, the parameter can go through these problematic areas, and get to the global minimum, speed up the process and even suppress oscillating around the minimum [4][11]. The momentum can be defined in several ways; however, it always uses some kind of *momentum (hyper)parameter*. In basic momentum, the parameter has values in interval $<0; 1>$, most commonly value of 0.9 [4][11]. The momentum is usually different for each parameter. The most commonly used momentum is *Adam* (Adaptive Moments), which combines simple momentum with a learning rate adaptation, where the learning rate is lowered by time using a weighted average of parameter derivatives [11][45]. [4]

Determining performance of the model is difficult, as every task has its individual goal and application. One of the simplest and most common measurements is accuracy, as it is usable in several problems, e.g. regression or classification [11][45]. As performance measure can serve any loss function, which is suitable for the task [11]. **Accuracy** says the fraction of correctly classified examples in a given test dataset [11][45]. However, in applications where someone's health or life is at risk, such as illness diagnosis, the mistake of classifying a healthy patient as ill is less serious than classifying an ill person as healthy. For that reason, other measurements than accuracy are used. For binary classification, a confusion matrix is used, although it can be adapted for classification in general. **Confusion matrix** represents more precise results of the testing, as **true positive**, **false positive**, **true negative** and **false negative** rates, illustrated in Figure 5.9 ⁶ [45]. True positive stands for classifying an example from class 1 into the correct class (class 1), for example, classifying an ill person as ill. False positive means the model assigned the example to class 1, even though it belongs to class 0, e.g. says the person is ill, although is healthy, and vice versa 5.9. [4]

		Actual Values	
		1 (positive) e.g. ill	0 (negative) e.g. healthy
Predicted Values	predicted 1 (positive)	True Positive	False Positive
	predicted 0 (negative)	False Negative	True Negative

Figure 5.9: Confusion matrix of two classes, with an example, where lass 1 indicates ill person and 0 indicates healthy person.

⁶It is assumed that class labelled '1' is called positive and the other class is called negative.

From the confusion matrix, many other measurements can be computed, including accuracy (Equation 5.9), precision (Equation 5.10), recall also called true positive rate (Equation 5.11), false positive rate (Equation 5.12), false discovery rate (Equation 5.13) or F-score (Equation 5.14). Assume an example in Figure 5.9, where class 1 indicates an ill person and class 0 stands for a healthy person. **Precision** estimates the probability that a patient marked as ill is truly ill. **Recall** estimates the probability that an ill person will be identified as ill. A **false positive rate** is an expectation that a healthy person is marked as ill. The **false discovery rate** estimates a fraction of patients marked as ill but who are healthy. **F-score** is a geometric mean of precision and recall. The selection of a measurement depends on the task, its context, and its risks. [4][7][45]

To plot model performance, a ROC⁷ curve can be used, depicted in Figure 5.10. The ROC curve plots a true positive rate against a false positive rate, allowing a comparison of these metrics. The comparison and trade-off of these metrics is common practice, especially in biomedicine, as one class is usually more critical than the other. The best possible classification model would touch the left top corner, having 100% true positives and 0% false positives outputs. [4][45]

$$Accuracy = \frac{TruePositive + TrueNegative}{TruePositive + FalsePositive + TrueNegative + FalseNegative} \quad (5.9)$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \quad (5.10)$$

$$Recall = True\ positive\ rate = \frac{TruePositive}{TruePositive + FalseNegative} \quad (5.11)$$

$$False\ positive\ rate = \frac{FalsePositive}{FalsePositive + TrueNegative} \quad (5.12)$$

$$False\ discovery\ rate = \frac{FalsePositive}{FalsePositive + TruePositive} \quad (5.13)$$

$$F - score = \frac{2 * precision * recall}{precision + recall} \quad (5.14)$$

5.4 Neural Network Generalisation and Overfitting: Techniques and Validation

The ultimate goal is to create a network that generalises. **Generalisation** means it is able to predict the correct outcome even for new, unseen data [4][7][11][45]. When adequately designed (in the first step of training), the network usually gives very good results for the training data. However, the outputs the model gives on new, unseen data, can be worse, and the model can not perform well [4]. It is referred to as **overfitting** and is caused by overtraining the model, i.e. training it for too many epochs⁸, or by using a model that is too complex [4][7]. The overtraining causes the network not to learn general features, but it starts to learn microfeatures specific only to the training data [7]. That means it cannot predict the new data because it has different microfeatures – it does not generalise [8].

⁷Receiver Operating Characteristic

⁸Epoch is a term for training a model on all data in the dataset. One epoch equals one pass through the entire dataset.

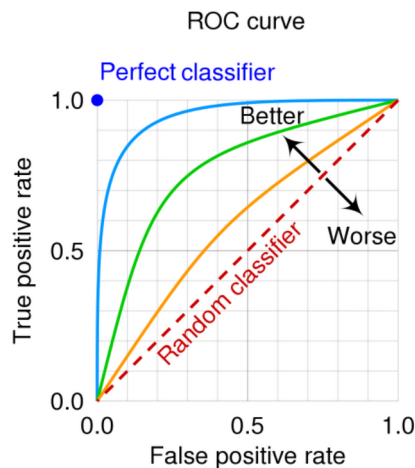


Figure 5.10: Illustration of ROC curve. The dashed line represents a random classifier with an accuracy of around 0.5, leading to a true and false positive rate of 0.5. The blue line represents the best classifier in the example. A perfect classifier would have a true positive rate of value 1.0 and a false positive rate value of 0.0.

Various techniques can avoid or lower the impact of overfitting, like early stopping, regularisation, dropout, data augmentation, parameter sharing, residual connections, multitask learning, model bagging or model averaging [4][11].

5.4.1 Network Size Reduction

Network size reduction is the simplest regularisation method. Bigger networks can learn faster; however, they are prone to overfitting [11]. On the other hand, smaller networks require longer training time with a lower risk of overtraining [8]. The size of the network is given by the number of trainable parameters, often referred to as model capacity [8][11]. The higher the model's capacity, the higher its memorisation capacity [8]. Therefore, to force the model to remember general features, the model can be sized down by lowering the layer count, their size, the number of weights in layers, changing the type of connection between layers or the size of the filter or kernel in case of some layer types [8]. In this process, it is important to validate the model – if the model is far too small, it cannot predict the data as a result [8]. This method can be time-consuming, as there are no general rules and searching for the right network architecture is different for each problem. On the other hand, this method is as important as the detector of model overfitting because the model's quality is strongly connected with its topology. However, models perform the best when their capacity corresponds with the complexity of the task [11].

5.4.2 Early Stopping

Early stopping is a method that measures the performance of the network during training, using validation dataset mentioned in chapter 5.3. The performance is done usually after each epoch. Thanks to the validation, the training can be stopped just before overfitting occurs, i.e. when training accuracy is growing, but validation accuracy is stagnating or lowering, resulting in generalising model. [4][7][6][11][45]

There are other methods called **cross-validation**, where there are multiple runs, always with different dataset splits to the training and validation part. In the result, every sample is used once for validation and (number of runs – 1)-times for training. [4][7][11]

5.4.3 Model Learning Regularisation

All the methods directly avoiding overfitting are referred to as regularisation. The main idea is to regulate how much or which information can be stored in the network and force the model to store the most descriptive information.

Weight regularisation or **weight decay** balances the network’s complexity by keeping the values of the weights low [4][8][6][11][45]. This forces the network to use all inputs a little than use only part of the input values a lot [6]. Values of the weights are given by initialisation and loss function; the regularisation adds a specific penalty to the loss function and influences the resulting weights of the network [4]. There are two types of weight regularisation: L1 and L2. The **L1 regularisation**, also called **Lasso regression**, adds the sum of the absolute values of parameters: Equation 5.15, where the λ hyperparameter stands for the relative importance of the regularisation [4][6][11][45]. The **L2 regularisation**, also called **Ridge regression**, adds the sum of squared values of parameters: Equation 5.16 [4][11][45]. The L1 version allows parameters to be 0, so it can remove insufficient parameters and make the model smaller [6], so it is used to identify which features contribute to the decision [11][45]. The L2 version is computationally faster as the result can be computed using simple math operations and functions [4].

$$lossfunction = Error(y, \hat{y}) + \lambda \sum_{j=0}^J |w_j| \quad (5.15)$$

$$lossfunction = Error(y, \hat{y}) + \lambda \sum_{j=0}^J w_j^2 \quad (5.16)$$

Dropout is a method that can be described as adding noise to the network. For each input, random neurons are „removed“ by setting their activation function to 0, and the activation functions of the rest of the neurons are divided by the probability of neuron removal. This forces the network to not depend only on a few neurons and find more ways to reach the right result [6]. Simultaneously, it helps the network to capture significant patterns and ignore random input features [8]. This technique usually provides good results, and it is common practice to use dropout in the network architecture. [4][11]

Transfer learning is a term for incorporating an already trained network, called *pre-trained network*, into a new network. The idea of transfer learning is that for a particular task, e.g. object classification from images, the learned parameters and representations might be useful for another similar task, e.g. animal classification. Especially in image processing, there are similar features in the early layers that can be shared. So, the input and hidden layers of the pre-trained network are loaded, and new output layers are connected to them. Then, usually, only the added layers are trained to produce the desired output. It was shown that transfer learning can improve model performance, especially in the case of small datasets, and speed up training. However, the tasks need to have the same inputs, and there should be some similarities between them. [4] [11]

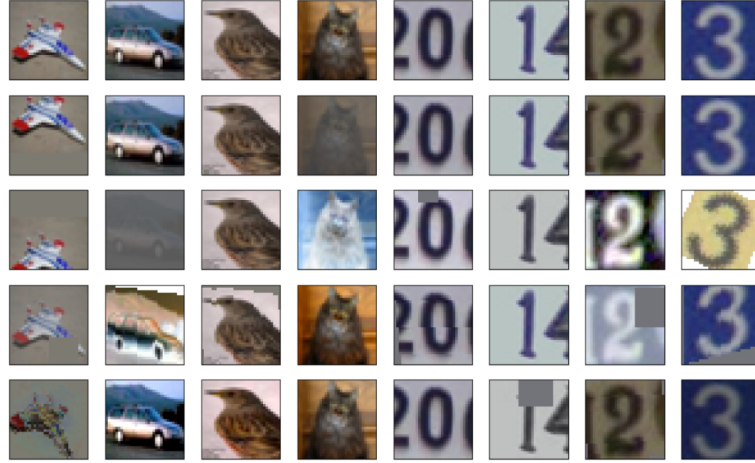


Figure 5.11: Example of data augmentation – shift, rescale, RGB rescale, adding noise. Image is taken from https://bair.berkeley.edu/blog/2019/06/07/data_aug/ in article 1000x Faster Data Augmentation by Daniel Ho, Eric Liang, Richard Liaw, accessed on 4. 4. 2024.

5.4.4 Data Augmentation: How to Extend Dataset without Collecting New Data

In general, having the largest dataset possible, with all possible inputs, is the best technique to create a network with excellent accuracy. However, it is not always possible to create such a large dataset, and even if it is, it would be very time-consuming. For most problems and data, there is a method called data augmentation that allows the programmer to enlarge the dataset without collecting new data. It is generally used for image data, but to some extent, it can be applied to another input type with some changes [4][45]. [11]

Data augmentation expands the dataset by transforming the existing data. For example, for images, the data can be rotated, translated, scaled, RGB scaled, zoomed or cropped, or some noise is added, as illustrated in Figure 5.11. [4]

5.5 Data Preprocessing and Feature Engineering

Data need to be preprocessed in order to feed them into neural network. The preprocessing is specific to the data type, although there are some common practices. The values should be low, typically in interval $< -1; 1 >$ or $< 0; 1 >$ and `float32` type. High values can cause more drastic changes in network weights, which is not desirable, and usually, when more features are examined, there are different value intervals for each property, disturbing the learning process. It is important to convert all features so they have a mean value of 0 and a standard deviation of 1 for the interval $< -1; 1 >$ or a mean value of 0.5 and a standard deviation of 0.5 for the $< 0; 1 >$ interval. A process called normalisation is used to convert actual data into these low values, keeping their distribution. [8]

Raw data are often too broad or complex for model training and must be preprocessed. The right preprocessing can highly influence the outcome. The principle of feature engineering is simplifying the problem and the input. Nowadays, most features are engineered by the network itself; however, choosing which data and format will be put into the network input is still essential.

5.6 Convolutional Neural Networks

Convolutional neural networks are networks with specific architecture designed for image processing. The images can be stored as a list of integers, where each integer is processed by one neuron; however, this approach is not effective and leads to a loss of structure and context of the image. Hence, the convolutional approach was invented. The convolutional layer processes the image as a 2D array and tries to detect some features. In the early layers, the simple and more detailed features are detected; in the deeper layers, more complex features are distinguished. It can be imagined that the early layers detect edges (horizontal, vertical, diagonal, ...), circles or squares; later, object like the eye can be detected, and the deep layers distinguish faces. Hence, they can be understood as analyses of hierarchical abstraction, building the complex features on the simple ones at previous layers: first, they detect edges, then detect eyes, and later, they detect faces [4]. However, this is only an approximate simplification to grasp the concept; in reality, the layer can detect anything, even things that seem not to make sense.

The feature detection is done by *filters*, also called *kernels* [4] [6]. Filters are small 2D arrays or matrices, usually with dimensions (2, 2) up to (7, 7) [6]. It is suggested to keep the size of the filter low, as it keeps a small number of parameters [6]. To improve the representational power, it is better to use more small filters than less large filters [6]. The values in the array determine what the filter detects, illustrated in Figure 5.12, as the maximum output of the filter appears when the input corresponds to the values of the filter [4] [45]. The centre of the filter is gradually placed on each pixel of the input image, computing the output value for this pixel, generating the *output matrix* also called *feature map* [6] [7]. The feature map indicates where the filter's pattern is located in the input [6]. The output value for the pixel is determined by the dot product of the filter and the values of the surrounding pixels in the image, depicted in Figure 5.13. If the image is colourful, the convolution is computed for each channel separately, and the outputs are summed into one output matrix. In each convolutional layer, one or more filters are present. Each filter generates its own feature map, and if there are multiple inputs (e.g. in hidden layers), the filter processes all of them separately [6]. The values of the filters can be specified before the network training but are usually considered trainable parameters, and their value changes during the learning process.

The main advantage of the filters is that the filter's pattern can be recognised anywhere in the figure meanwhile it is trained only once. So, the network can recognise a horizontal edge or a nose anywhere in the picture without the need to see all possible positions of the element in the frame [4] [11] [45]. This also allows using smaller datasets as there are fewer parameters than in other network designs [4]. Another advantage is that with larger input images, the number of parameters increases a little because there is no need for more parameters, m and the only difference is larger feature maps. [4].

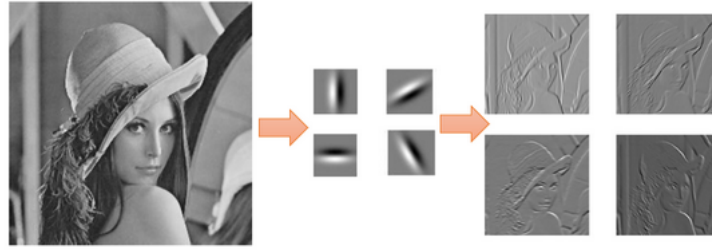


Figure 5.12: Results of application of filters (vertical, horizontal and two diagonals) to a grayscale image. The results correspond to the order of filters.

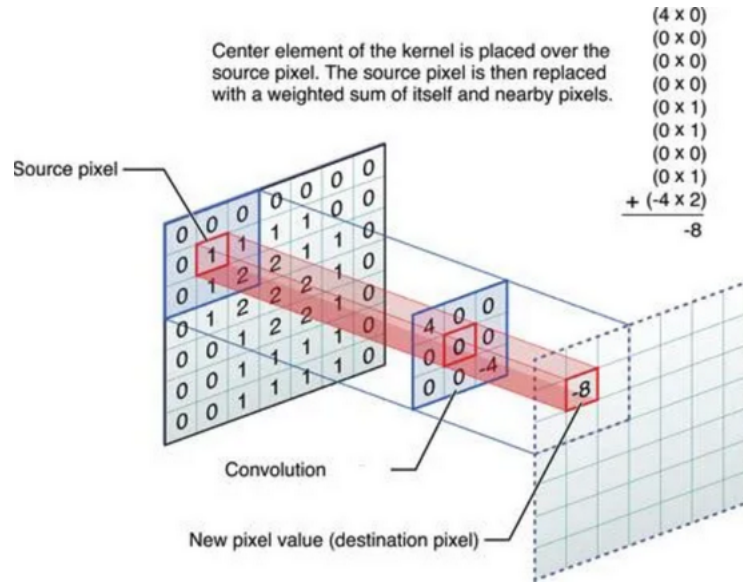


Figure 5.13: The computation of the red pixel using convolution and filter. Each pixel value is multiplied by the corresponding filter value, and these results are summed to resolve the value of the red pixel in the output. The figure was taken from the article 6 Basic Things to Know about Convolution by Madhushree Basavarajaiah, accessed on 23. 2. 2024 from: <https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-dae5e1bc411>.

This way, low-resolution patterns can be recognised. To detect larger features, either the filter must be scaled up, or the image, respectively the feature map, must be scaled down [45]. The convolution operation using a larger filter would be computationally expensive. Hence the scale down of the feature matrix is the used practice. The feature map is scaled down by **pooling layer**, usually *MaxPooling* or *AveragePooling* layer [4] [6] [45]. The basic idea is to divide each feature map into equal tiles of some size, represent these tiles by one number (maximum or average value, respectively) and connect them into a new condensed feature map [6].

The pooling layer is similar to the convolutional layer, with the difference it does not go through all the pixels in the input but skips some [4]. How many pixels are skipped is determined by *stride* [4]. By skipping pixels in the input, the output matrix is necessarily

smaller. Additionally, the output value is not the dot product of the filter and input pixels, but simply the maximal or average value of them, respectively [45]. Their size should be also small, usually between (2, 2) and (6, 6) [45], which define the step between two processed pixels in input. The positive side effect of the pooling layer is that it sharpens the features [6].

After the pooling layer, another convolutional layer can be connected to detect features in the condensed image and vice versa.

Summed up, the convolutional layers extract features using convolution, and the pooling layer downsamples the feature maps. There can be several consecutive convolutional layers, which can bring more detailed representations [11] [6]. A non-linear activation function before the pooling layer can be inserted before the pooling layer [11]. The convolutional layer stacking can bring more rich representations [6]. The main goal can be understood as detecting high-level features or patterns with a resolution as low as possible. Hence, the resulting feature maps (that are later connected to the output layer) usually look totally random; however, for the network, they contain the key points of the input. The logical presumption is the deeper the network, the more sophisticated the patterns can be. The structure of a convolutional neural network is shown in Figure 5.14. The dense layers close to inputs combine information across the whole image [4].

Notice that filters always need some pixels surrounding the central pixel. This brings a problem with the edge pixels, which have no neighbouring pixels on one or two sides. This can be solved in two ways, by a hyperparameter called *padding*: [4]

- No padding, i.e. the edge pixels are left out. This solution is simple, without any additional steps, but it leads to a smaller output feature map, as it misses the edge pixels.
- Padding with some value, usually zero [6] or the average value in the image.

The convolutional layers can also have stride, the same as pooling layers, somehow filling the downsampling role of the pooling layer [4]. However, it is not usually used, and it is recommended to use a stride of size 1, which means processing all pixels [6].

The usage of convolutional neural networks is classification, detection of objects, segmentation, synthesis of new images, improving image resolution, depth prediction, and others [4].

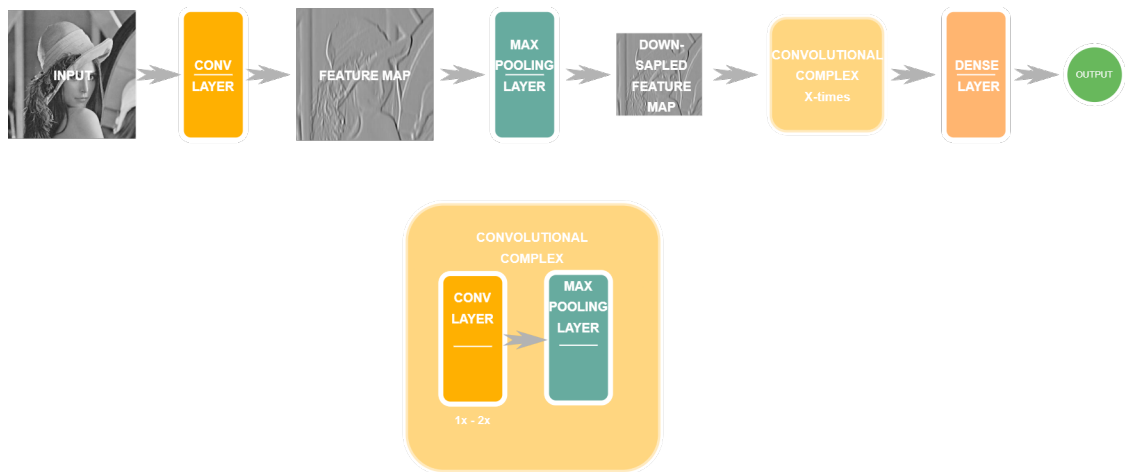


Figure 5.14: General structure of the convolutional network. The input is processed by several convolutional complexes, consisting of one or two convolutional layers and one max- or average-pooling layer. The convolutional layers generate a feature map, and pooling layers downsample it. At the end of the network, there are several dense layers that process the whole last feature map.

Chapter 6

Utilised Programs and Models

As the protein stability datasets are generally small, this work aims to utilise existing algorithms and applications to improve the prediction of the effect of on protein stability. The goal is to make the prediction fast; thus, the utilisation of force fields and other computationally demanding algorithms is not considered.

6.1 ProteinMPNN – Deep Learning-Based Protein Design Method

The ProteinMPNN is a deep learning based sequence designing software, published in 2022 by J. Dauparas et al. [9]. ProteinMPNN generates an amino acid sequence that should fold into a given 3D protein structure. It allows one to design the whole sequence or only part of it.

According to the paper, sequences generated by ProteinMPNN are predicted to fold more accurately to native protein backbone than native sequences, and it has a sequence recovery¹ of 52.4%, as illustrated in Figure 6.1. Compared to other machine learning methods, it generates reasonable sequences for all monomers, assemblies and protein-protein interfaces. The sequences generated by ProteinMPNN should improve the properties of the protein, including stability.

It uses a message-passing neural network consisting of three encoders, three decoders and 128 hidden layers, which predict the sequence based on several protein features:

- distance of residues, measured using C_α (main carbon of molecule)
- orientation and rotation of frame $C_\alpha - C_\alpha - C_\alpha$
- backbone dihedral angles
- distance between nitrogen, main carbon, carbon, oxygen and virtual main carbon of next residue
- encoder edge updates
- random encoding (instead of encoding from N to C terminus), shown in Figure 6.2
- sequence context of ± 32 residues

¹Sequence recovery is reconstructing the protein from experimental data.

- indication whether interacting residues are from the same or different chains

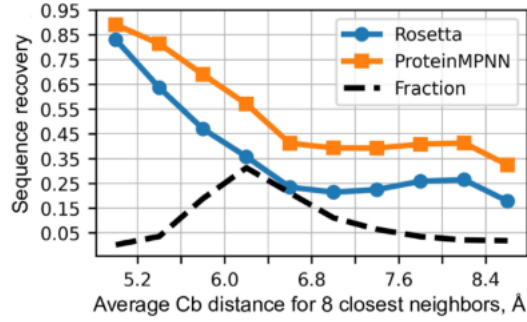


Figure 6.1: Sequence recovery rate of ProteinMPNN, compared to Rosetta based on the average distance of 8 closest residues. The average sequence recovery rate is 52.4%, compared to 32.9% for Rosetta. Figure taken from the article Robust deep learning based protein sequence design using ProteinMPNN [9].



Figure 6.2: ProteinMPNN encoding/decoding. Figure taken from the article Robust deep learning based protein sequence design using ProteinMPNN [9].

The neural network structure is shown in Figure 6.3. The authors also decided to use Gaussian noise during training. This allowed the neural network to focus more on the general properties of the backbone than on local details, achieving robustness of the network, especially for proteins with less detailed 3D structure than atomic level.

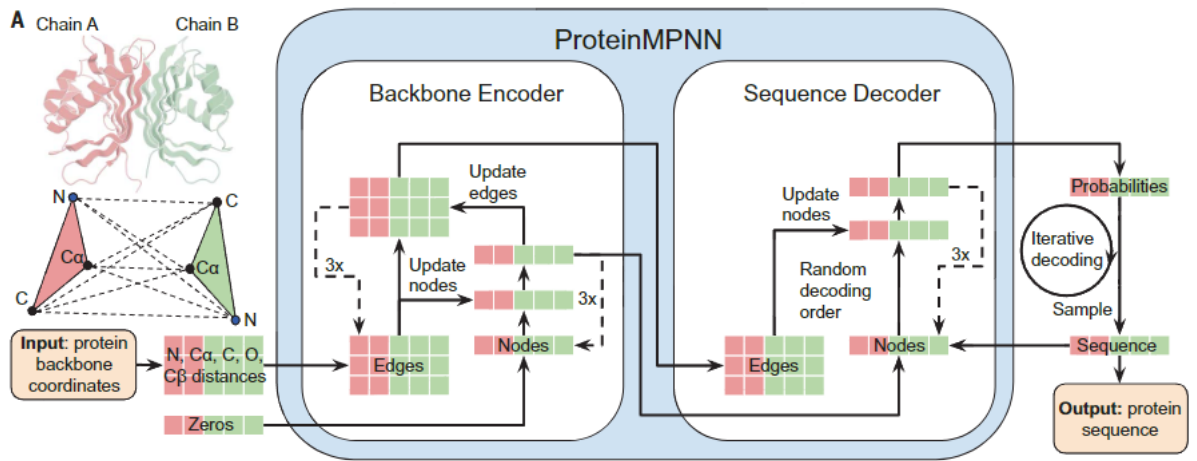


Figure 6.3: The ProteinMPNN network design. Figure taken from the article Robust deep learning based protein sequence design using ProteinMPNN [9].

6.2 Clustal Omega

Clustal Omega is a widely used application for multiple sequence alignment by F. Sievers and D. G. Higgins [39]. Clustal Omega is fast, especially with a lower number of input sequences. According to the authors, the accuracy of the alignment is high compared to the other algorithms, see Figure 6.4.

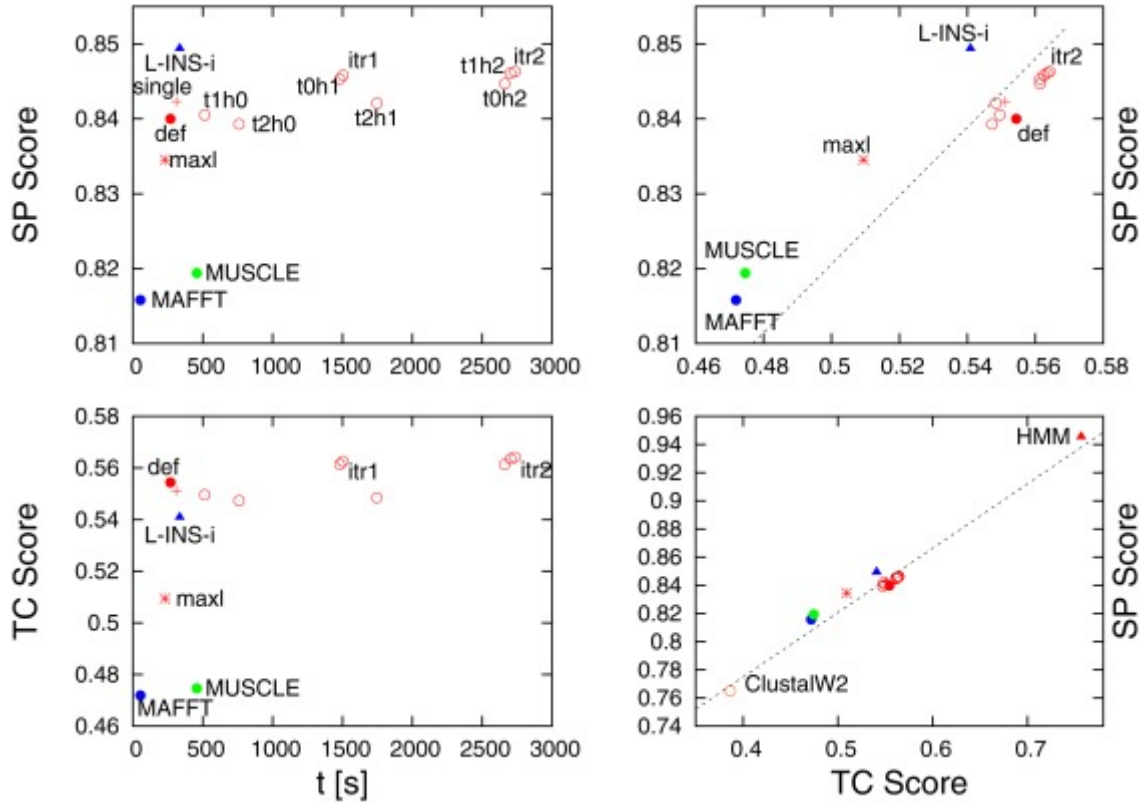


Figure 6.4: Performance of Clustal Omega compared to other software. The solid red dot represents the default settings of Clustal, and other red marks represent various settings. The left top figure shows the SP score of the algorithms. *SP score* stands for the sum-of-pairs and express the fraction of residues pairs that agree in the reference and test alignment. The higher the SP score, the greater the similarity between the sequences. The left bottom figure shows the TC score of the algorithms. *TC score* or Total Column evaluate the fraction of residue columns that are perfectly aligned. Graphs on the right show comparison of SP and TC scores. Figure taken from the article Clustal Omega for making accurate alignments of many protein sequences [39].

6.3 ProMod3

ProMod3 is a tool that can generate a protein structure using homology, i.e. modelling mutant structure based on the mutant sequence and wild-type sequence with known structure. It was presented Gabriel Studer et al. [44]. There are multiple modules that implement various tasks, such as whole sequence modelling or conformation ranking.

The module of interest is the *modelling module*, which models the whole protein structure. The main workflow of the model uses an alignment and a template structure². The structure is used to construct the initial model of the target sequence (the sequence of mutant). Gaps in the alignment are relaxed, and the probable conformation is computed. Sidechains are reconstructed by dihedral angle computation, speeded up by libraries containing the most common and all possible values for amino acids. After that, steric clashes and irregularities are handled by energy minimisation or gradient descent algorithm. The authors state that ProMod3 brings improvement compared to the existing tools. The comparison was made using the IDDT score, which indicates how well a predicted protein structure corresponds to the experimental data, and by MolProbity score, which identifies and quantifies irregularities or errors in protein structure. The ProMod3 increased in IDDT score by 1.51, where a higher value indicates better results, and decreased MolProbity score by 1.37, suggesting an improvement in the protein structure's quality. This means the ProMod3 results are closer to the experimentally measured structures, and the results contain fewer structural irregularities compared to the other methods.

Even though superior and more precise tools like AlphaFold are available, ProMod3 provides results almost immediately, typically within seconds, that are both reliable and of high quality. It stands out as one of the most computationally efficient tools for predicting mutant protein structures. Given the trade-off between the precision and computational time is important, ProMod3 emerges as the optimal choice for this work.

²Template structure is a reference structure for the modelling, usually the wild-type protein structure.

Chapter 7

Design of the Predictor

In such an experimental work, there are many possible approaches, with no recommendation on the best ones or where to begin. After several discussions and experiments, two approaches were selected. The first approach is utilising prior applications, particularly ProteinMPNN, and their combinations. The second approach is the implementation of the new neural network, utilising existing software only for feature extraction and balancing the dataset.

7.1 Dataset

The dataset used in this work was curated in Loschmidt Laboratories and is derived from the FireProt^{DB} [43]. FireProt^{DB} contains data from four sources [43], depicted in Figure 7.1:

- ProTherm database
- ProtBank database
- data collected in Loschmidt Laboratories
- manual mining of the scientific literature

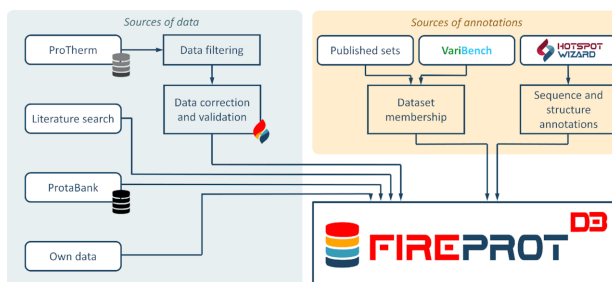


Figure 7.1: Diagram illustrating the aggregation of data into FireProt^{DB}. Figure taken from article FireProtDB: database of manually curated protein stability data [43].

The records were filtered to obtain only single-point mutations, which are not insertion or deletion, with measured $\Delta\Delta G$ or ΔT_m values. Additionally, the protein must have

a PDB code or SwissProt accession code. The mutation residue number corresponds to the Protein Data Bank sequence. [43]

The dataset consists of protein PDB code, a mutation is formatted as **current amino acid - position - new amino acid**, e.g. A26L, and measured $\Delta\Delta G$ or ΔT_m value (never both), which represents the ΔG or T_m value difference against the wild-type protein. Both $\Delta\Delta G$ and ΔT_m values can be used to determine whether the mutation is stabilising or destabilising, however, the two values cannot be easily converted between each other. The stated values in the dataset are averages or medians of multiple measurements for a particular mutation [43].

There are mutations of 262 monomeric proteins¹, from which 5 421 mutations are destabilising and 1 996 mutations are stabilising.

The original dataset had problems discussed in chapter 4.7 – was unbalanced with a higher count of destabilising mutations and was relatively small. This was resolved by the inclusion of the inverse mutations, which can be understood as a type of data augmentation. Incorporating inverse mutations into a dataset is approach lately recommended by several studies, e.g. [25] [31]. The inverse mutations build on the fact that the $\Delta\Delta G$ change in one direction should be in the exact opposite to its reverse mutation, e.g. L $\rightarrow$ A on residue 25 with $\Delta\Delta G$ value of -1.5 have inverse mutation A $\rightarrow$ L on residue 25 with $\Delta\Delta G$ value of 1.5 . In the inverse mutation, the originally mutated sequence is considered a wild-type sequence, and the original wild-type sequence is considered a mutant sequence to keep the correct calculation of the $\Delta\Delta G$ value.

For the data augmented dataset, the structures were generated by ProMod3 and Clustal Omega tools. The process is described in chapter 8.3. During the data augmentation, some entries were removed, because the process of generating mutant structure failed. The final count of entries was 10 118 from 160 proteins.

From the augmented initial training set of 160 proteins and 10 118 mutations, 15% of the proteins, that is, mutations of 26 proteins, were extracted for the validation. By transferring all examples of the specific protein, it is made sure there are no connected examples in the training and validation samples. These proteins were chosen randomly. Note, that each protein has different count of mutations, so the entry count ratio can be different. The training set contains 9554 entries, half stabilising and half destabilising. The validation set contains 564 entries.

The test dataset contains 219 entries, from which 110 are stabilising and 109 are destabilising.

7.2 Design Utilising ProteinMPNN

The first approach is a script utilising and combining several existing software for the prediction, especially ProteinMPNN. The authors claim that ProteinMPNN should generate mutations that enhance protein properties, including stability, and that it is performing the best on single-side mutations [9]. The main idea is to employ ProteinMPNN to construct an initial pool of potentially stabilising mutations to be evaluated by computationally demanding algorithms like Rosetta. This would fasten the calculations and lower expenses as the Rosetta would not be used for mostly destabilising mutations.

¹Monomer is a protein consisting only of one polypeptide chain.

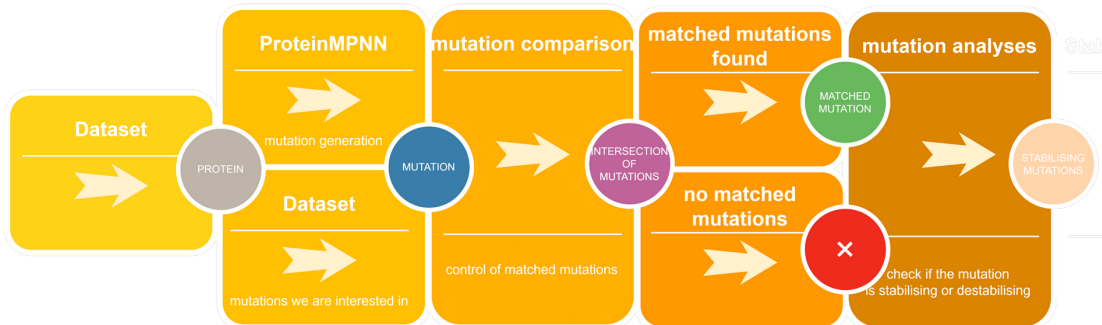


Figure 7.2: The pipeline of the algorithm utilising ProteinMPNN network. The input is a dataset of protein mutations, and the output is whether the mutation is stabilising or not. The decision is made based on the mutations generated by ProteinMPNN, considering these mutations stabilising.

The workflow is illustrated in Figure 7.2. The input is a protein PDB code² and mutation in format **current amino acid - residue number - new amino acid**, for example A24L. The protein code is given on the ProteinMPNN input, which generates multiple mutated sequences, which should enhance the properties of the protein. Next, the mutated sequences are scanned for the desired mutation. If a sequence with this mutation (on the same residue and mutated to the appropriate amino acid) is found, the mutation should enhance the stability of the protein, thus is considered stabilising. The ProteinMPNN generates multiple sequences; thus, the more mutated sequences with the desired mutation, the greater the output's certainty.

The application design contains these parts:

1. Processing the dataset to obtain protein PDB codes, protein mutations, and their measured $\Delta\Delta G$ or T_m (for test purposes).
2. Running ProteinMPNN for listed PDB codes.
3. ProteinMPNN results analyses – searching for desired mutations in output sequences and marking the mutations as stabilising and destabilising.
4. Test script to measure the performance of this algorithm, comparing the results to the dataset stability measurements.

Additionally, a combination of ProteinMPNN and other software with low computing demands, like FoldX and DDMut, can be implemented to improve the accuracy of the resulting model. The idea is that different approaches and algorithms can complement each other.

²PDB code is a simple substitute for the protein name, under which the protein can be found Protein Data Banks.

7.3 Convolutional Neural Network using Protein Properties Graphs

The second approach implemented in this work is training a convolutional neural network. After several discussions, the 2-dimensional graphs of selected protein features were chosen as an input. These features will be computed for both wild-type and mutant proteins. Hence, the mutant structure must be generated using Clustal Omega and ProMod3 software.

The pipeline is shown in Figure 7.3. The protein PDB code and mutation are extracted from the dataset. A wild-type protein sequence and structure are downloaded from the Protein Data Bank. The sequence is mutated, and Clustal Omega generates the sequence alignment. This alignment and wild-type structure is used as an input for the ProMod3 calculation, which predicts the mutant structure. Features are then extracted from the mutant and wild-type structures and graphs are generated and stored. These graphs are then fed to the convolutional neural network, which classify the mutation as stabilising or destabilising.

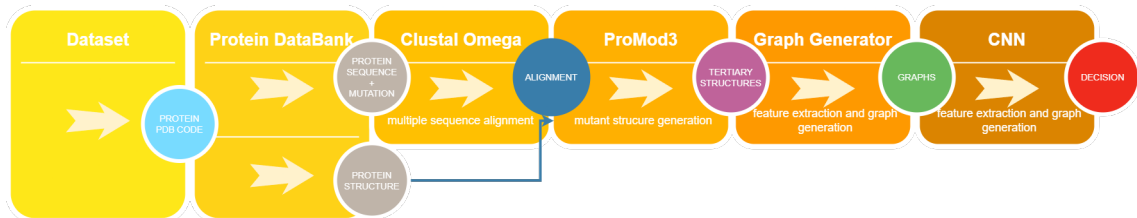


Figure 7.3: The design of pipeline utilising Clustal Omega, ProMod3 and convolutional neural networks. First, protein PDB codes and mutations are extracted from the dataset, and the protein sequence and structure are downloaded from the Protein Data Bank site. From the protein sequence and mutation, the mutant sequence is created, and the alignment of these two sequences is generated using Clustal Omega. This alignment and the wild-type protein structure are fed to ProMod3 to compute the structure of the mutant. From both of these structures, input graphs are generated and are given to the neural network. The neural network output predicts if the mutation is stabilising or destabilising.

The design of the implementation consists of several parts:

1. Extracting data from the database and downloading sequence and structure from Protein Data Bank.
2. Running Clustal Omega and ProMod3 to generate mutant structure.
3. Feature extraction from structures and graph generator.
4. Convolutional neural network model, including model definition and network training.

Due to the presence of two distinct measurements in the dataset, we decided to implement a classifier to categorize data into two classes: stabilising and destabilising. Only if the resulting classifier will have a good performance, a regression will be implemented. For the regression, the dataset would require processing. This could involve using only

records with $\Delta\Delta G$ values or only those with ΔT_m values, or alternatively, applying a suitable conversion to harmonize the data. Note, that this conversion does not exist and the conversion rate is different in different environments.

7.3.1 Convolution Neural Network Design

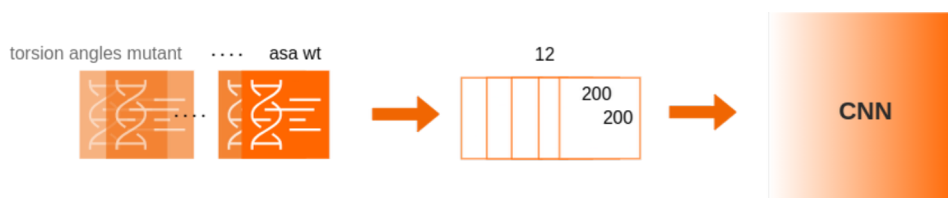
One of the main tasks in the construction of a well-performing neural network is the design, validation and comparison of different architectures and approaches. The networks can differ in input shape, types of layers or their count, used activation or loss functions and many other aspects.

It was decided to test five types of input, depicted in Figure 7.4:

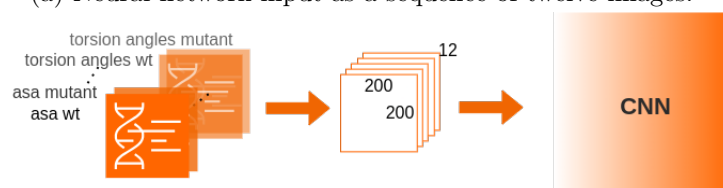
- CNN with one input, graphs stack as a sequence of images.
- CNN with one input, graphs are stored in one image as a third channel.
- CNN with multiple inputs with each graph as one input.
- CNN with multiple inputs with paired graphs as a sequence.

The network structure design is always variable as it is different for each problem or task, and it cannot be estimated which structure will provide the best performance on the available data. The network generally consists of several pairs of convolutional layers – max pooling, where the kernels have small sizes, i.e. 2-5 in each dimension. If there are multiple inputs, e.g. each graph is one input, there should be multiple branches of the same design, which are later concatenated. After that, dense or fully connected layers with a slowly decreasing count of neurons process the output of the convolutional part. The last layer should be a dense layer with one output and softmax activation function or other function that can set the output value between 0 and 1, which expresses the probability of class 1. Between these layers, dropout layers can be inserted.

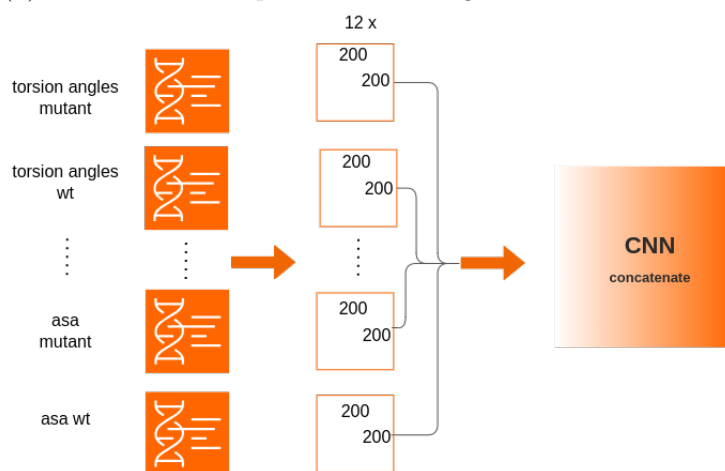
Based on chapter 5.1.1, the activation functions of hidden dense layers can be sigmoid, ReLU or some variants of the ReLU. Probably, some regularisation described in chapter 5.4 will have to be implemented, such as dropout layers and weight regularisation.



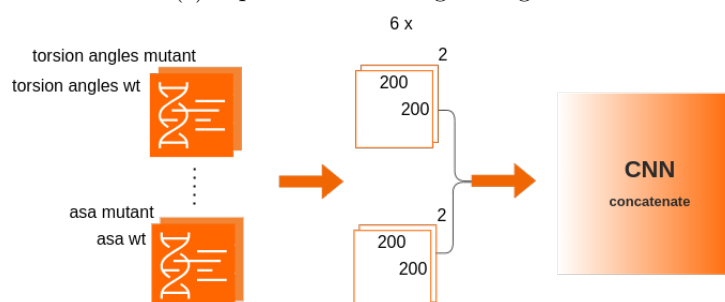
(a) Neural network input as a sequence of twelve images.



(b) Neural network input as twelve images in one, as channels.



(c) Input as twelve single images.



(d) Input as six images, each pair of wild-type and mutant saved in channels.

Figure 7.4: Designs of neural network inputs.

Chapter 8

Implementation, Training and Results

Two distinct approaches were implemented – script utilising ProteinMPNN, and convolutional neural network model utilising ProMod3 and Clustal Omega tools. Both approaches utilised the same dataset, described in the previous chapter. In the first subchapter, the former approach is described. The rest of the chapter focuses on the latter one.

8.1 Implementation Utilising ProteinMPNN

Different settings of the ProteinMPNN algorithm were tested to find the one that is the best for our goal. The *seed* values were tested at interval $<1; 60>$. Usually, steps of size 5 were done, and only promising values were tested in more detail. For the *sampling temperature*¹, the suggested values 0.1, 0.15, 0.2, 0.25, and 0.3 were evaluated, and also higher values up to 0.5 were tried out, as they should bring more diversity. The noise² values were tested on interval $<0; 0.8>$.

To choose the best-performing settings, the confusion matrix mentioned in chapter 5.3 was utilised. Considering class 1 as stabilising and class 0 as destabilising, the measurements were accuracy, the whole confusion matrix, precision, false positive rate and false discovery rate. As the best-performing values were chosen a seed value of 25, sampling temperature value of 0.25 and noise value of 0.5.

Unfortunately, the results were not viable for further use, as can be seen in Figure 8.1. It is apparent, that the output keeps the distribution of the dataset for both $\Delta\Delta G$ and ΔT_m values. Thus, the ProteinMPNN does not generate mainly stabilising mutations. Also, it is obvious that the mutations wrongly marked as stabilising are not even close to being neutral, i.e. being close to the difference of value 0.

¹Sampling temperature argument expresses at which temperature the protein was sampled, as it influences its properties. [9]

²Noise represents the standard deviation of Gaussian noise to add to backbone atoms. [9]

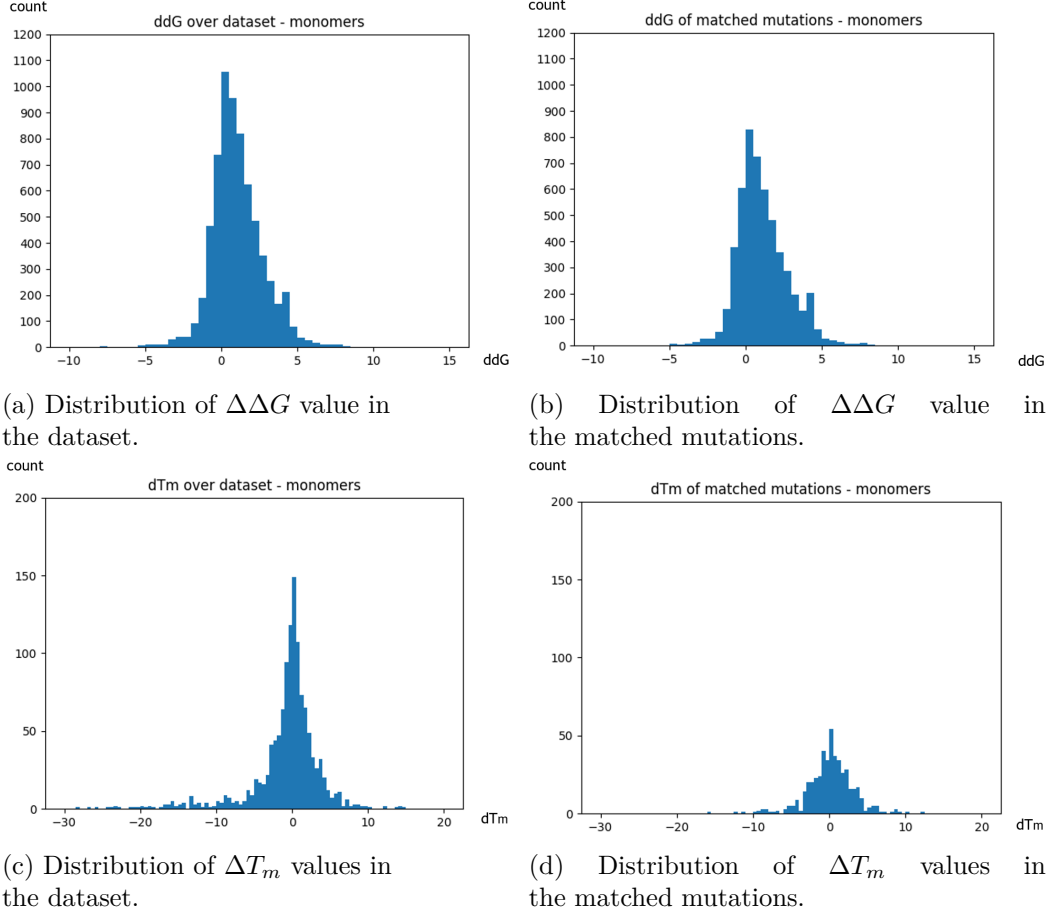


Figure 8.1: The comparison of distribution of values in the dataset and ProteinMPNN results. (a) and (b) shows a distribution of $\Delta\Delta G$ value over the whole dataset and the output, i.e. the matched mutations, respectively. The horizontal axis represents $\Delta\Delta G$ or ΔT_m value, the vertical axis represents count. Note, that the lower the $\Delta\Delta G$ value, the more stabilising the mutation is. (c) and (d) display the distribution of ΔT_m values over the dataset and the output, respectively. The higher the ΔT_m value, the more stabilising the mutation is. The dataset contained $\Delta\Delta G$ or ΔT_m value, not both, so the count of data points is not balanced.

Several other experiments with ProteinMPNN and a combination with other software were done. Especially the combination of ProteinMPNN and FoldX was tested to either see if the false stabilising mutations are actually close to neutral or to use the FoldX total protein energy approximation to filter out the false stabilising mutations. These hypotheses were declined, based on the results of the experiments.

The results of all experiments with ProteinMPNN were unsatisfying, the accuracy was low, equivalent to the chance. It was demonstrated that ProteinMPNN is not suitable for selecting stabilising mutations. Thus, the ProteinMPNN was rejected for prediction of the effect of mutation on protein stability and the rest of the work focused on the neural network model described in chapter 7.3

8.2 Amino Acid Properties Chosen as Input Features

The physicochemical properties of amino acids can be described in many ways, such as hydrophobicity, charge, and size. Containing all these properties in the input for the neural network would lead to a vast dataset, complicating the training of the neural network with high memory demands. Furthermore, some of the properties are correlated – e.g., the size of the amino acid and its mass; therefore, using both of these matrices would be redundant, and it could negatively affect the neural network training. Hence, a properties correlation analysis was done by colleagues in Loschmidt Laboratories to choose the most important properties. Based on the correlation analyses and their expert insight we agreed on six properties:

- torsion angles
- accessible surface area
- hydrophobicity
- isoelectric point
- cysteine bridges
- contact map

These properties will be computed for the close 3D area of the mutated residue, based on the tertiary structure of the protein. The size of the area was chosen at a distance of 10Å from the C_α of the mutated residue.

8.3 Data Preparation, Feature Extraction and Input Graphs

The chosen dataset must be cleaned and processed to generate input images in the desired format. The dataset is stored in an CSV file³, containing protein PDB code, a mutation in format `current amino acid - position - new amino acid` and its influence on protein stability as $\Delta\Delta G$ or ΔT_m value⁴. First, the structure of the wild-type protein is downloaded from RCSB Protein Data Bank⁵, using functions implemented in file `prepare_pdb.py`. Part of this process is editing the file’s content, like deleting lines not containing atom description (other information is not important to us) or changing the indexing of residues to begin on 1. The structure is stored in PDB file⁶, which is a standard for files containing atomic coordinates of the protein. To generate a structure of the protein mutant, the sequence alignment of the wild-type and mutant protein must be generated. The sequence of wild-type protein is obtained from PDB structure, the sequence of the mutant is just altered wild-type sequence, and their alignment is generated by `Clustal Omega`. Then, structures of mutants are generated, using `ProMod3`. This process is implemented in `get_mutants.py`.

³CSV file format is a file format to store records, where commas separate values and newlines separate records.

⁴The description of these metrics is contained in chapter 3.3.

⁵Can be seen at <https://www.rcsb.org/>.

⁶PDB file type serves for storing protein sequence and structure using atom coordinates. It can also contain other information based on the author. The current format documentation can be found on page <https://www.wwpdb.org/documentation/file-format>.

Finally, input graph generation is implemented in the script `feature_extraction_functions.py`. As mentioned above, protein features of interest are torsion angles, hydrophobicity, accessible surface area, distance matrix/contact map, isoelectric point and cysteine bridges.

Torsion angles and **accessible surface area** are computed using corresponding Biopython library functions. **Hydrophobicity** and **isoelectric point** values are taken from GenomeNet Database Resources⁷. **Distance map** is calculated as the distance of centres of geometric masses of each two residues. **Cystein bridges** are found by locating all cysteines and computing torsion angles and distance between each pair of them – angle between $C_\beta - S_\gamma - S_\gamma - C_\beta$ atoms must be 90 ± 15 degrees and distance between atoms in cysteine bridge must be 2.05 ± 0.05 Å between SG atoms, according to article Enhancing protein stability with extended disulfide bonds [20]. Torsion angle graphs have ϕ angle values on axis x and ψ angle values on axis y. Other features have x and y coordinates on the same-named axes, and their values are expressed by the colour of the point (the darker the colour, the lower the value).

Because analysing the whole protein is not necessary and would bring noise (in the context of our approach), the 3D space of the protein can be analysed only closely to the mutated residue. Most forces influence other molecules in a few Å, and even Wan der Waals forces are negligible beyond 5 Å. To ensure none of the influences are overlooked, the distance of 10 Å was defined as the threshold. Therefore, the graphs contain values of the residues in a 10 Å radius.

These graphs are computed for the original wild-type $\rightarrow$ mutant direction, labelled as described in the dataset, and in reverse direction mutant $\rightarrow$ wild-type with opposite label (changing the mutation to be reversed, e.g. A54V to V54A, assigning it to the original wild-type protein).

Due to the fact that each input consists of several independent files, the data loader has to manage their correct linkage. It is possible to preprocess files by merging them outside the data loader and CNN; however, saving these structures into files is overly complicated as they have uncommon shapes. Using a custom data loader is easier and faster without the need to store preprocessed files in the computer for each input type.

The simplest way to load and preprocess CNN input and attach the graphs for each input is using the CSV file. The CSV file contains a list of paths to files containing graphs and their labels in a specific order, as can be seen in Figure 8.2. Each input consists of six types of graphs, each occurring twice, so every input consists of twelve successive records. CSV file generation is described in file `run_cnn.py` in function `create_csv()`.

⁷<https://www.genome.jp/ftp/db/community/aaindex/aaindex1>

	FileName	Label	ClassName
0	./data/train/destab/1EY0_wt_41_asa_destab.png	0	destab
1	./data/train/destab/1EY0_G41T_asa_destab.png	0	destab
2	./data/train/destab/1EY0_wt_41_cys_bridges_destab.png	0	destab
3	./data/train/destab/1EY0_G41T_cys_bridges_destab.png	0	destab
4	./data/train/destab/1EY0_wt_41_dist_matrix_destab.png	0	destab
5	./data/train/destab/1EY0_G41T_dist_matrix_destab.png	0	destab
6	./data/train/destab/1EY0_wt_41_hydrophobicity_destab.png	0	destab
7	./data/train/destab/1EY0_G41T_hydrophobicity_destab.png	0	destab
8	./data/train/destab/1EY0_wt_41_iso_destab.png	0	destab
9	./data/train/destab/1EY0_G41T_iso_destab.png	0	destab
10	./data/train/destab/1EY0_wt_41_torsion_angles_destab.png	0	destab
11	./data/train/destab/1EY0_G41T_torsion_angles_destab.png	0	destab
12	./data/train/destab/1PGA_wt_15_asa_destab.png	0	destab

Figure 8.2: An example of CSV record of data for custom data loader. Each twelve following lines belong to one input sample, every time in a specific order, every type of protein for wild-type and then for mutant – accessible surface area, cysteine bridges, distance matrix, hydrophobicity, isoelectric point, and torsion angles. Each record contains its index, path to the file, numerical label and class name. In the screenshot, mutation G41T of protein 1EY0 can be seen; the last record is the first graph of another input sample.

8.4 Data Loaders

For each design stated in chapter 7.3, a custom data loader had to be implemented. **Keras** and **Tensorflow** libraries offer some data loaders, but none is suitable for our data.

All data loaders are implemented in file `data_generator.py` and consist of a function `load_samples()` for obtaining the images. This function reads the CSV file and loads data. The data loading process differs for each type of input. In the end, the function returns a list of samples, that is, a list of lists of figures, including their labels. Next, after calling `data_generator()` function, for each epoch, it generates batches from loaded data by converting the input images into shades of grey and resizing them to a specified size. The resulting shape also depends on the input type. The data are shuffled before each epoch to ensure stochasticity and diversity in the training. The shuffling can be turned off; however, it is not recommended, as it can lead to worse training. This is especially true for the used data, as the CSV files are organised by label.

Data Loader for Sequence Input

Dataloader for one sequence input design is implemented in class `DataGeneratorSequenceIn1`. The `load_samples()` function loads twelve following graphs for each sample, and the data generator ties the twelve images together. This can be understood as a sequence of images with two dimensions – x dimension and y dimension. The shape of the input is (batch size, 12, size of dimension x, size of dimension y).

Example of usage of the custom data loader for one-sequence input:

```
data_gen = DataGeneratorXIn1(graph_types_per_sample_count=6, img_size=200,
                             scsv_file_path='./data/monomers/csv/training.csv')
```

```
train_generator = data_gen.data_generator(batch_size=50, epochs=10,
    shuffle_data=True)
x, y = next(train_generator)
print(x.shape)
print(x)
```

Its output is shown in figure 8.3

```
(47, 12, 200, 200)
[[[ [1. 1. 1. ... 1. 1. 1.] | [[1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   ...                           ...
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.] ]]]

[[[ [1. 1. 1. ... 1. 1. 1.] | [[1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   ...                           ...
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.] ]]]

[[[ [1. 1. 1. ... 1. 1. 1.] | [[1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   ...                           ...
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.] ]]]

[[[ [1. 1. 1. ... 1. 1. 1.] | [[1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   ...                           ...
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.]
   [1. 1. 1. ... 1. 1. 1.]    [1. 1. 1. ... 1. 1. 1.] ]]]] ]]
```

Figure 8.3: An example of the output of data loader, specifically sequence input data loader. The first line is the shape of the first batch (the batch size in this example is 47, 12 graphs in one input, and each has a size of 200 x 200); the rest are records. Note that the values in the list are always 1., because these are values of pixels, and almost all input graphs are empty on their edges, different values are often somewhere in the middle of the graph.

Data Loader for Input as One Image with Twelve Channels

Data loader for storing graphs into one image as its channels is described in class `DataGeneratorChannelsIn1`, and it is very similar to one sequence input version. The data loader also gets twelve consecutive graphs; however, it loads all twelve images into one image with three dimensions. This is not uncommon; all colourful images have three dimensions – the x dimension, y dimension and channel dimension, where colours are stored, typically red, green and blue, as can be seen in Figure 8.4 [4]. Images with twelve channels cannot be displayed or easily stored as images, but it is easy to work with them as arrays of floats. As a result, the input data have shape (batch size, image size, image size, 12), which seems to be almost the same as the previous input shape, although in neural network processing, there is a difference. Especially convolutional layers expect shape (axis x size, axis y size, colour channels), and this input is processed precisely this way. The input of the previous data loader is handled with different layers and approaches.

The usage is the same as in the previous data loader, with the difference that class `DataGeneratorChannelsIn1` is called for `data_gen` initialisation.



Figure 8.4: RGB channels of a colourful image. Each channel corresponds to the value of the corresponding colour. These three colours can together create any colour with the right values.

Data Loader for Input as Twelve Single Images

Data loader for data as twelve images is implemented in class `DataGeneratorXInputs`. There are two implementations: one converts the images into greyscale, and the second keeps the colour channels of the images. The former was used for our own neural network, and some cases of transfer learning, and the latter was used for most of the transfer learning experiments. The `data_generator` function loads twelve lists of examples, each for one type of graph, and then connects them. Hence, the output is a list of length 12 containing twelve input batches. These inputs have shape `(batch size, image size, image size, 1 or 3)`; where the last number of the shape is 1 for greyscale and 3 for colourful inputs. This dataloader also has function `test_data` for loading data during model evaluation.

Data Loader for Input as Six Single Images with Two Channels

Data loader for input as six images with two channels is implemented in class `DataGeneratorMultipleInputs2In1`. It is similar to the previous one, with the difference the output list has only a length 6, as each of the inputs is for a different amino acid property, and the images have two channels, where the first one is the graph of the property for the wild-type protein and the second one is the graph for the protein mutant. The shape of the inputs is (batch size, image size, image size, 2).

8.5 Neural Network Implementation

Tens of neural network structures were tested for each input type. The network always consisted of several convolutional layers with max-pooling layers or average-pooling layers. If there are multiple inputs, there are several concatenated input heads, followed by a few dense layers follow. The output is a dense layer with a sigmoid activation function.

First, a network with three blocks of convolutional layer and max-pooling complexes was implemented, followed by three dense layers, shown and described in Figure 8.5. By default, the ReLU activation function was used in all possible layers except the last one, as it is recommended by literature [4]. The padding in convolutional layers was set to 'valid'. The default loss function was binary cross-entropy, the metrics binary accuracy and no optimiser. This model structure was tested several times with different settings, after which a few layers were added, either convolutional-max-pooling block, dense layers or both. The tested settings were some combinations of these:

- size of the convolutional filter set on (3, 3), (5, 5) or (7, 7), depending on the depth of the layer
- size of the max-pooling or average-pooling filter set on (2, 2) and (3, 3)
- max-pooling and average-pooling
- activation function ReLU, sigmoid, softplus and leaky ReLU
- loss function binary cross-entropy and logistic regression
- none and Adam optimiser
- number of epochs from 5 to 30; however, the training was often stopped earlier, as the loss stagnated on the same value for three epochs

Usually, one run was done without an optimiser and the other with the Adam optimiser. Similarly, most of the runs were done with the max-pooling layers. Also, the binary cross-entropy loss function was usually used, as the logistic regression did not seem to bring any improvement.

However, none of the tested models did learn anything. As mentioned earlier, the loss value stagnated around value of 0.6 almost every time. The binary accuracy was 0.52 at the best, which can not be considered as a satisfying result. Networks up to the size of 100 layers for one input and to the size of 50 for multiple inputs were tested. The network was not overtraining, so it was not caused by small dataset.

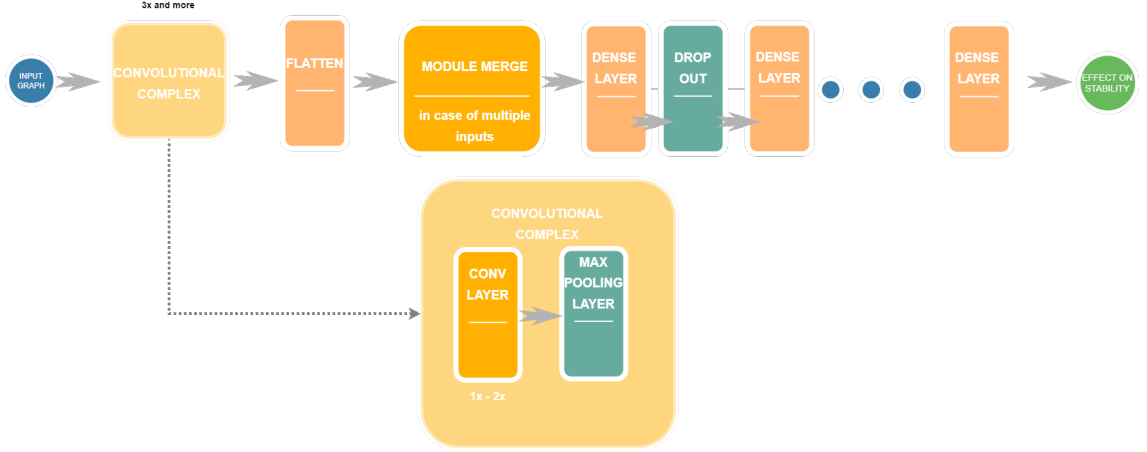


Figure 8.5: General design of the implemented neural networks. The input is processed by several convolutional complexes. If there are multiple inputs, there are more heads, which are later merged. Then, the feature map is processed by several dense layers with dropout layers. The last layer is a dense layer with one output.

8.6 Neural Network using Transfer Learning Implementation

After the unsuccessful attempts with a completely new neural network, the pre-trained neural networks for image processing were utilised. The hesitations about this approach were that the networks are pre-trained on photos or images with some meaning; meanwhile, in this work, the images are graphs, often containing only a few dots. Hence, the concern was that this approach would not bring the desired result. However, this approach proved as suitable. The tested networks for transfer learning are⁸:

- VGG16
- VGG 9
- ResNet50
- ResNet50V2
- InceptionResNetV2
- EfficientNetV2B0
- EfficientNetV2S
- EfficientNetV2M
- EfficientNetV2L

These pre-trained networks input is always one image with a specific size, either greyscale or colourful. So, for the transfer learning experiments, only a design with twelve inputs

⁸The list of pre-trained networks available in `Keras` library was taken from web article <https://keras.io/api/applications/>, accessed on 9. 1. 2024.

was applied. As illustrated in Figure 8.6, the network always has twelve heads, where each leads to a copy of the pre-trained model. From these pre-trained models, the top layer is removed, and the training is turned off. The layers must also be renamed to avoid naming conflict. The last layer of the network is flattened, and the individual models are concatenated. This concatenated layer is connected to several dense and dropout layers. The last layer is also a dense layer with one output and the sigmoid activation function. The tested settings were:

- number of dense layers from 3 to 10
- size of the dense layers
- none and Adam optimiser
- activation function of dense layers ReLU, sigmoid, softplus and leaky ReLU
- loss function binary cross-entropy and logistic sigmoid

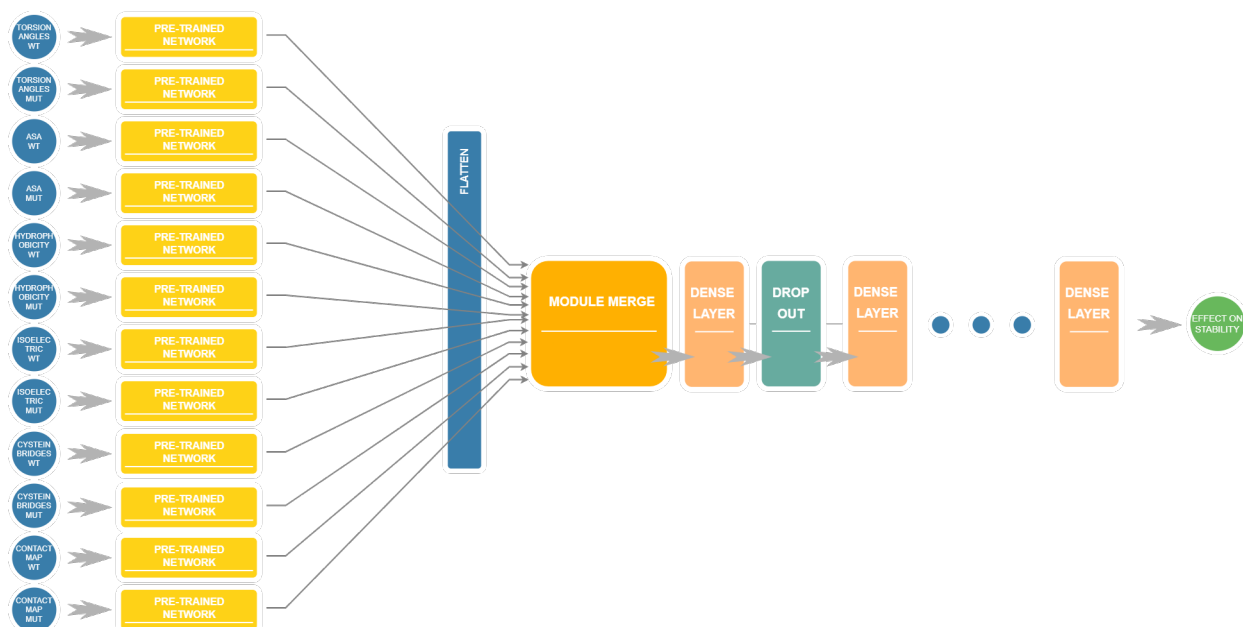


Figure 8.6: General design of transfer learning neural network. The model has twelve inputs, each for one protein property either for wild-type or mutant protein. Each input is processed by a pre-trained network, which is flattened at the end. The parts are concatenated and followed by dense layers and dropout layers.

To include multiple copies of the same pre-trained network, the layers of the copy must be renamed. Surprisingly, there is no such possibility directly in the libraries and the renaming is done manually by creating a new layer object. This process is implemented in `rename layers` function, and was taken from Navidreza Asadis' implementation⁹ and edited.

⁹The code is available at <https://nrasadi.medium.com/change-model-layer-name-in-tensorflow-keras-58771dd6bf1b>, created on 5. 5. 2023 and accessed on 25. 2. 2024.

First, all the pre-trained network models were run a few times to find if some of them could be useful. Only one network got an accuracy higher than 0.52: **ResNet50V2** with an accuracy of 0.86. This model was chosen for tuning, but during the tuning, it became obvious that there were some complications, likely shallow points or local minimum, described in more detail in the next chapter 8.7. The success with training the model was not repeated until 33rd attempt of a full training session of the training. After this discovery, other tests were done with other pre-trained models and even with newly designed neural networks described in the previous chapter. Some of these tests led to network learning; however, the **ResNet50V2** model seemed to give the best results, and the training was often successful compared to the other models.

With some adjustments of the settings described higher, the accuracy of the network on the test dataset hit a value of 0.99. Hence, regularisation was implemented:

- dropout layers between different dense layers with probability from 0.1 to 0.5
- L1 regularisation with importance value from 1^{-6} to 1^{-1}
- L2 regularisation with importance value from 1^{-5} to 1^{-2}
- L1L2 regularisation with importance value from 1^{-4} to 1^{-2}

8.7 Complications with Neural Network Training

The main complication is the apparent **existence of a plateau** – shallow point, saddle point or local minima. Local minimum in so high-dimensional space is not probable, although saddle points with gradient zero can occur [4] [6] [11]. The zero gradient causes the network not to learn [4]. To overcome saddle points in neural network learning, it is recommended to use a dynamic learning rate or some optimiser [4] [6] [45]. Adding an optimiser helped to defeat the plateau in some of the cases. However, it did not solve the problem entirely, as the training process depends on the weight value initialisation and how the data shuffled between epochs¹⁰. This led to complicated hyperparameter testing, as each scenario required several attempts, up to 20, to train the network successfully. One training session took several hours, depending on how fast the training loss stagnated on the same value. This highly slowed the model tuning and made it more complicated.

There was also a **problem with a training-validation dataset split**. At first, random proteins were separated from the whole dataset and put into the validation dataset. This led to a situation where the validation dataset was easier to fit, and the validation accuracy was higher than the training accuracy. It surely was not an indication that the model fit the data very well – with this belief, a few test samples were given to the network, which proved it gave worse results on the test data. Hence, I analysed the proteins in the training dataset and moved one of the more complex proteins to the validation dataset. This caused the model not to see enough complex protein examples, and it caused dropping both the test and validation accuracy without the desired result. It was due to the fact that there are mostly simple proteins in the dataset, and removing all the examples of one complex protein from the training dataset highly influenced the model’s training. To keep the model’s performance, it was decided to continue with the simple validation dataset, even though it was harder to choose the best model. This led to the decision to create a second validation

¹⁰As the used mini-batch gradient descent count the loss on several examples, the learning process can be influenced by the order of training data.

dataset, consisting of the data from the original validation dataset mixed with a few data taken (and removed) from the test dataset.

Additionally, almost all regularisation techniques led to lowering the validation accuracy score during training, but also to lowering the performance on the second validation dataset. Thus, the regularisation did not have the desired effect, i.e. overcoming overfitting and improving the performance on validation dataset.

8.8 Results

As the final model was chosen models, with validation accuracy of 0.8. The model has twelve inputs, each for one graph. Each of these inputs are connected to the ResNet50V2 pre-trained network. The outputs of these pre-trained networks are concatenated, followed by two dense layers with ReLU activation and of size 256 and 128, respectively, one dropout layer with probability 0.2 and a final dense layer with sigmoid activation function. The model contained Adam optimiser and no weigh decay regularisations. Then, this model was tested by test dataset, containing unseen data, with some difficult examples. The test dataset contains 219 entries with 110 stabilising and 109 destabilising mutations.

The model has binary accuracy of 0.7001 and binary crossentropy loss of 1.7 on the test data. The confusion matrix is:

	True Positive	True Negative
Predicted Positive	0.68	0.27
Predicted Negative	0.32	0.73

Precision, i.e. probability that a mutation marked as stabilising is truly stabilising, is 72%. False positive rate is 27%, and 28% of destabilising mutations are marked as stabilising. The ROC curve of the model is show in Figure 8.7.

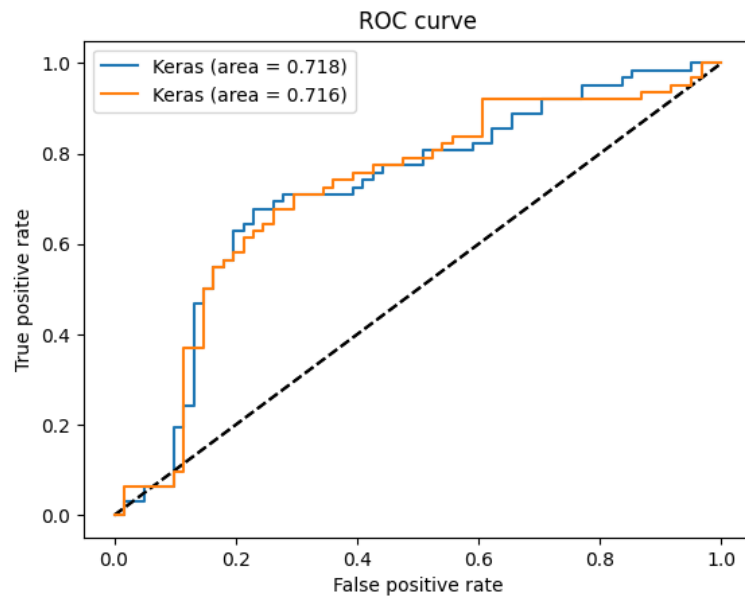


Figure 8.7: The ROC curve of the model.

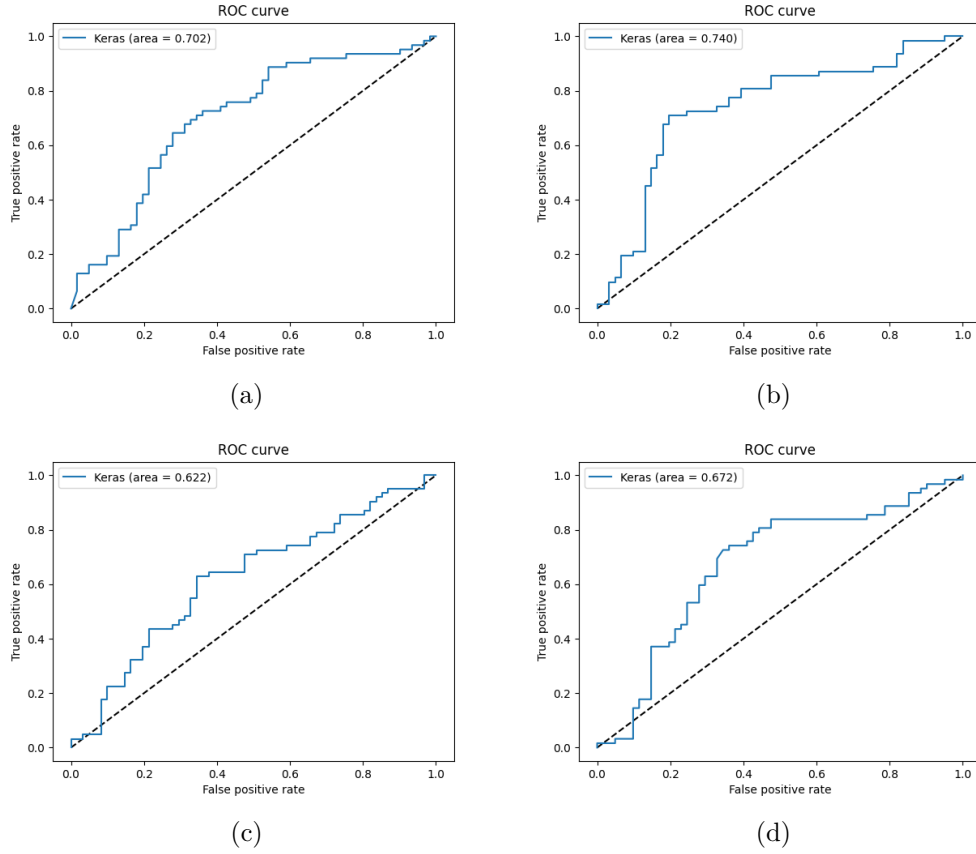


Figure 8.8: ROC curves of some of representative models, which were not chosen as final ones. (a) A model with two dense layers and L2 regularisation. Its validation accuracy was 0.76, with bias towards destabilising mutations. (b) ROC curve of model incorporating 3 dense layers and validation accuracy of 0.78. This model was another close candidate for the final model. (c) ROC curve of model with two dense layers, before which is always drop layer, and with L1L2 regularisation. The validation accuracy was 0.69 (d) Model with 2 dense layers and 1 dropout layer with propability of 0.1. Its validation accuracy was 0.72.

Providing an exhaustive description of all successfully trained models would be impractical due to their quantity. Therefore, Figure 8.8 shows representative examples of other models illustrated by their ROC curves.

Chapter 9

Conclusion

The goal of this work was to create a predictor of stabilising and destabilising protein mutations, utilising neural networks and existing tools. This aim was fulfilled.

To understand the concept of protein stability, I did an in depth examination of the protein structure, folding and stability, including bonds forming the proteins. These information are summarised in chapters 2 and 3. The current methods for predicting protein stability changes upon mutations were analysed. The methods for the prediction or improving protein properties and their evaluation is stated in chapter 4. I also did research on machine learning methods with focus on convolutional neural networks, including measurements of performance, summed up in chapter 5. The design of the experiments, dataset and network are described in chapter 7, and the final implementation, together with results, is described in chapter 8.

Two approaches were explored: one utilizing the ProteinMPNN tool, and the other employing a convolutional neural network architecture, which incorporates the ProMod3 framework and utilizes a pre-trained ResNet50V2 network. The former approach results are not viable, the latter model classifies the mutations with an accuracy 0.7. While it is possible that better results could be achieved with this dataset, it's worth noting that the performance observed may already represent the upper limit achievable using this dataset [25]. The model fulfills the role of selecting stabilising mutations for further analyses. Additionally, the result is not only the final class, but also the score on which it was decided, i.e. the percentual value of how sure the network is the input belongs to the stabilising class.

The training and tuning time was surely underestimated, same as the possible shallow areas and other problems connected with the training. Due to the computational time spent on the classification network tuning, the regression variant was not explored. However, the result is valuable and suitable for the desired use.

To improve the performance of the network, there is a possibility to train the network using more data. There is also a possibility to convert the measurements of mutation influence on protein stability and change the task to regression. For the regression, only the last layer of the network can be changed and trained. However, the improvement of the network is limited mostly by the amount of available data, which is the current problem of mutation focused proteomics.

Bibliography

- [1] ADDABBO, R. M., DALPHIN, M. D., MECHA, M. F., LIU, Y., STAIKOS, A. et al. Complementary Role of Co- and Post-Translational Events in De Novo Protein Biogenesis. *The Journal of Physical Chemistry B*. 2020, vol. 124, no. 30, p. 6488–6507. DOI: 10.1021/acs.jpcc.0c03039. Available at: <https://doi.org/10.1021/acs.jpcc.0c03039>.
- [2] ANFINSEN, C. B. Principles that Govern the Folding of Protein Chains. *Science*. 1973, vol. 181, no. 4096, p. 223–230. DOI: 10.1126/science.181.4096.223. Available at: <https://www.science.org/doi/abs/10.1126/science.181.4096.223>.
- [3] BERG, J. M., TYMOCZKO, J. L. and STRYER, L. *Biochemistry*. 5th ed. W.H. Freeman and Co., 2002. ISBN 0716730510.
- [4] BISHOP, C. M. and H., B. *Deep Learning: Foundation and Concepts*. Springer, 2023. ISBN 978-3-031-45467-7.
- [5] BRAMER, S. V. *Widener University: CHEM 135*. Widener University, january 2024.
- [6] BUDUMA, N., BUDUMA, N. and PAPA, J. *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*. 2nd ed. O’Reilly Media, Incorporated, 2022. ISBN 978-1-492-08218-7.
- [7] CANNATARO, M., GUZZI, P. H., AGAPITO, G., ZUCCO, C. and MILANO, M. *Artificial Intelligence in Bioinformatics: From Omics Analysis to Deep Learning and Network Mining*. Elsevier, 2022. ISBN 978-0-12-822952-1.
- [8] CHOLLET, F. and PECINOVSKÝ, R. *Deep learning v Jazyku Python: Knihovny Keras, tensorflow*. 1st ed. Grada Publishing, 2019. ISBN 9788024731001.
- [9] DAUPARAS, J., ANISHCHENKO, I., BENNETT, N., BAI, H., RAGOTTE, R. J. et al. Robust deep learning-based protein sequence design using ProteinMPNN. *Science*. 2022, vol. 378, no. 6615, p. 49–56. DOI: 10.1126/science.add2187.
- [10] DECARO, C., MONTANARI, G. B. and AL. et. Machine Learning Approach for Prediction of Hematic Parameters in Hemodialysis Patient. *IEEE Journal of Translational Engineering in Health and Medicine*. september 2019, vol. 7. DOI: 10.1109/JTEHM.2019.2938951. ISSN 2168-2372. Available at: <https://ieeexplore.ieee.org/document/8839068>.
- [11] GOODFELLOW, I., BENGIO, Y. and COURVILLE, A. *Deep Learning*. MIT Press, 2016. ISBN 9780262035613.

- [12] GROMIHA, M. M. *Protein Bioinformatics: From Sequence to Function*. 1st ed. Academic Press, 2010. ISBN 9788131222973.
- [13] HOCHBERG, G. K. A. and THORNTON, J. W. Reconstructing Ancient Proteins to Understand the Causes of Structure and Function. *Annual Review of Biophysics*. may 2017, vol. 46, p. 247–269. DOI: 10.1146/annurev-biophys-070816-033631.
- [14] HOU, Q., ROOMAN, M. and PUCCI, F. Enzyme Stability-Activity Trade-Off: New Insights from Protein Stability Weaknesses and Evolutionary Conservation. *Journal of Chemical Theory and Computation*. june 2023, vol. 19, no. 12, p. 3664–3671. DOI: 10.1021/acs.jctc.3c00036.
- [15] JIAJIA, F., SU, J., WANG, P., YUANYUAN, Y., WANG, Q. et al. Enzymatic processing of protein-based fibers. *Applied Microbiology and Biotechnology*. december 2015, vol. 99. DOI: 10.1007/s00253-015-6970-x.
- [16] LAZARIDIS, T. and KARPLUS, M. Effective energy functions for protein structure prediction. *Current Opinion in Structural Biology*. 2000, vol. 10, no. 2, p. 139–145. DOI: [https://doi.org/10.1016/S0959-440X\(00\)00063-4](https://doi.org/10.1016/S0959-440X(00)00063-4). ISSN 0959-440X. Available at: <https://www.sciencedirect.com/science/article/pii/S0959440X00000634>.
- [17] LAZARIDIS, T. and KARPLUS, M. Effective energy functions for protein structure prediction. *Current Opinion in Structural Biology*. 2000, vol. 10, no. 2, p. 139–145. DOI: [https://doi.org/10.1016/S0959-440X\(00\)00063-4](https://doi.org/10.1016/S0959-440X(00)00063-4). ISSN 0959-440X.
- [18] LEVINTHAL, C. How to fold graciously. *Mossbauer Spectroscopy in Biological Systems Proceedings*. 1969, vol. 67, no. 41, p. 22–26. Available at: https://faculty.cc.gatech.edu/~turk/bio_sim/articles/proteins_levintal_1969.pdf.
- [19] LI, B., YANG, Y. T., CAPRA, J. A. and GERSTEIN, M. B. Predicting changes in protein thermodynamic stability upon point mutation with deep 3D convolutional neural networks. *PLoS Computational Biology*. november 2020, vol. 30. DOI: 10.1371/journal.pcbi.1008291.
- [20] LIU, T. e. a. Enhancing protein stability with extended disulfide bonds. *Proceedings of the National Academy of Sciences*. may 2016, vol. 113, p. 5910–5915. DOI: 10.1073/pnas.1605363113.
- [21] MECHA, M. F., HUTCHINSON, R. B., LEE, J. H. and CAVAGNERO, S. Protein folding in vitro and in the cell: From a solitary journey to a team effort. *Biophysical Chemistry*. 2022, vol. 287, p. 106821. DOI: 10.1016/j.bpc.2022.106821. ISSN 0301-4622. Available at: <https://www.sciencedirect.com/science/article/pii/S0301462222000631>.
- [22] MENDES, J., GUEROIS, R. and SERRANO, L. Energy estimation in protein design. *Current Opinion in Structural Biology*. 2002, vol. 12, no. 4, p. 441–446. DOI: [https://doi.org/10.1016/S0959-440X\(02\)00345-7](https://doi.org/10.1016/S0959-440X(02)00345-7). ISSN 0959-440X. Available at: <https://www.sciencedirect.com/science/article/pii/S0959440X02003457>.
- [23] MODARRES, H. P., MOFRAD, M. R. and SANATI NEZHAD, A. Protein thermostability engineering. *RCS Advances*. 2016, vol. 6. DOI: 115252-115270.

- [24] MODARRES, H. P., MOFRAD, M. R. and SANATI NEZHAD, A. Protein thermostability engineering. *RSC Adv.* The Royal Society of Chemistry. 2016, vol. 6, p. 115252–115270. DOI: 10.1039/C6RA16992A. Available at: <http://dx.doi.org/10.1039/C6RA16992A>.
- [25] MONTANUCCI, L., MARTELLI, P. L., BEN TAL, N. and FARISELLI, P. A natural upper bound to the accuracy of predicting protein stability changes upon mutation. *Bioinformatics*. 2019, vol. 35, no. 9, p. 1513–1517. DOI: 10.1093/bioinformatics/bty880.
- [26] MUSIL, M., KONEGGER, H., HON, J., BEDNAR, D. and DAMBORSKY, J. Computational Design of Stable and Soluble Biocatalysts. *ACS Catalysis*. 2019, vol. 9, no. 2, p. 1033–1054. DOI: 10.1021/acscatal.8b03613.
- [27] PACE, C., FU, H., FRYAR, K. and AL. et. Contribution of hydrogen bonds to protein stability. *Protein science : a publication of the Protein Society*. may 2014, vol. 23. DOI: 10.1002/pro.2449. Available at: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4005716/>.
- [28] PACE, C. N., SCHOLTZ, J. M. and GRIMSLEY, G. R. Forces Stabilising Proteins. *FEBS Letters*. may 2014, vol. 588, p. 2177–2184. DOI: 10.1016/j.febslet.2014.05.006.
- [29] PANCOTTI, C., BENEVENUTA, S., BIROLO, G., ALBERINI, V., REPETTO, V. et al. Predicting protein stability changes upon single-point mutation: a thorough comparison of the available tools on a new dataset. *Briefings in Bioinformatics*. january 2022, vol. 23, no. 2, p. bbab555. DOI: 10.1093/bib/bbab555. ISSN 1477-4054. Available at: <https://doi.org/10.1093/bib/bbab555>.
- [30] POREBSKI, B. T. and BUCKLE, A. M. Consensus protein design. *Protein Engineering, Design and Selection*. july 2016, vol. 29, p. 245–251. DOI: 10.1093/protein/gzw015.
- [31] PUCCI, F., BERNAERTS, K. V., KWASIGROCH, J. M. and ROOMAN, M. Quantification of biases in predictions of protein stability changes upon mutations. *Bioinformatics*. november 2018, vol. 34, no. 21, p. 3659–3665. DOI: 10.1093/bioinformatics/bty348.
- [32] RODRIGUES, C. H. M., PIRES, D. E. V. and ASCHER, D. B. DynaMut: predicting the impact of mutations on protein conformation, flexibility and stability. *Nucleic Acids Research*. july 2018, vol. 46, p. 350–355. DOI: 10.1093/nar/gky300.
- [33] ROHL, C. A., STRAUSS, C. E., MISURA, K. M. and BAKER, D. Protein Structure Prediction Using Rosetta. In: *Numerical Computer Methods, Part D*. Academic Press, 2004, vol. 383, p. 66–93. Methods in Enzymology. DOI: [https://doi.org/10.1016/S0076-6879\(04\)83004-0](https://doi.org/10.1016/S0076-6879(04)83004-0). ISSN 0076-6879. Available at: <https://www.sciencedirect.com/science/article/pii/S0076687904830040>.
- [34] ROSE, G. D. Protein folding - seeing is deceiving. *Protein Science*. 2021, vol. 30, no. 8, p. 1606–1616. DOI: 10.1002/pro.4096.
- [35] ROSE, G. D., FLEMING, P. J., BANAVAR, J. R. and MARITAN, A. A backbone-based theory of protein folding. *Proceedings of the National Academy of Sciences*. 2006, vol. 103, no. 45, p. 16623–16633. DOI: 10.1073/pnas.0606843103. Available at: <https://www.pnas.org/doi/abs/10.1073/pnas.0606843103>.

- [36] SALI, A., SHAKHNOVICH, E. and KARPLUS, M. How does a protein fold? *Nature*. may 1994, vol. 369, p. 248–251. DOI: 10.1038/369248a0.
- [37] SCHYMKOWITZ, J., BORG, J., STRICHER, F., NYS, R., ROUSSEAU, F. et al. The FoldX web server: an online force field. *Nucleic Acids Research*. july 2005, vol. 33, p. 382–388. DOI: 10.1093/nar/gki387.
- [38] SIDDIQUI, K. S. Defying the activity–stability trade-off in enzymes: taking advantage of entropy to enhance activity and thermostability. *Critical Reviews in Biotechnology*. 2017, vol. 37, no. 3, p. 309–322. DOI: 10.3109/07388551.2016.1144045.
- [39] SIEVERS, F. and HIGGINS, D. G. Clustal Omega for making accurate alignments of many protein sequences. *Protein Science*. september 2017, vol. 27, p. 135–145. DOI: 10.1002/pro.3290.
- [40] SINDEN, R. R. DNA–Protein Interactions. In: SINDEN, R. R., ed. *DNA Structure and Function*. San Diego: Academic Press, 1994, p. 287–325. ISBN 978-0-08-057173-7.
- [41] SNUSTAD, D. P., SIMMONS, M. J. and RELICHOVÁ, J. e. *Genetika*. 2nd ed. Masarykova Univerzita, 2017. ISBN 978-80-210-8613-5.
- [42] SOSNICK, T. R. and BARRICK, D. The folding of single domain proteins—have we reached a consensus? *Current Opinion in Structural Biology*. 2011, vol. 21, no. 1, p. 12–24. DOI: 10.1016/j.sbi.2010.11.002. ISSN 0959-440X. Available at: <https://www.sciencedirect.com/science/article/pii/S0959440X10001740>.
- [43] STOURAC, J., DUBRAVA, J., MUSIL, M., HORACKOVA, J., DAMBORSKY, J. et al. FireProtDB: database of manually curated protein stability data. *Nucleic Acids Research*. november 2020, vol. 49, D1, p. D319–D324. DOI: 10.1093/nar/gkaa981. ISSN 0305-1048. Available at: <https://doi.org/10.1093/nar/gkaa981>.
- [44] STUDER, G., TAURIELLO, G., BIENERT, S., BIASINI, M., JOHNER, N. et al. ProMod3—A versatile homology modelling toolbox. *PLOS Computational Biology*. january 2021, vol. 17. DOI: 10.1371/journal.pcbi.1008667.
- [45] TRAPPENBERG, T. P. *Fundamentals of Machine Learning*. 1st ed. Oxford University Press, 2020. ISBN 9780198828044.
- [46] VARELA, A. E., LANG, J. F., WU, Y., DALPHIN, M. D., STANGL, A. J. et al. Kinetic Trapping of Folded Proteins Relative to Aggregates under Physiologically Relevant Conditions. *The Journal of Physical Chemistry B*. 2018, vol. 122, no. 31, p. 7682–7698. DOI: 10.1021/acs.jpcb.8b05360. Available at: <https://doi.org/10.1021/acs.jpcb.8b05360>.
- [47] WORTH, C. L., PREISSNER, R. and BLUNDELL, T. L. SDM—a server for predicting effects of mutations on protein stability and malfunction. *Nucleic Acids Research*. july 2011, vol. 39, p. 215–222. DOI: 10.1093/nar/gkr363.