Predicting Protein Stability with Machine Learning

A Master's Project in Biophysical Chemistry

Lucas Carlsson

September 2023

Abstract

Protein stability is a property of high importance and is of interest in a variety of fields. It determines if a protein has its native fold and can be of influence in certain diseases such as Parkinson's and Alzheimer's disease [1]. It can also be of interest in an industrial setting to optimise the stability of enzymes in certain physicochemical environments. Recent developments in machine learning have yielded novel methods able to predict protein characteristics with surprising accuracy solely from sequence information. However, few such models have been proposed for predicting protein stability. The aim of this project was to create a model able to predict protein stability from sequence information, by utilising a multiple sequence alignment based protein language model. Different models were developed to predict two quantities relating to protein stability, Gibbs free energy of unfolding and the heat denaturation temperature. Due to limited training data the performance of the models predicting Gibbs free energy was poor. One of the models predicting heat denaturation temperature, proved more promising, with higher performance than a previously published model trained on similar data. However its ability to predict conventionally obtained heat denaturation temperatures was poor.

Acknowledgements

I would like to start with expressing my gratitude to Ingemar, Sofia Andersson and Vera for all of the help and valuable input that I have received as well as for the interesting discussions we have had. I also want to thank Kristine, Frida and Sofia Canas for reviewing my thesis in detail and providing feedback. Furthermore I want to thank the rest of the members of André lab for the pleasant months working together with you.

Contents

1 Introduction

The introduction of modern sequencing techniques has for the last decades led to a rapid increase in the number of known proteins. However, protein characterising techniques that obtain information on protein properties such as structure, function and stability have not improved as significantly, leading to a large divide in knowledge between experimentally characterised and uncharacterised proteins.

To bridge the gap between sequence information and protein properties, computational models based on the artificial intelligence technique deep learning have been developed. The perhaps most famous example is AlphaFold, that can predict protein structures with impressive results. However, computational models for estimating protein stability, has not yet had a breakthrough.

For most proteins to function, the proteins have to be folded into a three-dimensional structure. The stability of this structure depends on the protein's amino acid sequence as well as the chemical and physical environment surrounding the protein. Knowledge of a proteins' stability is of importance in a wide array of settings. During development of new thermostable enzymes in industrial settings more thermostable enzymes are sought after since they can be used at higher temperatures, which increases the catalytic activity. It can also be useful while designing therapeutic antibodies to make them more stable and therefore suitable for treatment, as well as study evolutionary lineages of proteins [2]. However, conducting stability experiments can be a costly and time consuming endeavour, especially if a large number of proteins require characterisation.

There is therefore an interest in estimating the stability with *in silico* methods, which can be faster, cheaper and can be automated. The commonly used stability prediction models to date are molecular dynamics simulations, which require a three-dimensional structure of the protein and can be time consuming [2]. There has also been development of models based on sequence information, however the vast majority predict changes in stability upon point mutation, therefore requiring a value of the original protein's stability.

The growth of large-scale sequence datasets as well as the recent development of natural language models have enabled the creation of protein language models, able to represent proteins in a novel manner. One such protein language model is the multiple sequence alignment, MSA transformer that by learning how to predict multiple sequence alignments could make predictions about protein structure [3]. The models represent the amino acids and their dependencies to each other in a complex manner, which can be utilised by other models for extracting features [4].

The aim of this project was to create a model able to predict the stability of a protein from a multiple sequence alignment. The model was developed utilising machine learning and transfer learning of the MSA transformer published by Rao et al. [3].

2 Background

2.1 Proteins

A protein is a chain of amino acids in a specific order, or differently phrased, an amino acid sequence. Directly after being synthesised by the ribosome the protein has no function and in order to gain its function it has to be folded into a three-dimensional structure. The protein's function arises from the amino acids being arranged in a manner that creates a local physicochemical environment that allows the function to be carried out. As an example, for enzymes to catalyse a reaction, the active site must be formed to allow interactions with substrate that stabilise the reaction's intermediate states.

Most proteins fold spontaneously by random motion of the unordered chain, with non-covalent interactions between amino acids and between amino acids and the surrounding environment guiding the chain into a functional *native* fold. The main interactions are van der Waals interaction, hydrogen bonds as well as the hydrophobic effect.

2.2 Thermodynamics of Proteins

The change of state between the folded, native state and the unfolded denatured state is commonly approximated as a reversible two-state system, see Equation 2.1 . In the two-state system a protein is considered to either have the native fold, N, or an unfolded denatured structure, U. The approximation only allows for *one* native state and populations of intermediates are neglected. Therefore, proteins that do not have the native fold are considered unfolded, U. However, worth noting is that both states are so called *macrostates*, meaning that they are sets containing numerous configurations of protein. There is therefore not merely one configuration that is categorised as native, rather all configurations that can carry out the protein function are considered native.

$$N \rightleftharpoons U \tag{2.1}$$

The physical quantity determining the populations of the native or unfolded states at equilibrium is the difference in Gibbs free energy, $\Delta G \equiv G_U - G_N$. It can be related to the equilibrium constant, K, through Equation 2.2, where R is the gas constant and T is the temperature.

$$\Delta G = -RT \ln K \tag{2.2}$$

The equilibrium constant, K, in turn describes the relative population of the two states at equilibrium through Equation 2.3. Where X_N, and X_U are the fractions of native and unfolded proteins respectively, at temperature T. The previous restriction

of proteins exclusively being in the native or unfolded states, gives $X_N + X_U = 1$.

$$K = \frac{[U]}{[N]} = \frac{X_U}{X_N} = \frac{1 - X_N}{X_N} \tag{2.3}$$

Equal populations of the N and U give $X_N = 0.50$, which corresponds to $\Delta G = 0$. A majority of native proteins, $X_N > 0.50$, corresponds $\Delta G > 0$, while a majority of unfolded proteins $X_N < 0.50$ correspond to $\Delta G < 0$. This enables utilising ΔG as a metric for the stability of a protein's native fold.

How ΔG depends on temperature is described by Equation 2.4, which further relates ΔG to the difference in enthalpy of the two states ΔH and the difference in entropy of the states ΔS. Enthalpy, ΔH, can be understood as the difference in energy of non-covalent interactions, while ΔS is a measurement of the difference in how ordered the system is.

$$\Delta G = \Delta H - T\Delta S \tag{2.4}$$

The main contribution to ΔG is the hydrophobic effect, which is caused by the repulsion of water molecules on hydrophobic amino acids due to not being able to integrate into the water's hydrogen bond network. The lack of interaction causes hydrophobic residues to have an apparent attraction to each other, thus stabilising the folding [5].

The effect of hydrogen bonds on folding is more complicated. A typical protein can form hundreds of hydrogen bonds due to containing hydrogen bond donors and acceptors, both in the backbone as well in the hydrophilic side chains. In an unfolded state the hydrogen bonds are intermolecular, interacting with surrounding water molecules or other solutes, however in a folded state a considerable proportion of the hydrogen bonds are intramolecular, with interactions within the protein. Due to hydrogen bonds being formed regardless of folding, its effect on ΔH is comparatively small [5].

Repulsion of amino acids can be caused by over-packing, leading to a repulsive van der Waals interaction, as well as charge-charge interactions of amino acids of the same charge. However, the main destabilising effect arises from the change in conformational entropy, ΔS_{conf}. The difference in entropy is due to the fact that from a statistical perspective, the folded state is highly unlikely to form if each possible conformation of a protein is considered [5].

Integrating thermodynamic identities enables relating the enthalpy and entropy at a given temperature, $\Delta H(T)$ and $\Delta S(T)$, to the enthalpy and entropy of a reference temperature, $\Delta H(T_0)$ and $\Delta S(T_0)$, by utilizing the heat capacity ΔC_p, which is generally weakly dependent on temperature [2]. See Equations 2.5 and 2.6.

$$\Delta H(T) = \Delta H(T_0) + \int_{T_0}^{T} dT' \Delta C_p = \Delta H(T_0) + \Delta C_p (T - T_0) \tag{2.5}$$

$$\Delta S(T) = \Delta S(T_0) + \int_{T_0}^{T} dT' \frac{\Delta C_p}{T'} = \Delta S(T_0) + \Delta C_p \ln (T/T_0) \tag{2.6}$$

Choosing the heat denaturation temperature, T_m, as reference temperature enables the utilization of the following relation $\Delta S(T_m) = \Delta H(T_m)/T_m$, derived from Equation

2.4. This is due to T_m being the temperature where half of the protein population is denatured or $\Delta G(T_m) = 0$.

Combining Equations 2.5 and 2.6 with Equation 2.4 and T_m as the reference temperature, gives rise to Equation 2.7, which describes how the stability, ΔG, depends on temperature, T, based on the parameters $\Delta H_m, T_m$ and ΔC_p.

$$\Delta G(T) = \Delta H_m \left[\frac{T_m - T}{T_m}\right] - \Delta C_p \left[T_m - T\left(1 - \ln\left[\frac{T}{T_m}\right]\right)\right] \tag{2.7}$$

Figure 2.1 shows how ΔG depends on T, according to Equation 2.7 and how alterations of thermodynamic properties affects the stability. Alterations of ΔH_m or ΔC_p change the maximal value of the ΔG- curve as well as at which T the maxima is located, while alterations of T_m *slides* the ΔG-curve along the axis of the temperature.

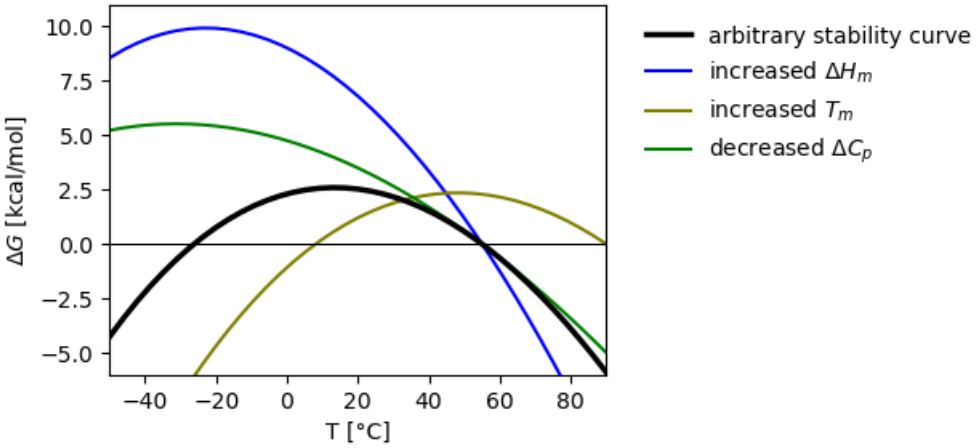

Figure 2.1: The thick black line is plotted based on the thermodynamic values $\Delta H_m = 40\text{kcal mol}^{-1}$, $T_m = 55\text{K}$ and $\Delta C_p = 0.9\text{kcal mol}^{-1}\,\text{K}^{-1}$. The other lines are variations of the thick black line where a single parameter has been altered. The blue line has ΔH_m altered to 80kcal mol^{-1}, while the olive line has T_m set to 90°C and the green line has ΔC_p set to $0.4\text{kcal mol}^{-1}\,\text{K}^{-1}$.

Protein stability is commonly quantified with either ΔG approximated at 25 $°C$, or the T_m value. Which of the two quantities is used depends on convention as well as the experimental method used. However as seen in Figure 2.1 do the two quantities correspond to different properties that both can be argued quantify protein stability, but different aspects of it.

Rees and Robertson [6] summarised different empirical findings on how ΔH_m and ΔC_p could be approximated by the peptide length N, and constructed an expression for how $\Delta G(T)$ could be approximated from T_m and the protein sequence length, N, alone.

$$\Delta G(T) = (2.92N + 0.058N(T_m - 333))\left[\frac{T_m - T}{T_m}\right] - 0.058N\left[T_m - T\left(1 - \ln\left[\frac{T}{T_m}\right]\right)\right]\ \text{kJ mol}^{-1} \tag{2.8}$$

2.3 Multiple Sequence Alignments

Due to genetic information, in the form of DNA/RNA, being passed on from one generation to the next imperfectly, divergence from a common ancestor can arise. The types of mutations occurring during reproduction are point mutations, insertions, deletions and translocations. If the mutation severely impacts the organism's fitness, by for example destabilising a vital protein, it is unlikely for the organism to reproduce due to the disadvantage it poses. However, mutations that do not negatively impact the fitness can be passed on to coming generations.

By aligning biological sequences, namely DNA, RNA, and protein, similarities and differences of sequences can be identified, which enables detection of homology. Higher order information such as structure, function or phylogeny can also be inferred by aligning a sequence to databases of already studied sequences. Sequence alignment is done by finding similarities between the sequences and introducing gaps to increase the degree of alignment between the sequences. In evolutionary terms a gap can be thought of as an insertion or deletion mutation, and a mismatch of a nucleotide or residue as point mutations. An example of a pairwise alignment can be seen in Figure 2.2.

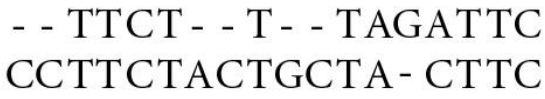

Figure 2.2: An example of pairwise alignment of two DNA sequences. The alignment algorithm tries to maximize the number of paired nucleotides by introducing gaps, which are shown as dashes(-). ([7], Creative Commons Attribution License)

Aligning multiple sequences through a multiple sequences alignment, MSA, enables extraction of more information than a pairwise alignment, by comparing sequences from multiple families. If a residue or region is well aligned among a wide array of proteins, it is considered conserved, due to not being replaced during the protein's evolution, and is therefore most likely to be of high importance for the proteins' function, structure or stability[8]. This can be rephrased as MSAs containing information of properties such as stability. An example of how this can be used is that over 20 examples in literature, reported by Sternke et al. [9], had engineered *de novo* proteins with stability higher than any of the in-going sequences in an MSA by simply utilising the MSAs *consensus* sequence. The consensus sequence of an MSA is constructed by choosing most common amino acid in each position of the sequence.

Viewing mutations on a protein level, contrary to nucleotide level, requires taking the properties of the amino acids into consideration. This is done through a scoring system where alignment of amino acids with similar physicochemical properties give a positive score, whereas alignment of dissimilar properties gives a negative score. Two popular scoring systems, or as more commonly called substitution matrices, are Blocks Substitution Matrix, BLOSUM, and Point Accepted Mutation, PAM matrix.

UniProtKB is a curated protein sequence database containing 249 million entries as of writing [10]. It is a good candidate for aligning unknown sequences to find homologous sequences, however the large size poses problems. In order create a representative of UniProtKB with reduced size Uniclust was created. It clusters sequences of UniProtKB with sequence similarities above a certain threshold and chooses a representative sequence for the cluster [11]. Utilising Uniclust for constructing MSAs therefore enables alignment of a wide representation of sequences.

2.4 Machine learning

Machine learning, ML, is a field that utilises computer programs to process data, extract information from it and make predictions about some unknown property. The program is developed through a generic model being trained on data, or phrased differently, the program learns from exposure to data [12].

Deep learning is a sub-field of ML focusing on the use of so called deep Neural Networks, NNs. Deep NNs are considered a state-of-the-art method in ML and can successfully model complex relationships between variables. Deep NNs can for instance be used for determining what object is present on an image, a so-called image classification problem [12].

To successfully grasp how an NN is built, it is helpful to understand the concept of linear regression. Linear regression utilises a parametric model in order to find a linear function, f, that from the input variables, $\mathbf{x}$, is able to predict an output, $f(\mathbf{x})$, that is close to the real value, y. In linear regression the output, $f(\mathbf{x})$, is a continuous variable, and the aim of the regression is to minimize the error, ϵ, which is the difference between the predicted value of input, $f(\mathbf{x})$, and the real value y, see Equation 2.9.

$$y = f(\mathbf{x}) + \epsilon \tag{2.9}$$

The linear function, f, consists of the parameters $\mathbf{w}$ and b, the slope and intercept, see Equation 2.10. The *learning* of the model is performed by altering the model parameters, $\mathbf{w}$ and b, to minimise the error, ϵ, until an optimal linear function, f, is found. This model is then able to perform predictions on new input data, $\mathbf{x}$, by utilising the at this stage already learned model parameters, $\mathbf{w}$ and b.

$$f(\mathbf{x}) = w_1x_1 + w_2x_2 + ... + w_mx_m + b = \mathbf{w}^T\mathbf{x} + b \tag{2.10}$$

If multiple outputs of the model, $\mathbf{f}$, are needed, logistic regression can be carried out in parallel through matrices, see Equation 2.11.

$$\mathbf{f}(\mathbf{x}) = \begin{matrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{matrix} = \begin{matrix} w_{1,1}x_1 + w_{1,2}x_2 + ... + w_{1,m}x_m + b_1 \\ w_{2,1}x_1 + w_{2,2}x_2 + ... + w_{2,m}x_m + b_2 \\ \vdots \\ w_{n,1}x_1 + w_{n,2}x_2 + ... + w_{n,m}x_m + b_n \end{matrix} = \mathbf{W}^T\mathbf{x} + \mathbf{b} \tag{2.11}$$

As previously is the learning of the model performed by optimisation of the model parameters, $\mathbf{W}$ and $\mathbf{b}$, to reliably describe how $\mathbf{x}$ relates to $\mathbf{y}$.

The term neural network stems from the fact that NNs were first developed to imitate how brains process information through biological neurons. A simple description of how a neuron processes information is as follows: the neuron obtains stimuli from multiple sources and internally processes the stimuli. If the stimuli is strong enough the neuron reacts with an output. Designing a mathematical model with similar properties, the artificial neuron, and constructing a network of such artificial neurons by aligning them in parallel and in sequence, yielded a general model suitable for ML [13].

The building block of an NN, the artificial neuron can be seen as a modification of linear regression. It consists of a linear function, $z = \mathbf{w}^T\mathbf{x} + b$, that imitates the reception and internal processing of stimuli, as well as an activation function $g(z)$, that imitates the evaluation of the combined signals and the output of the neuron. In deep learning there is furthermore a change in terminology, instead of describing $\mathbf{w}$ and b as slope and intercept, they are described as weights and bias. This is due to w_i determining the *weight* of influence x_i has on the artificial neuron, and b determining how strong *bias* the neuron has for activation.

There are many possible activation functions, g. Two common ones are the sigmoid function (Equation 2.12) and the Rectified Linear Unit, ReLU (Equation 2.13), which are plotted in Figure 2.3. What all activation functions have in common is non-linearity. If an activation function was linear, the artificial neuron would comprise of two linear functions following each-other, which can always be condensed into a single linear function.

$$g_{sigmoid}(z) = \frac{1}{1 + e^{-z}} \tag{2.12}$$

$$g_{ReLU}(z) = max(0, z) \tag{2.13}$$

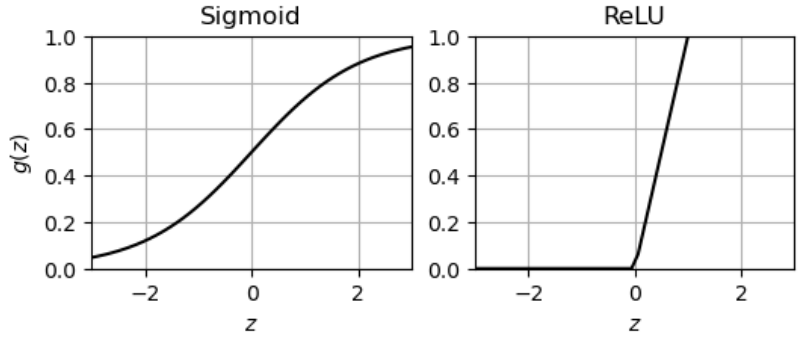

Figure 2.3: Two common activation functions. The vertical axis shows the output of the activation function $g(z)$, while the horizontal axis shows the result of the linear function, z.

An arbitrary NN can be broken down into so called layers. All NNs consist of an input layer, one or more hidden layers and an output layer. A common representation of a simple NN, with the different layers is shown in Figure 2.4. The x values represent input values to the network, the black circles artificial neurons or also called nodes, the arrows how inputs and outputs are connected to the artificial neurons. The output

of the NN is the predicted value $\hat{y}$. Each layer consists of a user-chosen number of artificial neurons in parallel.

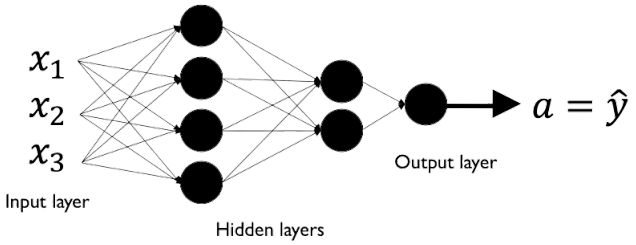

Figure 2.4: A basic fully connected feed forward artificial neural network, commonly called a multilevel perceptron, MLP.

The first layer in Figure 2.4 consists of four nodes, which all receive the three input values. The four nodes have their own weights and biases, similar to the multiple linear regression (Equation 2.11), resulting in four calculated values. The results from the linear functions are passed through the activation functions to yield the first layer's output values. The four output values of the first layer are then passed on to the second layer as input values. The second layer consists of two nodes, meaning that the linear function has four input variables and two output variables. The results of the linear function are once again passed through an activation function and the outputs are fed to the next layer. The output layer in this example consists of a node, that yields the output value of the whole NN. While performing regression the output node's activation function can be omitted, to not restrict which values the model can output.

The type of network with fully connected layers just described is commonly used in deep learning and is called a feed forward layer or a multilevel perceptron, MLP. The sizes of the layers vary depending on the number of input variables, output variables and the wanted complexity of the model. However it is common to use two or three layers [14].

Similar to linear regression, the mechanism of learning in deep learning is performed by gradually altering the model parameter values until a useful model is reached. This poses two requirements, firstly a way to evaluate how well a model is performing, and secondly how to alter the parameters based on the model performance.

The model evaluation is quantified through a loss function, L, that relates the predicted value of the model, $\hat{y}$, to the actual value, y. A *good* prediction leads to a low loss, while a *bad* prediction leads to a high loss. By utilising an optimisation algorithm, a minima of the loss function can be found, which if implemented correctly should correlate to a model predicting $\hat{y}$ closer to y.

A suitable optimisation algorithm is gradient descent, which is visualised in Figure 2.5. The gradient descent can be described by starting with a randomly initialised model, calculating the loss and the gradient at the initial state with respect to the model parameters. The parameters are then altered in the direction against the gradient. The loss and the gradient are calculated once again, and the processes is repeated until a sufficiently low loss is reached.

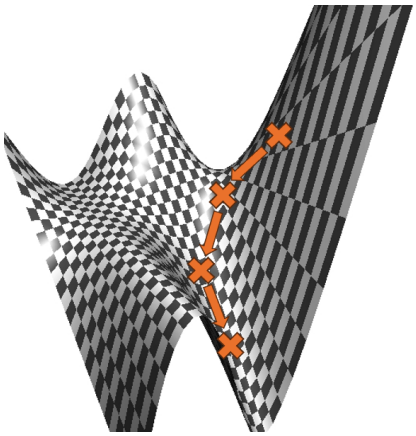

Figure 2.5: The checkered surface represents a loss landscape of an arbitrary two-dimensional model, meaning that the height of the surface corresponds to the loss at the given parameters. The orange crosses represent states of the model, while the orange arrows are steps against the gradient at a cross.

To efficiently calculate the output value of an NN as well as the gradients a method called backpropagation was developed. It calculates the result of the NN in what is called forward propagation. The gradients are calculated by utilising the chain rule of calculus and is calculated from the last layer backward and is therefore called backward propagation. An interested reader can read more about it in the book *Machine Learning: A first course for engineers and scientists* by Lindholm et al. [12].

A somewhat more mathematical description of the gradient descent is the cycle of the following four steps:

1. Compute the loss $L(\hat{y}, y)$.
2. Compute the partial derivative of the loss function for each weight, $\frac{\partial L}{\partial w_j}$, and bias $\frac{\partial L}{\partial b}$.
3. Compute the gradient step, $\Delta\mathbf{w}$ and Δb, by multiplying the partial derivatives with a user-defined learning rate, α: $\Delta\mathbf{w} = -\alpha\nabla_w L(\mathbf{w}, b)$ and $\Delta b = -\alpha\nabla_b L(\mathbf{w}, b)$.
4. Update the weights and biases: $\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}$ and $b := b + \Delta b$.

The learning rate, α, is a parameter set by the user which alters how much the parameters are changed during each iteration. If α is too low will the optimisation take an unnecessarily long time, but is it too large is there a risk that the optimisation will overshoot a loss minima and not be able to find useful parameter values.

So far the following parameters have been introduced: layer size, $\mathbf{w}, b$ and α. The parameters $\mathbf{w}$ and b are directly related to the model, since these are altered to train the model, while layer size and α remain constant during training or directly affect the training process. Parameters that are not an intrinsic part of the model, such as layer size and α are called hyperparameters.

In order to draw useful conclusions from a model developed by ML techniques, it is important to handle the available data wisely. The common practice is to separate the model development and the available data into three stages with accompanying datasets, namely training, validation and testing. The training set is the largest and is used to train the model through updating the model parameters. To ensure that the final model yields valuable results once deployed, the training data should, to the largest extent possible, represent the end user's data. The training and validation sets are used to develop the model, however the validation set is not used to optimise the model parameters, but rather to compare models during development and to tune hyperparameters. The test set ensures that the model has not overfitted to the training and validation sets and should therefore only be used once the model is ready to be deployed [14].

To determine if a model is overfitting, one compares the loss value from predictions of different datasets. During model development this is commonly done with respect to the duration of the training, which is quantified by *epochs*, the amount of times the training dataset has been used on the model. An epoch is therefore an iteration through the entire training set. The expected behaviour of a DL model is that at the beginning of training both the training and validation loss decrease, however if the training is allowed to carry on for long enough, the model becomes overly specialised on the training data and the loss value between the two sets will start diverging. An illustration of this behaviour can be seen in Figure 2.6.

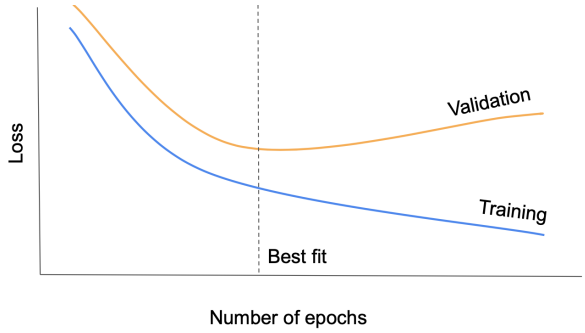

Figure 2.6: A schematic of overfitting. In the beginning of the training both training and validation loss decrease, which is associated with enhanced prediction performance. After a certain point the model reaches the *best fit* and more training leads to the model being over-specialised on the training data, which corresponds to an increased loss on the validation dataset.

Depending on the amount of available data the separation of data into training, validation sets looks different. The preferred method of separation is what is known as k-fold cross validation, which is illustrated in Figure 2.7. In k-fold cross validation the data that is not designated to the test dataset is perceived as training data. The training dataset is randomly partitioned into k number of folds. The model is trained k times with each of the folds being excluded from the training in each iteration in order to be used as validation. By averaging the performance of the folds an expected performance of the model can be obtained. The standard is to set k to 10, since it is deemed to offer reasonable trade off between bias and variation [14]. However, if the amount of available data is so large that training the model repeatedly is unfeasible,

can a single division into training and validation sets be necessary.

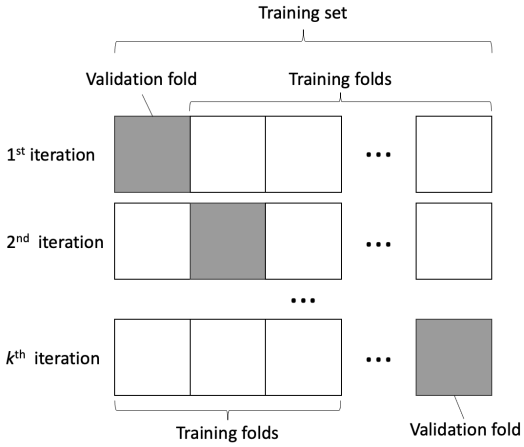

Figure 2.7: Illustration of how the training set is separated during k-fold cross validation.

Two commonly used concepts in ML are high variance and high bias. High variance corresponds to overfitting to training data and can be caused by the model being too complex. In order to decrease overfitting there are multiple options, namely obtaining more training data, reducing the complexity of the model, or increasing regularisation. There are multiple ways to regularise a model, some common options are L2-regularisation and dropout. L2-regularisation adds a term to the loss function penalising large weight values while dropout "shuts down" nodes with, the user-defined, p_{drop} probability. When done properly both methods ensure that the model cannot rely on specific nodes to make predictions and increases the model's robustness and generality. High bias corresponds to underfitting and is a symptom of a model too simple to learn patterns in the training data. High bias can be tackled by increasing the model size, increasing the number of input features or by decreasing regularisation [14].

As previously mentioned is the basis of ML creating models that learn from data. The learning described so far has been framed as using some features or input variables, x, in order to predict a property or output y. In order to train a model for this kind of prediction *labeled* data is needed, meaning that for each x there is a known corresponding y. Training a model to predict a label, y, from the features x is called supervised learning. However, availability of labeled data is often limited, compared to the amount of unlabeled data, and obtaining more can be an expensive and time consuming endeavour due to it often requiring manually labelling unlabeled data. Which is the current state within the field. There are a vast number of known protein sequences, with unknown properties such as stability.

This is where *unsupervised* learning is of interest, due to it enabling learning from unlabeled data. In unsupervised learning, the model is instead tasked to predict the input data. This is performed by giving the model incomplete inputs and letting the model predict the missing values. As previously are unsuccessful predictions penalised by parameter alteration, which forces the model to learn dependencies among the input features. This leads to the model having the ability, to perform abstractions of the input data, without a bias toward any label that supervised learning would restrict it to. This makes it of interest to use unsupervised models trained on large datasets,

such as the MSA transformer, for other task with much smaller amounts of available labeled data.

2.5 Sequential models

A protein has sequential information in the sense that it consists of amino acids in a specific order. It can be compared to sentences, where each amino acid can be thought of as a word. As the order of the words impact the meaning of a sentence, the order of amino acids impact the folding and properties of the protein.

So far the described architecture treats each data point as independent variables and therefore cannot interpret the order of sequences. However, two popular architectures able to interpret sequential information are recurrant neural network, RNN, and transformers.

An RNN processes the input sequence step by step, or in this context amino acid by amino acid. As it processes each step of the sequence it has two types of input and two types of output, as seen in Figure 2.8. One of the inputs stems from the sequence itself, while the other input is from the previous processing step in the same layer that is interpreting the sequence. The input from the previous step is not seen from the outside and is therefore called the hidden state. Similarly the outputs are to the next layer and to the hidden state to the next step of the same layer. This enables a transverse flow of information, which is impossible with a regular feedforward architecture, where information only flows in and out of the layer, and not within the layer.

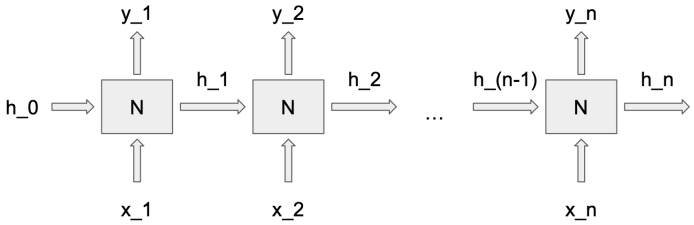

Figure 2.8: The structure of a generic RNN. The input sequence x, consists of the steps $x_1, x_2, ...x_n$. The boxes denoted N are the recurring network that receive inputs from the sequence step x_i as well as the previous step's hidden state, h_{i-1}. Each node in the figure outputs y_i, as well as a hidden state h_i.

The flow of information within the RNN layer enables learning sequential patterns, however a more sophisticated architecture is needed to learn long-range interactions. Two RNN architectures able to learn long-range interactions are long short-term memory cell, LSTM, and gated recurrant unit, GRU. An LSTM consists of recurring blocks that utilise two states passed between the steps, the hidden state and a cell state. The hidden state functions similarly as described in the previous paragraph, while the cell state is more complex. The cell state influences the hidden state and has mechanisms that can alter the input information or suppress information from being passed forward. GRU has mechanisms similar to LSTM, but utilises only one hidden state, making it less computationally heavy.

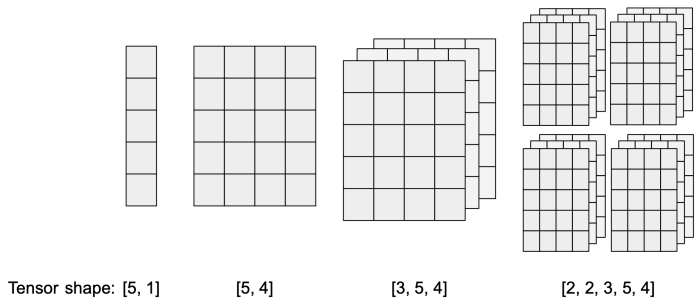

Figure 2.9: Visualisation of tensors with different dimensionalities and a description of their shapes. Each box represents a numerical value. The left-most tensor can also be called a row vector, and the second left-most tensor a matrix.

The transformer architecture was developed by Vaswani et al. [15]. Instead of interpreting the sequence step by step as RNN models do, it interprets the whole sequence at the same time. The sequential understanding of the model stems from an attention mechanism that quantifies how much attention each sequence step should pay to the other steps. This approach has two main benefits over the RNN architecture, firstly that even longer interactions can be learned, since the information does not need to be passed down through a large number of hidden states during the processing of the sequence. Secondly it can increase the speed of the model since it utilises vectorisation to a high degree. Shortly described vectorisation means that the model relies on structuring information in so called tensors and utilising matrix operations rather than breaking it up into repeating computations. A tensor is a multidimensional array, and can be viewed as a matrix of higher dimensions or a generalisation of a matrix. Matrices are two-dimensional in the sense that they are arranged by rows and columns, while a tensor can structure its elements in an arbitrary number of directions. Tensors therefore allow information to be ordered into dimensions higher than two. By utilising tensors and matrix operations the computer can save, load and compute more efficiently, than if the information was separate. A visualisation of how tensors can be seen in Figure 2.9.

2.6 MSA Transformer

The MSA Transformer is a protein language model developed by Rao et al. [3] utilizing MSAs. It was trained through unsupervised learning to learn complex dependencies and patterns in MSAs. The training was performed by masking residues in MSAs and letting the model predict the masked residues. The model consists of 100 million parameters and outperformed previous protein language models with 650 parameters. It is trained on 26 million MSAs, with an average of 1192 sequences per MSA. Rao et al. [3] showed that the model's representations of MSAs could be used for structure and contact prediction, even though it had only been trained on MSAs and had not been exposed to any structural information. It outperformed the previous state-of-the-art models with unsupervised training on sequence data. It had therefore learned to extract structural information from MSAs in order to more accurately predict them. Due to both structure and stability being of high importance for a protein to function,

it is likely that the MSA transformer also has learned to interpret protein stability from MSAs. The following section describes how the MSA transformer interprets and processes MSAs.

An MSA can be described as a two-dimensional matrix of M number of sequences of sequence length L. In order to make the MSA interpretable for a computer each position is represented by an integer. The vocabulary the MSA transformer utilises has 29 tokens, 20 of which are standard amino acids, five are non-standard amino acids, the remaining four are an alignment character, a gap character, a start token as well as a mask token.

Each position in the MSA was represented by an embedding vector of 768 elements. This enables the model to internally represent each amino acid residue through 768 features. The features can be viewed as the model's annotations of the residues. The usage of feature vectors enables comparing residues to one another. Two residues with similar properties should be close to each other in the vector space, while dissimilar residues have a larger distance in the vector space.

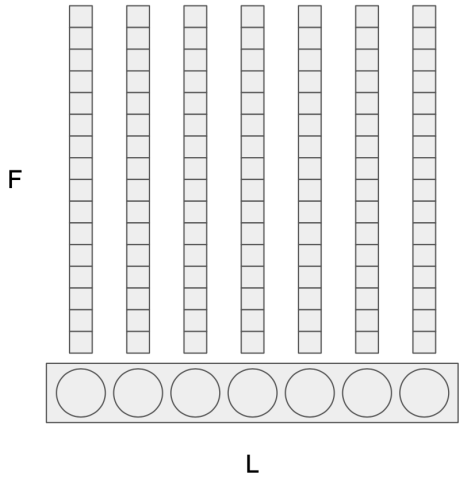

Figure 2.10: A representation of embeddings. The box at the bottom is a protein sequence, of length L, with each circle being a residue. The vertical objects are embedding vectors, with F number of features, describing the residues. The whole sequence is represented by a L by F two-dimensional matrix.

In the context of MSAs, attention can be interpreted as the importance amino acids have on each other. It can be in the form of two amino acids together being of importance for stability and therefore co-evolving. Due to MSAs' two-dimensional structure, the MSA transformer's attention mechanism would relate each position in the MSA to all other positions, meaning that the attention-maps would scale $O(ML)^2$ unless restrictions were put in place. The MSA Transformer therefore utilises two types of attention, tied row attention and column attention. This reduces the attention maps to $O(L^2)$ and $O(M^2L)$ for tied row attention and untied column attention respectively.

Furthermore, since the order of the residues affect the properties of a protein, the position has to be represented in the embedding. This is done through so called positional encoding, the embedding is altered by addition of a vector uniquely describing the residues position in the sequence. This enables the network to recognise the absolute position of a residue in the protein, but also the distance between two residues [14].

2.7 Experimental Characterisation of Proteins

The conventional methods for obtaining thermodynamic values of proteins are differential scanning calorimetry, DSC, and cosolvent denaturation experiments. To conduct conventional stability experiments the protein of interest must be isolated and the solution it is suspended in, must be well characterised.

DSC experiments measure the partial molar heat capacity, C_p, and its dependency on temperature. The definition of C_p is the amount of heat needed to increase the temperature of a mole of the protein with $1°C$, in a diluted solution at constant pressure. The experiment is conducted by heating the protein solution as well as a reference solution and comparing the difference in C_p between the two solutions. In the temperature range of a protein's unfolding there is a dramatic increase of C_p due to the amino acids absorbing more heat to shift into higher energy conformations. This excess C_p is directly associated to the unfolding of the protein and is used to estimate ΔH and T_m [5].

Cosolvents are polar molecules present in such high concentrations that they are considered a solvent, rather than a solute. A denaturant is a cosolvent that interacts more favourably with the protein backbone compared to the backbone's interaction with water, leading to a stabilisation of the unfolded state or a decrease of ΔG. Due to the cosolvent interacting weakly with the protein, the change in free energy is proportional to the denaturant concentration, C, leading to the Equation 2.14.

$$\Delta G(C) = \Delta G_0 + mC \tag{2.14}$$

Where ΔG_0 is the free energy of unfolding with none of the cosolvent species present and m is a constant describing the cosolvent species' effect on stability. The relation enables estimation of ΔG_0 by measuring ΔG at different C and performing a simple linear regression. This method of determining $\Delta G(T)$ is commonly called the linear extrapolation method. $\Delta G(C)$ can be determined by methods such as fluorescence spectroscopy or circular dichroism, spectroscopy. Fluorescence spectroscopy methods determining X_U utilise the fact that residues of unfolded proteins are exposed to a polar physicochemical environment, thereby broadening and shifting the emission spectra. Circular dichroism spectroscopy can be used to estimate concentration of secondary structures as well as the mobility of aromatic side-chains, both of which are related to the amount of unfolding. Even if the linear extrapolation method differs from DSC, the ΔG_0 values usually agree well with the ΔG values from the DSC experiments [5].

A high-throughput method for estimating T_m was developed by Leuenberger et al. [16]. The method obtains apparent T_m values on a proteome level by examining whole cell samples in shotgun mass spectrometry, MS. Shotgun MS methods are able to compare concentrations of proteins and peptide fragments between samples [17]. This is done by high precision determination of the molecular mass of protein fragments. The fragments are created by treating the protein samples with high specificity proteases, meaning that the protein chain is *cut* at sites with amino acid in a specific order. For known protein sequences these protein fragments and their molecular mass can easily be determined, due to the high specificity of the used proteases. A shotgun MS experiment therefore use sequence databases in order to map which proteins the fragments stem from.

The novelty of the method by Leuenberger et al. [16] is that it initially uses a thermostable broad-specificity protease with an increased activity for locally unfolded segments. By using the protease at different temperatures, it is argued that, it can be used to monitor local unfolding. The partially digested samples from the different temperatures are then run through the shotgun MS protocol, which involves the high specificity protease trypsin, to compare how the initial broad specificity protease altered the peptide fragments and their concentrations. The method therefore yields curves of how the relative intensity of peptide fragments changes at different digestion temperatures. The reported apparent T_m value is the temperature at which the concentration of protein unaffected by the broad-specificity protease was halved, compared to the concentration at the lowest temperature. For larger proteins consisting of multiple domains or folding units are apparent T_ms for each domain expected. The reported apparent T_m for each protein, is the lowest apparent T_m of the protein domains.

An alternative high-throughput method similar to the previously described one was performed by Jarzab et al. [18]. The method, thermal proteome profiling, TPP, is somewhat simpler due to it not using the broad-specificity protease and instead measuring the amount of protein precipitation as a function of temperature. This was performed by heating the sample and then centrifuging it to remove the precipitate. The soluble fraction was trypsinated and labeled and shotgun MS was used to determine the relative concentrations of each soluble protein species at the different temperatures. The temperature at which half of the protein concentration has been lost due to precipitation is therefore determined as the apparent T_m. The authors note that the apparent T_m obtained cannot be directly compared to the thermodynamically defined T_m, due to this method depending on unfolding and precipitation.

The high-throughput methods state apparent T_m values, rather than regular T_m, since the methods cannot assure reversibility between the native and unfolded states due to the digestion and precipitation.

2.8 Datasets

ProThermDB is a database published in November 2020 containing thermodynamic information on proteins. It currently holds about 31 500 entries with information about thermal heat denaturation temperature, T_m, free energy of unfolding from thermal and denaturant denaturing, ΔG, change of thermal heat denaturation temperature upon mutation, ΔT_m, change of free energy upon mutation, $\Delta\Delta G$, enthalpy of unfolding, ΔH, as well as change of heat capacity of unfolding ΔC_p. The data is collected from studies using conventional methods to study stability such as DSC, circular dichroism and fluorescence spectroscopy [19]. However, redundancy occurs, most entries are of mutants and not all entries contain ΔG or T_m values.

As previously mentioned did Leuenberger et al. [16] develope a method to estimate T_m on a proteome scale. The method was used on cell samples from four different species, namely *E. coli*, *S. cerevisiae*, *T. hermophilus* and human. Their values were published for public use.

The meltome atlas was published in April 2020 and contains values of 48 000 proteins from 13 species, obtained through TPP as described in the previous section. The samples were either whole cells or lysate.

2.9 Previous Studies

ProTstab was published in in 2019 by Yang et al. [20]. It utilises the ML method gradient boosting regression trees and is trained on data presented by Leuenberger et al. [16]. In 2022 the authors published ProTstab2, which utilises data also from the meltome atlas. Including the meltome atlas resulted in the amount of usable entries growing from 3 500 to roughly 35 000. Furthermore ProTstab2 utilises another method, light gradient boosting machine, which performed better during development. The models utilise features generated by protr, RECON, ProtDCal, ProtParam as well as amino acid group counts and frequencies [21].

A model successfully utilising the MSA Transformer for transfer learning is S-Pred by Hong et al. [22]. The model predicts secondary state structures, accessible surface areas and intrinsically disordered regions from embeddings and row attentions from the MSA Transformer. S-pred first processes the embeddings and row attentions as described in the Method Subheading 3.2.4. It is then passed through two bi-directional LSTM-layers. The last layers depend on which features to predict and if it was a classification or regression problem. See Figure 2.11 for a visualisation of the model architecture.

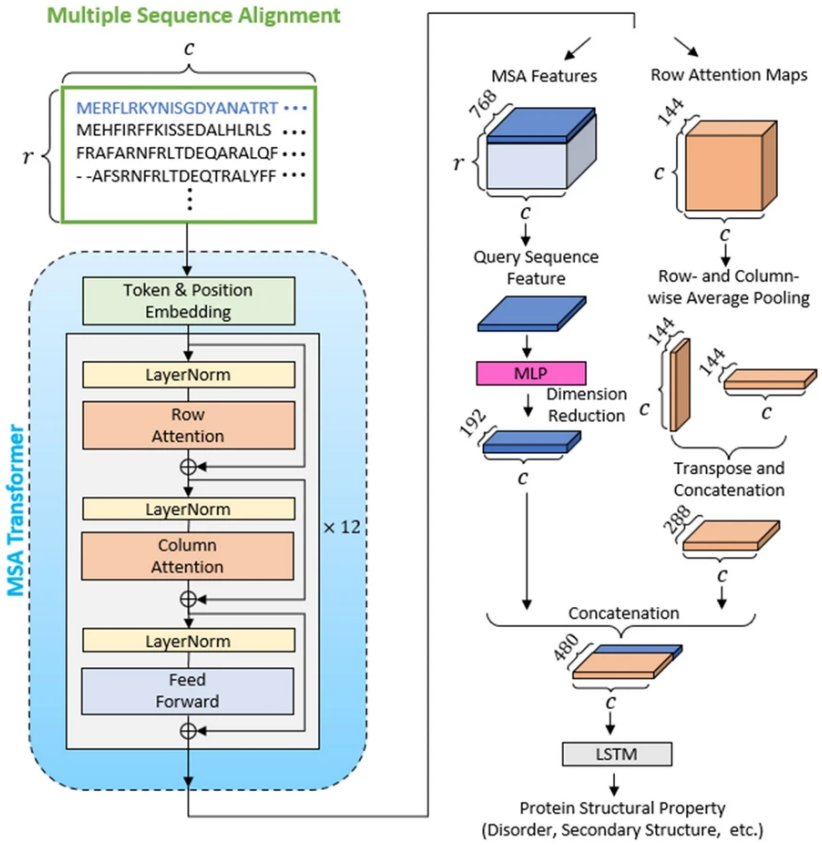

Figure 2.11: The architecture of the S-pred model. (Hong et al. [22], Creative Commons Attribution 4.0 International License.)

3 Method

3.1 Data and Model Handling and Availability

The models were implemented using PyTorch version `2.1.0.dev20230419 py3.9_cuda11.8_cudnn8.7.0_0`. The embeddings and attention maps were obtained by running the MSA transformer version `esm_msa1b_t12_100M_UR50S`. The developed models utilising the embeddings and attention maps were based on the code published by Hong et al. [22]. The developed models were trained on a NVIDIA GeForce RTX 4090 (24 GB) GPU card.

In order to enable replication of experiments the environment, data preprocessing scripts, models as well as training and testing sets was made available on GitHub on `https://github.com/luciduci2/stability_predict_masters_23`.

3.2 Data processing

3.2.1 ProThermDB

The ΔG values from thermal and chemical denaturation experiments were pooled together to obtain enough data. Some of the proteins in the database had multiple entries of stability measurements, due to conducting experiments at different conditions. Of the entries concerning the same protein but conducted at different conditions, only the mean of the entries from experiments conducted closest to pH 7 and temperature of 25 $^{\circ}C$ were used. The relative stability, $\Delta\Delta G$ and ΔT_m, of the mutants was further related to the absolute stability, ΔG and T_m, of the wild-type in order to estimate the absolute stability of the mutants.

3.2.2 Leuenberger's Data

No specific preprocessing was carried out on the dataset.

3.2.3 Meltome Atlas

For species with both whole cell and lysate data, the apparent T_m values of the lysate was used, since it can be argued that the chemical environment of the lysate is more similar to the chemical environment of conventional protein stability experiments, thus having higher resemblance to ProThermDB. However, if a species' entries only consisted of whole cell data, then it was also included to increase the amount of data.

3.2.4 All Datasets

The sequence information of the proteins was downloaded through UniProt's REST API [23] using the proteins' accession codes. The mutants' FASTA were then altered to match the described mutations. The length of the proteins were restricted to only allow amino acid chains longer than 40 residues, in order to mainly include proteins but also allow some polypeptides. Due to computational limitations only sequences shorter than 512 residues were included.

The imposed restrictions result in 4 666 number of entries from ProThermDB and 16 719 number of entries from the meltome atlas. See Table 3.1 for division between wild-type and mutant entries of ProThermDB and the meltome atlas. The data by Leuenberger et al. [16] resulted in 4 242 entries, however it was omitted during training and a subset of it was used during testing. ProThermDB and the meltome atlas were

Table 3.1: Number of stability measurements from the datasets after preprocessing, by the type of stability measurement and if the measurements are of a wildtype of mutant. The number of entries with both measurements, or the overlap between the two groups, in ProThermDB is included in the columns "Have both ΔG and T_m."

	ProThermDB			Meltome Atlas
	Have ΔG	Have T_m	Have both ΔG and T_m	Have T_m
Wild-type	296	519	177	16 719
Mutants	2 319	2 348	639	-
Total	2 615	2 867	816	16 719

separated into training, validation and testing data differently. In order to easily compare to ProTstab2 the test set of the meltome atlas was based on the test set constructed by Yang et al. [21] in the paper describing ProTstab2. This was also the approach for the data published by Leuenberger et al. [16]. It is worth noting that the test sets are not identical due to the size limitation imposed in this project. While the data by Leuenberger et al. was not used for training or validation, the meltome atlas was. Due to the size of the meltome atlas the remaining entries were split once into a training and validation set, rather than utilising k-fold cross validation. It was further motivated by a trial run utilising 5-fold cross validation, where each of the folds were nearly inseparable. The sorting of the meltome atlas into training and validation sets was randomised, with 10% of the non-test entries into the validation set and the other 90% into the training set.

ProThermDB was initially not separated into training, validation and testing, rather every entry was used during 10-fold cross validation. The dataset was divided first prior to running the tests run in Section 4.3 Models Trained on Meltome Atlas Predicting Conventional T_m. Before separating ProThermDB into training, validation and testing sets, the entries were clustered together based on sequence identity, through CDHit. The clustering was performed at default settings and clustered together protein with over 80% sequence identity, see Appendix A.3 for specifics. After clustering one tenth of the clusters containing wildtype entries with T_m values were labeled test clusters, and were not used in this project. The rest were labeled as training clusters and used

as validation.

3.2.5 Feature processing

By utilising the retrieved and altered sequences MSAs were constructed through the software HHblits[24] with uniclust30_2018_08[11] as reference database. The search was over three iterations. Otherwise MSA generation was run at default settings, see Appendix A.3 for specifics.

In order to limit computational cost the MSAs were set to have a depth, M, of maximally 128 sequences. To reduce the MSA depth the sequences were sampled through a method that maximizes the differences of the sequences in the reduced MSA. The method therefore maximises the so-called Hamming distance. Due to the MSA transformer utilising an embedding size of 768, the embedded output of the MSA transformer describing a protein has the shape $[M, L, 768]$. Since the first sequence in the MSA is the query sequence, the subset of the tensor corresponding to the first line of the MSA, should hold the most relevant information of the query protein. In order to reduce computational cost only the embedding of the query sequence was used, therefore reducing the tensors size to $[1, L, 768]$, which is inspired by Hong et al. [22].

Furthermore the MSA transformer consists of 12 layers, each consisting of 12 attention heads. The model outputs all attention maps from these layers. The row attentions are tied and have the shape $[12, 12, L, L]$. Since it is tied, all sequences in the MSA have the same identical row attention map. It relates the amount of attention payed to the other residues of the same sequence, for each attention head of the 12 layers. Like the embedding, the row attention was also manipulated to reduce the size of the tensors.

The two dimensions reflecting the 12 attention heads and 12 layers were flattened into one dimension by collating the 144 attention maps. Each attention map was averaged both column- and row-wise, resulting in a $[144, 1, L]$ sized tensor as well as a $[144, L, 1]$ sized tensor. The latter was reshaped and concatenated with the first to yield a tensor of size $[288, 1, L]$.

3.3 Models

Two types of network architectures both with similarities to S-pred were tested, one based on an RNN-layer and the other on a transformer-layer. Two types of models from each architecture type were tested, leading to in total four different models. All models use the preprocessed embeddings and attentions as described in the previous section. In order to condense the embeddings further to 192 features, an MLP of three fully connected layers were used, see Table 3.2 for specifics. The output of the MLP were then concatenated with the reshaped row attention tensor to create a tensor of shape $[1, L, 480]$. See Figure 3.1 for an illustration of the feature preprocessing and the initiating layers of the models.

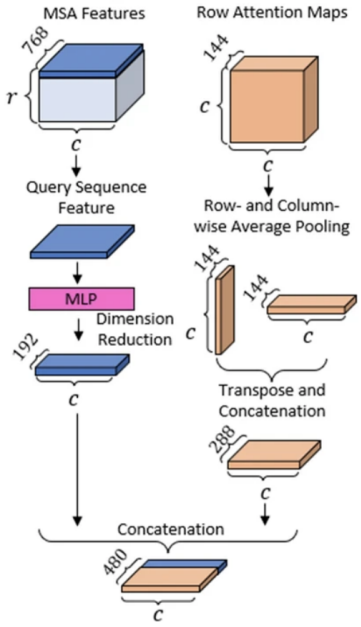

Figure 3.1: Visualisation of the initial part of all models. The upper blue box represents embeddings and illustrates how it is processed through selection of the query sequence's embeddings and further processing by the MLP. The orange box represents the tied row attention maps and how it was reduced by average pooling and later concatenated with the embedding information. (A cropped version of a figure by Hong et al. [22], Creative Commons Attribution 4.0 International License.)

Table 3.2: The MLP processing the embedding features. The layers were applied in descending order.

Layer	Description
Dropout	$p_{drop,1}$
Linear	Input features: 768, Output features: 384
Instance Normalisation	
LeakyReLU	Slope = 0.01
Dropout	$p_{drop,2}$
Linear	Input features: 384, Output features: 192
Instance Normalisation	
LeakyReLU	Slope = 0.01
Dropout	$p_{drop,2}$
Linear	Input features: 192, Output features: 192

The RNN models did not utilise dropout and therefore $p_{drop,1} = p_{drop,2} = 0$. Due to RNN layers being recurrent, the number of parameters of the RNN layer is independent of the input sequences' lengths. The number of parameters are instead affected by what kind of RNN is used, which was GRU, the number of hidden nodes, number of layers as well as if it is bidirectional, meaning that the sequence is also read backwards. Since the residues of a protein interact with residues both forwards and backwards in the sequence a bidirectional RNN layer was chosen. The two RNN models differed in model complexity with one model consisting of 3 million trainable parameters, while the other model consisted of 13 million trainable parameters. See Table 3.3 for details

regarding the differences in the RNN GRU layer for the two models.

Table 3.3: The differences of the RNN GRU layer for the two RNN models.

Hyperparameter	RNN 3 million parameters	RNN 13 million parameters
Number of input features	480	480
Number of hidden nodes	256	512
Number of layers	2	3
Bidirectional	Yes	Yes

Since the model's interpretation of the entire sequence was of interest and not specific parts, the output of interest from the RNN GRU layer was the hidden state of last processing step, in the forward direction, as well as the hidden state of the last processing step in the backward direction. The hidden states were concatenated and fed to a second MLP that reduced the number of elements down to a single value, which was the output. Figure 3.2 illustrates the flow of information in the latter part of the RNN models. The two RNN models differed in number of parameters in the last MLP, see Table 3.4 for details.

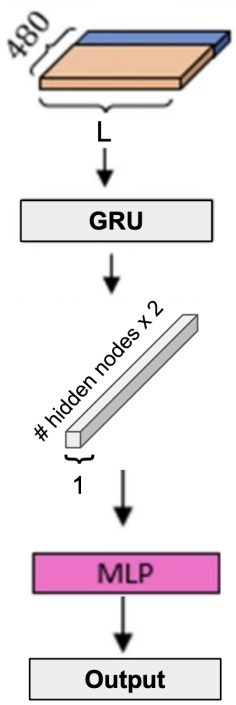

Figure 3.2: Illustration of the later part of the RNN models. At the top of the figure are the concatenated embedding and attention features of the shape $[1, L, 480]$, which correspond to the last part of Figure 3.1. The features are processed by the GRU based RNN. The most relevant hidden states' output by the GRU layers were concatenated into a tensor of shape $[1, 1, \#\text{hidden nodes in GRU} * 2]$ and fed to an MLP. The output of the MLP was the final prediction of the model. (An alteration of an image published by [22], Creative Commons Attribution 4.0 International License.)

The two transformer based models utilise the preprocessed features as described previously and illustrated by Figure 3.1 and Table 3.2. However, due to the transformer

Table 3.4: The MLP processing the output of the RNN layer. The layers were applied in descending order.

Layer	RNN 3 million parameters	RNN 13 million parameters
Linear	Input features: 512, Output features: 128	Input features: 1024, Output features: 512
LeakyReLU	Slope = 0.01	Slope = 0.01
Linear	Input features: 128, Output features: 64	Input features: 512, Output features: 256
LeakyReLU	Slope = 0.01	Slope = 0.01
Linear	Input features: 64, Output features: 1	Input features: 256, Output features: 1

not being recurrent, all of the sequences must be of the same length. In order to fulfil the requirement, the tensors were *padded* along the length dimension. The padding was performed by extending the tensor *after* the sequence with zeros up until the length dimension had 511 elements, before being fed to the MLP and being concatenated. The outputs of the MLP were still concatenated with the preprocessed row attentions yielding a feature tensor of size $[1, L, 480]$, with a constant L of 511. The reason why the sequence length was 511, rather than the intended 512 was due to a misunderstanding of the sequence length criteria while generating the embeddings and attentions of the MSA transformer.

The difference between the two transformer models are the dropout rates, p_{drop}, see Table 3.5. The dropout rates of the transformer model without regularisation are default values of the layers, while the dropout rates for the transformer model with regularisation where chosen by manually testing different values.

The concatenated tensor from the initial part of the model was fed into a transformer encoder, which output a tensor of the same size as the input, see Figure 3.3. The transformer encoder consisted of three encoder layers each consisting of 8 heads. The encoder performs dropout from a given dropout rate, $p_{drop,transformer}$, see Table 3.5, for which dropout rate was used during training of the two models. The output tensor is further processed by an MLP to reduce the number of features to 8 resulting in a tensor of size $[1, L, 8]$.

Table 3.5: The differences in dropout rates for two Transformer models.

Dropout rate	Transformer without regularisation	Transformer with regularisation
$p_{drop,1}$	0	0.1
$p_{drop,2}$	0	0.2
$p_{drop,transformer}$	0.1	0.3

The tensor was then reshaped to $[1, 8, L]$ to perform a one-dimensional convolution with a kernel-size of 25, and an output of 10 channels, which changes the tensor shape to $[1, 10, 487]$. The features were then flattened to a vector of shape $[1, 4870]$. Lastly a second MLP processed the features to yield the output of the model.

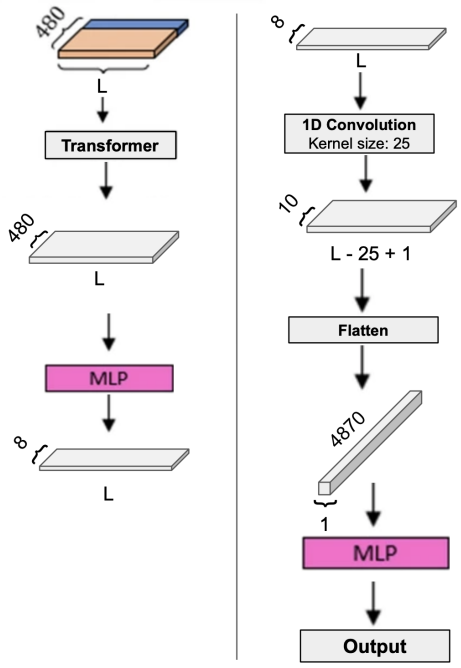

Figure 3.3: The later part of the transformer models. The top left tensor is the same tensor as the last one in Figure 3.1. The right column is the continuation of the processing of the left column. (An alteration of an image published by [22], Creative Commons Attribution 4.0 International License.)

Table 3.6: The MLP processing the output of the transformer models. The layers were applied in descending order.

Layer	Description
Linear	Input features: 4870, Output features: 4870
LeakyReLU	Slope = 0.01
Dropout	$p_{drop,2}$
Linear	Input features: 4870, Output features: 512
LeakyReLU	Slope = 0.01
Linear	Input features: 512, Output features: 1

3.4 Model development

The tested models were trained for 20 epochs and utilised the loss function mean square error, MSE. To update the model parameters the optimiser `torch.optim.AdamW` was used. It utilises stochastic gradient descent, based on gradient descent as previously described, and decoupled weight decay regularisation, which has a similar effect to L2-regularisation. The optimiser requires defining the hyperparameters learning rate and weight decay, of which multiple values were tested. By manually testing different values while training the transformer model to predict values on ProThermDB it was found that learning rate 1×10^{-4} and the weight decay 0.01 yielded the most stable model with lowest validation loss. However when applied on data from the Meltome atlas other values were found to yield better results. The learning rate and weight decay used for the final transformer and RNN models are shown in Table 3.7.

Table 3.7: The learning rate and weight decay used during training of the RNN and transformer based models.

	Learning rate	Weight decay
RNN models	5×10^{-4}	1×10^{-4}
Transformer models	1.05×10^{-4}	3×10^{-4}

4 Results

4.1 Models Trained on ProThermDB

The performance of the models during training were tracked by plotting the root mean square error, RMSE, of training and validation as a function of the number of epochs. See Appendix A.2 for definition of RMSE as well as other metrics.

Initially the transformer model was trained on all ΔG values from ProThermDB. Utilising all entries in 10-fold cross-validation for 15 epochs of training, resulted in the performance seen in Figure 4.1. With each epoch of training the model's predictions on the training set improve, which is seen as the constant decrease of RMSE on the training dataset. The predictions of the validation set also continuously improve, however after the 6th epoch the improvements are marginal.

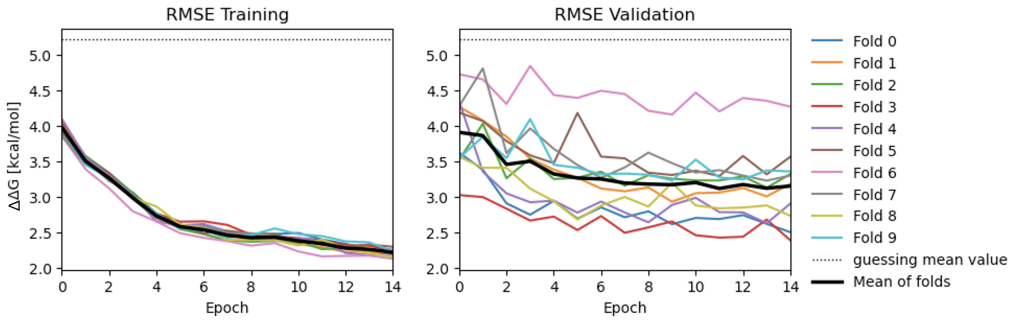

Figure 4.1: Progression of RMSE during training and validation with 10-fold cross validation to predict ΔG.

The model performs better than the baseline, which is defined as the RMSE value equivalent to a model constantly guessing the mean stability value of the dataset. However, it was not considered that proteins with high degree of similarity, for example a wildtype protein and its mutants, could be in both the training set and the validation set.

In order to study how this aspect affects the performance, entries were clustered together based on 80 % sequence similarity. The clusters were then the basis for the 10-fold cross validation. The clustering did not affect the training performance, as seen in Figure 4.2, however the average validation RMSE was above the baseline. The model therefore suffers from high variance.

Training the model to instead predict T_m values from ProThermDB, returned similar results. The baseline RMSE differs from the baseline when predicting ΔG, due to being a different unit, but is defined in the same manner. During clustered 10-fold cross-validation the training RMSE was below the baseline value, however, the validation

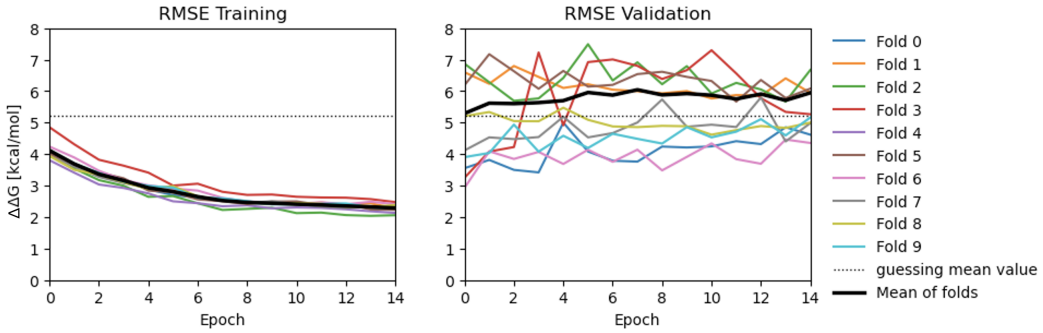

Figure 4.2: Progression of RMSE during training and validation with 10-fold cross validation to predict ΔG, with clustering. Note that the y-axis has shifted compared to Figure 4.1.

RMSE converged with it, as can be seen in Figure 4.3. Therefore also this model suffers from high variance.

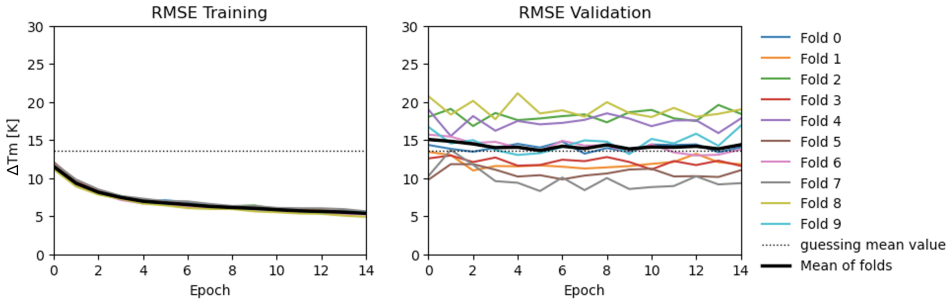

Figure 4.3: Performing 10-fold cross validation to predict T_m from ProThermDB. The baseline performance, *guessing mean value*, in the right sub-figure heavily overlaps with the other lines.

4.2 Models Trained on Meltome Atlas

In order to train a model on more data the meltome atlas was used. At first 5-fold cross validation was used. However due to all folds performing similarly, and in order to save time, it was decided to create a fixed validation set. See Figure A.1 for the behaviour of 5-fold cross validation.

Training a model with the transformer architecture on the meltome atlas for 20 epochs led to the progression of RMSE seen in Figure 4.4. The RMSE during training decreased rapidly while the validation RMSE remained constant. A new baseline was introduced, which was the RMSE value obtained by a model constantly guessing the mean stability value of the proteins belonging to the same species. The validation RMSE was below the first baseline, corresponding to guessing the whole sets mean stability, but higher than the new baseline. In order to decrease the the variance of the model it was tested to increase the regularisation by increasing the dropout rate as well as the weight decay parameter. The change of hyperparameters delayed the

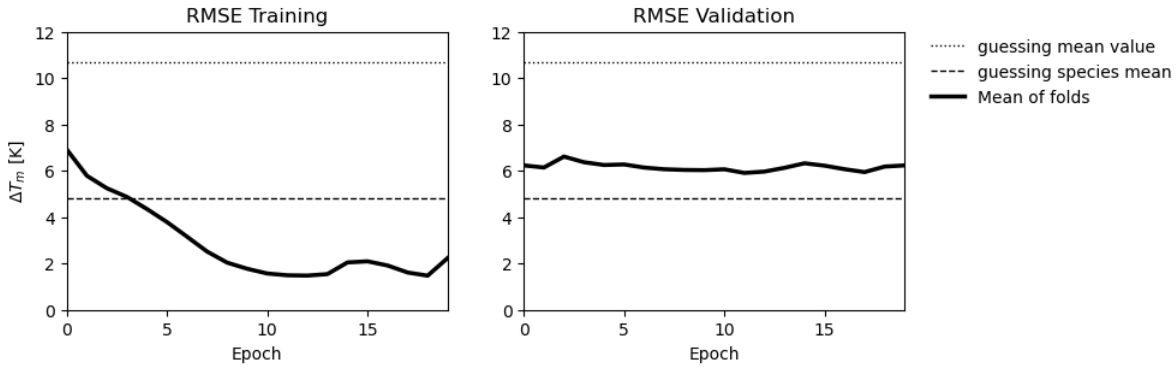

Figure 4.4: Progression of RMSE during training of the transformer without regularisation. The dotted line corresponds to the loss value obtained by solely guessing the mean stability of the dataset, the dashed line corresponds to the loss of guessing the mean stability of the entries' species.

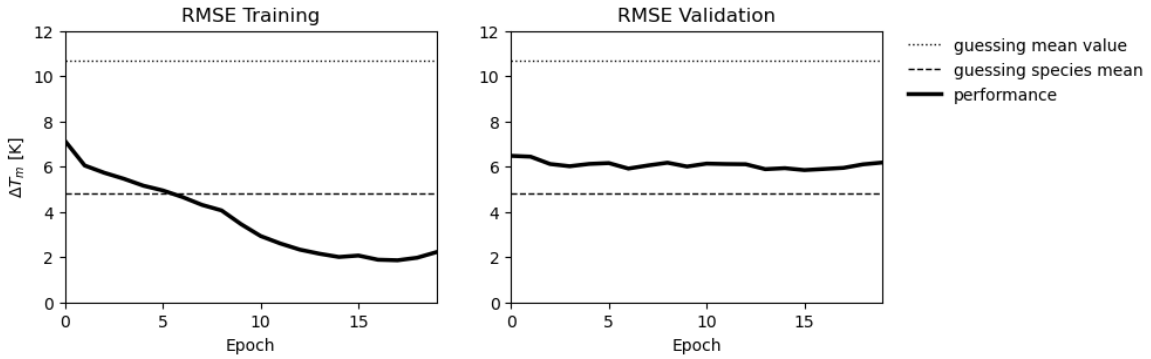

Figure 4.5: Progression of RMSE during training of the transformer model with regularisation. The dotted line corresponds to the loss value obtained by solely guessing the mean stability of the dataset, the dashed line corresponds to the loss of guessing the mean stability of the entries' species.

overfitting in the sense that it took more epochs for the training RMSE to reach the minimum value, see Figure 4.5. However, the validation performance remained largely unchanged.

In order to see how the complexity of the the model impacted the performance, a simpler architecture was utilised based on RNN. The two simpler models consisted of 3 and 13 million parameters and during training for 20 epochs without regularisation they performed as seen in Figure 4.6 and 4.7.

To more easily compare the four models' performance see Figure A.2. The validation performance after 20 epochs of training is shown in Table 4.1. As previously mentioned do the two RNN models differ by the amount of parameters, while the two transformer models differ in the degree of regularisation during training. The two RNN models perform better than the transformer models on the validation set, with lower MSE, RMSE and MAE, and higher PCC and R2 scores. The two RNN models score similarly with slightly higher performance for the model with 13 million parameters.

In order to inspect if the models exhibits overfitting to the validation data the, the test

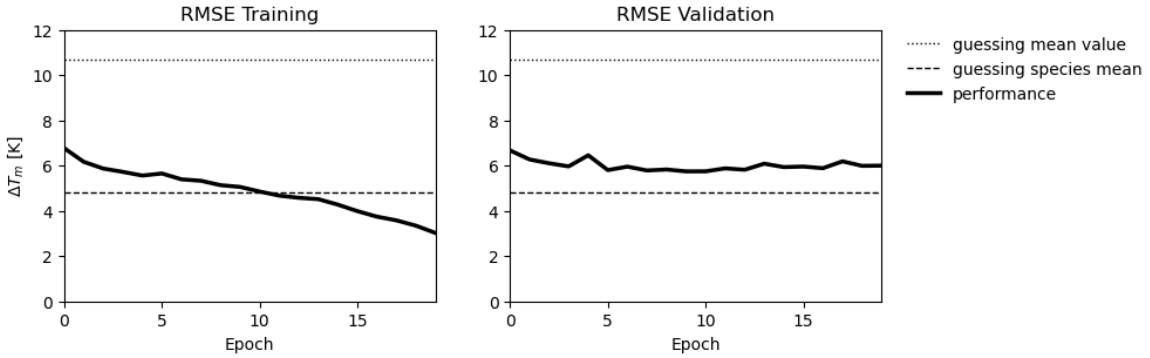

Figure 4.6: Progression of RMSE for a RNN model with 3 million parameters.The dotted line corresponds to the loss value obtained by solely guessing the mean stability of the dataset, the dashed line corresponds to the loss of guessing the mean stability of the entries' species.

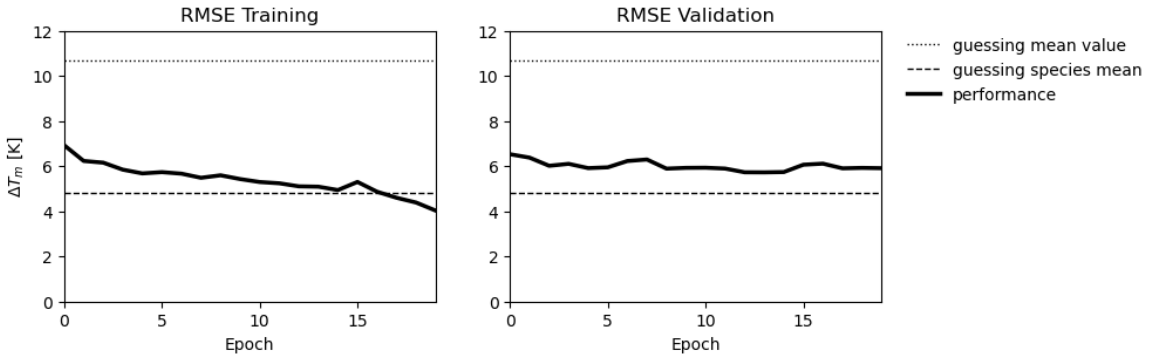

Figure 4.7: Progression of RMSE for a RNN model with 13.5 million parameters.The dotted line corresponds to the loss value obtained by solely guessing the mean stability of the dataset, the dashed line corresponds to the loss of guessing the mean stability of the entries' species.

Table 4.1: The performance of the models during validation. The best performance of a metric is shown in bold.

	RNN		Transformer	
Metric	3 mill	13 mill	With regularisation	Without regularisation
PCC	0.84	**0.85**	0.78	0.80
MSE	37.0	**36.0**	52.2	48.3
RMSE (K)	6.1	**6.0**	7.2	7.0
MAE (K)	4.7	**4.6**	5.4	5.4
R^2	0.70	**0.71**	0.58	0.61

dataset of the meltome atlas as well as the test dataset of Leuenberger's data. During testing the RNN models again perform better than the transformer based models, as well as ProTstab2, as can be seen by the lower MSE, RMSE and MAE, as well as the higher PCC and R2 in Table 4.2. How the models performed on entries of the test set belonging to the meltome atlas as well as the test set from Leuenberger's data, respectively, see Tables A.1 and A.2.

Table 4.2: The performance of the models while predicting the test dataset. It is a subset of the test dataset presented by Yang et al. [21] . The best performance of a metric is shown in bold.

	RNN		Transformer		
Metric	3 mill	13 mill	With regularisation	Without regularisation	ProTstab2
PCC	**0.86**	**0.86**	0.77	0.78	0.78
MSE	**43.3**	43.6	63.8	61.5	49.4
RMSE (K)	**6.6**	**6.6**	8.0	7.8	7.0
MAE (K)	**4.9**	**4.9**	5.7	5.7	5.3
R^2	**0.73**	0.72	0.59	0.61	0.61

4.3 Models Trained on Meltome Atlas Predicting Conventional T_m

In order to see how the four models trained on the meltome atlas performed while predicting conventional T_m values, the training dataset of ProThermDB was used as a sort of validation set. The performance of the models for predicting the conventional T_m is shown in Table 4.3. Guessing the mean T_m value of the training set results in a baseline RMSE of 17.3 K, which is reflected in the R^2 value being close to 0 or negative.

Table 4.3: Performance of models trained on TPP data to predict conventional T_m. The best performance of a metric is shown in bold.

	RNN		Transformer		
Metric	3 mill	13 mill	With regularisation	Without regularisation	ProTstab2
PCC	0.51	0.46	**0.52**	0.49	0.48
MSE	**277**	303	280	308	315
RMSE (K)	**16.6**	17.4	16.7	17.6	17.7
MAE (K)	**12.3**	12.7	12.4	13.0	12.5
R^2	**0.07**	-0.02	0.06	-0.04	-0.27

5 Discussion

The RNN based models performed better than ProTstab2 on the test set constructed from the meltome atlas and the data published by Leuenberger et al. [16], as seen by the lower MSE, RMSE and MAE and higher PCC and R^2 in Table 4.2. The RNN model with 3 million parameters scored the best, with an RMSE of 6.6 K and an R^2 of 0.73. Furthermore, the RNN models performed surprisingly well on the test set from the data published by Leuenberger et al., see Table A.2, considering that the models had not been trained on data from that dataset, nor on data gathered from the same experimental method. This in an indication that the T_m values published by Leuenberger et al. and the meltome atlas correlate relatively well and that the models could be of use for predicting the apparent T_m values obtained from high throughput methods.

Interestingly, the models trained on the meltome atlas performed poorly while predicting T_m values from ProThermDB, as seen by the R^2 values in Table 4.3 that are all close to 0. Regardless of the models' performances on validation and test sets of high throughput data. The R^2 of the best performing model, RNN with 3 million parameters, drops from 0.73 on the meltome atlas test set, down to 0.07 on the ProThermDB test set. The R^2 close to 0 reflects next to none predictive ability of the ProThermDB test set. The model is slightly more accurate than guessing the mean stability value for each prediction. This confirms that the apparent T_m used for training, poorly represents T_m. The two distributions are too dissimilar for the model to be of any use for predicting T_m values.

All of the validation performances, except the one utilising unclustered k-fold cross validation while predicting ΔG, Figure 4.1, converge with a value after very few epochs, and neither increase nor decrease a considerable amount as training continues. As described in the background and shown in Figure 2.6 is the expected behaviour of the validation loss an initial decrease, and then an increase as the model starts overfitting to the training set. The flatlining of the validation performance is not well described in literature but is deemed to be due to overfitting. If a model is too complex it can quickly learn to predict an average value for previously unseen proteins and develop detection of specific features for some of the proteins in the training dataset. These specific features make the proteins recognisable, but might not reflect protein stability in a broader sense, which would correspond to the stagnant validation performance.

This indicates that more actions could have been made to make a more generic model by utilising more training data, increasing regularisation, or decreasing model complexity. At the beginning of the project, only ProThermDB was used and utilising the meltome atlas was an attempt at the first approach, to use more training data.

Using the meltome atlas as training data did indeed improve the validation performance in the sense that the validation RMSE was lower than the baseline RMSE, of guessing the dataset's mean T_m. However, the RMSE was still higher than the baseline

corresponding to guessing the protein's species of origin's average protein stability. This indicates that the model could be of use to predict the apparent T_m of proteins with unknown species of origin or designed proteins. However, if the species of origin and the species' average apparent T_m is known, then a smaller error is expected by guessing the species' average apparent T_m, rather than the models' output. Furthermore the models were still overfitting, which was seen by the training loss being considerably smaller than the validation loss.

To decrease overfitting by increasing regularisation, different dropout rates and weight decay values were tested. As seen in Figure A.2 did the training loss of the regularised transformer model decrease slower than for the non-regularised model. However, the validation performance was only marginally improved by increasing regularisation or altering the other hyperparameters, as seen in Figure A.2 and Table 4.1. During development higher dropout rates were tested, however none of the tested combinations of dropout values affected the validation performance much.

By lastly using less complex models, with fewer trainable parameters, namely the RNN models, there was an improvement of validation performance, as seen by the lower MSE, RMSE and MAE and higher PCC and R^2 in Table 4.1. The difference in performance between the RNN models and the transformer based models was also seen during testing, see Table 4.2. This indicates that the RNN models indeed were more general and overfitted less. However, the models were still heavily overfitting. It would therefore be interesting to try even smaller models or more regularised RNN models to see how it impacts the performance.

When developing a model it is good to have a clear objective of what one wants the model to actually predict. If a protein's precipitation's dependency on temperature is wanted then the suitable way forward is to continue trying new methods to minimise the losses of the meltome validation set. However, if one requires a model with thermodynamically relevant predictions, one should try to look less at the performance of predicting data from high-throughput methods.

To investigate how well the RNN models capture knowledge of thermodynamic stability, one should train the network on the ΔG and T_m entries from ProThermDB. This wasn't done due lack of time. An even more interesting investigation would have been to use the RNN models trained on the meltome atlas and re-train them on data from ProThermDB. This would investigate if the pre-training on the meltome atlas gave the network some understanding of stability that could be transferred to predicting *non-apparent* T_m values. This was also not performed due to lack of time.

Obtaining more values of ΔG proved hard and is why models predicting ΔG were not investigated to the same degree as the T_m predicting models. The majority of published stability values in databases are changes of stability upon mutation, $\Delta\Delta G$ and ΔT_m, which require the wild-type's ΔG or T_m value in order to be used. Furthermore, a mutant's stability is generally marginally different to that of a wild-type, meaning that a database mainly containing stability values of mutants, with a small amount of wild-type proteins, will not cover enough of the sequential landscape to be useful for data-driven learning for predicting absolute stability.

An aspect that would have been beneficial to consider would have been the impact of multi-domain proteins on training and prediction. A protein domain is a unit of the protein chain that self-assembles independently from other domains and is self-stabilising. Most domains are 50 to 150 residues long, meaning that proteins over 200 residues long are likely to be multi-domain [25]. Due to this project including protein chains between 40 to 512 residues long, the models tried to learn to predict the stability of both single- and multi-domain proteins. In order to make the problem simpler, it would have been wise to train the model to predict the stability of single domain proteins. It is also worth noting that the apparent T_m published by Leuenberger et al. is the lowest apparent T_m of the protein domains. This means that if the dataset would have been included in training the model, there would have been entries where the apparent T_m only was directly connected to a certain part of the sequence. The other parts of the sequence would therefore not have contributed to the stability value in the database.

When it comes to ML is there always a need for more high quality data, and this project is not an exception. The high throughput methods contribute with large amounts of data, however, as shown is their relevance for predicting thermodynamic properties questionable. To develop a useful model from the chosen features with the help of ML, more thermodynamic data is needed, or another type of model that incorporates more empirical or physical knowledge into the prediction.

In this project has mainly the outputs or labels of the models been considered, but it is also important to assess the input features. All models developed in this project used the same input features, the query sequence embedding and a pooling of the row attentions related to the query sequence. Since no other input features were tested can no comparisons be made. To more properly investigate the MSA transformer's abilities, other feature preprocessing methods would have been interesting to test.

Furthermore, it is worth reconsidering the usefulness of MSAs for discovering the relationship between amino acid sequence and protein stability. MSAs do capture the mutations that have been able to withstand evolutionary selection, and most proteins must be stable in order to function. However, protein stability is not the only evolutionary pressure, and there is seldom an advantage for a protein to have *excess* stability. This means that an MSA based language model, like the MSA transformer, will have to learn about protein stability in order successfully predict MSAs, however fully describing protein stability may not be congruent with the protein language model's learning objective.

6 Summary

During the project, multiple models were developed to predict the quantities, ΔG and T_m, that are associated to protein stability. Due to the performance of the models being hindered by the limited amount of published thermodynamic data, data from high-throughput methods were also considered. A model trained on high-throughput data was more successful than a published equivalent at predicting high-throughput T_m values. However, due to the disconnect between T_m values obtained from high-throughput methods and the thermodynamic definition of T_m, the relevance of the model is questionable. However, the success at predicting the apparent T_m of the high-throughput methods indicates that if the model can be trained on larger datasets containing relevant data, it might be successful at predicting protein stability.

Bibliography

[1] Lene Clausen, Amanda B. Abildgaard, Sarah K. Gersing, Amelie Stein, Kresten Lindorff-Larsen and Rasmus Hartmann-Petersen. "Chapter Two - Protein stability and degradation in health and disease". In: *Advances in Protein Chemistry and Structural Biology*. Ed. by Rossen Donev. Vol. 114. Molecular Chaperones in Human Disorders. Academic Press, 1st Jan. 2019, pp. 61–83. DOI: 10.1016/bs.apcsb.2018.09.002. URL: https://www.sciencedirect.com/science/article/pii/S1876162318300580 (visited on 23/08/2023).

[2] Stepan Timr, Dominique Madern and Fabio Sterpone. "Chapter Six - Protein thermal stability". In: *Progress in Molecular Biology and Translational Science*. Ed. by Birgit Strodel and Bogdan Barz. Vol. 170. Computational Approaches for Understanding Dynamical Systems: Protein Folding and Assembly. Academic Press, 1st Jan. 2020, pp. 239–272. DOI: 10.1016/bs.pmbts.2019.12.007. URL: https://www.sciencedirect.com/science/article/pii/S1877117319302133 (visited on 10/07/2023).

[3] Roshan Rao, Jason Liu, Robert Verkuil, Joshua Meier, John F. Canny, Pieter Abbeel, Tom Sercu and Alexander Rives. *MSA Transformer*. Pages: 2021.02.12.430858 Section: New Results. 13th Feb. 2021. DOI: 10.1101/2021.02.12.430858. URL: https://www.biorxiv.org/content/10.1101/2021.02.12.430858v1 (visited on 02/05/2023).

[4] Tristan Bepler and Bonnie Berger. "Learning the protein language: Evolution, structure, and function". In: *Cell Systems* 12.6 (16th June 2021), 654–669.e3. ISSN: 2405-4712. DOI: 10.1016/j.cels.2021.05.017. URL: https://www.sciencedirect.com/science/article/pii/S2405471221002039 (visited on 13/07/2023).

[5] Bertil Halle, Kaare Teilum, Sara Linse, Kristofer Modig and Ingemar Andre. *Biophysical Chemistry*. Lund: Division of Biophysical Chemistry, Faculty of Engineering, Lund University, 2022.

[6] Douglas C. Rees and Andrew D. Robertson. "Some thermodynamic implications for the thermostability of proteins". In: *Protein Science : A Publication of the Protein Society* 10.6 (June 2001), pp. 1187–1194. ISSN: 0961-8368. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2374017/ (visited on 09/05/2023).

[7] Scott Lloyd and Quinn O. Snell. "Hardware Accelerated Sequence Alignment with Traceback". In: *International Journal of Reconfigurable Computing* 2009 (26th Jan. 2010). Publisher: Hindawi, e762362. ISSN: 1687-7195. DOI: 10.1155/2009/762362. URL: https://www.hindawi.com/journals/ijrc/2009/762362/ (visited on 18/08/2023).

[8] Maria Chatzou, Cedrik Magis, Jia-Ming Chang, Carsten Kemena, Giovanni Bussotti, Ionas Erb and Cedric Notredame. "Multiple sequence alignment modeling: methods and applications". In: *Briefings in Bioinformatics* 17.6 (1st Nov.

2016), pp. 1009–1023. ISSN: 1467-5463. DOI: 10.1093/bib/bbv099. URL: https://doi.org/10.1093/bib/bbv099 (visited on 07/08/2023).

[9] Matt Sternke, Katherine W. Tripp and Doug Barrick. "Chapter Seven - The use of consensus sequence information to engineer stability and activity in proteins". In: *Methods in Enzymology*. Ed. by Dan S. Tawfik. Vol. 643. Enzyme Engineering and Evolution: General Methods. Academic Press, 1st Jan. 2020, pp. 149–179. DOI: 10.1016/bs.mie.2020.06.001. URL: https://www.sciencedirect.com/science/article/pii/S0076687920302500 (visited on 27/08/2023).

[10] *UniProtKB*. UniProt. URL: https://www.uniprot.org/uniprotkb?query=* (visited on 21/08/2023).

[11] Milot Mirdita, Lars von den Driesch, Clovis Galiez, Maria J. Martin, Johannes Söding and Martin Steinegger. "Uniclust databases of clustered and deeply annotated protein sequences and alignments". In: *Nucleic Acids Research* 45 (D1 4th Jan. 2017), pp. D170–D176. ISSN: 0305-1048. DOI: 10.1093/nar/gkw1081. URL: https://doi.org/10.1093/nar/gkw1081 (visited on 25/05/2023).

[12] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten and Thomas B. Schön. *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022. URL: https://smlbook.org.

[13] Larry Hardesty. *Explained: Neural networks*. MIT News | Massachusetts Institute of Technology. 14th Apr. 2017. URL: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (visited on 23/08/2023).

[14] Sebastian Raschka, Yuxi Liu, Vahid Mirjalili and Dmytro Dzhulgakov. *Machine learning with PyTorch and Scikit-Learn: develop machine learning and deep learning models with Python*. Expert insight. Birmingham Mumbai: Packt, 2022. 741 pp. ISBN: 978-1-80181-931-2.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. *Attention Is All You Need*. 5th Dec. 2017. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762[cs]. URL: http://arxiv.org/abs/1706.03762 (visited on 02/05/2023).

[16] Pascal Leuenberger, Stefan Ganscha, Abdullah Kahraman, Valentina Cappelletti, Paul J. Boersema, Christian von Mering, Manfred Claassen and Paola Picotti. "Cell-wide analysis of protein thermal unfolding reveals determinants of thermostability". In: *Science* 355.6327 (24th Feb. 2017). Publisher: American Association for the Advancement of Science, eaai7825. DOI: 10.1126/science.aai7825. URL: https://www.science.org/doi/10.1126/science.aai7825 (visited on 02/06/2023).

[17] Bruno Domon and Ruedi Aebersold. "Mass Spectrometry and Protein Analysis". In: *Science* 312.5771 (14th Apr. 2006). Publisher: American Association for the Advancement of Science, pp. 212–217. DOI: 10.1126/science.1124619. URL: https://www.science.org/doi/full/10.1126/science.1124619 (visited on 27/08/2023).

[18] Anna Jarzab, Nils Kurzawa, Thomas Hopf, Matthias Moerch, Jana Zecha, Niels Leijten, Yangyang Bian, Eva Musiol, Melanie Maschberger, Gabriele Stoehr, Isabelle Becher, Charlotte Daly, Patroklos Samaras, Julia Mergner, Britta Spanier, Angel Angelov, Thilo Werner, Marcus Bantscheff, Mathias Wilhelm, Martin Klingenspor, Simone Lemeer, Wolfgang Liebl, Hannes Hahne, Mikhail M. Savitski and Bernhard Kuster. "Meltome atlas—thermal proteome stability across the tree of life". In: *Nature Methods* 17.5 (May 2020). Number: 5 Publisher: Nature Publishing Group, pp. 495–503. ISSN: 1548-7105. DOI: 10.1038/s41592-020-0801-4. URL: https://www.nature.com/articles/s41592-020-0801-4 (visited on 10/07/2023).

[19] Rahul Nikam, A Kulandaisamy, K Harini, Divya Sharma and M Michael Gromiha. "ProThermDB: thermodynamic database for proteins and mutants revisited after 15 years". In: *Nucleic Acids Research* 49 (D1 8th Jan. 2021), pp. D420–D424. ISSN: 0305-1048. DOI: 10.1093/nar/gkaa1035. URL: https://doi.org/10.1093/nar/gkaa1035 (visited on 12/07/2023).

[20] Yang Yang, Xuesong Ding, Guanchen Zhu, Abhishek Niroula, Qiang Lv and Mauno Vihinen. "ProTstab – predictor for cellular protein stability". In: *BMC Genomics* 20.1 (Dec. 2019), p. 804. ISSN: 1471-2164. DOI: 10.1186/s12864-019-6138-7. URL: https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-019-6138-7 (visited on 26/05/2023).

[21] Yang Yang, Jianjun Zhao, Lianjie Zeng and Mauno Vihinen. "ProTstab2 for Prediction of Protein Thermal Stabilities". In: *International Journal of Molecular Sciences* 23.18 (Jan. 2022). Number: 18 Publisher: Multidisciplinary Digital Publishing Institute, p. 10798. ISSN: 1422-0067. DOI: 10.3390/ijms231810798. URL: https://www.mdpi.com/1422-0067/23/18/10798 (visited on 10/07/2023).

[22] Yiyu Hong, Jinung Song, Junsu Ko, Juyong Lee and Woong-Hee Shin. "S-Pred: protein structural property prediction using MSA transformer". In: *Scientific Reports* 12.1 (16th Aug. 2022). Number: 1 Publisher: Nature Publishing Group, p. 13891. ISSN: 2045-2322. DOI: 10.1038/s41598-022-18205-9. URL: https://www.nature.com/articles/s41598-022-18205-9 (visited on 26/04/2023).

[23] Andrew Nightingale, Ricardo Antunes, Emanuele Alpi, Borisas Bursteinas, Leonardo Gonzales, Wudong Liu, Jie Luo, Guoying Qi, Edd Turner and Maria Martin. "The Proteins API: accessing key integrated protein and genome information". In: *Nucleic Acids Research* 45 (W1 3rd July 2017), W539–W544. ISSN: 0305-1048. DOI: 10.1093/nar/gkx237. URL: https://doi.org/10.1093/nar/gkx237 (visited on 07/07/2023).

[24] Martin Steinegger, Markus Meier, Milot Mirdita, Harald Vöhringer, Stephan J. Haunsberger and Johannes Söding. "HH-suite3 for fast remote homology detection and deep protein annotation". In: *BMC Bioinformatics* 20.1 (14th Sept. 2019), p. 473. ISSN: 1471-2105. DOI: 10.1186/s12859-019-3019-7. URL: https://doi.org/10.1186/s12859-019-3019-7 (visited on 02/05/2023).

[25] Dong Xu and Ruth Nussinov. "Favorable domain size in proteins". In: *Folding and Design* 3.1 (1st Feb. 1998), pp. 11–17. ISSN: 1359-0278. DOI: 10.1016/S1359-0278(98)00004-2. URL: https://www.sciencedirect.com/science/article/pii/S1359027898000042 (visited on 30/09/2023).

A Appendix

A.1 Notations and Symbols

Notation	Unit	Description
α	-	Learning Rate
b	-	Bias
ΔG	[kcal/mol]	Gibbs free energy of unfolding
$\Delta\Delta G$	[kcal/mol]	Change in Gibbs free energy from mutation
ΔH	[kcal/mol]	Enthalpy of unfolding
ΔH_m	[kcal/mol]	Enthalpy of unfolding at the heat denaturation temperature T_m
ΔS	[kcal/mol/K]	Entropy of unfolding
ΔC_p	[kcal/mol/K]	Change in specific heat capacity of unfolding
DL	-	Deep Learning: an ML techniques that utilise deep neural networks
DSC	-	Differential Scanning Calorimetry: an experimental method
GRU	-	Gated Recurrent Network, a type of RNN.
LSTM	-	Long Short-Term Memory, a type of RNN
ML	-	Machine Learning
MLP	-	Multilayer Perceptron: a fully connected feedforward neural network
MS	-	Mass Spectrometry
MSA	-	$\Delta\Delta G$
NN	-	Neural Network
ReLU	-	Rectified Linear Unit: a non-linear activation function
RNN	-	Recurrent Neural Network
T_m	[K] or [$^\circ C$]	The heat denaturation temperature
w	-	Weights

A.2 Metrics

In order to compare models more easily different metrics are useful. The metrics reflect how well a model's predicted values $\hat{Y}$ correlates to the real values Y in different ways. The Pearson correlation coefficient (PCC) measures the linear dependency between two variables. For N number of samples it is calculated in the following manner:

$$\text{PCC} = \frac{cov(Y, \hat{Y})}{\sigma_Y \sigma_{\hat{Y}}} = \frac{\sum_{i=1}^{N}[(\hat{y}_i - \mu_{\hat{y}})(y_i - \mu_y)]}{\sqrt{\sum_{i=1}^{N}(\hat{y}_i - \mu_{\hat{y}})^2}\sqrt{\sum_{i=1}^{N}(y_i - \mu_y)^2}}$$

Where $cov(Y, \hat{Y})$ is the covariance between the real and predicted values, σ the standard deviation of the defined variable and μ the mean value of the defined variable. PCC can vary between -1 and 1, where 1 corresponds to a perfectly linear positive correlation, 0 corresponds to no correlation and -1 to a negative perfect correlation.

In order to quantify the size of the errors in a more interpretable manner, mean square error (MSE), root mean square error (RMSE) and mean absolute error (MAE) can be used. The metrics are calculated by the following equations:

$$\text{MSE} = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}$$

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}$$

$$\text{MAE} = \frac{\sum_{i=1}^{N}|y_i - \hat{y}_i|}{N}$$

MSE and RMSE both square the errors, which accentuates larger errors' contribution to the metrics. MAE on the other hand uses the absolute of the error, meaning that all errors' contribution to the metric is proportional to the size of the error. The metrics RMSE and MAE have the same unit as y and $\hat{y}$, while MSE has the squared unit, making the result of RMSE and MAE often more intuitive.

R^2, also known as coefficient of determination, is commonly used to quantify the goodness-of-fit of a model. It this context it describes how well the predicted value describes the real value, with an R^2 of 1 corresponding to a perfect fit, with no prediction errors. An R^2 of 0 corresponds to the prediction errors being so large that using the mean of the real values, μ_y, as a predictor would be equally successful. R^2 can be calculated by the following expression:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \mu_y)^2}$$

A.3 Commands

CDHit version 4.8.1 was used to cluster proteins based on sequence identity through the line below, which was run in the terminal. The options can be described as the following: `-i all_seqs.FASTA` defines the FASTA file of the sequences to be aligned, `-o all_seqs80`, the name of the output file, `-c 0.8`, the sequence identity threshold, `-n 5`, the word length, `-M 1600`, the allowed max memory in Mb and lastly `-T 8`, the number of threads or CPUs.

```
cdhit -i all_seqs.FASTA -o all_seqs80 -c 0.8 -n 5 -d 0 -M 1600 -T 8
```

HH-blits version 3.3.0 was used to generate MSAs, with the command below used for each MSA. The options meant the following: `-i seq.FASTA`, the file containing the query sequence, `-oa3m seq.a3m`, the name of the output MSA file in a3m format. Both `seq.FASTA` and `seq.a3m` were different for each protein, these are just example names. The option `-n 3` meant that the MSA was created through 3 iterations, `-d dirLib` was used to specify the path of the uniclust30 database and lastly `-cpu 4` specified that 4 CPUs could be used for the MSA generation.

```
hhblits -i seq.FASTA -oa3m seq.a3m -n 3 -d dirLib -cpu 4
```

A.4 Figures and tables

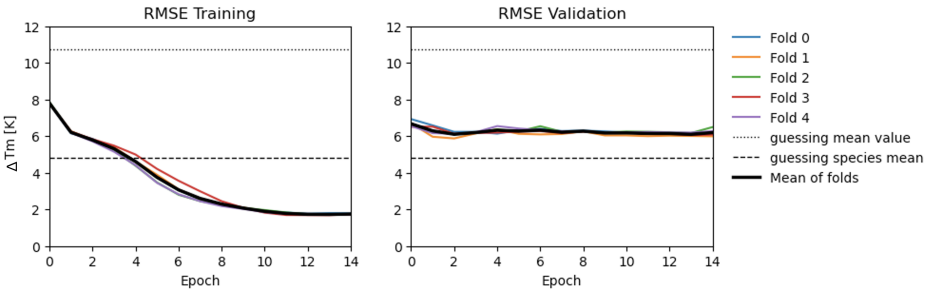

Figure A.1: The performance during 5-fold cross validation on the Meltome atlas for the transformer model without regularisation.

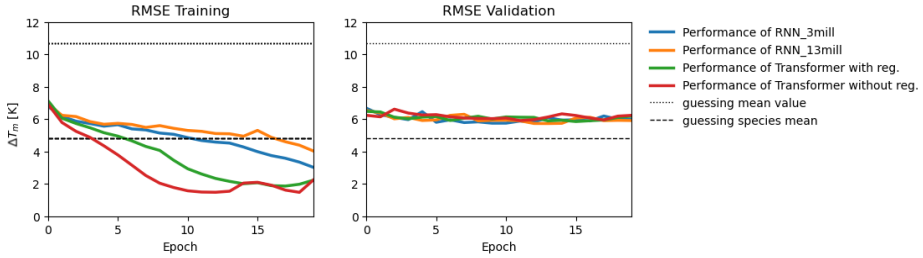

Figure A.2: The progression of RMSE during training of the four models trained on data from the meltome atlas. In the figure regularisation is shortened to *reg.*

Table A.1: The performance of the models while predicting the test set of the meltome atlas . It is a subset of the test dataset presented by Yang et al. [21] . The best performance of a metric is shown in bold.

	RNN		Transformer		
Metric	3 mill	13 mill	With regularisation	Without regularisation	ProTstab2
PCC	**0.86**	**0.86**	0.77	0.78	0.79
MSE	36.6	**36.0**	54.2	53.6	52.0
RMSE (K)	**6.0**	**6.0**	7.3	7.3	7.2
MAE (K)	4.6	**4.5**	5.4	5.4	5.5
R^2	**0.73**	0.72	0.58	0.59	0.60

Table A.2: The performance of the models while predicting a subset of the test set of ProTstab (not ProTstab2). It is a subset of the test dataset presented by Yang et al. [21] . The best performance of a metric is shown in bold.

	RNN		Transformer		
Metric	3 mill	13 mill	With regularisation	Without regularisation	ProTstab2
PCC	0.80	**0.82**	0.71	0.74	0.79
MSE	97.1	96.6	131.3	117.1	**91.4**
RMSE (K)	9.9	9.8	11.5	10.8	**9.6**
MAE (K)	**7.2**	7.3	8.4	7.6	7.3
R^2	0.53	0.53	0.36	0.43	**0.56**