Peptide Retention Time Prediction using Artificial Neural Networks

SARA VÄLJAMETS
Peptide Retention Time Prediction using Artificial Neural Networks

SARA VÄLJAMETS

Master’s Thesis in Mathematical Statistics (30 ECTS credits)
Master Programme in Applied and Computational Mathematics (120 credits)
Royal Institute of Technology year 2016
Supervisor at KTH: Lukas Käll, Timo Koski
Examiner: Timo Koski

TRITA-MAT-E 2016:49
ISRN-KTH/MAT/E--16/49-SE

Royal Institute of Technology
SCI School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci
Abstract

This thesis describes the development and evaluation of an artificial neural network, trained to predict the chromatographic retention times of peptides, based on their amino acid sequence. The purpose of accurately predicting retention times is to increase the number of protein identifications in shotgun proteomics and to improve targeted mass spectrometry experiment. The model presented in this thesis is a branched convolutional neural network (CNN) consisting of two convolutional layers, followed by three fully connected layers, all with leaky rectifier as the activation function. Each amino acid sequence is represented by a matrix $X \in \mathbb{R}^{20 \times 25}$, with each row corresponding to a certain amino acid and the columns representing the position of the amino acid in the peptide. This model achieves a RMSE corresponding to 3.8% of the total running time of the liquid chromatography and a 95% confidence interval proportional to 14% of the running time, when trained on 20,000 unique peptides from a yeast sample. The CNN predicts retention times slightly more accurately than the software ELUDE when trained on a larger dataset, yet ELUDE performs better on smaller datasets. The CNN does however have a considerable shorter training time.
Peptid retentionstids prediktering med artificiella neuronnät

Sammanfattning

Det här examensarbetet beskriver utvecklingen och utvärderingen av ett artificiellt neuronnät som har tränats för att prediktera kromotografisk retentionstid för peptider baserat på dess aminosyrasekvens. Syftet med att prediktera retentionstider är att kunna identifiera fler peptider i ”shotgun” proteomik experiment och att förbättra riktade masspektrometri experiment. Den slutgiltiga modellen i detta arbete är ett konvolutions neuronnät (CNN) bestående av två konvolutions lager följt av tre lager med fullt kopplade neuroner, alla med ’leaky rectifier’ som aktiveringsfunktion. Varje aminosyrasekvens representeras av en matris \( X \in \mathbb{R}^{20 \times 25} \), där varje rad representerar en specifik aminosyra och kolumnerna beskriver aminosyrans position i peptiden. Den här modellen uppnår ett kvadratiskt medelfel motsvarande 3.8\% av körtiden för vätskekromatografin och ett 95\% konfidensinterval motsvarande 14\% av körtiden, när CNN modellen tränas på 20 000 unika peptides från ett jästprov. CNN modellen presterar marginellt bättre än mjukvaran ELUDE när de är tränade på ett stort dataset, men för begränsade dataset så presenterar ELUDE bättre. CNN modellen tränar dock avsevärt mycket snabbare.
Acknowledgements

I would like to thank my supervisors Lukas Käll and Timo Koski for their help and feedback on my work. I would also like to thank Heydar Maboudi Afkham for his supervision and input throughout the project. I would like to thank Matthew The for his help on running ELUDE. Finally, I would like to thank Marcus for proofreading.
## Contents

1 Introduction .......................... 1
   1.1 Motivation & Contribution .... 1
   1.2 Outline ........................ 3

2 Artificial Neural Networks ............ 4
   2.1 FeedForward Neural Networks .. 5
   2.2 Activation Functions .......... 8
   2.3 Convolutional Layers .......... 9

3 Training Deep Neural Networks ...... 11
   3.1 Cost Function ................ 11
   3.2 Gradient Descent and Stochastic Gradient Descent ...... 11
   3.3 Backpropagation .............. 12
   3.4 Dropout ...................... 15

4 Data ................................ 16
   4.1 Data Representations .......... 17
   4.2 Mirror Images ................. 19
   4.3 Scaling ....................... 20

5 Evaluation .......................... 21

6 Implementation ....................... 23

7 Model Development .................... 24
   7.1 Layer Architecture .......... 25
   7.2 Selecting Hyperparameter Values ..... 26
   7.3 Structure of Final Model, Net 11.3 ... 31

8 Results ............................. 32

9 Discussion ........................... 37
   9.1 Comparison of the Different Implemented Models .... 37
   9.2 Comparison of Net 11.3 and ELUDE ................. 37
   9.3 Comment on the Structure of Net 11.3 .............. 38
   9.4 Future Work & Possible Improvements ............. 39

10 Conclusion ......................... 40

A Complete Results from First and Second Round of Hyperparameter Experiments 41
B Earlier Models

B.1 Linear Regression .......................................................... 42
B.2 AA Frequency & AA Pair Frequency ANN ............................... 43
B.3 Single Matrix Representation CNN .................................... 44
1 Introduction

The study of proteins, proteomics, is central to enhancing our understanding of biological systems. It is also a field that at the offset faces a series of challenges. Firstly, there is a vast number of different proteins. In the human body alone over 20 000 protein coding genes have been identified, some with the ability to encode hundreds of different proteins. Another challenge is the ratio of concentrations of these different proteins. In human plasma it is estimated that these ratios amount to 1 in $10^{12}$ for certain proteins, yet the scarce proteins may be as important to study as the abundant ones [2].

Mass spectrometry (MS) based methods are today the most popular tools for analysing protein content of biological mixtures [19]. In MS the specimen to be analysed is ionised and then accelerated through an electric or magnetic field, sorting the ions based on their mass to charge ratio. Inferences about the specimen can than be made from the resulting spectrum. Shotgun proteomics combines MS with an initial liquid chromatography (LC) step. First, the proteins are enzymatically broken down into peptides, short chains of amino acids. The peptides are then dissolved in water (solvent A) and poured into a tube (the LC column) filled with hydrophobic silica beads. Due to their generally hydrophobic nature the peptides bind to the silica beads. A hydrophobic solvent B is then gradually introduced into the column, usually with a linear gradient. Each peptide will release from the silica beads at a certain concentration of solvent B. This allows for the peptides to be gradually fed to the MS, as the peptides will elute at different times depending on their properties. The time interval from the introduction of solvent B to the elution of a peptide is referred to as a peptide’s chromatographic retention time [21].

Accurate prediction of retention times can benefit the field of proteomics both by increasing the number of peptide identifications and by increasing the reliability of those identifications. This thesis explores the possibilities of applying deep neural networks to this task.

1.1 Motivation & Contribution

There are two main application that make retention time prediction relevant to shotgun proteomics. Firstly, predicting retention times can improve peptide identification. Identifying peptides from MS is nontrivial and the proportion of spectra from LC/MS analyses that can be confidently matched to a peptide is generally around 50%. This low yield is due to several factors, such as limitations on protein database searches and incorrect mass and charge assignment [2]. Many attempts at dealing with these issues are underway and utilizing peptide retention time prediction is one such approach. The experimental retention time of the peptide can be compared to the predicted retention time of the identified peptide as an additional verification of the identification [19]. The second application relates to the large variation in abundance between peptides. High-abundance peptides are favoured in most identification methods. As a result
targeted MS is gaining ground, to improve identification of more scarce peptides [19]. Knowledge of when certain peptides will elute and enter the mass spectrometer allows the calibration to be changed to fragment the mass to charge ratio corresponding to the peptides of interest [19].

For a given setup the timing of different peptides has proven to be highly consistent between runs [18], [21]. This reproducibility suggests that the factors determining retention times for a given setup almost solely relate to the peptides themselves [21]. A peptide constitutes a sequence of amino acids connected by peptide bonds and in addition has a three-dimensional conformation. The sequence should hold virtually all the information required, although in particular for longer peptides the three-dimensional structure may affect the elution.

The elution process is dependent on the choice of solvent gradient, among other features, and retention time is therefore specific to each experimental setup. However, there is little variation in the setup and conditions of LC, making interlaboratory comparisons possible, if the settings can be calibrated for [2].

Several attempts at retention time prediction have previously been published. It was first considered in 1980 by Meek [18]. The prediction then entailed summing the contribution of each amino acid’s residue to the retention time. However, this model had limited predictive power and it became apparent that the order of amino acids affects the retention time. This led to the development of a series of predictors based on different methods, yet which all included more complex feature engineering, utilizing domain knowledge, to better represent the data [13], [19], [22], [23]. Notable among then is the retention time predictor ELUDE, first described in [21], using kernel regression to predict retention times, based on 60 features derived from the peptides’ amino acid composition. Petritis et al. [18] applied artificial neural networks to this problem representing the data as nearest neighbour pairs. However, neither of these predict with accuracy close to the high level of consistency in retention times for repeated experiments. There is still considerable room for improvement, particularly for longer peptides.

The methods applied to this problem have congregated more and more towards the realm of machine learning, where the performance is heavily dependent of the choice of data representation used [1]. Recent advances in deep learning have shown promising results from learning a feature representation from the data, rather than engineering them heuristically [1]. Deep neural networks can discover intrinsic structures in complex data by learning a representation through multiple layers, to increasing levels of abstraction [17]. Perhaps the key challenge in the retention time prediction is engineering sufficiently descriptive features. A deep learning approach may therefore have the potential to find a better representation and thereby enable the learning of a model with improved performance. Furthermore, through this representation learning arti-
ficial neural networks are adept at modelling complex non-linear function, as long as they are reasonably continuous. A detailed discussion on the particular advantages of applying convolution networks to this problem will follow in Section 2.3. In short however convolution networks are designed to handle array structured data and for such data the train is easier due to the model using a smaller number of connections and parameters [12].

1.2 Outline

This report will detail the work done for this project on applying artificial neural networks to the task of peptide retention time prediction. Firstly, an introduction to artificial neural networks will be given. The training of these networks will also be discussed at length in Section 3. The data used for the training of these models will then be described, along with the chosen data representations. The evaluation metrics applied when presenting the result in Section 8 will also be presented in detail. Section 7 then explains the methodology employed in the model development. The final model structure is illustrated in Section 7.3. Finally, the results of the implemented models are presented and compared to the benchmark model ELUDE, followed by a brief conclusion.
2 Artificial Neural Networks

The development of artificial neural networks was, as the name suggests, inspired by their biological counterpart. The neuroscience inspired name is furthermore suitable as artificial neural networks are today viewed as one of the key methods in the field of Artificial Intelligence (AI). Artificial neural networks have for example had great success in natural language processing and computer vision, as well as teach computers to play games such as Go [3], [12], [25]. However, the resemblance to biological neural networks is in practice limited. The goal of most research on artificial neural networks is not to model the brain, it is rather guided by many mathematical and engineering disciplines [8].

Although artificial neural networks were introduced in the 1950s it took five decades for these methods to gain the high level of popularity they hold today. This was due to several factors, one among them being the modelling limitations of the first network presented, the one layer perceptron. However, already in 1989 standard multilayer feedforward network with as few as one hidden layer (one layer less than the network described in Section 2.1 and illustrated in Figure 1) was proved capable of approximating any Borel measurable function from one finite dimensional space to another, to any desired degree of accuracy, given that the hidden layer contains sufficiently many units [10]. As such artificial neural networks can be viewed as a class of universal approximators. There is of course a distinction between what a network theoretically can represent and what learning algorithms are able to learn. Furthermore, there is a question of size to consider the sufficient number of units stated in the theorem might be enormous [8].

Neither the main principles of these methods, nor the algorithms for training them, have changed greatly since the 1980s even though the views on these methods have change significantly. This shift can mainly be attributed to the considerable improvements in computer power that have occurred in recent years [6]. Another contributing factor, is the increasing size of available datasets which has allowed for successful training networks that generalise well [12]. These two developments, with the aid of a small number of algorithmic changes, have enabled successful training of large neural networks, consisting of many layers, deep neural networks in other word.

Its is these deep neural networks which, to a great extent, account for the dramatic improvement in such areas as computer vision and speech recognition [17]. This success is mainly due to abilities discussed in Section 1.1 of discovering intrinsic structures in data by learning the data representation required for classification/evaluation from the raw data. The basic theory of artificial neural networks will be presented in the following sections.

Throughout this report the term ‘neural networks’ will be used to refer to artificial
neural networks, not biological. The terms neural network, artificial neural network, ANN and deep neural network will be used almost interchangeably, with the only distinction that a deep neural network refers to an artificial neural network with a large number of layers.

2.1 FeedForward Neural Networks

The aim of artificial neural networks is to approximate some function \( y = f^*(x) \) with \( y = f(x) \) where the input \( x \) can take many different forms and the target variable \( y \) can be either categorical or continuous. These models are called feedforward networks because information flows through the function in a chain [8]. Models that include information being passed "backwards" are called recurrent neural networks and will not be considered in this report.

Neural network layers generally consist of several units, or neurons, performing vector to scalar computations in parallel, thus forming a directed interconnected network, as seen in Figure 1.

Most functions used in the hidden layer neurons consist of two components:

\[
\begin{align*}
  z &= w^T x + b \\
  \phi(z) &= f(z).
\end{align*}
\]

Firstly, a linear transform is applied to the input vector \( x \) as in Equation 1, where \( w \) is a vector of weights and \( b \) is a scalar bias term. A non-linear activation function, Equation 2, is then applied to the result of 1. The function \( \phi(z) \), generally referred to as the activation function or the squashing function, takes on different shapes and will be discussed in Section 2.2.

A neural network with two hidden layers, each consisting of \( k \) units, and a linear output layer will now be described. The structure of this network is shown in Figure 1. The networks implemented for this project contain more layers however, the principles are the same and this smaller network is therefore presented for clarity.

Given a data set of \( n \) samples where each sample \( i \) takes the form \((x^i, y^i)\), with \( x^i \in \mathbb{R}^m \) \( y^i \in \mathbb{R} \) the network will have an input layer as follows:

\[
\begin{align*}
  Z_0^1 &= 1 \\
  Z_1^1 &= x_1^i \\
  Z_2^1 &= x_2^i \\
  \vdots \\
  Z_m^1 &= x_m^i.
\end{align*}
\]
Figure 1: Structure of network described in Section 2.1 with each circle representing a neuron and the arrows representing directed connections between the neurons.
where $Z_0^1$ is included to account for the bias term. The first hidden layer is then

\[
f^{(1)} = \begin{cases} 
Z_0^2 = 1 \\
Z_1^2 = \phi(w_{10}^1 Z_0^1 + w_{11}^1 Z_1^1 + w_{12}^1 Z_2^1 + \ldots + w_{1m}^1 Z_m^1) \\
Z_2^2 = \phi(w_{20}^1 Z_0^1 + w_{21}^1 Z_1^1 + w_{22}^1 Z_2^1 + \ldots + w_{2m}^1 Z_m^1) \\
\vdots \\
Z_k^2 = \phi(w_{k0}^1 Z_0^1 + w_{k1}^1 Z_1^1 + w_{k2}^1 Z_2^1 + \ldots + w_{km}^1 Z_m^1), 
\end{cases}
\]

where each equation corresponds to a neuron in the layer. This first hidden layer is followed by:

\[
f^{(2)} = \begin{cases} 
Z_0^3 = 1 \\
Z_1^3 = \phi(w_{10}^2 Z_0^2 + w_{11}^2 Z_1^2 + w_{12}^2 Z_2^2 + \ldots + w_{1k}^2 Z_k^2) \\
Z_2^3 = \phi(w_{20}^2 Z_0^2 + w_{21}^2 Z_1^2 + w_{22}^2 Z_2^2 + \ldots + w_{2k}^2 Z_k^2) \\
\vdots \\
Z_k^3 = \phi(w_{k0}^2 Z_0^2 + w_{k1}^2 Z_1^2 + w_{k2}^2 Z_2^2 + \ldots + w_{kk}^2 Z_k^2), 
\end{cases}
\]

\[
f^{(3)} = \begin{cases} 
Z_4^4 = w_{00}^3 Z_0^3 + w_{10}^3 Z_1^3 + w_{20}^3 Z_2^3 + \ldots + w_{kk}^3 Z_k^3. 
\end{cases}
\]

As each neuron in a layer takes the complete output of the previous layer as input, these types of layers are referred to as fully connected layers. Considering these equations in matrix notation, with a weight matrix $W^L$ for each layer $L$ where $W^1 \in \mathbb{R}^{k \times m+1}$ contains the weights from Equations 4, $W^2 \in \mathbb{R}^{k \times k+1}$ contains the weights from Equations 5 and $W^3 \in \mathbb{R}^{k+1}$ contains the weights from Equations 6 with $Z^1 \in \mathbb{R}^{m+1}$. The output of the network $Z^4$ is thus:

\[
Z^4 = f^{(3)}(f^{(2)}(f^{(1)}(x))) = W^3 Z^3 = W^3 \phi(W^2 Z^2) = W^3 \phi(W^2 \phi(W^1 Z^1)).
\]

As the final output layer, $f^{(3)}(Z^4)$ attempts to drive the output towards the target value $y_i$ for each given $x_i$ during training. The previous layers $f^{(1)}, f^{(2)}$ however do not have directly specified targets and as a consequence these layers are referred to as ‘hidden’. These layers can be seen as preprocessing of the data in a high dimensional space to learn a representation to give as input to the linear output layer. Instead of deciding on a kernel or manually engineering the features the hidden layers of the network are learning them. The strength of neural networks comes from these hidden layers [8].
2.2 Activation Functions

This section introduces several commonly used activations functions, variations of $\phi(z)$ as was seen in Equations 2, 4, 5.

The first activation function introduced was a simple step function as shown in Figure 2a. This is part of the perceptron, Equation 7, the simplest neural network which was presented by Rosenblatt in 1958 [24].

$$\phi(z) = \begin{cases} 1 & \text{if } w^T x + b > 0 \\ 0 & \text{else} \end{cases}$$  \hspace{1cm} (7)

A multilayer perceptron was the network considered in [10], when neural networks were proved to be universal approximators. However, other architectures have since been developed and have for certain problems proven to train faster and have a greater likelihood of reaching a better and more stable solution [16].

![Activation Functions](image)

Figure 2: Examples of commonly used activation functions in artificial neural networks.

Traditionally, the most commonly used activation function has been the sigmoid function, Equation 8, plotted in Figure 2b. However, in recent years with the success of deep neural networks sigmoid activation functions have slightly fallen out of favour for use in hidden layers. The reason for this is that the sigmoid function is prone to saturation during training. For a sigmoid to output 0 it has to be pushed to a regime where the gradient approaches zero, thereby preventing gradients to flow backward and prevent the lower layers from learning useful features [6]. (Training of neural networks will be discussed in Section 3.) Sigmoid functions can however still be useful for the right problem and with a suitable initialisation of the weight:

$$\phi(z) = \frac{1}{1 + \exp(-z)} .$$  \hspace{1cm} (8)

Hyperbolic tangent is another commonly used activation function:

$$\phi(z) = \tanh(z) .$$
As clearly seen in Figure 2c the hyperbolic tangent is in a linear regime around zero, which is an improvement on the sigmoid function. However, it still has the problem of saturation in other regions.

Studies by Glorot et al. [7] among others, have shown that training of neural networks proceeds better when the neurons are either off or operating mostly in a linear regime. Following these observations a new activation function has gained popularity, a "sharp" sigmoid function. This activation function is called Rectifier and takes the form:

\[ \phi(z) = \max(0, z) \]  

(9)

This non-linearity has the advantage of considerably shorter training time, due to it not being prone to the same saturation problem as the sigmoid and hyperbolic tangent functions [12]. A theoretical objection to this function is that it is not differentiable at 0, which was also one of the reasons why it took some time for it be be widely adopted. However, for implementation this does not present a challenge as the value zero can simply be approximated with a value close but not equal to zero [8].

One problem that can arise with rectifier activation functions is that they can "get stuck" on zero slopes as the gradient based methods used cannot learn on examples when the neuron is not activated [8]. A development of the rectifier has been suggested to deal with this disadvantage:

\[ f(z) = \max(\alpha z, z) \]  

(10)

Equation 10 is called a leaky rectifier, where the parameter \( \alpha \) determines the "leakiness". The neuron is then never completely unactivated. There are plenty of other options for activation functions exist, however so far no other function has proven to perform significantly better than than the rectifier on a wider range of problems [8].

2.3 Convolutional Layers

Artificial neural networks with at least one convolutional layer are referred to as convolutional neural networks (CNNs), or simply convolutional networks. Convolutional networks are designed to handle data in the form of multiple arrays, be they two dimensional (2D) or three dimensional (3D) [17]. These types of layers have gained huge successes within computer vision, with the network designed by Krizhevsky et al. for the ImageNet competition being a notable milestone [12], as well as the early success of LeCun with reading hand-written digits, presented in [15]. Convolutional networks have however also been successfully applied in other areas, for example for constructing data representations of the game board when teaching computers to play the game Go [25] and for natural language processing [3].

The convolution operation is described in Figure 3. This example is for an input matrix of size \( X \in \mathbb{R}^{3 \times 4} \) and a filter size of \( 2 \times 2 \). A convolutional layer will have
Elements of Output Matrix

\[ \phi(a\alpha + b\beta + e\gamma + f\delta) \quad \phi(b\alpha + c\beta + f\gamma + g\delta) \quad \phi(c\alpha + d\beta + g\gamma + h\delta) \]

\[ \phi(e\alpha + f\beta + i\gamma + j\delta) \quad \phi(f\alpha + g\beta + j\gamma + k\delta) \quad \phi(g\alpha + h\beta + k\gamma + l\delta) \]

Figure 3: This figure shows a convolution operation of a $2 \times 2$ filter, or kernel, on a $3 \times 4$ input matrix. Each of the expressions directly above are elements of the output matrix, which in this case has the dimensions $2 \times 3$. The first output element is obtained when the filter operates on the top left corner of the input matrix. The other elements are similarly obtained when the filter is applied to different regions of the input.

The network then learns different parameter values of $\alpha, \beta, \gamma, \delta$ for the different filters in the same way as neurons in standard fully connected layer will have different parameter values. These filters can vary in size, [12] for example reported using filters ranging from $11 \times 11$ to $3 \times 3$. The difference between these layers and the layers described in the Section 2.1 is that several neurons in a convolution layer share parameters and that each neuron is only connected to a small region on the input (the difference between the size of the input and the filter size is generally larger). When comparing a CNN with a standard ANN of equal size, the CNN therefore has considerably fewer connections and parameters and is consequently easier to train. At the same time the CNN’s theoretical best performance is usually only slightly worse [12]. This allows for the construction of a larger network, with the potential of improving the performance.

Convolutional layers are usually paired with a subsequent pooling layer, the most common kind being max-pooling layers. This layer does not perform any learning. For each $k \times k$ region in it outputs the maximum of that region. Pooling filters with $k = 2$ are commonly used. Pooling layers reduce the size of the data for each layer, which is particularly useful in computer vision where the input data is generally of a large size. It can also introduces a degree of invariance to input translations [1].
3 Training Deep Neural Networks

The training of the networks is handled as a classical supervised machine learning problem. A labeled data set is used, in this case meaning that the retention times are known. A portion of the data is set aside to be used as test set, used to compare with the predicted values to evaluate performance.

Training of neural networks is almost always done with gradient based algorithms, which requires a cost function to be specified. This section will describe the cost function used, the training algorithm and in addition a number of techniques used to improve training.

3.1 Cost Function

As for any optimization method a measure quantifying the difference between the predicted value and the target has to be defined. Minimizing this cost function is the aim of the training process. The retention time problem is a regression problem and consequently mean square error is a natural choice as a cost function:

\[ c = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \]

where \( y \) is the known target and \( \hat{y} \) is the predicted value, with the index \( i \) specifying the data point in the data set of \( n \) points.

As a measure to avoid overfitting a regularisation term is added to the cost function, by the same principles as in ridge regression:

\[ c = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \left( \sum_{l=1}^{L-1} \sum_{k=1}^{m} \sum_{p=1}^{w_{lp}} w_{lp}^2 + \sum_{p=1}^{m} w_{lp}^2 \right), \]  

(11)

where the sum of the square of all weights in the network are added as a penalty term. Note that these indices are adapted to the network example given in Section 2.1, it is however for example not necessary for all hidden layers to have the same number of neurons \( k \). The value of the regularisation parameter \( \lambda \) determines the extent to which the regularisation term influences the training.

3.2 Gradient Descent and Stochastic Gradient Descent

As an initial step in training a neural networks all weights in the network have to be set to a starting value. This is usually done by randomly initialising the weights to values close to zero. With this random set of parameter values the network can perform predictions for given input data. After this step however we need a technique for knowing how to change the weights in order to improve these prediction. Gradient descent is
a simple and commonly used technique to iteratively completing this task. It is based on calculating how small perturbations in the parameter values in each layer will affect the output, and changes the weights in the direction in weight space that reduces the cost function the most.

For each weight this computation is performed:

\[ w_{jp}^{l+1} \leftarrow w_{jp}^l - \eta \frac{\partial c}{\partial w_{jp}^l}, \]  

where \( t \) is the index for the iteration in training, and \( \eta \) specifies the learning rate. This straightforward expression becomes more complicated when considering how \( c \) depends on \( w_{jp}^l \). As the layers are connected in a chain \( 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \), requires knowledge of how \( Z_2 \) affects \( Z_3 \) and how \( Z_3 \) affects \( Z_4 \). The next section will describe how the resulting chain derivatives are dealt with in practice.

Using the basic gradient descent approach entails passing through the entire training set in each iteration and then updating the weights according to the average gradients. This method is today referred to as batch gradient descent [16]. A more commonly used approach is Stochastic Gradient Descent (SGD) which in each iteration randomly selects one or a small number of data points (a mini batch) on which to update the weights. The main advantage of this method is that it speeds up the learning.

An additional technique often used to further increase the learning rate is momentum. As indicated by the name, momentum favours subsequent learning steps that go in the same direction in weight space. Denoting the change in weight as \( \Delta w_{jp}^{l+1} = \eta \frac{\partial c}{\partial w_{jp}^l} \) gradient descent with momentum is given by:

\[ w_{jp}^{l+1} = w_{jp}^l + m \Delta w_{jp}^l, \]

where \( m \) is the momentum constant. This means that the weights are updated according to a moving average of gradients, not just the last result. This smooths the process of training, by decreasing oscillations and effectively increasing the learning rate in directions of low curvature. It thereby has the potential to increase the learning speed [16].

### 3.3 Backpropagation

The calculation of the derivative in Equation 12 often requires a summation over several chains of derivatives. The number of these weights for which the gradients have to be computed is also generally very extensive. These factors create the need for an effective algorithm to compute these derivatives. Backpropagation, an abbreviation of "backward propagation of errors" is by far the most commonly used algorithm for this purpose. Here follows an explanation of backpropagation, using the network example
introduced in Section 2.1, with \( k = 2 \). The layer wise modular approach of explaining is inspired by de Freita lectures in [4]. This approach has been chosen as it is consistent with how the training algorithm is generally implemented.

To start let us consider the cost function as a final layer to the network described in Equations 3-6. This gives the additional layer \( Z_5 \):

\[
c(W) = Z_5 = \sum_{i=1}^{n} [y_i - Z_4^i]^2 = [y_i - W^3 \phi(W^2 \phi(W^1 Z_1))]^2.
\]

For \( k = 2 \) the dependence of the cost function \( c \) can then be described as follows:

\[
c(W) = Z_5 \left( Z_4 \left[ W^3, Z_2^3 \left( W_2^2, Z_1^2 \{ W_1^1, Z_1 \}, Z_2^2 \{ W_2^1, Z_1 \} \right) \right], \right.
\]

\[
Z_2^3 \left( W_2^2, Z_1^2 \{ W_1^1, Z_1 \}, Z_2^2 \{ W_2^1, Z_1 \} \right) \right). \tag{14}
\]

\[
\left. \frac{\partial c(W)}{\partial W_1^1} = \frac{\partial Z_5}{\partial Z_4} \frac{\partial Z_4}{\partial Z_3} \frac{\partial Z_3}{\partial Z_2} \frac{\partial Z_2}{\partial W_1^1} + \frac{\partial Z_5}{\partial Z_4} \frac{\partial Z_4}{\partial Z_3} \frac{\partial Z_3}{\partial Z_2} \frac{\partial Z_2}{\partial W_1^1}. \tag{16}
\right.
\]

Even for this simple network Equation 16 requires the calculation of several partial derivatives. An important fact to note is also that \( \frac{\partial Z_3}{\partial Z_1} \) is used twice. The partial derivative of \( c \) with respect to \( W_2^3 \) will also to a large extent consist of the same partial derivatives. The backpropagation algorithm allows for computations of the derivatives recursively and without having to redo the computation of any single derivatives.
Figure 4: Three messages pass through each layer in the network; a forward pass of function values, a backward pass of derivatives and in case of layer parameters output of gradients with respect to the different parameters.

The workings of the algorithm can easily be considered for full layers. Figure 4 shows the flow of information through a layer. For each layer three separate quantities have to been determined. These quantities are the function the layer computes, the derivative of the layer with respect to the inputs and finally the derivatives with respect to the parameters.

The function values are passed from layer $L$ to layer $L + 1$ through the network. This is referred to as the forward pass:

$$Z^{L+1} = f(Z^L).$$

This is the functionality required to compute a prediction from input. The partial derivative with respect to the input are passed from layer $L + 1$ to $L$, backwards through the network. This is referred to as the backward pass:

$$\delta^L_i = \frac{\partial c(W)}{\partial Z^L_i} = \sum_j \frac{\partial c(W)}{\partial Z^{L+1}_j} \frac{\partial Z^{L+1}_j}{\partial Z^L_i} = \sum_j \delta^{L+1}_j \frac{\partial Z^{L+1}_j}{\partial Z^L_i},$$

where $i$ is the index of the unit in layer $L$ and the index $j$ specifies the output of the layer. The upper bound of the summations change depending on the size of the layers as follows:

$$L = 2: \quad i \in [1, m], \quad j \in [1, k]$$
$$L = 3: \quad i \in [1, k], \quad j \in [1, k]$$
$$L = 4: \quad i \in [1, k], \quad j = 1.$$

Finally, the derivatives with respect to the parameters are computed, as shown in the
example Equation 16:

$$\frac{\partial c}{\partial W_i^L} = \sum_j \frac{\partial c}{\partial Z_j^{L+1}} \frac{\partial Z_j^{L+1}}{\partial W_i^L} = \sum_l \delta_j^{L+1} \frac{\partial Z_j^{L+1}}{\partial W_i^L}.$$  

With each layer in the network performing these three functions all the information required to apply SGD are obtained in an effective way for each iteration in the training. There are also today a wide range of software that offer implementation of training through backpropogation, hence these derivatives need not be computed by hand [14].

3.4 Dropout

Ensemble learning, the technique of combining several models, is frequently used for different machine learning techniques to improve performance and has been proven to significantly reduce variance. However, in the case of deep neural networks it often becomes too computationally expensive to train several networks [12]. Using dropout, first described in [9] is a computationally inexpensive solution that produces similar effects to ensemble learning.

The dropout layer can be viewed as a filtering layer. The output $Z_j^L$ for each $j$ and each $L$ among the hidden layers is set to 0 with a probability $p$ ($p = 0.5$ is usually given as a default). Thus, the network samples a different architecture for each input. This produces a more robust solution less prone to overfitting by reducing advanced co-adaptions in the network. Neurons can not rely on the presence of others to the same extent [12].
4 Data

The data used in this project has been obtained from LC/MS experiments on a yeast sample. The retention time has been recorded and the observed spectrum has been matched to a peptide. The peptide is recorded as a sequence of letters, each letter corresponding to an amino acid. The retention time of each peptide is recorded in minutes.

The data set used for the training and the evaluation of the models is a collection of results from five runs with the same setup, each containing approximately 14 000 peptides. The running time of the experiment was approximately 263 minutes for each run. There is a considerable overlap of peptides between the runs which reduces the number of peptides that can be used as there can be no duplicates between the training and test sets. In total the data set contains 24953 unique peptides, 4000 of which were set aside for the final evaluation of the models, as a test set.

There are several sources of error in the data sets that are important to consider when evaluating the performance of the models. As previously discussed there is an element of uncertainty when matching observed spectrum to peptides. The data sets used do however have a very low false discovery rate. All peptide identification with a posterior error probability higher than 1% have been removed from the set. However, we still expect to have some misclassified peptides in the set. This is one factor that can be expected to decrease the accuracy of predictions.

Another issue to consider is in-source fragmentation. During the ionisation process prior to the MS step of the analysis peptides sometimes break into smaller parts. This means that the peptides analysed in the MS are not the same ones that eluted from the chromatography column [19]. Such smaller peptides have generally been removed from the data set, however there may still be occurrences.

Analysing the retention times of the peptides identified in all five runs gives an indication of the extent to which these sources of error may affect the results. Figure 5 shows the retention times of the 4866 peptides found in all five runs of the experiment. As expected the majority of the peptides are aligned between the runs, although with a slight shift. However, the peaks show that some peptides are reported with larger variations in retention time. If we classify outliers as peptides with a retention time that deviates with 10 min or more from the average of the other four runs, the approximated percentage of incorrectly classified peptides is 0.25%, thus lower than the FDR. The root mean square error (Equation 17) of the duplicates shown in Figure 5 is 2.5 min. This can be viewed as the minimum achievable error level.
Perhaps the opposite benchmark to consider is the performance of the null model. For the training data set, if predicting all retention times as the sample mean the RMSE is 61.2 min. We thus expect the model to have a higher error than 2.5 min and to, after successful learning, have a considerably lower test error than 61.2 min.

4.1 Data Representations

The performance of a statistical learning model is always heavily dependent on what features or what more general data representation is applied to it. Initial tests with amino acid frequency and pair frequency as features were performed. In the case of amino acid frequency the input vector has the dimensions \( x \in \mathbb{R}^{20} \), one element for each amino acid. In the case of pair frequency \( x \in \mathbb{R}^{210} \), one element for each possible amino acid pair. Further details on these representations are found in Appendix B.

However, as the aim of the project was to utilize neural networks’ ability to learn representations of the data, a more general form was sought. A matrix representation was therefore applied. For each peptide used to train or evaluate the model a matrix \( X \in \mathbb{R}^{20 \times k} \) was constructed, where the 20 rows each represent an amino acid and the \( k \) columns represent the position in the sequence.

As a toy example, a peptide with the amino acid sequence EHSENEHKESGK would generate the following matrix:
This representation still leaves two degrees of freedom, the order of the amino acids and the choice of $k$. Ideally, we wish the rows to be ordered in such a way that amino acids which together in sequence strongly affect hydrophobicity are represented by adjacent rows. The reason for this being that important patterns in the matrix might be found more easily with small convolutional filters. The linear combination of the amino acids’ independent contributions offers a crude model for predicting a peptide’s retention time. However, the internal ranking of the elements of the ordinary least squares estimate, each corresponding to a different amino acids, give an indication of their hydrophobic properties. A comparison of these weights were therefore used to order the amino acids, resulting in the following order,

$$\text{H R K N Q G S C T E A D P V Y M I L F W}.$$  

A more detailed explanation of this sorting is found in Appendix B.1. This is not a perfect solution, however it is likely better than simply using the alphabetical order and furthermore we expect the neural networks to learn which patterns are important even if important patterns do not consist of direct diagonals.

The other degree of freedom, the choice of $k$ determines the maximum peptide length that the model can handle and also the number of columns of each peptide matrix, regardless of sequence length. A more elegant solution would be to have matrices of variable number of columns. However, the number of methods that can handle input of variable length is unfortunately today limited and for convolutional networks the size of the input has to be fixed. Figure 6 shows the distribution of peptide lengths in the main datasets used, the second dataset shows a similar distribution. As shown in the figure the majority of the peptides are of lengths in the range $k = [7, 20]$. Choosing $k = 50$ will allow the model to handle all peptides in this dataset. However, the model will have a considerable larger number of parameters and can be expected to train slower. To the opposite extreme, selecting $k = 10$ will give a smaller model and with few excess columns in each matrix, however the model can only use approximately 30% of the data. With these factors in mind $k = 25$ was selected as a standard maximum length, allowing for the inclusion of 97.2% of the dataset. It should be noted that
the exclusion of the longest peptides is likely to slightly improve the observed accuracy of the model. Previous studies have shown that retention time predictions are less accurate for longer peptides, removing them from the validation set thereby introduces a slight bias [27].

![Distribution of peptide lengths in the training set.](image)

Figure 6: Distribution of peptide lengths in the training set.

### 4.2 Mirror Images

A technique often used in computer vision to extend a dataset is to use multiple copies of the same data, yet with added modifications, such as rotating the images or slightly shifting pixel values [12]. One such technique that can be applied to this problem is using mirror images of the sequences, effectively doubling the dataset. The matrix for the mirror image of the peptide (EHSENEHKESGK) shown in the previous section will take the form:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

As this is the same peptide, the target retention time during training is the same as for the original matrix.
4.3 Scaling

Scaling is a common practice in most machine learning methods. The target values, that is the retention times of the peptides, are scaled prior to training. The training set is scaled to have approximately mean zero and unit variance. This is achieved by subtracting the mean of the set and then dividing by the sample standard deviation.
5 Evaluation

This section presents the measures used to evaluate the performance of different models in this project.

A natural measurement to consider is the root mean square error:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i} (y_i - \hat{y}_i)^2}.$$  \hspace{1cm} (17)

This metric is equivalent to the cost function, if the penalty term is ignored, and it is thus directly correlated with the quantity being minimised in the optimisation. However, following the discussion in Section 4 we expect approximately 1% of the peptides to be incorrectly classified, meaning that the retention time prediction for them may yield a considerably larger error. Such outliers can affect the RMSE. This is one reason why considering a confidence interval, rather than the average result, can also be useful.

A confidence interval with a significance level of $\alpha = 0.05$ is employed to evaluate the models. This can be expected to exclude outliers, resulting from for example incorrect identifications. The confidence interval can be estimated by a empirical nonparametric confidence interval based on the whole sample. This measure is also useful in practice if using the retention time predictor to confirm peptide identifications. For a certain time $t$ let:

$$S_t(y, \hat{y}) = \{ i : |y_i - \hat{y}_i| < t \}.$$  \hspace{1cm} (18)

The confidence interval is then given by $2t$ when the following equation is satisfied:

$$|S_t| = \alpha \times n.$$  \hspace{1cm} (19)

With the conditions of Equations 18 and 19 being fulfilled and given $\alpha = 0.05$, 95% of the predicted retention times will be within $t$ time units from the actual values. It should be noted that this is the confidence interval for a whole data set. It is not unique to each data point. It is thus a measure of the uncertainty on the whole set evaluated, not for a specific given prediction. The confidence interval can be a useful measure if using the prediction results to verify a peptide match in a new experiment. Given that the peptide match is correct there is then a 95% probability of the predicted retention time falling inside this interval.

To make comparisons of results from different runs easier, it is advisable to scale the measurements. This can be done by considering the above mentioned measures relative to the difference between the highest and the lowest retention time in the set. Ideally this scaling should be done with respect to a fixed set of peptides to mark a known interval. However, using the retention times of the first and last peptides to elute gives a estimate of the running time used. Hence a relative confidence interval can be specified as

$$\Delta t_r^{95\%} = \frac{2t_{\text{r}}}{\max(y) - \min(y)}.$$
where \( \max(y) \) and \( \min(y) \) are the lowest and highest retention times observed in the training set.

A final measure, that has been considered in previous publications on retention time prediction, is the correlation coefficients between predicted and observed retention times [20]. The sample correlation is calculated as follows:

\[
\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i.
\]

\[
r = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2} \sqrt{\sum_{i=1}^{n} (\hat{y}_i - \bar{\hat{y}})^2}}.
\]

These four metrics will be used in the presentation of the results in Section 8 and in the discussion that follows in Section 9.
6 Implementation

There are today many software packages for working with neural networks, such as Torch, Theano, and packages for MATLAB. I have chosen to use Lasagne, a light-weight library for building and training neural networks in Theano [14]. Lasagne is an open-source project started in 2014 that offers implementation of different network structures and also includes optimisation packages. Lasagne enables the use of GPU, through the CUDA toolkit, during training, considerable improving the speed of learning.

All code for the project has been written in Python.
7 Model Development

This section describes the development of a convolutional network taking the matrix representation of peptides, discussed in Section 4.1, as input. Models based on simpler feature vectors, developed at early stages of this project, will be presented in Section 8 for comparison, yet will not be discussed in this section. Details of these models are instead found in Appendix B.

The selection of artificial neural networks as the method to be applied to the task of retention time prediction still leaves a large amount of choices to be made regarding structure of the network and values of hyperparameters to be used. In general, there are unfortunately not yet many definitive guiding theoretical principles for the design of hidden units, although it is an extremely active area of research [8]. Recommendations found in literature are often based on experimental findings, in essence trial and error. The model development, central in this project, has been based on common practices and a series of experiment. These experiments, evaluated using a validation set consisting of 3906 peptides, will be presented in this section and will be discussed further in Section 9.

The aim of the model development is to determine a structure that can be successfully trained to accurately predict retention times of peptides. This requires a model that is sufficiently complex to represent the problem, a network of sufficient size. In addition to this, the optimisation method has to manage to minimize the cost function sufficiently. The success of these two factors will be manifested by a low training error. However, as the goal is to predict retention times for unknown peptides, a third criteria is introduced; the model has to generalise well to new data. This is achieved by employing methods that regularise the model, avoiding overfitting to the training set. We hence arrive at the classical trade off in statistical learning between bias and variance [11]. A larger model poorly regularised will yield high variance and low bias, whilst a small model with strong regularisation will yield low variance, yet high bias. This high variance is manifested by being very sensitive to the data used to train the model. The ideal solution is a trade off between these two. For neural networks the most successful approach, when viewing breakthroughs such as [12] and [25], appears to be constructing a large network and then attempting to strongly regularise it with convolutional layers, techniques such as dropout, and the use of a large training set.

Two additional factors that have to be considered are limitations on time and computer power. Increasing the network size can often be compensated for by also increasing the training set, thereby improving the regularisation. However, this brute force approach will inevitably increases the required computer power and the time for training the network [8]. Even with few limitations on time and access to supercomputers, faster learning requiring less resources can only be seen as a positive factor and also perhaps has the added benefits of being more comprehensible and more elegant. For this pro-
ject however, training time has not been a central concern. All attempted network structures have been trained in less than 60 min, even when using the full training set.

### 7.1 Layer Architecture

As discussed in Section 4 the data can be represented in a 2D array form with limited loss of information. This fact and the benefits of convolution networks presented in Section 2.3 favours the use of convolutional layers. The common practice for CNNs is to alternate convolutional layers with pooling layers for the first few layers of the network and to have layers of fully connected 'standard' neurons in the final, top layers. This general structure has been the starting point of the model development in this project.

As discussed in Section 4.1 mirror images of the peptide arrays can easily be created, effectively doubling the data set, with the potential of improving the generalisation of the network. An alternative option, that was been the main focus of this project, is to give both mirror images as input to the network simultaneously. This is possible as they share the same target value. For this purpose a branched network was implemented, as illustrated in Figure 7. Two branches consisting of convolution layers, followed by fully connected layers were setup to each take one of the matrices as input. The output of these branches was then jointly fed through a series of fully connected layers before reaching the single linear output neuron. The results of this branched network will be compared to those of the standard unbranched network in Section 8. Advantages that can be expected are improved generalisation and a possible decrease in training time, compared to training the network on twice the number of data points.

The activation function to be used in the fully connected layers, as well as in the convolution layers, also had to be determined. As presented in Section 2.2 there are definite advantages of using the leaky rectifier activation function and consequently the tests in this project focused on implementations involving this nonlinearity. However, results from using other activation functions will be presented as reference in Section 7.2, after the hyperparameter values have been set.

This concludes the discussion of the general structure being tested. What remains is determining the values of all hyperparameters such as number of layers, convolution filter size and pooling layers. These will be discussed in the next section.
7.2 Selecting Hyperparameter Values

A summary of the hyperparameters to be determined regarding the structure of the network have been summarised in Table 1, along with the range of values tested. Several factors make this task more challenging. Firstly, these parameters cannot be optimised individually, for example the best width of layers to use is likely to depend on the chosen number of layers and vice versa. This co-dependence naturally makes the process of selecting hyperparameter values more involved. Secondly, as there is a stochastic element to the training results between trials vary. The average performance should therefore be considered from a set of trials, as well as the variation in performance to give an indication of the stability of the model. The training of multiple instances of each network structure becomes time consuming and the number of structures tested must therefore be limited.

The results for some of the initial test rounds are shown in Table 8 in Appendix A. These tests suggested that the use of convolution and pooling layers did not improve the performance of the model. Increasing the number of convolution layers decreased the performance and one of the best performing networks was the network without convolution layers altogether. As a consequence of these results the next test round focused on one and two convolution layers and removing the pooling operations between layers. The structures tested in this round all had the same setup of fully connected layers, $3 \times 30$ neuron in each branch, followed by $1 \times 15$ neurons in the merged layer (one of the best performing structures from the initial tests). A selection of these results are summarised in Table 9 in Appendix A. In the majority of trials removing the pooling layers improved the performance of the network.

The RMSE of the majority of the networks was in the range of 10-11 min, suggesting
that the performance is not particularly sensitive to the choice of convolution filter size. However, to proceed one structure was chosen, that with the best performance Table 9. This network is presented in Table 2.

Net 11.1 was chosen as it had the lowest RMSE of the tested networks. Using Net 11.1 as a foundation, the affect of changing the number of fully connected layers and the width of these layers were then investigated more thoroughly. Figure 8 summarises the findings by plotting RMSE as a function of the number of layers and by plotting average RMSE as a function of the width of the layers. Each structure was trained 10 times and the sample standard deviation among these 10 results has also been plotted.

Graph 8a shows that up until 35 neurons the performance of the model is improved when adding neurons to each layer. The training error continues to decrease slightly even after 35. The variation between model performances after re-training also decreases with the increase in neurons.

Graph 8b shows that the performance of the network is best for 2 - 4 layers. When more layers are added both the test and training error are increased and greater variations in the results are observed. As depth in networks are generally more challenging for optimisation methods than widths increasing the number of epochs for the training is one example of a measure that might improve the performance of deeper networks. However, preliminary test on this suggest that the networks in many cases start to overfit when adding more epochs.
<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Net 11.1</th>
<th>Notation in Figure 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv layer 1</td>
<td></td>
<td>a)</td>
</tr>
<tr>
<td>No. conv filters</td>
<td>30</td>
<td>a)</td>
</tr>
<tr>
<td>Filter size</td>
<td>4x2</td>
<td>a)</td>
</tr>
<tr>
<td>Pooling</td>
<td>No</td>
<td>a)</td>
</tr>
<tr>
<td>Conv layer 2</td>
<td></td>
<td>a)</td>
</tr>
<tr>
<td>No. conv filters</td>
<td>30</td>
<td>a)</td>
</tr>
<tr>
<td>Filter size</td>
<td>4x4</td>
<td>a)</td>
</tr>
<tr>
<td>Pooling</td>
<td>No</td>
<td>a)</td>
</tr>
<tr>
<td>Number of ReLU layers in branches</td>
<td>3</td>
<td>b)</td>
</tr>
<tr>
<td>Width of ReLU layers in branches</td>
<td>30</td>
<td>b)</td>
</tr>
<tr>
<td>Number of final ReLU layers</td>
<td>1</td>
<td>c)</td>
</tr>
<tr>
<td>Width of final ReLU layers</td>
<td>15</td>
<td>c)</td>
</tr>
<tr>
<td>Final training error</td>
<td>8.2 (min)</td>
<td></td>
</tr>
<tr>
<td>Test error</td>
<td>9.7 (min)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: The structure of the best performing network from the first two rounds of trials. The reported errors are the average error of three separate trainings of the network, trained and evaluated on the same data.

Figure 8: Both plots show the results of tests performed on a network with convolution layer structure according to Table 2. In (a) RMSE is plotted as a function of the number of neurons per fully connected layer, in a network containing four fully connected layers. (b) shows RMSE as a function of the number of fully connected layers of width 30. The layers have alternately been added in the branches and the final layers, (corresponding to b) and c) in Figure 7). Both graphs give the mean RMSE (min) obtained over 10 trials. The marked grey areas show the sample standard deviation of these 10 trials.
<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Net 11.1</th>
<th>Net 11.2</th>
<th>Net 11.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv layer 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. conv filters</td>
<td>30</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>Filter size</td>
<td>4x2</td>
<td>4x2</td>
<td>4x2</td>
</tr>
<tr>
<td>Conv layer 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. conv filters</td>
<td>30</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>Filter size</td>
<td>4x4</td>
<td>4x4</td>
<td>4x4</td>
</tr>
<tr>
<td>Number of ReLU layers in branches</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Width of ReLU layers in branches</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>Number of final ReLU layers</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Width of final ReLU layers</td>
<td>15</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>Final training error</td>
<td>8.1</td>
<td>6.2</td>
<td>6.0</td>
</tr>
<tr>
<td>Test error</td>
<td>10.1</td>
<td>10.1</td>
<td>10.0</td>
</tr>
<tr>
<td>Test error STD</td>
<td>0.9</td>
<td>0.5</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Table 3: The results of the final trials from which Net 11.3 was selected as the main model.

Given the results in Figure 7 a final round of tests were performed, altering Net 11 according to the findings. The results are presented in Table 3. Note that these results are more reliable than those presented in Tables 2 as the average RMSE is calculated from the results of 10 trials, instead of three. The difference in RMSE on the test set between the three structures is negligible. However, Net 11.3 has a lower variation between re-trainings and also has the lowest training error and was consequently chosen as the main model.

There are also a series of hyperparameters pertaining to the training of the networks that can be adjusted. These are summarised in Table 4. These values have been taken from literature and different values have not been tested to any great extent, with the exception of the dropout level.

All reported results are for networks training during 50 epochs. This limit was set heuristically as many structures were found to overfit when running the training for 100 epochs or more. This limit on the number of epochs also has the advantage of decreases the learning time, enabling more tests to be performed.

Figure 9 clearly shows the advantage for a network of this depth of using the leaky rectifier (Equation 10) as activation function, rather than the traditional sigmoid function (Equation 8). The training error obtained with the sigmoid network is approximately that of the null model and the test error is slightly higher. This clearly shows that the training of the sigmoid network is unsuccessful. It should however be noted that this
<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate, $\eta$</td>
<td>0.01</td>
</tr>
<tr>
<td>Momentum, $m$</td>
<td>0.9</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0.001</td>
</tr>
<tr>
<td>No. of Epochs</td>
<td>50</td>
</tr>
<tr>
<td>Dropout level</td>
<td>0.1 - 0.5</td>
</tr>
</tbody>
</table>

Table 4: Hyperparameters pertaining to the training of the network. $\eta$ and $m$ are explained in Equations 12 and 13. $\lambda$ refers to the parameter determining the weight of the L2 regularisation used in the cost function, see Equation 11.

is not the best performing sigmoid network for this task. A smaller network performs better than the sigmoid version of Net 11.3, although the training of a ReLU network is generally always easier.

Figure 9: Comparison of learning curves of Net 11.3 with sigmoid and rectifier activation functions.

Finally, tests were performed for different levels of dropout applied to Net 11.3, the results of which are presented in Table 5. As seen in the table, none of the tested levels of dropout improved the performance of the final model. When the number of epochs was increased the results improved slightly, for 200 epochs an test error of 11.6 min was obtained for 50% dropout level. Yet the performance was still better without dropout. Consequently, no dropout was applied in the training of the final model.
<table>
<thead>
<tr>
<th>Dropout level</th>
<th>Validation Error</th>
<th>STD Validation Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>10.3</td>
<td>0.4</td>
</tr>
<tr>
<td>0.4</td>
<td>11.5</td>
<td>0.5</td>
</tr>
<tr>
<td>0.5</td>
<td>12.6</td>
<td>0.6</td>
</tr>
<tr>
<td>0.6</td>
<td>14.4</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Table 5: Results achieved when applying different levels of dropout between the fully connected layers of Net 11.3.

### 7.3 Structure of Final Model, Net 11.3

![Diagram of Net 11.3](image)

Figure 10: Structure of selected network, Net 11.3. The performance of this network will be evaluated in Section 8.
8 Results

In this section the performance of the model Net 11.3, the development of which was described in Section 7, is presented. Its performance will also be compared to that of earlier stage models implemented for this project and to the performance of the ELUDE software [18].

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Training Error (min)</th>
<th>Test Error (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null Model</td>
<td>61.6</td>
<td>62.5</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>22.0</td>
<td>22.4</td>
</tr>
<tr>
<td>AA Frequency ANN</td>
<td>15.5</td>
<td>15.9</td>
</tr>
<tr>
<td>AA Pair Frequency ANN</td>
<td>7.7</td>
<td>14.4</td>
</tr>
<tr>
<td>Unbranched CNN</td>
<td>10.4</td>
<td>11.7</td>
</tr>
<tr>
<td>Branched CNN (Net 11.3)</td>
<td>6.0</td>
<td>10.0</td>
</tr>
</tbody>
</table>

Table 6: The performance of the different model types implemented. The errors are given in terms of RMSE and are the averages of 10 trials. Details on the different models used can be viewed in Appendix B. The network run for the Single CNN has the same structure as the main model Net 11.3, with the exception of only having one branch.

Figure 11: Scatter plot of predicted retention times using Net 11.3 as a function of the observed retention times for the 3906 peptides in the test set.
Table 7: Comparison of relative confidence intervals and sample correlation between Net 11.3 and ELUDE.

<table>
<thead>
<tr>
<th>Model</th>
<th>$\Delta t^{90%}$</th>
<th>Correlation</th>
<th>Training Data</th>
<th>Training Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELUDE</td>
<td>17.4-26.6 %</td>
<td>0.92-0.97</td>
<td>See [19],[20]</td>
<td>-</td>
</tr>
<tr>
<td>ELUDE</td>
<td>21.1%</td>
<td>Na</td>
<td>Yeast 40cm, 1850 samples</td>
<td>-</td>
</tr>
<tr>
<td>ELUDE</td>
<td>17%</td>
<td>0.98</td>
<td>Main set 20 000 samples</td>
<td>28 h</td>
</tr>
<tr>
<td>Net 11.3</td>
<td>23%</td>
<td>0.97</td>
<td>Main set, 2000</td>
<td>40 s</td>
</tr>
<tr>
<td>Net 11.3</td>
<td>14%</td>
<td>0.98</td>
<td>Main set, 20 000 samples</td>
<td>5 min</td>
</tr>
</tbody>
</table>

Figure 12: Histogram showing the deviance of the predicted retention time from the observed elution time for the test set, using Net 11.3 and ELUDE.
Figure 13: Average RMSE of predictions with Net 11.3 and ELUDE, plotted as a function of peptide length.
Figure 14: RMSE plotted as a function of training size used to train Net 11.3. All network instances were trained over 50 epochs. Adjusting the number of epochs to give a constant level of data exposure during training (e.g. setting the number of epochs to 500 for 2000 training peptides for an even comparison with 50 epochs for 20,000 training peptides) did not significantly change this result. In several cases the results were indeed poorer, due to overfitting of the networks. Training on 2000 peptides over 500 epochs still gave an average RMSE of 15 min.
Figure 15: Relative confidence interval plotted as a function of training size, for a constant number of 50 epochs.
9 Discussion

9.1 Comparison of the Different Implemented Models

Table 6 summaries the performances of the different model types implemented, sorted in order of test error value. The structures of the earlier stage models of the project are all presented in Appendix B. As expected all models perform better than the null model. The performance of the linear regression model and the amino acid (AA) frequency ANN show that retention times can be predicted with reasonable accuracy without having information of the complete structure.

Training the same network on a vector representation of the AA pair frequency noticeably decreases the training error, showing that the model is more adept at representing the problem. The test error is however only slightly lower, suggesting that the model fails to generalise as well as the CNNs. It may however well be possible to further reduce the test error slightly by fine tuning parameter values. The unbranched CNN further improves the performance on the test set and finally, the best scoring model in Table 6 is the branched CNN Net 11.3, the development of which was described in Section 7. Figure 11 shows the predicted retention time using Net 11.3, plotted as a function of observed retention time.

The branched network was implemented in the hope of improving generalisation by effectively building two data representations through the branched layers of each peptide, thereby improving the chance of the network picking up the important structural elements of the data and reducing the risk of overfitting. It is however slightly surprising that the branched structure produced such a low training error compared to the unbranched structure. It was also hoped that the training time would be reduced, compared to feeding the network with twice the amount of data (when adding mirror images separately). However, the training time was only reduced by 17%. A more effective implementation of the network could probably reduce the training time of the branched structure.

It should be noted that more time was spent optimising the performance of the branched CNN structure than on any of the other models. Early evaluation of the model types however suggested that this structure had the greatest potential and to limit the scope of the project focus was therefore put on its development. It is unlikely that the ranking of the models would change even if more time was spent attempting to improve the performance of the other models.

9.2 Comparison of Net 11.3 and ELUDE

Table 7 shows how the performance of Net 11.3 compares to the benchmark model ELUDE. With a small training set ELUDE performs better than Net 11.3. However, as seen in Figure 15 the performance of Net 11.3 improves significantly when the train-
ing set is extended. ELUDE is not designed to handle large training sets and does not show the same improvement in accuracy when more data is added. Increasing the training set from approximately 2000 peptides to 20 000 reduces the confidence interval from 21% to 17% for ELUDE. The corresponding change for Net 11.3 is from 23% to 14%. A comparison of the deviance from the observed retention times is found in Figure 12.

Figure 13 shows that ELUDE has a more even accuracy for different peptide lengths, compared to Net 11.3. Net 11.3 performs significantly better for shorter peptides than for longer ones. This explains the decrease in accuracy for peptides with higher observed retention times seen in Figure 11 as longer peptides generally have longer retention times [27]. The limited exposure of the network to longer peptides during training is presumably one reason for the trend seen in Figure 13. As seen in Figure 6, the number of long peptides is limited in this dataset. A potential solution would be simply up-sampling the longer peptides during training, although this method has not been explored.

9.3 Comment on the Structure of Net 11.3

During the background research for this project, no similar applications of convolutional neural networks were found. There is therefore no clear benchmark with which to compare the structure of this network. The ImageNet CNN presented in [12] contains five convolution layers, alternated by pooling layers and followed by three fully connected layers. All layers had a larger width than the layers in Net 11.3. The number of fully connected layers found to be optimal is the same. However, the design of convolutional layers differs.

The absence of pooling layers is perhaps one of the more noticeable features of this network, as they are a very common practice. The difference in the input dimensions is one explanation for this difference. The ImageNet input for example is a 256 × 256 pixel image where each pixel has an RGB value. Our input is a sparse 20 × 25 matrix. The pooling operation, which each time throws away 75% of the information is likely better suited for larger input dimension and less sparse data, where the loss of some information throughout the network may even be desirable. It is also better suited for input that is less sensitive to translations, again such as images.

The lack of improvement gained from dropout is another noteworthy finding. Several sources have reported that introducing dropout of 50% between the fully connected layers improve the performance of neural networks. In the case of Net 11.3 however, the performance decreased with the increase in dropout level, as reported in Table 5. The most probably explanation for this is that the network is not sufficiently large to benefit from the regularising effect of dropout. It is possible that dropout is more successful for still larger networks, trained over more epochs.
9.4 Future Work & Possible Improvements

The reproducibility of experimental retention times, estimated for this experimental setup to a RMSE of 2.5 min, is four times lower than the RMSE obtained with Net 11.3. This suggests that it ought to be possible to construct a model that performs even better.

Adding more data to the training set improves the performance of Net 11.3, as seen in Figures 14 and 15. However, the performance appears to start converging at 18 000, although unfortunately there is not more data to fully confirm this. A convergence could be due to the fact that the number of layers and the width of the layers have been optimised for this amount of data, as seen in Figures 8a and 8b. For a larger dataset an increased network size may well yield the minimum RMSE, compared to the minimum seen in Figures 8a and 8b. A larger network might then improve upon this prediction accuracy.

In addition to increasing the training set, an endless amount of time can be spent on optimising the structure hyperparameters as well as testing different values of learning parameters. Net 11.3 is likely to be suboptimal, even for this given training size. However, most trials during the model development only gave slight variations in performance and the optimal network of this general structure is therefore unlikely to perform considerably better than Net 11.3.

There are also a number of improvements regarding other model capabilities, which can be considered. One desired improvement would be the construction of a model which allows for variable input sizes, without requiring a maximum length being set. However, convolutional networks are as yet unfortunately limited in this respect. This would therefore require the use of some other method or some clever preprocessing of the data.

Another improvement would be having the predictor estimate the uncertainty of each predicted retention time separately, rather than the confidence interval being based on the entire sample. As seen in Figure 13 there is for instance a strong positive correlation between peptide length and RMSE. There are also other factors that are likely to affect the confidence of the estimate, such as similarity of the peptide to peptides in the training set. The difficulty of the prediction is dependent of the peptide in question, and this fact could potentially be incorporated in the model.

An interesting additional test on this model would be exploring its capability of recalibrating to new experimental conditions using only a small calibration dataset. This could likely be achieved by initializing the network with the current parameter values of Net 11.3, before starting training with a small calibration dataset. This would however requires access to an additional dataset, obtained from a separate experimental setup.
10 Conclusion

This thesis has applied convolutional neural networks to the task of predicting liquid chromatographic retention times of peptides. The aim of which is to increase the number of protein identifications in shotgun proteomics and to improve targeted mass spectrometry experiment. The final network was a branched convolutional neural network consisting of two convolutional layers of 40 filters each, followed by three fully connected of width 50 and a final single linear output neuron. Convolutional networks are designed to handle data in array format and each amino acid sequence was therefore represented by a matrix $X \in \mathbb{R}^{20 \times 25}$, with each row corresponding to a type of amino acid and the columns representing the position of the amino acids in the peptide. The two mirror images of this representation were fed simultaneously to the network, through one branch each, merging before the final two layers.

This model achieved a RMSE of 10.0 min, 3.8% of the total running time of the liquid chromatography and a 95% confidence interval proportional to 14% of the running time, when trained on 20 000 unique peptides from a yeast sample. The software ELUDE, used as a benchmark, produced a confidence interval of 17% when trained and evaluated on the same data. ELUDE is still preferable when the amount of data is limited, however for larger training sets the presented CNN performs better, particularly when taking into consideration the considerably shorter training time. An interesting extension to this work would be testing the CNN’s capability to re-calibrate to a new experimental setup by initialising training on the calibration dataset with the current parameter values of the network.

Although the CNN presented in this thesis improves upon the performance of ELUDE when trained on a large dataset, there is still theoretically potential to develop a predictor with higher accuracy, given the reproducibility of results in liquid chromatography.
Table 8: A selection of the results from the initial hyperparameter tests. The general network structure can be viewed in Figure 7. The errors reported are the mean RMSE (min) from three trials. Convolution filters of size $3 \times 3$ were used for all networks, with $2 \times 2$ pooling layers after each convolutional layer.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
<th>Net 6</th>
<th>Net 7</th>
<th>Net 8</th>
<th>Net 9</th>
<th>Net 10</th>
<th>Net 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of conv layers</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>No. conv filters per layer</td>
<td>15</td>
<td>15</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>No. of ReLU layers</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Width of ReLU layers</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>No. of final ReLU layers</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Width of final ReLU layers</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Final training error</td>
<td>9.1</td>
<td>12.6</td>
<td>9.5</td>
<td>9.4</td>
<td>11.0</td>
<td>7.9</td>
<td>7.9</td>
<td>8.7</td>
<td>7.5</td>
<td>7.6</td>
<td>7.0</td>
</tr>
<tr>
<td>Test error</td>
<td>11.4</td>
<td>14.3</td>
<td>11.1</td>
<td>12.8</td>
<td>15.0</td>
<td>10.9</td>
<td>11.6</td>
<td>13.1</td>
<td>11.0</td>
<td>11.0</td>
<td>10.9</td>
</tr>
</tbody>
</table>

Table 9: Tests on convolutional layers. All networks in the table have the same structure of ReLU layers: 3 layers of 30 neurons each in the branches, followed by one joint layer consisting of 15 neurons.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
<th>Net 6</th>
<th>Net 7</th>
<th>Net 8</th>
<th>Net 9</th>
<th>Net 10</th>
<th>Net 11</th>
<th>Net 12</th>
<th>Net 13</th>
<th>Net 14</th>
<th>Net 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter size conv layer 1</td>
<td>3x3</td>
<td>2x2</td>
<td>3x3</td>
<td>2x2</td>
<td>2x2</td>
<td>4x2</td>
<td>4x2</td>
<td>4x2</td>
<td>4x2</td>
<td>5x2</td>
<td>5x5</td>
<td>5x5</td>
<td>5x5</td>
<td>3x3</td>
<td></td>
</tr>
<tr>
<td>Pooling</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Filter size conv layer 2</td>
<td>-</td>
<td>-</td>
<td>3x3</td>
<td>2x2</td>
<td>3x3</td>
<td>3x3</td>
<td>2x2</td>
<td>4x4</td>
<td>2x2</td>
<td>4x4</td>
<td>5x5</td>
<td>5x5</td>
<td>5x5</td>
<td>5x5</td>
<td>5x5</td>
</tr>
<tr>
<td>Pooling</td>
<td>-</td>
<td>-</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Final training error</td>
<td>7.4</td>
<td>7.0</td>
<td>8.2</td>
<td>7.8</td>
<td>7.8</td>
<td>8.4</td>
<td>8.4</td>
<td>8.5</td>
<td>8.0</td>
<td>8.2</td>
<td>8.6</td>
<td>11.6</td>
<td>8.8</td>
<td>8.1</td>
<td></td>
</tr>
<tr>
<td>Test error</td>
<td>10.3</td>
<td>10.4</td>
<td>11.6</td>
<td>10.7</td>
<td>10.9</td>
<td>11.0</td>
<td>10.1</td>
<td>9.8</td>
<td>11.6</td>
<td>9.7</td>
<td>10.2</td>
<td>12.8</td>
<td>11.5</td>
<td>11.0</td>
<td></td>
</tr>
</tbody>
</table>

A Complete Results from First and Second Round of Hyperparameter Experiments
B Earlier Models

This section describes the models implemented in this project prior to the final branched structure to which Section 7 is dedicated. The test and training errors of the models presented in this section are summarised in Table 6 in Section 8.

B.1 Linear Regression

The first two model types implemented have the same data representation. A vector $x_i \in \mathbb{R}^{20}$ is constructed for each peptide $i$ where each element of $x_i$ corresponds to the count of one of the amino acid’s:

\[\text{A R N D C E Q G H I L K M F P S T W Y V},\]

in this given order. As a toy example a peptide of the sequence EHSENEHKESGK will give the vector:

\[x = (0 \ 0 \ 1_N \ 0 \ 0 \ 4_E \ 0 \ 1_G \ 2_H \ 0 \ 0 \ 2_K \ 0 \ 0 \ 0 \ 2_S \ 0 \ 0 \ 0) .\]

With a standard linear regression the retention time of a peptide can be estimated as:

\[\hat{y}_i = \hat{\beta}x_i + \text{bias} .\]

Where $\hat{\beta}$ is the ordinary least square estimate,

\[\hat{\beta} = \left(\sum_{i=1}^{n_{trc}} x_i x_i^T\right)^{-1} \left(\sum_{i=1}^{n_{trc}} x_i y_i\right) .\]

estimated from the all $n$ data points in the training set.

The weight vector $\hat{\beta}$ was also used to determine the ordering of rows in the peptide matrix representation. The relative values of the elements of $\hat{\beta}$ indicate the extent to which each amino acid contributes to the hydrophobicity of the peptide. The amino acid which has the highest corresponding value in $\hat{\beta}$ is represented by the first row in the matrix and the amino acid with the lowest value is subsequently represented by the last row in the matrix.
<table>
<thead>
<tr>
<th></th>
<th>AA Frequency</th>
<th>Pair Frequency ANN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of ReLU layers</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Width of ReLU layers</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Final training error</td>
<td>15.5</td>
<td>7.7</td>
</tr>
<tr>
<td>Test error</td>
<td>15.9</td>
<td>14.4</td>
</tr>
<tr>
<td>Test error STD</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>$\Delta_{t_{r}}^{95%}$</td>
<td>24%</td>
<td>22%</td>
</tr>
</tbody>
</table>

Table 10: The structure of the unbranched ANN, presented for comparison in Table 6, taking either AA frequency or AA pair frequency as input.

### B.2 AA Frequency & AA Pair Frequency ANN

An artificial neural network was also designed to take the vector $x_i \in \mathbb{R}^{20}$ as input. From the initial trials the network structure presented in Table 10 produced the most accurate predictions. This same network was also employed to predict retention times when given amino acid pair frequency as input, rather than the singular AA count. This gives an input vector $x_i \in \mathbb{R}^{210}$ as there are 210 possible unique combination of 20 amino acids.
Table 11: The structure of the unbranched CNN, presented for comparison in Table 6. This network has an approximate training time of 6 min, compared to the branched structure which trains in 5 min.

B.3 Single Matrix Representation CNN

A series of convolutional networks of unbranched structure have been implemented and evaluated in this project. However, for the purposes of comparing to the branched network a similar structure was chosen for the evaluation in Table 6. Furthermore, none of the other tested unbranched CNNs showed significantly better performance.

The network is presented in Table 11.

<table>
<thead>
<tr>
<th>Single Matrix Representation CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv layer 1</td>
</tr>
<tr>
<td>No. conv filters</td>
</tr>
<tr>
<td>Filter size</td>
</tr>
<tr>
<td>Conv layer 2</td>
</tr>
<tr>
<td>No. conv filters</td>
</tr>
<tr>
<td>Filter size</td>
</tr>
<tr>
<td>Number of ReLU layers</td>
</tr>
<tr>
<td>Width of ReLU layers</td>
</tr>
<tr>
<td>Final training error</td>
</tr>
<tr>
<td>Test error</td>
</tr>
<tr>
<td>Test error STD</td>
</tr>
<tr>
<td>$\Delta t_{95%}$</td>
</tr>
</tbody>
</table>
References


