Double Descent for Sequence-to-Expression Models

Scott Florence

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

Double descent is a phenomenon observed in machine learning models where validation loss initially decreases, then increases, and eventually decreases again as effective model complexity is increased. This can be used to overtrain sequence-to-expression models that take DNA sequences as input and try to predict protein expression *in silico*. Overtraining these models by increasing training epochs increases the effective model complexity, which pushes the model into the over-parameterised regime where double decent takes place. In this project, sequence-to-expression models were overtrained to observe double descent across a wide range of datasets, training, and validation sample sizes, and architectures. We also observed that, under certain model and training set criteria, double descent improves the performance of low-N sequence-to-expression models.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics Committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Scott Florence*)

Acknowledgements

I would like to thank my supervisor, Diego Oyarzún for the opportunity to work on this project and for welcoming me into the Biomolecular Control Group.

I would like to thank Evangelos Nikolados for guiding me through the world of sequence-to-expression models and for his invaluable expertise.

Finally, I would like to thank my friends and family for their continued support throughout my time at university.

Table of Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Project Objective . . . 2

2 Background **4**
- 2.1 Sequence-to-Expression Models . . . 4
  - 2.1.1 DNA Encoding Strategies . . . 4
  - 2.1.2 Non-deep Machine Learning Methods . . . 6
  - 2.1.3 Deep Machine Learning Methods . . . 7
- 2.2 Double Descent . . . 9
  - 2.2.1 Bias-Variance Trade-Off . . . 9
  - 2.2.2 The Double Descent Phenomenon . . . 10
  - 2.2.3 Epoch-wise Double Descent . . . 12
- 2.3 Project Timeline . . . 13

3 Data **15**
- 3.1 Sequence-To-Expression Datasets . . . 15
  - 3.1.1 Training & Validation Data . . . 16
  - 3.1.2 Held-Out Test Set . . . 18
  - 3.1.3 Mutational Series Dataset . . . 19
- 3.2 Preprocessing . . . 20
- 3.3 Candidature for Double Descent . . . 21

4 Methodology **23**
- 4.1 Sequence-to-Expression Architectures . . . 23
  - 4.1.1 Components . . . 23
  - 4.1.2 DeepSEA . . . 24
  - 4.1.3 Nikolados et al. CNN . . . 25
  - 4.1.4 Vaishnav et al. CNN . . . 26
- 4.2 Training Procedure . . . 28
  - 4.2.1 Data Partitioning . . . 28
  - 4.2.2 Training Specifications . . . 28
  - 4.2.3 Evaluation . . . 28

5 Results **30**
- 5.1 Double Descent . . . 30

5.2 Double Descent Across Sample Sizes 31
5.3 Double Descent Across Architectures 32
5.3.1 Double Descent on the Nikolados et al. CNN 32
5.3.2 Double Descent on the Vaishanv et al. CNN 33
5.4 Double Descent Improving Performance 34
5.4.1 Double Descent Across Validation Sizes 34
5.4.2 Evaluation on Held-Out Test Set 36
5.4.3 Evaluation on Mutational Series 37

6 Conclusions **38**
6.1 Accomplishments 38
6.2 Research Question 39
6.3 Future work 39
6.3.1 Further research into Double Descent 39
6.3.2 A General Double Descent Sequence-to-Expression Model . . 39
6.3.3 Formulation of Double Descent Classifier 40

Bibliography **41**

Chapter 1

Introduction

1.1 Motivation

Sequence-to-expression models have become a growing interest in the biotechnology community. The ability to predict protein expression from DNA sequences *in silico* gives researchers a measurable understanding between sequence and expression, which enables effective strain optimisation.

These machine learning models aim to predict gene expression levels based on input DNA sequences. The models take advantage of the fact that the sequence of a gene can provide insight into its function and the level of expression in a cell. Sequence-to-expression models allow us to further our understanding of which genes are expressed in certain cell types, which can help identify the underlying molecular mechanisms that drive cell differentiation and disease progression. Developing a sequence-to-expression landscape using machine learning models drastically reduces the cost of strain development as it eliminates costly rounds of prototyping and characterisation needed to identify high production DNA sequences.

Although sequence-to-expression models have been shown to generalise well on unseen data (Vaishnav et al., 2022), they require enormous amounts of data to do so. DNA sequence to expression data, at a scale of $>$20,000,000 samples, which is required for these large models, can incur a large cost for research groups who want to train these models on their own proprietary data.

The paper by Nikolados et al. (2022) addresses this issue by training sequence-to-expression models that generalise with low amounts of training sequences. These predictors are trained on $\sim$4,000 sequences and, in some cases, develop state-of-the-art accuracy. Although these models generalise well, they may not be extracting as much information from the data as possible.

A recent paper by OpenAI (Nakkiran et al., 2019) popularised a phenomenon named double descent, which proposed that as *effective model complexity* increases, validation loss initially decreases, then increases, and eventually decreases again. To induce this double descent phenomenon, we need to ensure that the effective model complexity is sufficiently larger than the number of training samples.

Given that sequence-to-expression models are more economically viable when we train on a low number of sequences, and double descent occurs only when the number of samples is sufficiently smaller than the model size, this makes ”low-N” sequence-to-expression models a strong candidate for double descent, which can improve the performance of deep learning models.

1.2 Project Objective

Low-N sequence-to-expression models operate in the *over-parameterised regime* where the number of training samples is much less than the effective model complexity. To induce double descent, we must overtrain the model to increase it's effective model complexity and, in turn, push the model deeper into the over-parameterised regime.

Although it has been hypothesised, deep double descent has not been documented in the literature to exist for a deep regression task such as sequence-to-expression models. We must first prove its existence across sequence-to-expression models, then observe under what conditions it improves model performance.

To increase the effective model complexity, there are two main ways shown by Nakkiran et al. (2019) which is altering the architecture by increasing the number of parameters to induce double descent (model-wise double descent) and training the model for more epochs (epoch-Wise double descent). Given that Model-Wise double descent involves us increasing the parameters of models that have already been fine-tuned for sequence-to-expression tasks, the simplest method to prove the existence of double descent would be through epoch-wise double descent.

The research question of this project is:

- Does epoch-wise double descent exist in sequence-to-expression models, and can it be used to improve overall model performance?

Therefore, the main objectives of this project are:

- Show epoch-wise double descent occurs empirically across different datasets and training sample sizes.
- Show epoch-wise double descent occurs empirically across different sequence-to-expression architectures, both low-N and not.
- Show epoch-wise double descent can improve model performance by evaluating R^2 both before and after it occurs.

Epoch-wise double descent will be shown by plotting validation loss as a function of training epochs. Traditionally, this plot is used to determine after how many epochs we should stop training the model. The lower the validation loss is, the better the model generalises to unseen data, which can be translated into better model performance. The model has produced double descent when the validation loss, relative to the number of training epochs, initially decreases, then begins to increase, and finally decreases again. In the case where the second descent minima are lower than the first descent, we will evaluate a held-out test set both before and after double descent has occurred to show how model accuracy has improved.

The main accomplishment of this project is that we show empirically that double descent does occur in sequence-to-expression models across datasets, training set sizes, and architectures. Epoch-wise double descent also, under certain model and training set criteria, improves the performance of a low-N sequence-to-expression model.

Chapter 2

Background

This chapter provides a literature review of machine learning methods for sequence-to-expression modelling as well as the origination of the double descent phenomenon. First, we discuss how protein expression is predicted from DNA sequences using current deep and non-deep machine learning techniques. Then, we introduce the double descent phenomenon by discussing the classical bias-variance trade-off and how double descent contradicts conventional wisdom. Finally, double descent is discussed in a deep learning context and how it is achieved.

2.1 Sequence-to-Expression Models

Traditionally, strain engineering required large libraries of sequence variants where researchers would select a subset of the top producing sequences for scale-up or iterative-design. Recently, a model-guided approach has emerged that makes use of sequence-to-expression regressors that learn the shape of the phenotype landscape, allowing iterative querying of the model to produce high-yield sequences.

When compared to other methods of sequence-to-expression modelling (LaFleur et al., 2022), machine learning has become a favourite amongst researchers for its ability to produce highly accurate regressors. These machine learning models take a set of inputs and try to predict a continuous output variable. In the case of sequence-to-expression models, the set of inputs is a DNA sequence with a numerical encoding, and the continuous output variable is the protein expression. The process by which the expression is predicted from the DNA sequence can be done using deep or non-deep machine learning methods.

2.1.1 DNA Encoding Strategies

Sequence encoding is required to transform the given DNA string into a numerical encoding that can be interpreted by a machine learning model. Methods of sequence encoding have been shown to impact the accuracy of regressors, with some encoding methods increasing the performance by 80% when evaluated on the same regressor with

an inferior encoding method (Nikolados et al., 2022). Encodings can be considered at three resolutions: global, subsequence, and single base. These resolutions describe what part of the sequence each individual encoding represents (Table 2.1).

Encoding Strategy	**Resolution**	**Dimension**
biophysical properties	global	8
k-mer ordinal	subsequence	4^k
k-mer counts	subsequence	$L-k+1$
one-hot binary	single base	$4L$
one-hot ordinal	single base	L

Table 2.1: **DNA encoding methods used for sequence-to-expression models.** Each encoding method is shown with its corresponding resolution, i.e., what part of the sequence each encoding represents, as well as the dimension of the resulted encoding. The dimension of the encoding is also the input dimension of our models.

Global resolution encoding is where each sequence is described by its biophysical properties. Each sequence has eight biophysical properties: AT content (%AT), codon adaptation index (CAI), ramp bottleneck position (Btl-p) and strength (Btl-s), mean hydropathy index (MHI), and minimum free energy (MFE-1, MFE-2, MFE-3). These properties provide a comprehensive description of sequence features, encompassing four distinct levels of granularity: nucleotide sequence (which relates to %AT), codon sequence (which relates to CAI, Btl-p, Btl-s), amino acid sequence (which relates to MHI), and secondary mRNA structure (which relates to MFE-1, MFE-2, MFE-3). Each of these properties can be encoded numerically and fed into a machine learning model.

At subsequence resolution, we consider overlapping k-mers encodings. A k-mer is a nucleotide sequence of certain length, where each k-mer is given a unique integer value between 1 and 4^k (k-mer ordinal), or the number of occurrences of each k-mer is recorded (k-mer counts).

Finally, at single base resolution, we consider two versions of one-hot encoding, binary and ordinal. Binary one-hot encoding is where each base of the sequence is allocated a column in a matrix, with a one at the position corresponding to the base and a zero everywhere else. The resulting matrix is of size $4 \times L$ where L is the length of the sequence. Ordinal one-hot encodes each base with a unique integer value i.e. A = 1, C = 2, G = 3, and T = 4. The resulting encoding has length L. Figure 2.1 gives a graphical representation of the possible DNA sequence encodings.

Nikolados et al. (2022) investigates each of these encodings across combinations of non-deep machine learning models and training sample sizes. It was observed that binary one-hot encoding consistently yielded better model performance when compared to the other encoding methods evaluated on the same model. For a Random Forest model trained on 2,865 sequences using biophysical properties encoding, the resulting model performance was $R^2 = 0.44$ but when the encoding method was changed to binary one-hot, the performance increased to $R^2 = 0.79$. A similar increase in performance was shown across all other models and training sample sizes.

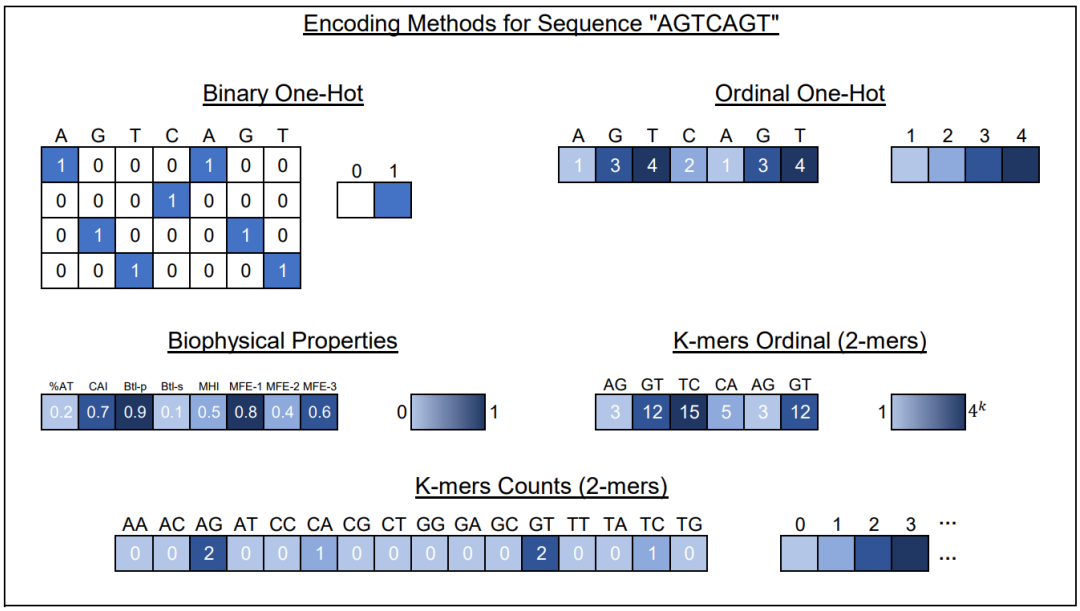

Figure 2.1: **DNA encoding strategies for the sequence "AGTCAGT".** Each encoding method is shown with the corresponding values that would be produced. Binary one-hot encodes a 4×7 matrix, ordinal one-hot encodes a 7-feature array, biophysical properties encodes a 8-feature array, *k*-mers ordinal using 2-mers encodes a 6-feature array, and *k*-mers counts using 2-mers encodes a 16-feature array.

2.1.2 Non-deep Machine Learning Methods

Non-deep machine learning methods such as multilayer perceptrons (Rumelhart et al., 1985), support vector machines (Drucker et al., 1996), random forest regressors (Breiman, 2001) and ridge regressors (Hastie et al., 2009) have found applications in finance (Khaidem et al., 2016; Kim, 2003), logistics (Yang et al., 2020) and energy (Liu et al., 2021) and biotechnology is no exception.

Nikolados et al. (2022) showed that mildly accurate models with $R^2 \geq 0.5$ can be obtained when training random forest regressors or support vector machines on approximately 1,000 sequences with binary one-hot encoding. It was also shown that, compared to all other non-deep models, random forest performed best when trained on greater than 1,000 sequences (Figure 2.2). Furthermore, the same model can achieve an accuracy of $R^2 \geq 0.75$ for different mutational series and encoding strategies.

Comparable results were observed by Höllerer et al. (2020) who trained ridge regressors, random forests, and k-nearest neighbours (Altman, 1992) on a varied number of DNA sequences. For all training sizes, from 2,500 to ∼248,000 sequences, random forest with binary one-hot encoding outperformed all other models, achieving an accuracy of $R^2 = 0.835$ in some cases.

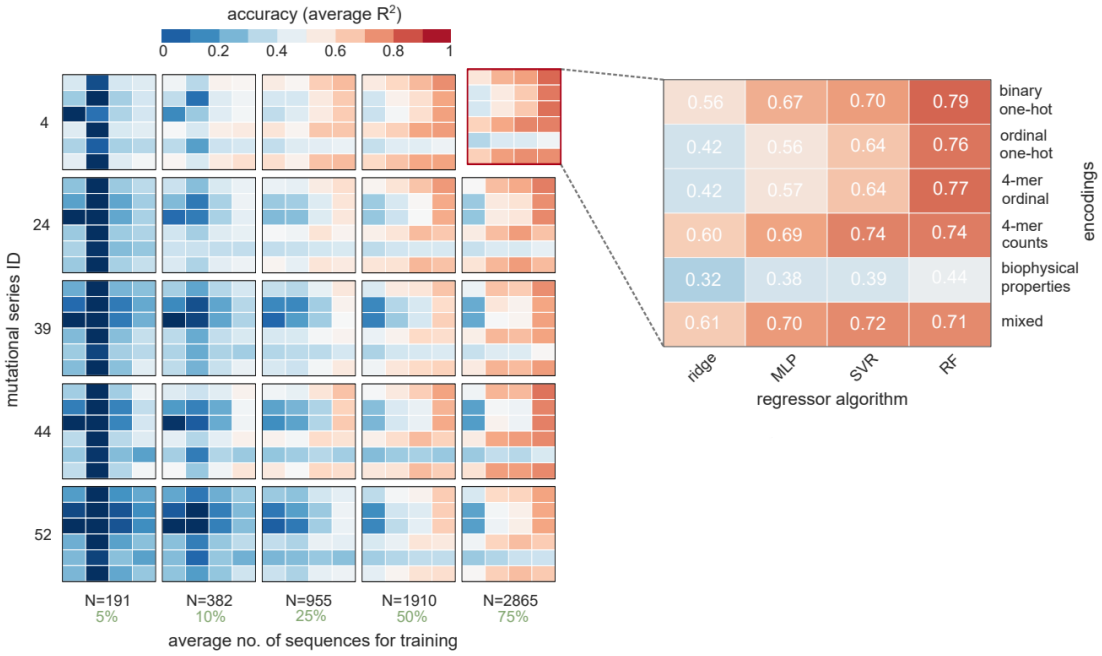

Figure 2.2: **Impact of encoding strategy and model choice across different training sample sizes and mutational series.** The choice of non-deep regressor as well as the encoding strategy are shown to be the key contributors to the overall model R^2. This was shown across various training sizes and mutational series. Random forest paired with binary one-hot encoding outperforms all other combinations. Figure from Nikolados et al. (2022).

2.1.3 Deep Machine Learning Methods

Although non-deep machine learning methods achieve reasonable accuracy, the biggest leaps in performance come from the use of deep machine learning models (Cuperus et al., 2017) such as residual neural networks (ResNet) (He et al., 2016; Xie et al., 2017b) and convolutional neural networks (CNN) (Xie et al., 2017a). The ability to consume a large corpus of sequence-to-expression data and detect highly nonlinear correlations between sequence and expression gives deep models the power to be highly accurate regressors that can be incorporated into optimisation algorithms that search for high expression sequences (Gupta and Zou, 2019).

Deep sequence-to-expression models such as DeepSEA (Zhou and Troyanskaya, 2015), DeepAtt (Li et al., 2020) and DanQ (Quang and Xie, 2016) have been proposed in recent literature and can achieve extremely high accuracy with many sequences. However, a more generalised CNN model proposed by Vaishnav et al. (2022), outperforms the previous models (Figure 2.3), and achieves state-of-the-art accuracy (Pearson's r = 0.960, $P < 5 \times 10^{-324}$, n = 61,150) when trained on the same number of sequences.

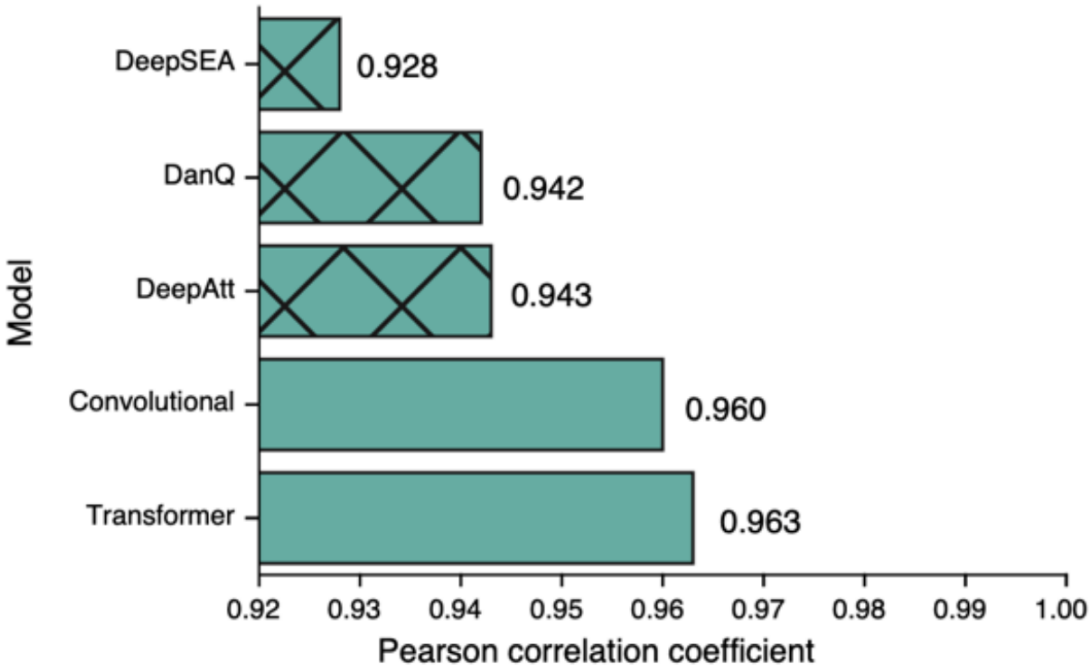

Figure 2.3: **Comparison of performance across deep sequence-to-expression model architectures.** Deep sequence-to-expression models can achieve state-of-the-art Pearson correlation coefficients when trained on tens of millions of sequences. The CNN proposed by Vaishnav et al. (2022) has shown to outperform other models in the literature by a considerable amount. Although a transformer model was also constructed, this will not be considered. Figure from Vaishnav et al. (2022).

Whilst laboratory automation (Carbonell et al., 2019) and reduction in DNA sequencing cost have made acquisition of large datasets more feasible, most laboratories cannot afford to sequence tens of millions of DNA samples to train large regressors, which develops the need for ”low-N” predictors. These predictors are capable of producing highly accurate results while being trained on only a few thousand sequences, which is more achievable for most research labs.

Nikolados et al. (2022) showed that deep learning improves accuracy without more data. They proposed a CNN, designed with Bayesian optimisation, that, regardless of training sample size, was consistently more accurate than all non-deep machine learning models. The model achieved accuracy of $R^2 \geq 0.6$ consistently when trained on as few as 1,000 sequences and even achieved state-of-the-art accuracy of $R^2 = 0.87$ for some sequence sub-spaces. Although this model focuses on operating in the low-N sequence space, it can achieve high accuracies of $R^2 = 0.82$ when trained on 160,000 sequences.

2.2 Double Descent

2.2.1 Bias-Variance Trade-Off

When discussing supervised machine learning models, specifically for predictive modelling, the prediction error can be broken down into two components: the error due to bias and the error due to variance. The bias-variance trade-off is the conflict in trying to minimise both bias and variance error.

Bias error is the difference between the model's expected predictions and the correct value of what we are trying to predict. A model with high bias can fail to capture underlying patterns and relations, i.e., underfitting.

Variance error is the statistical disparity between the model's predictions and a given value. This highlights how sensitive the model is to slight changes in the training set. A model with high variance may focus on random noise in the training data rather than generalising the dataset, i.e., overfitting.

Traditionally, to build a good predictive model, we must balance overfitting and underfitting such that the total error is minimised. (Geman et al., 1992; Hastie et al., 2009). As we increase the complexity of the model, the total variance will increase, and the model will begin to overfit. On the other hand, if the model isn't complex enough, the bias is high, and the model will underfit. Conventional wisdom in machine learning suggests there is an optimum model complexity where the increase in bias is equivalent to the decrease in variance and model performance is maximised (Figure 2.4).

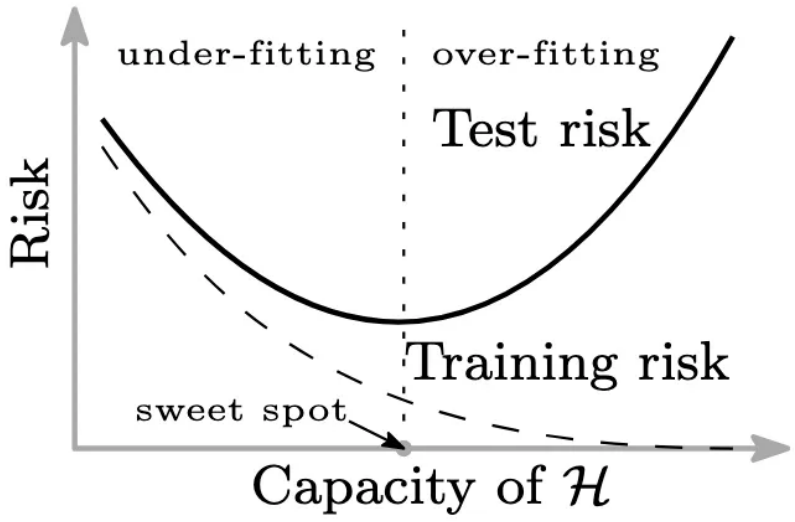

Figure 2.4: **The classical U-shaped bias-variance trade-off curve.** This curve represents the conventional wisdom of a machine learning training schedule. The x-axis is the capacity of a given machine learning model $\mathcal{H}$ i.e., the complexity of that model. The y-axis is the risk metric used to evaluate model performance; for regressors, its usually Mean Squared Error or Mean Absolute Error. Figure from Belkin et al. (2019).

2.2.2 The Double Descent Phenomenon

Recent research in machine learning has revealed a surprising phenomenon called *double descent*, which challenges the conventional wisdom that increasing model complexity always leads to worse performance due to overfitting. Unlike traditional models, modern deep learning models exhibit a double descent curve where performance initially decreases as model complexity increases, but then improves again beyond a certain point (Figure 2.5).

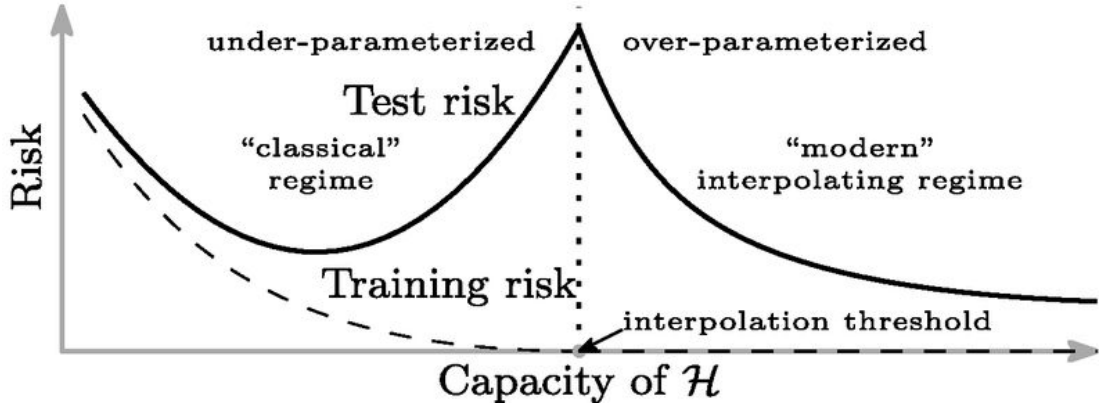

Figure 2.5: **A generalisation of the double descent curve.** The double descent curve incorporates the classical U-shaped curve in the ”classical” regime while also showing the second descent that occurs in the ”modern” interpolating regime. The point at which the first ascent stops, and the second descent starts is called the interpolation threshold, which occurs when training error is approximately zero. Figure from Belkin et al. (2019)

When model complexity is insignificant compared to the number of training samples, it is said to be under-parameterised and exhibits traditional bias-variance trade-off behaviour, i.e., we operate in the ”classical” regime (Figure 2.5). As we approach the model complexity being equal to the number of training samples, we arrive at the interpolation threshold, where the training error is effectively zero and the test error reaches a peak. At this point, a double descent can occur if we continue to increase the model complexity so that it is greater than the number of training samples. The second descent occurs when the model is over-parameterised and begins to reduce the test error of the model while the train error remains at zero, i.e., we operate now in the ”modern” interpolating regime (Figure 2.5).

The phenomenon has been previously observed in non-deep machine learning models (Opper, 1995; Advani and Saxe, 2017; Spigler et al., 2019) but the notion of ”double descent” was first put forward by Belkin et al. (2019) where it was shown to exist for non-deep machine learning methods such as 2-layer neural networks and decision trees. Since then, there have been attempts to mathematically explain double descent for simple linear regressors (Nakkiran, 2019; Belkin et al., 2020) by isolating and understanding the phenomenon in a simple setting.

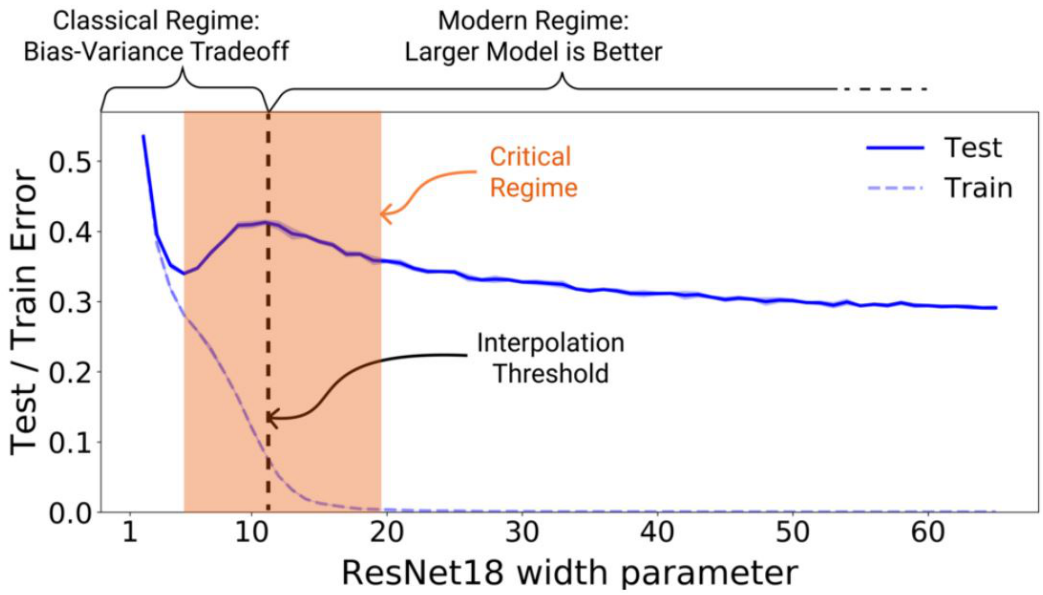

Figure 2.6: **Observation of model-wise double descent for ResNet18 models.** Model-wise double descent was observed by training 64 ResNet-18 models with varying width parameters to increase the EMC of the model. Each model was trained on the CIFAR-10 dataset with 15% label noise for 4,000 epochs. We observe that the critically parameterised regime occurs at a width parameter of 11, where the test loss peaks. The test loss at width parameter 64 is lower than the previous low in the under-parameterised regime, meaning model-wise double descent produced a more accurate model. Figure from Nakkiran et al. (2019).

Deep double descent was first introduced by Nakkiran et al. (2019) which showed that double descent was a robust phenomenon that can be observed on a varied number of datasets and architectures and proposed a general double descent that is more than simply increasing the number of model parameters to increase model complexity. An effective model complexity (EMC) of training procedure was defined as the maximum number of samples needed to achieve close to zero training error, i.e., at what point we reach the interpolation threshold. The definition of EMC as well as the General Double Descent Hypothesis is:

Effective Model Complexity: *The* Effective Model Complexity (EMC) *of a training procedure* $\mathcal{T}$, with respect to distribution $\mathcal{D}$, *and parameter* $\varepsilon > 0$ *is defined as:*

$$EMC_{\mathcal{D},\varepsilon} := \max\{n \,|\, \mathbb{E}_{S\sim\mathcal{D}^n}\left[Error_S(\mathcal{T}(S))\right] \leq \varepsilon\} \tag{2.1}$$

Generalised Double Descent hypothesis: *For any natural data distribution* $\mathcal{D}$, *neural-network-based training procedure* $\mathcal{T}$, *and small* $\varepsilon > 0$, *if we consider the task of predicting labels based on n samples from* $\mathcal{D}$ *then:*

Under-parameterized regime: *If* $EMC_{\mathcal{D},\varepsilon}$ *is sufficiently smaller than n, any perturbation of* $\mathcal{T}$ *that increases its effective complexity will decrease the test error.*

Over-parameterized regime: *If $EMC_{\mathcal{D},\varepsilon}$ is sufficiently larger than n, any perturbation of $\mathcal{T}$ that increases its effective complexity will decrease the test error.*

Critically-parameterized regime: *If $EMC_{\mathcal{D},\varepsilon} \approx n$, then a perturbation of $\mathcal{T}$ that increases effective complexity will decrease* **or increase** *the test error.*

Nakkiran et al. (2019) continues by showing that double descent can be observed as a function of EMC rather than strictly model complexity. They observe an "epoch-wise double descent", where the architecture of the model is fixed but the EMC is increased by increasing the number of training epochs. They also observe a "model-wise double descent" for CNNs and ResNets in which the model architecture is altered to increase the EMC, which produces a double descent curve (Figure 2.6).

Nakkiran et al. (2019) concludes that the perception of "more data is better" proposed in Belkin et al. (2019) may be false and that more data may lead to worse test accuracy, if we are currently operating in the over-parameterized regime, as increasing the number of training samples will decrease the gap between EMC and the number of samples (n). This gap is what allows double descent to improve model performance. Double descent occurs only when we do not employ an early stopping technique and allow the model to overfit by removing all regularisation.

2.2.3 Epoch-wise Double Descent

As previously mentioned, epoch-wise double descent is observed when we keep the model size fixed and increase the number of training epochs. A training epoch is the number of times the training data is passed through a machine learning model. For the first few training epochs, the model operates in the under-parameterised regime where the EMC is less than the number of training samples present. As we increase EMC, by letting the model train for more epochs, we hit the critically parameterised regime where the EMC is equal to the number of training samples and test loss reaches a peak. Finally, as we continue to let the model train for longer, the EMC increases further, pushing us into the over-parameterised regime where the test loss has again decreased while the train error remains close to zero.

In most cases of model training, an early stopping technique is used, which watches out for test loss increasing as we increase the number of training epochs and reverts the model back to a point where test loss is lower. This technique is only useful for models whose EMC is in the critically parameterised regime, i.e., the EMC is equal to the number of training samples, as once the model is fully trained, it will have a peak test loss. For this reason, early stopping is removed during epoch-wise double descent, as we do not want the model to stop training once the critically parameterised regime is reached; rather, we want to push it further into the over-parameterised regime where we can exhibit double descent.

Nakkiran et al. (2019) observed the effect of epoch-wise double descent across varied model sizes (Figure 2.7). They show that for models that have a low EMC, double descent does not occur when we increase the number of training epochs because the number of training samples is never less than the EMC. For intermediate sized models, i.e., when EMC $\approx n$, double descent does occur, but the second descent is higher than the first, meaning we are better off stopping model training. Epoch-wise double descent is only useful when we have large models where EMC $> n$ as the test loss after the second descent is lower than the first, which translates into better overall model performance.

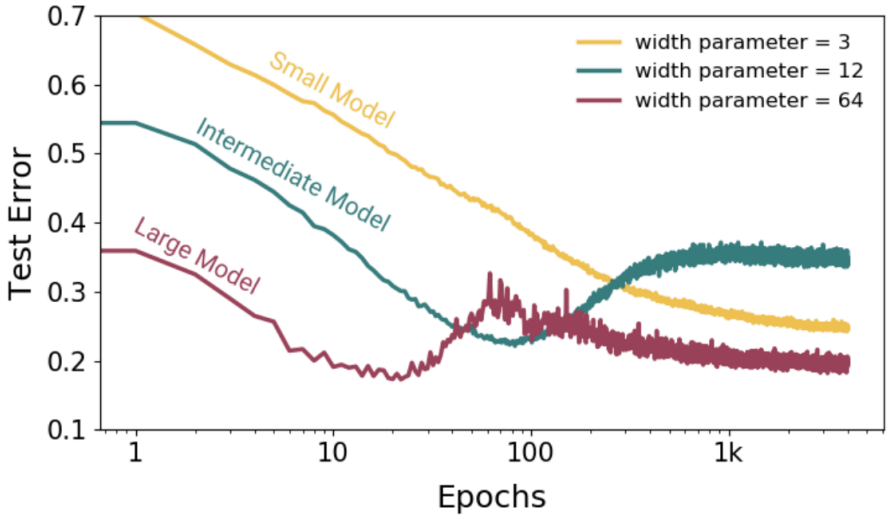

Figure 2.7: **Observation of epoch-wise double descent for ResNet18 models.** Epoch-wise double descent was observed by training three different ResNet18 models, all with varying EMC trained, on the CIFAR-10 dataset with 20% label noise. Models with a lower width parameter have a lower EMC meaning double descent does not occur. Larger models exhibit epoch-wise double descent as their EMC is sufficiently larger than the training data size. Figure from Nakkiran et al. (2019).

Although epoch-wise double descent can occur on calibrated datasets such as CIFAR-10 and CIFAR-100 (Krizhevsky, 2009), artificial label noise must be applied (Nakkiran et al., 2019). Label noise of 10-20% on these datasets allows double descent to occur, and the second descent is more prominent than the first. Whilst epoch-wise double descent does not occur on datasets with zero noise, this gives precedence for datasets with a natural noise component to use epoch-wise double descent to improve model performance.

2.3 Project Timeline

The Project Timeline can be detailed as follows:

1. **Investigate epoch-wise double descent across different sequence-to-expression datasets.** Previous papers that observe the epoch-wise deep double descent phenomenon use models trained on CIFAR-10 and CIFAR-100 datasets for classifi-

cation tasks. I propose to use deep regression models trained on sequence data to predict protein expression and determine if epoch-wise double descent occurs across different sequence-to-expression datasets.

2. **Investigate if training sample size can affect double descent for sequence-to-expression models.** When operating in the over-parameterised regime, we must have sufficiently fewer training samples than EMC to achieve epoch-wise double descent. I propose to investigate if epoch-wise double descent occurs for a varied training sample size that can be used for low-N models.

3. **Investigate the effect model architecture has on epoch-wise double descent for sequence-to-expression models.** As shown by Nakkiran et al. (2019), model size can play a role in epoch-wise double descent as EMC needs to be sufficiently larger than the number of training samples. I propose to investigate if epoch-wise double descent is architecture-specific or occurs across all sequence-to-expression architectures.

4. **Show under what conditions epoch-wise double descent can improve model performance.** Previous work has been done to predict protein expression using sequence data in deep models. I propose to employ epoch-wise double descent to overtrain the model such that overall test accuracy is improved when compared to test accuracy before double descent occurs.

Chapter 3

Data

To investigate double descent in sequence-to-expression models, we needed a dataset consisting of DNA sequences and their corresponding protein expressions. We used multiple datasets, including complex media, defined media, and mutational series, to overtrain, validate, and test our models. To ensure our models could process the data and make predictions on the continuous output variable, we encoded the datasets into a numerical format suitable for our regressors. This chapter covers the details of our training and validation datasets, as well as the held-out test set, the preprocessing steps, and the candidature for double descent.

3.1 Sequence-To-Expression Datasets

Sequence-to-expression datasets consist of millions of DNA sequences and their corresponding yellow fluorescent protein (YFP) expressions. These sequences are promoter DNA, which defines where transcription of a gene by RNA polymerase begins, and corresponding YFP expression is then measured (de Boer et al., 2020). All sequences in our datasets are 110 bp long, meaning there are 110 bases (A,C,G, and T) that make up our sequence, and the corresponding expression is represented by a floating-point value. (Table 3.1).

DNA Sequence (110bp)	Protein Expression
TGCATTTTTT...TACGGCTGTT	15.15532
ATGATGGCAG...ACACTGCACA	12.34555
AGCTCTGTTA...TCAGCACCGT	7.361053
GTTTACATAA...AAGGTGAATA	10.25913
AGTCAAGGCG...GAGTCATGAT	0.113132

Table 3.1: **Example sequences and their corresponding expression values are used for training, validating, and testing our models.** Each sequence used for training is 110 base pairs, initially represented as strings but encoded numerically before being input into the model. Our expression values remain as they are and represent the continuous output variable our regressors are trying to predict.

3.1.1 Training & Validation Data

To ensure comparability of results, we used the dataset from Vaishnav et al. (2022), as this gave us the freedom to change the model architecture whilst ensuring meaningful results would be returned, as this dataset has been used to train various sequence-to-expression models in the past. This dataset consists of sequences from a defined and complex media.

During the measurement of protein expression, the promoter sequences are cloned into an episomal low-copy-number YFP expression vector, which is then transformed into yeast. The resulting yeast is then cultured in a desired medium for later measurement of YFP expression. The term defined or complex medium refers to the chemical composition of the media given to the yeast during the culturing process. A defined medium is one whose chemical composition is exactly known, whereas a complex medium has an unknown chemical composition.

The defined medium dataset consists of more than 20,000,000 sequences and their corresponding YFP expressions. These sequences were randomly generated from a defined medium lacking uracil (SD-Ura). The complex medium consists of more than 30,000,000 sequences and their protein expression, which have been randomly generated from Y8205 S. cerevisiae measured in complex (YPD).

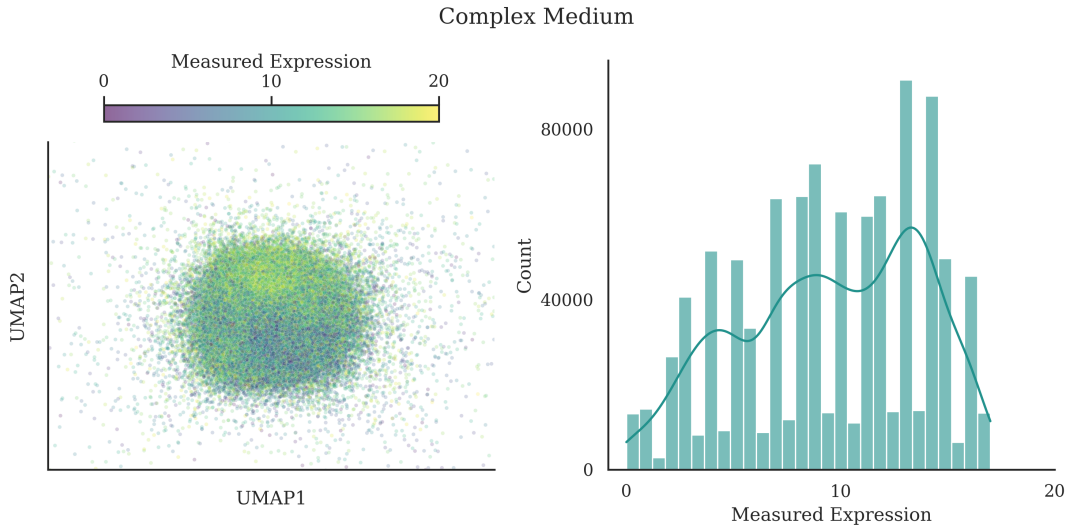

Figure 3.1: **The complex medium sequence space.** The UMAP of the complex medium sequence space (left) was generated by randomly sampling 1,000,000 sequences from the dataset and encoding them as overlapping 4-mers. The UMAP demonstrates that the sequence space is characterised by a single continuous cluster of sequences. The distribution of expression values (right) is characterised by a multi-modal, left-skewed distribution. Specifically, the distribution indicates that sequences with higher expression values are more prevalent in the dataset, but there are still a substantial number of sequences with lower expression values present in the data.

In this study, we utilised Uniform Manifold Approximation and Projection (UMAP) (McInnes et al., 2018) to visualise the complex and defined medium sequence spaces, as depicted in Figures 3.1 and 3.2. To reduce the dimensions of each sequence, we encoded them as overlapping 4-mers, which was chosen to align with Nikolados et al. (2022) who employed a similar strategy to visualise the mutational series dataset. To obtain an overview of the sequence space while minimising computational time, we randomly sampled 1,000,000 sequences for each medium. The resulting UMAP visualisations showed a single, continuous cluster for both defined and complex mediums. Additionally, we plotted the distribution of expression values, which revealed a multi-modal left-skewed distribution for complex medium and a slight bi-modal left-skewed distribution for defined medium. This indicates that a greater number of sequences have expression values around 15.

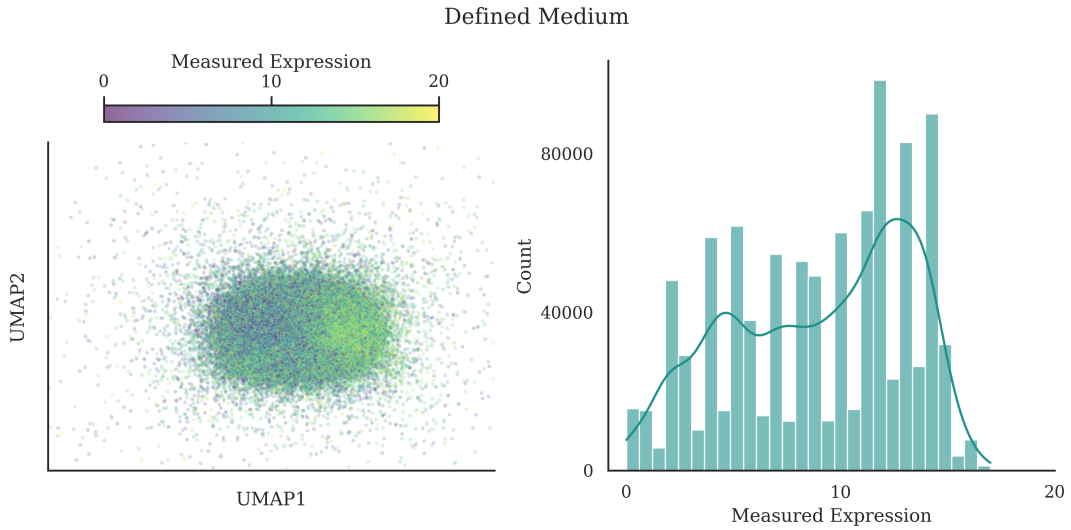

Figure 3.2: **The defined medium sequence space.** Much like Figure 3.1, the UMAP (left) was generated by randomly sampling 1,000,000 sequences from the defined medium dataset and encoding them as overlapping 4-mers. The sequence space is characterised by one continuous cluster of sequences. The distribution of expression (right) shows a slight bimodal, left-skewed distribution. This indicates high expression sequences are more prevalent in the dataset, but low expression sequences are still abundant.

In Chapter 5, the models were trained and validated using data randomly sampled from the defined and complex medium datasets. Random sampling was used to obtain an unbiased sample of the entire sequence space while maintaining the same distribution of expression values present in the full dataset. To prevent data leakage, where a sequence-expression pair appears in both the training and validation sets, the samples were compared against each other prior to training.

While the training data is used solely to overtrain our sequence-to-expression models, the validation data is evaluated by the model after every epoch to get a validation error. Traditionally, validation error is used to determine at what epoch training should stop to avoid overfitting. In our case, this loss is then plotted against training epochs to see if

double descent has occurred for this training procedure.

3.1.2 Held-Out Test Set

Once we have observed double descent in our models, we need a way to determine if double descent has improved model performance. Although we can observe if double descent has improved performance by checking if the validation error is lower after the second descent than the first, we need a quantifiable metric of improvement for the model. We can use a held-out test set to evaluate the model both before and after the double descent to check if model performance improved.

Vaishnav et al. (2022) provides us with a held-out test set that consists of 61,150 native promoter sequences from yeast grown in complex medium. As mentioned, these sequences will be evaluated to produce an R^2 to quantify how well our model generalises to unseen data before and after the double descent has occurred. Figure 3.3 presents a UMAP of the held-out test set's sequence space as well as a distribution of their expression values.

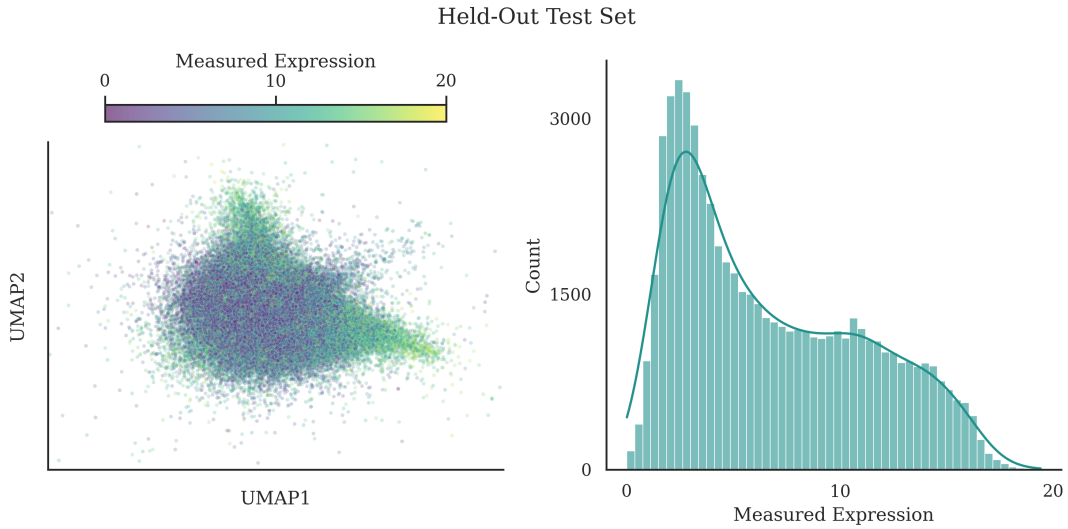

Figure 3.3: **The held-out test set sequence space.** The UMAP of the held-out test set sequence space (left) was generated by encoding all sequences in the set as overlapping 4-mers. The UMAP demonstrates that the sequence space is characterised by a single continuous cluster of sequences, which is like that of the defined and complex media. The distribution of expression values (right) is characterised by a right-skewed distribution. Specifically, the distribution indicates that sequences with lower expression values are more abundant in the test dataset.

The UMAP presented in Figure 3.3 was generated using the same parameters as in the previous figures, with the exception that the entire test set was dimensionality reduced. Our analysis reveals that the sequence space of the test set is like that of the training and validation sets, which is evident in the continuity of the cluster observed in Figures 3.1 and 3.2. However, a notable difference between the test set and the training and

validation sets is the distribution of expression values. Specifically, we observe a right-skewed distribution of expression values in the test set, which indicates a prevalence of lower expression sequences. Nonetheless, we note that the test set still contains enough high expression sequences, which ensures that the testing remains representative of the sequence space.

3.1.3 Mutational Series Dataset

In the paper by Nikolados et al. (2022), they provided a dataset of *Escherichia coli* (E. coli) sequences from 56 different mutational series. During the construction of sequence-to-expression datasets, a seed sequence is used to generate multiple sequences in the same area of the sequence space by way of mutation. A mutational series is defined as all sequences generated from a single seed sequence, and hence these sequences cluster in the same part of the sequence space. Given that each of these mutational series operates in the same part of the sequence space, we can use the sequences as a simulation of how these models may be used by researchers, who have access to only a few thousand sequences and want to predict protein expressions from these DNA sequences.

The dataset consists of $\sim$4,000 sequence and expression pairs for each mutational series. These pairs were partitioned into train, validate, and test sets to train our models and investigate the phenomenon of double descent. Specifically, we selected five mutational series from the dataset highlighted by Nikolados et al. (2022) as they followed distinct expression distributions.

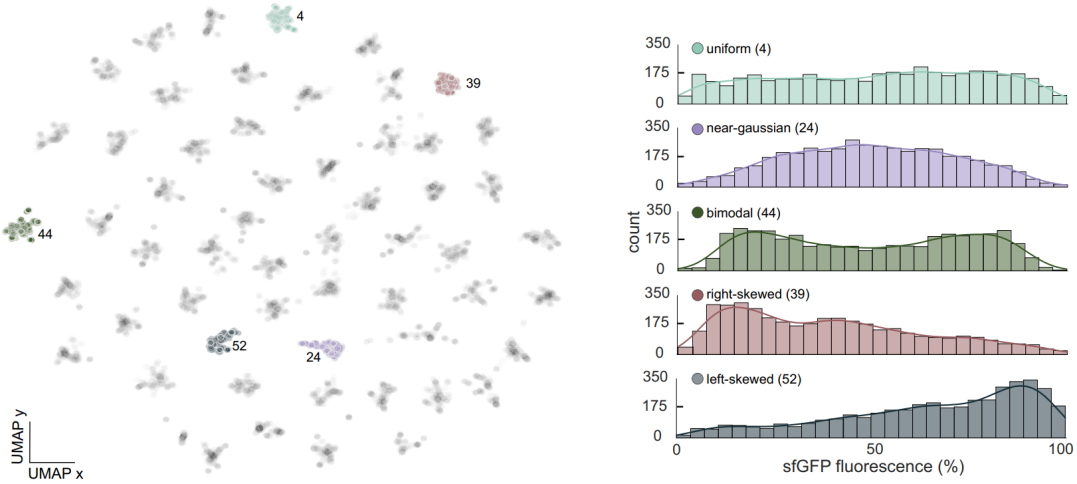

Figure 3.4: **Mutational Series sequence space.** The UMAP visualisation of the mutational series dataset (left) shows 56 distinct clusters, each representing a mutational series. To generate the visualisation, 2,400 sequences were sampled from each series and encoded as overlapping 4-mers. The five series that will be used for training, validation, and testing of our model are highlighted in the visualisation. Additionally, the expression values for these five series (right), reveal that each cluster we plan to train on has a unique distribution. Figure from Nikolados et al. (2022).

In Figure 3.4, we present the sequence space of all 56 mutational series in our dataset, along with the 56 clusters formed by these series. To generate this UMAP visualisation, 2,400 sequences were sampled from each series and encoded as overlapping 4-mers. In addition, Figure 3.4 provides insights into the expression distribution of the five mutational series selected for our overtraining experiments. These series were chosen by Nikolados et al. (2022) based on their distribution types, which include uniform, near-Gaussian, bimodal, right-skewed, and left-skewed. The visualisation provides a comprehensive overview of the sequence space and expression distribution for the selected mutational series, which will be used to evaluate our model's performance under different conditions.

3.2 Preprocessing

Preprocessing input data is standard practice in most machine learning applications. In our case, preprocessing of inputs was required as models cannot read input encoded as strings, such as DNA sequences. Therefore, we must transform our strings into a numerical equivalent that the model can understand.

As previously mentioned in Chapter 2, DNA sequences have a variety of encoding methods that have been used across the literature. Although each encoding method has its merits, Nikolados et al. (2022) showed that binary one-hot improved R^2 accuracy of many non-deep and deep machine learning models when compared to other encoding methods. This method is also the most popular amongst the sequence-to-expression model literature (Höllerer et al., 2020; Vaishnav et al., 2022), so it's useful for comparability of results.

Binary one-hot encoding involves transforming each base pair into an array of length 4, where each index of that array has either a 0 or 1. The value at each index is dictated by which base pair we are encoding. For example, if we were to encode base pair A, the associated array is $[1,0,0,0]$ and for base pair T, the associated array is $[0,1,0,0]$. This method of encoding follows for base pairs C and G.

The standard length of a sequence in the dataset is 110 base pairs long, meaning that once binary one-hot encoding is applied, the input shape for a 1-D convolutional layer is (110, 4) and (1, 110, 4) for a 2-D convolutional layer. Figure 3.5 visualises the preprocessing pipeline that occurs before model training.

Label normalisation is a technique used in many machine learning applications to bring the mean value of a label (in our case, the expression value) close to 0 and the standard deviation close to 1. Although this is a popular preprocessing method for regression tasks, it was not employed in Vaishnav et al. (2022) or Nikolados et al. (2022) so for comparability of results, this was omitted.

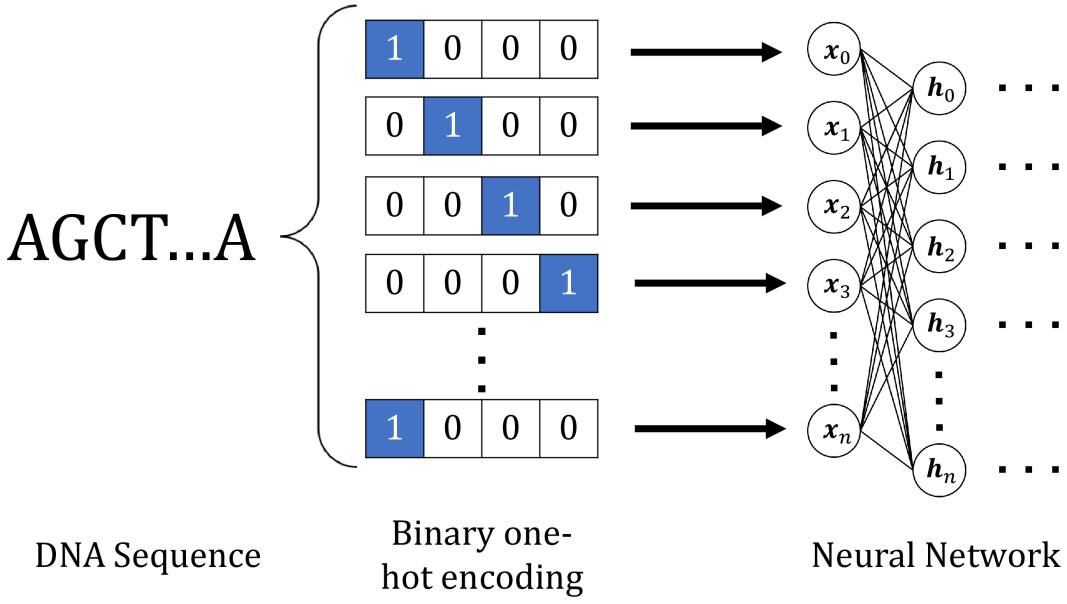

Figure 3.5: **The sequence encoding pipeline.** The raw sequences must be encoded in a numerical format before being passed to a model. Each model we train in Chapter 5 follows this preprocessing pipeline. We first transform each base pair into its binary one-hot equivalent, and we then collate these arrays to form a 4×110 matrix, which is then transposed (for 1-D layers) or reshaped (for 2-D layers) before being passed to our machine learning models.

3.3 Candidature for Double Descent

Double descent was observed by Nakkiran et al. (2019) on calibrated datasets such as CIFAR-10 and CIFAR-100 with artificial label noise added to induce the phenomenon. For epoch-wise double descent, the second descent only occurs after label noise has been added, meaning that the amount of noise present in the dataset is a contributing factor. Therefore, sequence-to-expression models are an excellent candidate for observing the phenomenon.

Sequence-to-expression data has a natural noise component due to the sequencing process itself. It was found in a study conducted by Schirmer et al. (2016) that substitution errors were the primary source of noise in genomic sequencing. During the process of sequencing, when DNA fragments are amplified through cycles of synthesis for the detection of the incorporated nucleotides, the DNA polymerase sometimes incorporates the wrong nucleotide into the DNA strand, which leads to a substitution error. For example, the polymerase incorporates cytosine instead of thymine, resulting in a C-to-T substitution error. These errors contribute to the overall noisiness of the dataset we use to train the models.

As detailed by Nakkiran et al. (2019), a noisy dataset is more likely to exhibit the double descent phenomenon, as when we are overfitting, the model starts to learn the noisiness of the data. This benefits us, as the noisiness is an inhernet part of the dataset and should be learned to produce a more accurate model. The double descent phenomenon can then be exploited to decrease the overall test loss of the model and improve performance.

Nakkiran et al. (2019) proposed that double descent only occurs when the effective model complexity exceeds the number of training samples by a significant margin. This is particularly relevant to sequence-to-expression models, which require a large amount of sequence data to train effectively. By leveraging the over-parameterized regime, where the effective model complexity is much greater than the number of training samples, we can create more accurate models with less data, making this approach more feasible for researchers with limited resources.

Chapter 4

Methodology

In this chapter, we present the methodology used for overtraining sequence-to-expression models to produce epoch-wise double descent. First, we give an overview of each CNN component, which builds our sequence-to-expression models. Next, we present the model architectures used for overtraining, which have regularisation and drop-out layers removed from them. Finally, we present the training procedure and evaluation metrics used to determine if double descent has occurred and what the improvement was.

4.1 Sequence-to-Expression Architectures

Model architecture is a substantial component of EMC. Each model architecture has different hyperparameters that contribute to the overall complexity of the model. The deep double descent phenomenon in Nakkiran et al. (2019) was an empirical observation that relied on different architectures of classifiers, such as CNNs and ResNets. In this section, we present multiple architectures in the literature that will be overtrained to confirm double descent's existence across sequence-to-expression models.

4.1.1 Components

A typical sequence-to-expression CNN architecture consists of sequentially alternating layers that extract sequence features and try to predict a continuous output variable, i.e., protein expression. Each layer of a CNN executes a linear transformation of the output from the previous layer by multiplying by a weight matrix and applying a nonlinear transformation. The three basic layers of a CNN are the convolutional layer, the pooling layer, the and fully connected (dense) layer.

A convolutional layer computes output by one or two-dimensional convolution operations that have a specified number of kernels, i.e., weight matrices. Each convolution output is then transformed non-linearly by an activation function, which is then passed to the next layer of the model. A convolution is defined as follows:

$$convolution(X)_{ik} = \sigma(\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} W_{mn}^{k} X_{i+m,n}) \tag{4.1}$$

where X is the input of the convolution, i is the index of the output position, k is the index of the kernels, W^k is an $M \times N$ weight matrix where M is window size, N is input channels, and σ is an activation function that does the non-linear transformation.

A maximum pooling layer downsamples the previous convolutional layers of the feature map by taking the maximum value of a sub-region of the map. This sub-region, or pooling window, traverses the feature map and selects the maximum value in each region. The result is a smaller output feature map, which improves the computational efficiency of the model. A pooling operation is defined as follows:

$$pooling(X)_{ik} = \max(\{X_{iM,k}, X_{(iM+1),k}, ..., X_{(iM+M-1),k}\}) \tag{4.2}$$

where X is the input of the pooling, i is the index of the output position, k is the index of kernels, and M is the sub-region or pooling window size.

A dense layer performs a linear operation on the input and applies a non-linear activation function. The layer is referred to as "fully connected" as all input neurons are connected to all output neurons. A dense operation is defined as follows:

$$dense(X) = \sigma(WX + b) \tag{4.3}$$

where X is the input of the dense layer, σ is the activation function, W is the weight matrix, and b is the bias vector.

An activation function σ is applied to the output of each neuron in a convolutional or dense layer. It introduces non-linearity into the network, allowing it to learn and model complex, non-linear relationships between input features. The most common activation functions used in our models are ReLU, Sigmoid, and Linear:

$$ReLU(X) = \max(0, X) \tag{4.4}$$

$$Sigmoid(X) = \frac{1}{1 + e^{-X}} \tag{4.5}$$

$$Linear(X) = X \tag{4.6}$$

The stride of a convolutional layer refers to the distance the filter moves after each convolution operation. A stride size of one means that the filter slides over one element after each convolution. Padding, on the other hand, determines the additional values added to the input data to maintain its original size and shape during the convolution operation. If a padding value of "SAME" is used, the amount of padding is applied uniformly to all elements of the input data, ensuring that the output feature maps have the same dimensions as the input data.

4.1.2 DeepSEA

The DeepSEA architecture was proposed by Zhou and Troyanskaya (2015) for predicting chromatin effects of noncoding variants. DeepSEA was able to predict chromatin features with high accuracy, achieving a median area under the curve (AUC) of 0.958,

which surpassed the state-of-the-art k-mer SVM at the time. This model was chosen as a basic sequence-to-expression architecture that has been previously used as a benchmarking model (Vaishnav et al., 2022). This gives us a base architecture to investigate if double descent occurs, whilst ensuring our results are comparable with recent literature.

The implementation of the architecture, as shown in Table 4.1 and Figure 4.1, was taken from `https://github.com/ljw-struggle/Bioinfor-DeepATT` with the L1 and L2 Kernel Regularisation, Kernel MaxNorm Constraint, and Dropout Layers all removed to induce overfitting.

Blocks	Hyperparameter	Value
Convolutional (1-3)	number of filters	[320, 480, 960]
	kernel size	8
	strides	1
	padding	SAME
	activation	ReLU
MaxPool (1-2)	pool size	4
	strides	4
	padding	SAME
Dense (4-5)	units	925
	activation	[ReLU, Sigmoid]
	bias	True
Dense (final)	units	1
	activation	linear
	bias	True

Table 4.1: **The hyperparameters used in our implementation of the DeepSEA architecture.** Our architecture is divided into blocks, with each block corresponding to a specific section of layers in the model. The values in square brackets denote the varying parameters for each layer within the block. It's important to note that both the Convolutional and MaxPool layers in our architecture are 1-D.

Figure 4.1: **Visualisation of our DeepSEA model.** Our DeepSEA model begins with two 1-D convolutional and MaxPool layers, followed by a final 1-D convolutional layer that does not use a MaxPool layer. Next, a flatten layer is utilised to reduce the dimensions for input into the dense layer. Two additional dense layers follow the flatten layer. Finally, the model includes an output layer with linear activation.

4.1.3 Nikolados et al. CNN

The CNN proposed by Nikolados et al. (2022) was designed with Bayesian Optimisation to find a single architecture for all mutational series in the paper. The resulting architecture is of similar complexity to models proposed by Zhou and Troyanskaya

(2015) and Vaishnav et al. (2022) and produces excellent prediction accuracy's with a low number of training samples. The architecture achieves $R^2 = 0.82$ for 160,000 training sequences, and in some cases, $R^2 = 0.87$ for as few as 1,000 sequences.

Although this architecture is similar to DeepSEA, its ability to produce excellent prediction accuracy's well into the over parameterised regime makes it a strong candidate for double descent. The architecture, as shown in Table 4.2 and Figure 4.2, was implemented with TensorFlow by following the architecture specification provided in the paper. Again, my implementation of the model removes the regularisation and dropout layers to induce overfitting.

Blocks	Hyperparameter	Value
Convolutional (1-3)	number of filters	256
	kernel size	13
	strides	(1, 1)
	padding	SAME
	activation	ReLU
MaxPool (1-3)	pool size	(2, 2)
	strides	None
	padding	SAME
Dense (4-7)	units	256
	activation	ReLU
	bias	True
Dense (final)	units	1
	activation	linear
	bias	True

Table 4.2: **The hyperparameters used in our implementation of the Nikolados et al. (2022) architecture.** Our architecture is divided into blocks, with each block corresponding to a specific section of layers in the model. The values in square brackets denote the varying parameters for each layer within the block. It's important to note that both the Convolutional and MaxPool layers in our architecture are 2-D.

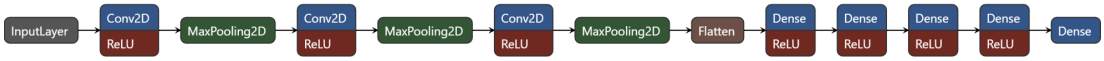

Figure 4.2: **A visualisation of the Nikolados et al. (2022) CNN.** The model consists of three 2-D convolutional blocks, each followed by a MaxPool layer. After the three convolutional blocks, a flatten layer is employed to reduce the output's dimensionality and prepare it for input into the dense layers. The model has four dense layers, each utilising a ReLU activation function, and a final dense layer with a linear activation function.

4.1.4 Vaishnav et al. CNN

The CNN proposed by Vaishnav et al. (2022) was created to build a fitness landscape that maps each DNA sequence to its associated fitness. This model was used to predict protein expressions from DNA sequences, helping to understand evolution from

sequence to expression whilst also being used for designing sequences with high and low expression. This model achieves state-of-the-art accuracy (Pearson's r = 0.960, $P < 5 \times 10^{-324}$, n = 61,150) on the complex medium described in Chapter 3 for which this model was curated.

The implementation of the architecture, as shown in Table 4.3 and Figure 4.3 was taken from `https://codeocean.com/capsule/8020974/tree/v1` with the L2 Kernel Regularisation and Dropout Layers removed to induce overfitting. This architecture also employs forward and reverse strand convolutional layers, which take two inputs, a sequence in its normal form and a sequence that has been reversed, as well as a bias term added to each layer, which is different from the other architectures.

Blocks	**Hyperparameter**	**Value**
Forward Convolutional (1-2)	number of filters	256
	kernel size	[(1, 30), (30, 1)]
	strides	(1, 1, 1, 1)
	padding	SAME
	activation	ReLU
Reverse Convolutional (1-2)	number of filters	256
	kernel size	[(1, 30), (30, 1)]
	strides	(1, 1, 1, 1)
	padding	SAME
	activation	ReLU
Convolutional (3-4)	number of filters	256
	kernel size	(30,1)
	strides	(1, 1, 1, 1)
	padding	SAME
	activation	ReLU
Dense (5-6)	units	256
	activation	ReLU
	bias	True
Dense (final)	units	1
	activation	linear
	bias	True

Table 4.3: **The hyperparameters used in our implementation of the Vaishnav et al. (2022) architecture.** Our architecture is divided into blocks, with each block corresponding to a specific section of layers in the model. The values in square brackets denote the varying parameters for each layer within the block. It's important to note that both the Convolutional and MaxPool layers in our architecture are 2-D.

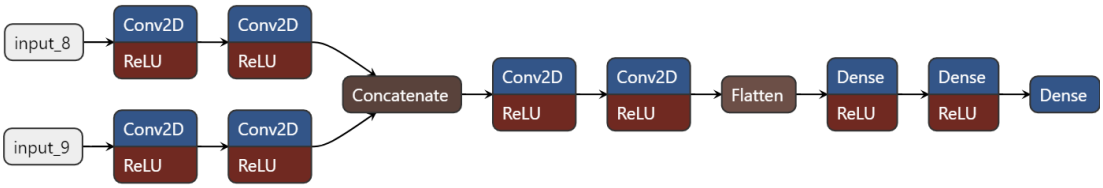

Figure 4.3: **A visualisation of the Vaishnav et al. (2022) CNN.** The model begins with two convolutional layers, one used for forward strands and the other for reverse strands. These layers are then joined together with a concatenation layer. We then employ two further convolutional layers, which are then passed to a flatten layer to reduce the output dimensionality. Finally, we have two ReLU-activated dense layers and a final linearly activated dense layer.

4.2 Training Procedure

4.2.1 Data Partitioning

Double descent occurs when our models are sufficiently larger than the dataset, i.e. $EMC > n$. To achieve this, we must sub-sample our defined and complex media dataset such that the number of training samples is sufficiently smaller than the EMC. As mentioned in Chapter 3, we accomplish this through random sampling. This method decreased our chance of sampling errors when compared to non-probability methods, as well as reducing the bias in our samples.

Our training and validation data is randomly sampled from the entire dataset to meet our needs for certain training procedure sizes. The training sequences are then compared with the validation sequences to ensure there is no data leakage across the samples.

4.2.2 Training Specifications

A batch size of 64 was used throughout the entire model training process, and models were compiled with an Adam optimizer (Kingma and Ba, 2017) with a learning rate of 0.0001; this was done to ensure consistency with Nakkiran et al. (2019).

Since we are focused on epoch-wise double descent, we increased the number of training epochs to 4,000 for our early results and dialled it back to 1,000 after we observed double descent.

4.2.3 Evaluation

To determine if epoch-wise double descent has occurred, we must plot the validation error in terms of the number of training epochs. We will know if it has occurred if the validation error decreases, increases, then decreases again relative to the number of training epochs. Traditionally, this plot of validation error and training epochs would show us after how many epochs we should stop training, but in our case, it's the main

visualisation method of double descent. The loss metric we use for our regressors is mean squared wrror (MSE) which is defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4.7}$$

Where n is the number of data points, Y_i is the observed value and $\hat{Y}_i$ is the predicted value. The main advantage of using MSE over other metrics such as mean absolute error (MAE), is that it ensures our trained model has no outlier predictions with huge errors. This is due to MSE putting a larger weight on these errors due to the squaring of the function.

Although MSE is a key metric in determining if model performance is increasing or decreasing, we need a quantifiable metric for evaluation to observe if model performance is better or worse after double descent. The metric we use is R^2 which is defined as follows:

$$R^2 = 1 - \frac{\sum_i(\hat{y}_i - \bar{y})^2}{\sum_i(y_i - \hat{y})^2} \tag{4.8}$$

Where y_i is the i-th element of the validation or test set. $\hat{y}_i$ is the model's prediction of y_i, and $\bar{y}$ is the mean of y_i. We would have $R^2 = 1$ for a perfect fit of the validation or test sets and an $R^2 = 0$ if the model predicted all sequences have the mean value $\bar{y}$ i.e a baseline model. Therefore, negative R^2 values indicate an inadequate model structure with worse predictions than the baseline model.

Chapter 5

Results

In this chapter, the results of our research are presented. First, we present our initial observation of double descent in sequence-to-expression models. Next, we show double descent occurring across different training sample sizes and architectures. Finally, we observe under what conditions double descent improves the performance of sequence-to-expression models.

5.1 Double Descent

Given that double descent is an empirical evaluation, we must assess it across multiple sequence-to-expression datasets to confirm its existence. As mentioned in Chapter 3, sequence-to-expression models in the literature are trained on datasets from complex and defined media. Figure 5.1 shows double descent occurring when trained on both a defined and complex medium using the DeepSEA model.

The DeepSEA model used in Figure 5.1 was trained on 50,000 sequences and validated on 10,000 sequences per epoch. This train-validate split was chosen to follow Nakkiran et al. (2019) which observed double descent on the CIFAR-10 dataset, which has the same training data split. We can observe that the model's interpolation threshold occurs at approximately 10 epochs and reaches a new local minima at between 100 and 1000 epochs. It is imperative to note that models such as the CNN produced by Vaishnav et al. (2022) are solely trained for 5 epochs, which may result in potential inadequacies in model performance.

Although double descent occurs across sequence-to-expression datasets, the model accuracy after the second descent is on par, or worse, than the accuracy before. This is an indication that the EMC is not sufficiently larger than the number of training samples. Given that double descent occurs for both defined and complex media, we will continue to use only the complex media for the remainder of our results.

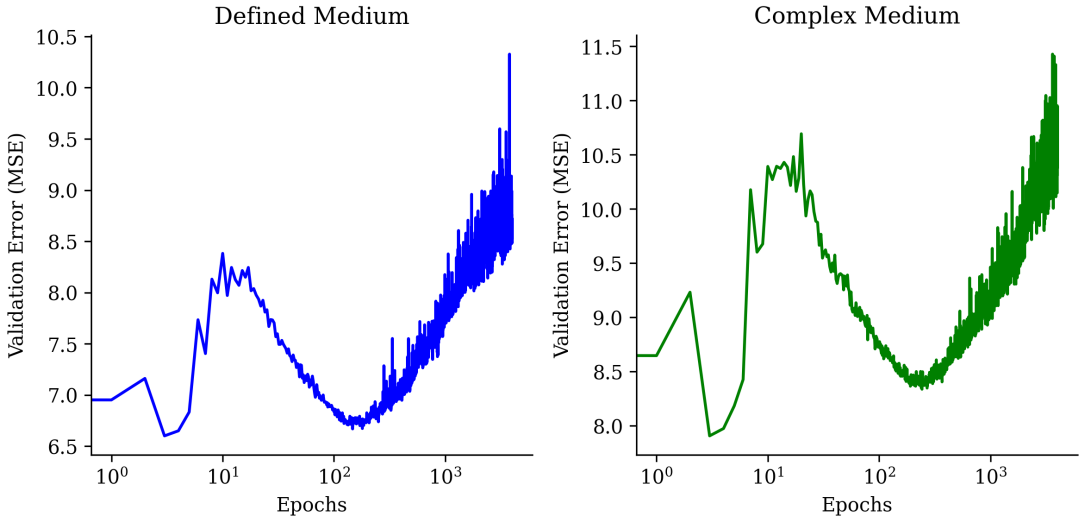

Figure 5.1: **Observation of double descent across datasets.** We trained the DeepSEA model on both a defined medium dataset (left) and a complex medium dataset (right). These models were trained on 50,000 samples from each dataset and validated on 10,000 samples.

5.2 Double Descent Across Sample Sizes

Our primary focus lies on the phenomenon of epoch-wise double descent enhancing the performance of low-N sequence-to-expression models. This is based on the premise that this approach provides us with a greater likelihood of observing the said phenomenon, as the EMC is expected to be significantly larger than the number of training samples. Therefore, we must determine if, for the DeepSEA model, double descent exists within the training sample bounds of low-N models and at what point the model no longer double descends. Figure 5.2 explores double descent for a varied training set size, with the DeepSEA model trained on 1,000 sequences to 1,000,000 sequences. Each model was validated on 10,000 sequences.

As shown in Figure 5.2, double descent still occurs when trained on 1,000,000 sequences, as the EMC is still greater than the number of training samples. When we reach 1,000 samples, we can see that double descent does not occur, but rather the model validation error steadily decreases until it plateaus at between 10 and 100 epochs. The underlying idea is that the EMC, being substantially larger than the number of training samples, results in the phenomenon of double descent being bypassed. As a consequence, the model consistently stays in the over-parameterised regime, instead of making a transition from the under-parameterised to the over-parameterised regime as depicted in Figure 5.1.

The most promising observation is the model trained on 10,000 samples. This model hits a peak in validation error at $\sim$15 epochs and reaches a new local minimum just before 1,000 epochs, which is approximately the same loss as reached at 9 epochs before the peak. This gives us grounds to investigate double descent at between 10,000 and 1,000 samples to find at what point double descent occurs and improves performance.

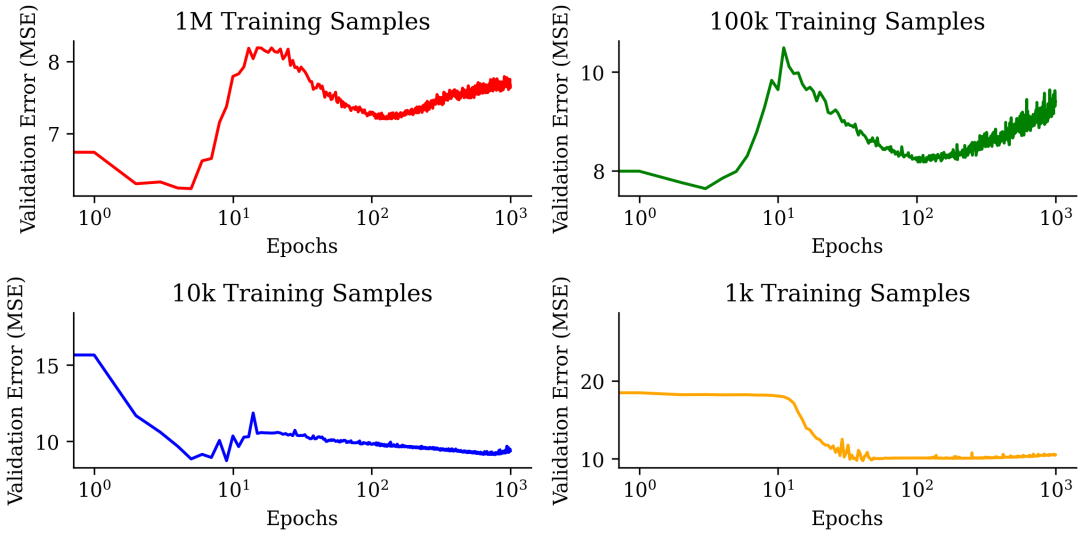

Figure 5.2: **Observation of double descent across training sample sizes.** We trained the DeepSEA model on varied training sample sizes of 1,000,000 (top-left), 100,000 (top-right), 10,000 (bottom-left), and 1,000 (bottom-right). These models were all validated on 10,000 different sequences, and we found that there exists a lower bound where double descent does not occur for sequence-to-expression models.

Note that the number of samples where double descent improves performance may be different for each model as the EMC will change.

5.3 Double Descent Across Architectures

To confirm the existence of double descent across sequence-to-expression models, we must show it is not architecture-specific. Given that Nakkiran et al. (2019) showed double descent for CNNs and ResNets, we have an indication that double descent is solely dependent on EMC the and number of training samples. However, we must observe that popular sequence-to-expression models have sufficient EMC to produce double descent in a low-N scenario. We decided to overtrain two models from the sequence to expression literature, specifically Nikolados et al. (2022) and Vaishnav et al. (2022).

5.3.1 Double Descent on the Nikolados et al. CNN

The model proposed by Nikolados et al. (2022) was trained on 56 different mutational series, with each mutational series containing roughly 4,000 sequences. In some cases, this model achieved state-of-the-art accuracy in a low-N scenario, meaning this model is a prime candidate for double descent as it can generalise well for a low number of sequences.

To be aligned with the paper by Nikolados et al. (2022) we overtrained the model on 4,000 training sequences from the complex medium and validated the model on 10,000 different sequences. We also trained the model on 10,000 sequences and validated it on

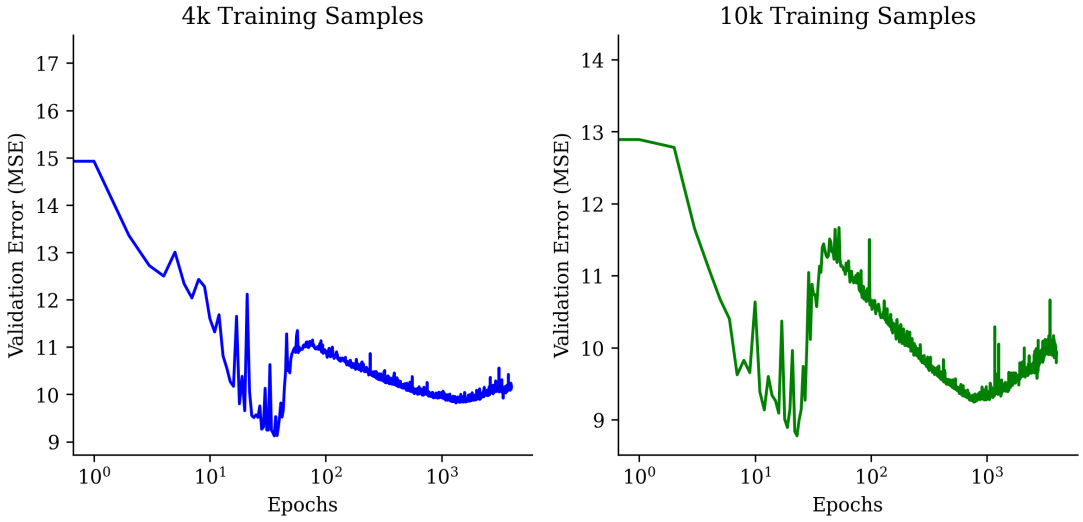

Figure 5.3: **Observation of double descent on the Nikolados et al. (2022) CNN.** We overtrained the CNN proposed by Nikolados et al. (2022) on 4,000 and 10,000 training samples and validated both models on 10,000 samples. We showed that double descent occurs in both cases, with the second descent on 10,000 training samples being more profound.

10,000 to get comparable results with Figure 5.2.

As shown in Figure 5.3 double descent does occur for the low-N sequence-to-expression model proposed by Nikolados et al. (2022). As we can see, double descent occurs for both 4,000 and 10,000 training samples, meaning it has a similar EMC to the DeepSEA (Zhou and Troyanskaya, 2015) model. Although this is promising, we still observe that double descent does not improve model performance for a model that is specifically designed for low-N tasks. This alludes to performance improvements coming from large sequence-to-expression models that are used for low-N tasks.

5.3.2 Double Descent on the Vaishanv et al. CNN

The model produced by Vaishnav et al. (2022) was originally designed to generalise over a large number of sequences, i.e., the entire 20,000,000 and 30,000,000 sequence datasets. Although this is the case, the model employs a forward and reverse strand layer, which increases the number of model parameters and, in turn, increases the overall EMC. Furthermore, this model doesn't employ any MaxPool layers, which lowers the number of parameters in the model.

Given this, the model is a prime candidate for double descent, as the EMC is much larger than previous architectures, pushing it further into the over-parameterised regime. As we are only concerned with low-N models, we trained the model on 4,000 and 10,000 sequences while validating on 10,000 sequences. As shown in Figure 5.4, this did not produce any meaningful results as the model was not able to generalise well. The promising results came from pushing the model further into the over-parameterised

regime when training on 1,000 sequences and validating on 10,000 sequences.

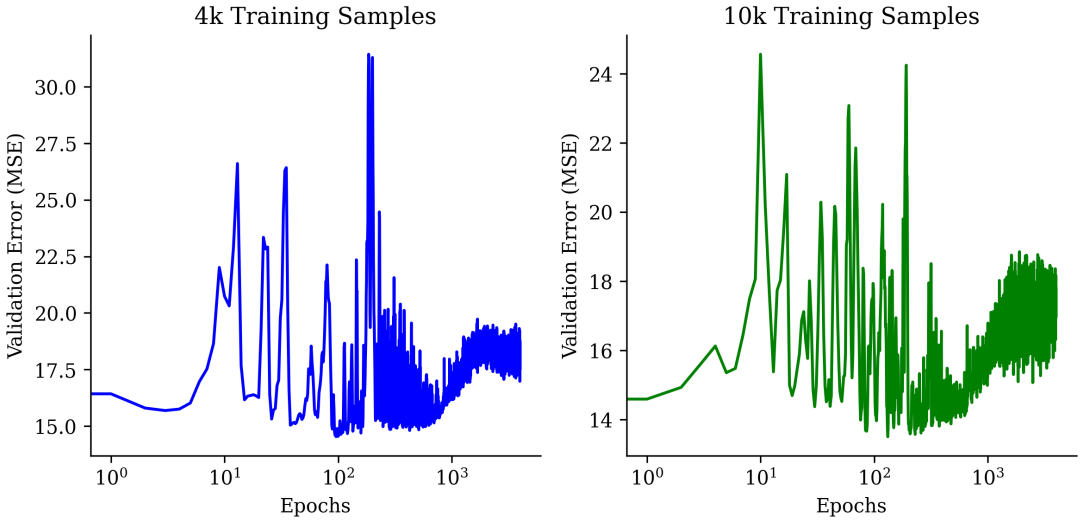

Figure 5.4: **Vaishnav et al. (2022) CNN unable to fit data.** We observed that the Vaishnav et al. (2022) CNN was unable to overtrain on 4,000 and 10,000 sequences from the complex medium. This was due to the number of sequences being too similar to the EMC of the model, causing the model to generalise poorly to new data. To observe double descent on this architecture, we must reduce the number of training sequences to push the EMC further into the over-parameterised regime.

As shown in Figure 5.5, double descent does not only occur, but the validation error improves after the second descent. We observe that, just after 10 training epochs, the local minima of the under-parameterised regime occurs. We observed that our model reaches a peak validation error between 10 and 100 epochs, following which, after the second descent, it gradually starts to plateau in terms of generalisation error beyond 1000 epochs. To quantify if double descent has improved model performance, we must look at a performance metric before and after the second descent, which has been evaluated on an unseen test set, to determine if model performance has actually improved.

5.4 Double Descent Improving Performance

As shown in Figure 5.5 double descent can be used to decrease the overall validation error of a model. While this is promising, we must investigate if this translates into increased performance (R^2) on a held-out test set as discussed in Chapter 3. Before we investigate, we must confirm that the improvement caused by double descent isn't correlated to the size of the validation data and that this improvement can be achieved with validation data of any size.

5.4.1 Double Descent Across Validation Sizes

Given the empirical nature of double descent, we want to decouple it from as many training factors as possible so that its existence can be justified solely based on EMC

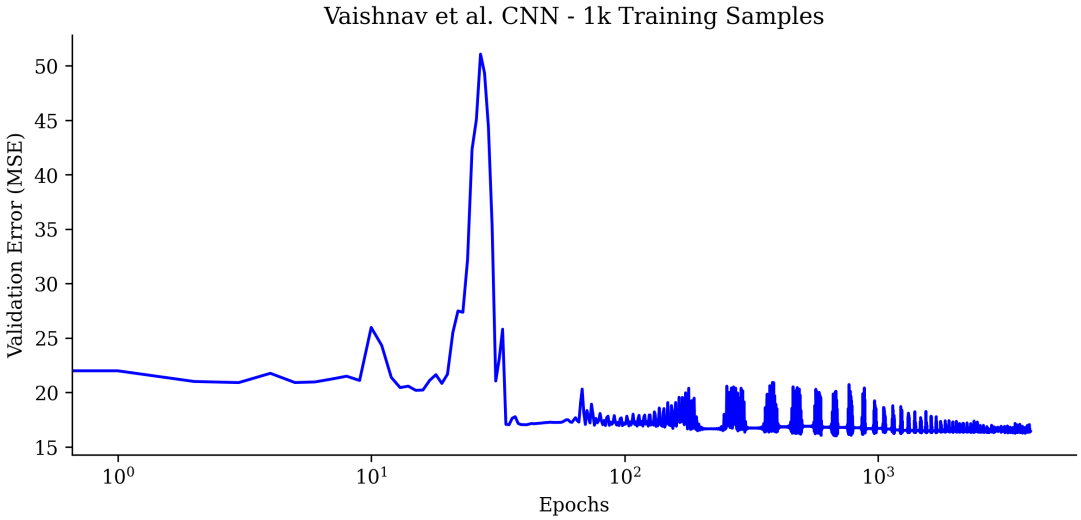

Figure 5.5: **Observation of double descent on the Vaishnav et al. (2022) CNN.** We observed double descent on the Vaishnav et al. (2022) when training on 1,000 sequences and validating on 10,000 sequences. We can see that between 10 and 1000 epochs a peak in validation error is reached, and after 1,000 epochs validation loss plateaus.

and the number of training samples. To do this, we must observe double descent across a wide range of validation sample sizes to determine if it is related to double descent or not. To do this, we trained the Vaishnav et al. (2022) CNN, which produced a double descent that improved validation loss. We trained this model on 1,000 sequences but varied the validation data size each time, starting at 5,000 validation sequences and decreasing it to 100.

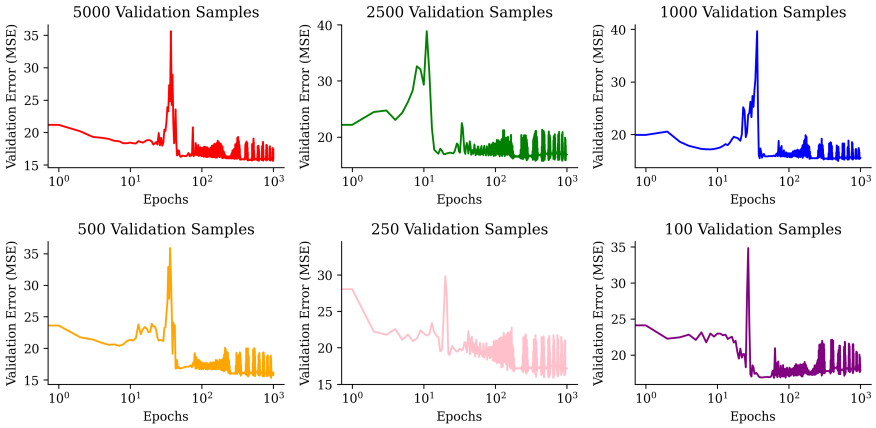

Figure 5.6: **Observation of double descent across many validation set sizes.** We observed double descent on validation sets of 5,000 sequences (top-left), 2,500 sequences (top-middle), 1,000 sequences (top-right), 500 sequences (bottom-left), 250 sequences (bottom-middle), and 100 sequences (bottom-right). The validation sets were used on the Vaishnav et al. (2022) CNN and trained on 1,000 sequences.

As shown in Figure 5.6, epoch-wise double descent is not dependent on validation data

size. In all cases, we observe an initial decrease in validation error, followed by a sharp increase and a final decrease, i.e., double descent. An important observation is that as validation size decreases, the validation loss after the second descent generalises poorly. Good model generalisation stops after 1,000 validation samples, as after that point the validation error after the second descent doesn't plateau like previous validation sample sizes.

5.4.2 Evaluation on Held-Out Test Set

Given that we have now decoupled double descent from validation set size, we must quantify our improvement in validation error using a more quantitative metric. To do this, we will train the Vaishnav et al. (2022) CNN on 1,000 samples and validate it on 10,000. At each local minima, both before and after the second descent, we will evaluate the entire held-out test set described in Chapter 3. This will generate an R^2 score that we can use to determine if the model's accuracy has improved or not.

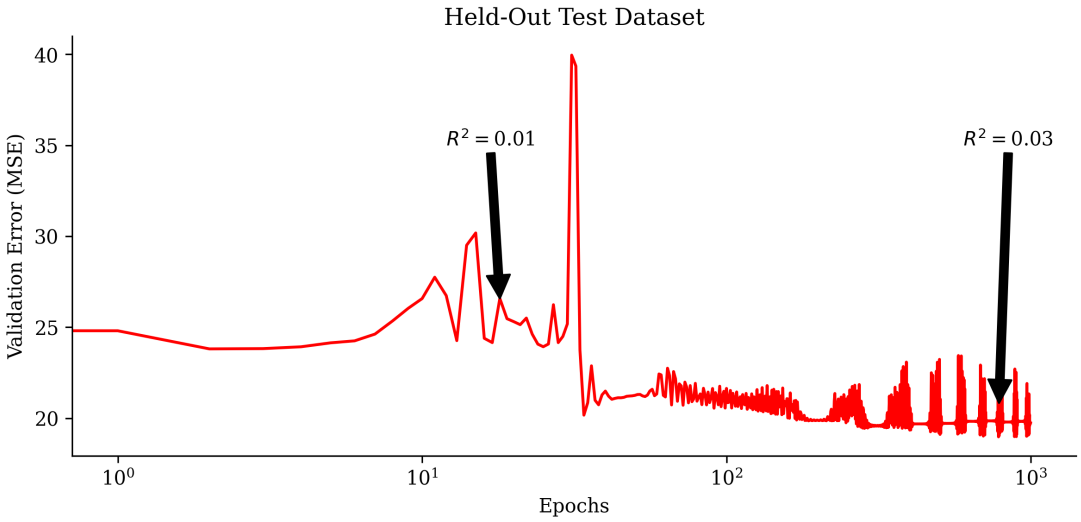

Figure 5.7: **Observation of double descent improving performance when evaluated on a held-out test set.** We observe that when the Vaishnav et al. (2022) CNN is trained on 1,000 sequences, and evaluated on a held-out test set, double descent improves the model's performance by 200%. We recorded the best R^2 both before and after double descent and saw an increase from $R^2 = 0.01$ to $R^2 = 0.03$. This performance increase isn't huge, but it gives us precedence to investigate further.

As shown in Figure 5.7, double descent does improve R^2 of the model by 200%. Although this improvement from $R^2 = 0.01$ to $R^2 = 0.03$ does not seem high, this is due to the model trying to infer sequences from a broader region of the sequence space. We trained the model on only 1,000 sequences from the entire sequence space but evaluated it on over 60,000, which contains more samples of the sequence space. Given that an improvement in R^2 has occurred, we have precedence to investigate model improvement on a model trained on sequences from a specific region of the sequence space.

5.4.3 Evaluation on Mutational Series

We have now observed that epoch-wise double descent does indeed improve the model performance of low-N sequence-to-expression models. Although this is promising, we have only observed this during the evaluation of the model on a sub-sample of the sequence space. In practice, this improvement in model performance would come from sequences that operate in a small region of the sequence space where few sequences are available.

To simulate this, we overtrained the Vaishnav et al. (2022) CNN on the five mutational series described in Chapter 3 using the training procedure proposed by Nikolados et al. (2022). Specifically, we trained the model on each mutational series with varying training size (5%, 10%, 25%, 50%, and 75% of the series) and reserved 10% of the series for validation and 10% for our held-out test set. We discarded the the remaining sequences. Figure 5.8 shows the most promising results, which came from models trained on mutational series 24 and 39, using 10% of the total series to train.

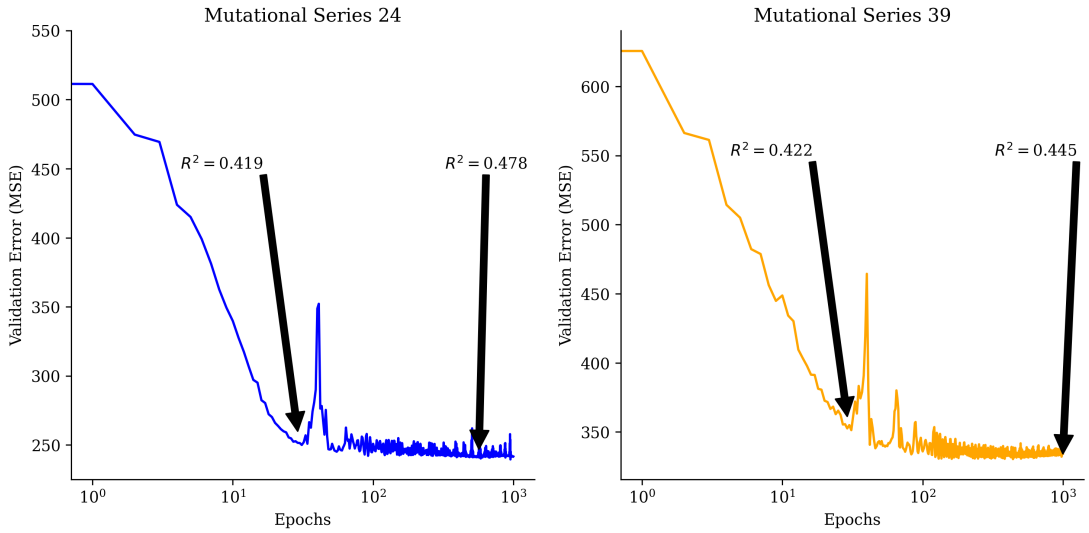

Figure 5.8: **Observation of double descent improving performance when trained and evaluated on the mutational series dataset.** Double descent was observed in two mutational series, namely series 24 (left) and series 39 (right), when we overtrained the Vaishnav et al. (2022) CNN on 10% of the total series while validating on 10% and testing on 10%. Mutational series 24 exhibited the most significant improvement in performance, resulting in a 14% increase in R^2. In contrast, mutational series 39 also showed a considerable improvement in performance, with an increase in R^2 of 5.4%.

As shown in Figure 5.8, double descent improves model performance on mutational series data, increasing the R^2 by 5.4% for series 39 and 14% for series 24. This performance increase comes from pushing the EMC of the Vaishnav et al. (2022) CNN further into over-the parameterised regime by training on $\sim$400 sequences. This gives a more realistic indication of double descent performance as the model only has to generalise to new data inside a mutational series (or a small cluster of the sequence space), which is where low-N sequence-to-expression models perform best.

Chapter 6

Conclusions

In this report, we present epoch-wise double descent across a wide range of sequence-to-expression models. We saw double descent occur in a range of datasets, training and validation sizes, and architectures. We observed that under specific model and training set conditions, epoch-wise double descent can be used to improve low-N sequence-to-expression model performance simply by training a model for longer.

6.1 Accomplishments

1. **Observed double descent empirically in sequence-to-expression models across different datasets.** We observed this phenomenon when trained on a dataset of both complex and defined media, which shows that the sequence data type does not play a part in the existence of double descent.
2. **Observed double descent empirically in sequence-to-expression models across different training and validation set sizes.** We show that double descent occurs for a wide range of training sample sizes, with the most profound decent coming from extremely low-N models that are trained on less than 10,000 samples. We also decoupled validation set size from double descent by showing that it does not affect the existence of the phenomenon.
3. **Observed double descent empirically in sequence-to-expression models across different architectures.** We show that double descent in sequence-to-expression models isn't architecture-specific, as it occurs in all tested architectures regardless of whether the architecture was designed for low-N sequence-to-expression models or not. We can conclude that while architecture plays a part in double descent, if we want to see performance gains, we must alter the number of training samples.
4. **Observed that double descent, under certain conditions, can improve model performance.** We saw that when sequence-to-expression models were only trained on a small sub-sample of the sequence space (using mutational series), epoch-wise double descent could improve model performance. For the Vaishnav

et al. (2022) CNN we found that double descent improved performance by up to 14% for some mutational series.

Overall, we show that the double descent phenomenon can be beneficial for researchers who want to improve their low-N sequence-to-expression model performance. Although this phenomenon does not improve accuracy in every case, it may be worth investigating by researchers to see if their specific model can take advantage of double descent.

6.2 Research Question

As stated in Chapter 1, the goal of this project was to observe double descent in sequence-to-expression models and see under what conditions, we could improve model performance. Based on the results from Chapter 5, we can conclude that the goal has been achieved. Not only does the double descent phenomenon exist in sequence-to-expression models, across a wide range of datasets, training and validation sizes, and architectures, but it can also be used to improve model performance for certain models that are trained on a low number of sequences.

6.3 Future work

6.3.1 Further research into Double Descent

Double descent can be observed empirically by overtraining sequence-to-expression models, but no formal definition or proof exists to justify the phenomenon. Nakkiran et al. (2019) formalises an EMC that hypothesises why general double descent occurs, but it is based purely on observation. A more rigorous definition of double descent in sequence-to-expression models is required to understand what the model generalises well to and what it doesn't when operating in the over-parameterised regime.

6.3.2 A General Double Descent Sequence-to-Expression Model

Double descent was shown to exist across a wide range of architectures in Chapter 5, but no architecture exists where double descent occurs across a wide range of low-N training sizes. We must develop an architecture that only operates in the over-parameterised regime, regardless of the size of the training samples. This can be achieved by stripping away model components such as MaxPool, which decreases the number of parameters of a model after each convolutional layer. By constructing an architecture with a large number of parameters, we can guarantee that EMC will be much greater than the number of training samples and force the model to operate in the over-parameterised regime.

6.3.3 Formulation of Double Descent Classifier

Although double descent can be used by low-N sequence-to-expression models to achieve better performance, there is no way to determine, before running the model, if double descent will improve model performance. A double descent classifier should be developed that, given number of model parameters, training epochs, training set size etc. would return if this is a viable candidate for double descent. This would initially take a large amount of computing but may be beneficial for researchers who don't have the time or resources to overtrain multiple models and determine for themselves if whether double descent can improve performance.

Bibliography

Madhu S. Advani and Andrew M. Saxe. High-dimensional dynamics of generalization error in neural networks. 2017. doi: 10.48550/ARXIV.1710.03667. URL https://arxiv.org/abs/1710.03667.

N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992. ISSN 00031305. URL http://www.jstor.org/stable/2685209.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, jul 2019. doi: 10.1073/pnas.1903070116. URL https://doi.org/10.1073%2Fpnas.1903070116.

Mikhail Belkin, Daniel Hsu, and Ji Xu. Two models of double descent for weak features. *SIAM Journal on Mathematics of Data Science*, 2(4):1167–1180, jan 2020. doi: 10.1137/20m1336072. URL https://doi.org/10.1137%2F20m1336072.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

Pablo Carbonell, Tijana Radivojevic, and Héctor García Martín. Opportunities at the intersection of synthetic biology, machine learning, and automation. *ACS Synthetic Biology*, 8(7):1474–1477, 2019. doi: 10.1021/acssynbio.8b00540. URL https://doi.org/10.1021/acssynbio.8b00540. PMID: 31319671.

Josh T Cuperus, Benjamin Groves, Anna Kuchina, Alexander B Rosenberg, Nebojsa Jojic, Stanley Fields, and Georg Seelig. Deep learning of the regulatory grammar of yeast 5' untranslated regions from 500,000 random sequences. *Genome Res*, 27(12): 2015–2024, November 2017.

Carl G. de Boer, Eeshit Dhaval Vaishnav, Ronen Sadeh, Esteban Luis Abeyta, Nir Friedman, and Aviv Regev. Deciphering eukaryotic gene-regulatory logic with 100 million random promoters. *Nature Biotechnology*, 38(1):56–65, Jan 2020. ISSN 1546-1696. doi: 10.1038/s41587-019-0315-8. URL https://doi.org/10.1038/s41587-019-0315-8.

Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. In M.C. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9.

MIT Press, 1996. URL https://proceedings.neurips.cc/paper/1996/file/d38901788c533e8286cb6400b40b386d-Paper.pdf.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

Anvita Gupta and James Zou. Feedback gan for dna optimizes protein functions. *Nature Machine Intelligence*, 1(2):105–111, Feb 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0017-4. URL https://doi.org/10.1038/s42256-019-0017-4.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, second edition, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Simon Höllerer, Laetitia Papaxanthos, Anja Cathrin Gumpinger, Katrin Fischer, Christian Beisel, Karsten Borgwardt, Yaakov Benenson, and Markus Jeschek. Large-scale dna-based phenotypic recording and deep learning enable highly accurate sequence-function mapping. *Nature Communications*, 11(1):3551, Jul 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-17222-4. URL https://doi.org/10.1038/s41467-020-17222-4.

Luckyson Khaidem, Snehanshu Saha, and Sudeepa Roy Dey. Predicting the direction of stock market prices using random forest, 2016. URL https://arxiv.org/abs/1605.00003.

Kyoung-jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1):307–319, 2003. ISSN 0925-2312. doi: https://doi.org/10.1016/S0925-2312(03)00372-2. URL https://www.sciencedirect.com/science/article/pii/S0925231203003722. Support Vector Machines.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Travis L. LaFleur, Ayaan Hossain, and Howard M. Salis. Automated model-predictive design of synthetic promoters to control transcriptional profiles in bacteria. *Nature Communications*, 13(1):5159, Sep 2022. ISSN 2041-1723. doi: 10.1038/s41467-022-32829-5. URL https://doi.org/10.1038/s41467-022-32829-5.

Jiawei Li, Yuqian Pu, Jijun Tang, Quan Zou, and Fei Guo. DeepATT: a hybrid category attention neural network for identifying functional effects of DNA sequences. *Briefings in Bioinformatics*, 22(3), 08 2020. ISSN 1477-4054. doi: 10.1093/bib/bbaa159. URL https://doi.org/10.1093/bib/bbaa159. bbaa159.

Yang Liu, Hongyu Chen, Limao Zhang, and Zongbao Feng. Enhancing building energy efficiency using a random forest model: A hybrid prediction approach. *Energy Reports*, 7:5003–5012, 2021. ISSN 2352-4847. doi: https://doi.org/10.1016/j.egyr.

2021.07.135. URL https://www.sciencedirect.com/science/article/pii/S2352484721006016.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2018. URL https://arxiv.org/abs/1802.03426.

Preetum Nakkiran. More data can hurt for linear regression: Sample-wise double descent, 2019.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt, 2019. URL https://arxiv.org/abs/1912.02292.

Evangelos-Marios Nikolados, Arin Wongprommoon, Oisin Mac Aodha, Guillaume Cambray, and Diego A. Oyarzún. Accuracy and data efficiency in deep learning models of protein expression. *bioRxiv*, 2022. doi: 10.1101/2021.11.18.468948. URL https://www.biorxiv.org/content/early/2022/11/02/2021.11.18.468948.

Manfred Opper. Statistical mechanics of learning: Generalization. pages 922–925, 1995.

Daniel Quang and Xiaohui Xie. DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic Acids Research*, 44(11):e107–e107, 04 2016. ISSN 0305-1048. doi: 10.1093/nar/gkw226. URL https://doi.org/10.1093/nar/gkw226.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Melanie Schirmer, Rosalinda D'Amore, Umer Z. Ijaz, Neil Hall, and Christopher Quince. Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data. *BMC Bioinformatics*, 17(1):125, Mar 2016. ISSN 1471-2105. doi: 10.1186/s12859-016-0976-y. URL https://doi.org/10.1186/s12859-016-0976-y.

S Spigler, M Geiger, S d'Ascoli, L Sagun, G Biroli, and M Wyart. A jamming transition from under- to over-parametrization affects generalization in deep learning. *Journal of Physics A: Mathematical and Theoretical*, 52(47):474001, oct 2019. doi: 10.1088/1751-8121/ab4c8b. URL https://doi.org/10.1088%2F1751-8121%2Fab4c8b.

Eeshit Dhaval Vaishnav, Carl G. de Boer, Jennifer Molinet, Moran Yassour, Lin Fan, Xian Adiconis, Dawn A. Thompson, Joshua Z. Levin, Francisco A. Cubillos, and Aviv Regev. The evolution, evolvability and engineering of gene regulatory dna. *Nature*, 603(7901):455–463, Mar 2022. ISSN 1476-4687. doi: 10.1038/s41586-022-04506-6. URL https://doi.org/10.1038/s41586-022-04506-6.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on*

Computer Vision and Pattern Recognition (CVPR), pages 5987–5995, 2017a. doi: 10.1109/CVPR.2017.634.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017b. doi: 10.1109/CVPR.2017.634.

Xi Yang, Mingrui Han, Hengliang Tang, Qian Li, and Xiong Luo. Detecting defects with support vector machine in logistics packaging boxes for edge computing. *IEEE Access*, 8:64002–64010, 2020. doi: 10.1109/ACCESS.2020.2984539.

Jian Zhou and Olga G. Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature Methods*, 12(10):931–934, September 2015. ISSN 1548-7091. doi: 10.1038/nmeth.3547. Funding Information: This work was primarily supported by US National Institutes of Health (NIH) grants R01 GM071966 and R01 HG005998 to O.G.T. This work was supported in part by the US National Science Foundation (NSF) CAREER award (DBI-0546275), NIH award T32 HG003284 and NIH grant P50 GM071508. O.G.T. is supported by the Genetic Networks program of the Canadian Institute for Advanced Research (CIFAR). We acknowledge the TIGRESS high-performance computer center at Princeton University for computational resource support. We are grateful to all Troyanskaya laboratory members for valuable discussions. Publisher Copyright: © 2015 Nature America, Inc. All rights reserved.