Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Øyvind Hansen Singsaas

Neural networks for genomic prediction in the wild

Master's thesis in Applied Physics and Mathematics (MTFYMA)
Supervisor: Stefanie Muff
January 2024

NTNU
Norwegian University of
Science and Technology

Øyvind Hansen Singsaas

Neural networks for genomic prediction in the wild

Master's thesis in Applied Physics and Mathematics (MTFYMA)
Supervisor: Stefanie Muff
January 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

NTNU
Kunnskap for en bedre verden

Department of Mathematical Sciences

Master thesis - TMA4900

Neural networks for genomic prediction in the wild

Øyvind Hansen Singsaas

Supervised by Stefanie Muff

2023

Summary

This graduate thesis investigates neural networks for genomic prediction using Single Nucleotide Polymorphisms (SNPs) from house sparrows. Despite the sucess of neural networks in various fields, they have not yet revolutionized genomic prediction. Understanding the genetic basis of phenotypes is crucial for fields like breeding programs and conservation biology.

t-distributed stochastic neighbour embedding (t-SNE) was compared with principal component analysis (PCA) for visualizing high-dimensional SNP data. t-SNE effectively revealed population structures by clustering individuals by their hatch island.

A total of 27 neural networks were trained on two house sparrow datasets containing overlapping samples and SNPs, and the same non-genetic variables. The datasets differed in size: the small dataset had 3032 samples and 182,854 SNPs, and the large dataset had 6092 samples and 66,018 SNPs. The networks predicted body mass, tarsus length, and body mass adjusted for non-genetic variables. A linear mixed model served as a benchmark. The consequences of modelling a wild population consisting of multiple subpopulations is explored by recording the model accuracy separated by the hatch island of the sample.

Three classes of feed forward neural network was tested, multi layered perceptrons, convolutional neural networks and locally connected neural networks. The effect of one-hot encoding applied to SNPs, input size and various approaches to feature selection was also investigated. The best approach to perform feature selection on the SNPs was to choose the SNPs most correlated with the phenotype. The three network types performed similarly when predicting the adjusted body mass. The best performing network, which was trained on the large data set, achieved a Pearson correlation between the predicted and true phenotype of 0.291 ($\sigma = 0.023$), outperforming the linear mixed model, which was trained on the small data set (0.272 (0.029)). None of the neural networks outperformed the linear mixed model on the small data set. One-hot encoding was rarely beneficial, but to performed well when combined with locally connected layers. The neural networks predicting the raw phenotypes (using both genetic and non-genetic data) slightly outperformed the linear model when predicting body mass (Neaural network : 0.347 (0.042), Linear model : 0.320 (0.037), but did worse on tarsus length (Neural network : 0.324 (0.029), Linear model : 0.385 (0.035)). The results in this thesis shows that neural networks can be a viable option for genomic prediction, especially when dealing with large data sets.

Sammendrag

Denne masteroppgaven utforsker bruken av nevrale nettverk for genomisk prediksjon på enkeltnukleotidpolymorfismer (SNP-er) fra en vill populasjon av gråspurv. Til tross for suksessen til nevrale nettverk, har de ennå ikke revolusjonert genomisk prediksjon. En forståelse av det genetiske grunnlaget for fenotyper er avgjørende for forskning innen bevaringsbiologi og avlsprogrammer.

"t-distributed stochastic neighbour embedding" (t-SNE) ble utforsket som et alternativ til prinsipalkomponentanalyse (PCA) for visualisering av høydimensjonal SNP-data. t-SNE ga en to-dimensjonal visualisering som i stor grad samsvarte med hvilken øy fuglene var fra.

Totalt 27 nevrale nettverk ble trent på to datasett av ulik størrelse, men med overlapp i SNP-er og individer. Det lille datasettet hadde 3032 individer med 182,854 SNP-er og det store datasettet hadde 6092 individer med 66,018 SNP-er. Fenotypene kroppsvekt, tarsuslengde, eller kroppsvekten etter justering for ikke-genetiske variabler ble predikert. Den uendrede fenotypen ble predikert av et nevralt nettverk som brukte både genetiske og ikke-genetiske variabler samtidig. Ytelsen til nettverkene ble sammenlignet med en lineær model. Konsekvensene av å modellere en vill bestand bestående av flere subpopulasjoner utforskes ved å sjekke ytelsen på et datasett separert etter individers opprinnelsesøy.

Tre klasser av nevrale nettverk ble testet, "multi layered perceptrons", "con- volutional neural networks" and "locally connected neural network". Effekten av å anvende "one-hot encoding" på SNP-er, antall SNP-er og ulike tilnærminger til redusere antall SNP-er før trening av nettverkene ble testet. Den beste tilnærmingen for å selektere SNP-ene var å velge basert på absolutt korrelasjon med fenotypen. De tre nettverkstypene presterte omtrent likt på den justerte kroppsvekten. Det beste ytelsen kom fra en "multi layered perceptrons" som oppnådde en Pearson-korrelasjon mellom den predikerte og sanne fenotypen på 0.291 ($\sigma = 0.023$) (på det store datasettet), og overgikk dermed den lineære modellen, som ble trent på det lille datasettet (0.272 (0.029)). Ingen av de nevrale nettverkene overgikk den lineære modellen på det lille datasettet. "One-hot encoding" var sjeldent gunstig, men presterte godt når den ble kombinert med et "locally connected layer". På de ujusterte fenotypene presterte nettverkene litt bedre enn den lineære modellen på kroppsvekt (Neuralt nettverk : 0.347 (0.042), ; Lineær modell : 0.320 (0.037), men dårligere på tarsuslengde (Neuralt nettverk : 0.324 (0.029), ; Lineær modell : 0.385 (0.035). Resultatene i denne avhandlingen viser at nevrale nettverk kan være et godt alternativ for genomisk prediksjon, spesielt når datasettet er stort.

Table of Contents

Summary **3**

List of Figures **4**

List of Tables **4**

1 Introduction **5**

2 Background **7**

2.1 Foundational biological concepts for genomic prediction 7
2.1.1 Phenotypic variation 7
2.1.2 QTLs and linkage disequilibrium 7
2.1.3 Gene actions: Dominance and epistatic effects 8
2.1.4 Breeding values 8
2.1.5 Heritability 9
2.2 The prediction problem 9
2.2.1 Animal models and relatedness based regression 9
2.2.2 Marker-based regression 10
2.3 Neural networks 10
2.3.1 Multi-layered perceptrons 11
2.3.2 Convolutional neural networks 12
2.3.3 Locally connected neural networks 15
2.3.4 Training 15
2.3.5 Regularization 16
2.4 Neural networks for genomic prediction 17
2.4.1 Capturing interactions 17
2.4.2 High-dimensional data ($n \ll p$) 18
2.4.3 Interpretability 19
2.5 One-hot encoding 19

3 Methodology **22**

3.1 Data description 22
3.2 Visualizing high-dimensional data 22
3.2.1 Principle Componen Analysis (PCA) 22
3.2.2 t-Distributed Stochastic Neighbour Embedding (t-SNE) 23
3.3 Adjusted phenotype: Accounting for non-genetic effects 25
3.4 Genomic Best Linear Unbiased Predictor 25
3.5 Feature selection: Pre-selecting SNPs 26
3.6 Neural networks predicting the adjusted body mass 27
3.7 Parallel neural network: Predicting on environmental and genetic data . . . 30
3.8 Model assessment and loss function 31
3.9 Testing for difference in model performance by hatch island 32

4 Results **34**

5 Discussion and Conclusion **39**

Bibliography **42**

A t-SNE for varying perplexity and metric **46**

B Variance explained in PCA **46**

C Six principle components of the large data set **47**

D Intersection of SNP selection methods by increasing SNPs **47**

E Adjusting body mass for non-genetic variables **48**

F Best_unif SNP selection **48**
G GBLUP using INLA **50**
H Parallel neural network **51**

List of Figures

1 Multi-layered perceptron 12
2 Sparsity of convolutional neural networks 13
3 Convolutional neural network 14
4 Illustration of One-hot encoding 20
5 Locally connected layer with one-hot encoded SNPs 20
6 Parallel neural network illustration 31
7 Cross-Validation procedure 32
8 PCA and t-SNE visualizations 34
9 Genetic flow between islands 35
10 Accuracy of models on small data set predicting adjusted body mass 36
11 Accuracy of models on large data set predicting adjusted body mass 37
12 Neural network performance separated by hatch island 38
13 Parallel neural network performance on tarsus length and body mass 39

List of Tables

1 Architecture of models fitted to the small data set 28
2 Architecture of models fitted to the large data set 29

1 Introduction

Biological systems are highly complex and the field of biology as become an increasingly data driven area of research. Advances in genomic sequencing, reducing cost and improving accuracy, have enabled the collection of large genomic data sets. One use case of these genomic data sets is modelling the functional relation between the genes and the phenotypes. Methods from machine learning are increasingly being employed to analyze genomic data. Advances in machine learning have revolutionized numerous fields of research involving large complex data. The most impactful machine learning method might be neural networks. This thesis continues previous work in my project thesis (TMA4500) and aims to contribute to the understanding of how neural networks is best applied to genomic data from a wild population for the purpose of genomic prediction.

The aim of genomic prediction is to predict the phenotype of an individual, often based on genome wide molecular markers in the from of single nucleotide polymorphisms (SNPs) (Meuwissen et al. 2001; Meuwissen et al. 2016). SNPs are locations in the DNA sequence where the variation in the observed nucleotides is above a certain threshold. Recording these locations preserve the variation in genetic material within a population, while leaving out the large majority which is identical and would therefore be useless for predicting the effect of genes on phenotypes as we have no observations where they are absent. The correlation of the SNPs with surrounding genes (linkage disequilibrium) lets us capture the effect of genes with a causal relation to the phenotype (quantitative trait loci), without them being directly included in the data. Next generation sequencing has made it possible, at least in principle, to construct models on whole genomes. Predicting on whole genomes would let the model directly estimate the effects of the causal mutations, rather than relying on their linkage disequilibrium with surrounding SNPs. Despite these upsides we are yet to see definitive improvements by whole genome regression compared to SNP based regression (Ros-Freixedes et al. 2022).

Genomic prediction and genomic selection has been widely used to accelerate and improve precision in the artificial selection process in plant and animal breeding (VanRaden 2020). The generation time can be shortened by sampling genetic material rather than having to wait to record phenotypes when they are expressed. Use cases include accelerating development of crops which are more resilient to the changing climate (Budhlakoti et al. 2022). In medicine there is a need for a better understanding of how genes affects the health outcomes of humans, such as poly-genetic disorders. Genomic prediction on wild systems has seen comparably less activity, although a few studies have been published (e.g, Ashraf et al. 2022; Hunter et al. 2022). Genomic prediction in wild populations is primarily focused on conservation biology and understanding evolutionary processes, letting us predict a species capacity to adapt to environmental changes. Genomic prediction on wild populations poses some unique challenges compared to studies on domestic populations. Wild populations often have higher genetic diversity compared to domestic populations, which have been selectively bred. This diversity can make it more difficult to identify relevant genetic markers for traits of interest and causes poor generalization between sub-populations.

In recent years, neural networks have made significant advancements in numerous fields, including image recognition (Khan et al. 2020) and natural language processing (OpenAI 2023). The unparalleled success of neural networks models in these areas has resulted in a growing interest in their application to various domains. However, in the field of genomic prediction using SNP data, the impact of neural networks has been limited. More

traditional statistical models have remained the dominant method for genomic prediction tasks (Montesinos-López et al. 2021). The effect of genes on phenotypes are known to have aspects that are poorly captured by the commonly used linear models. This thesis investigates the viability of the much more flexible neural networks, as they in principle should be able to model genetic effects that are poorly modeled by linear models. Until now, most studies on genomic prediction using machine learning techniques focus primarily on genomic selection within plant and animal breeding contexts (Li et al. 2018; Montesinos-López et al. 2021; Ros-Freixedes et al. 2022; Wang et al. 2022), as well in medicine, disease risk and complex trait prediction (Bellot et al. 2018; Kim et al. 2018).

Despite these potential upsides, linear models are still the dominant method, as the neural networks have yet to consistently outperform the linear models (e.g., Montesinos-López et al. 2021). This raises the question of whether neural network models are inherently unsuitable for genomic prediction tasks, or if the appropriate architecture and methodology have yet to be discovered. To address this question, this thesis focuses on an empirical analysis of a real data set from collaborators at the Centre for Biodiversity Dynamics (CBD) at NTNU using a data set of house sparrows in Northern Norway. A total of 28 models are trained to predict the phenotype tarsus length, body mass or an adjusted body mass (adjusted for non-genetic effects). These models all vary in architecture, input size and input encoding, giving insights into the what the best design choices are for such neural networks. Topics covered include SNP selection criteria for dimensionality reduction, network architecture types, SNP encoding and regularization techniques. In addition, two visualization techniques for exploring potential genetic clustering of the wild population is presented and discussed. These two methods are the commonly used principle component analysis (PCA) and the more novel t-distributed stochastic neighbour embedding (t-SNE).

NTNU has stated goal to be a leading actor tackling today's sustainability challenges, researching aspects ranging from new technology and innovative solutions, modelling and environmental impact assessments to research on consumer behavior and governance (https://www.ntnu.no/barekraft). This thesis contributes to the work of Centre for Biodiversity Dynamic by investigating new methods for genomic prediction, which is important for some ecological metrics used in conservation biology. Additionally, advancing the field of genomic prediction is beneficial for the development of crops that are better suited for the climate of tomorrow.

2 Background

2.1 Foundational biological concepts for genomic prediction

To understand the discussion of how neural networks can be suited for genomic prediction, a basic understanding of concepts in quantitative genetics is needed. These concepts has informing the design choices made in neural networks trained in this thesis. The terms introduced relates to the different ways genes acts on phenotypes and how we categorize them.

2.1.1 Phenotypic variation

Phenotypic variation refers to the differences in physical characteristics and traits observed among the individuals of a population, influenced by both genetic factors (G) and environmental conditions (E). Genetic factors refers to the genetic material (DNA). Environmental conditions are the various external factors and influences, such as climate, habitat, and resources, that impact the development and traits of an animal. Variation in G and E gives rise to the variation in phenotype (P). This separation of genetic and environmental effects means that the phenotype can be partitioned into these two components, following the equation $P = G + E$. The variance in phenotype is by extension expected to be the sum of the genetic and environmental variances ($\sigma^2_P = \sigma^2_G + \sigma^2_E$), assuming that these two components are independent. The assumption that the genetic and environmental variance is independent of each other is made in the most commonly applied statistical models for genomic prediction.

A more nuanced view of phenotypic variation includes the interactions between genes and the environment ($G \times E$) (Conner and Hartl 2004, p. 108). The $G \times E$ interaction refers to gene action being depending on the environment. Two individuals with different genetic material may express the same phenotype, but change the environment and the $G \times E$ interaction may give different changes in the phenotype of the individuals. This effect is also referred to as phenotypic plasticity. The presence of $G \times E$ interactions underlines the fact that the genomic prediction task, especially in wild systems, is highly dependent on the specific population and environment. The contribution of genetic variation, environmental variation, and their interactions to the phenotype variation will be different for each phenotype. Some phenotypes may be sufficiently modelled only considering genetic and environmental effects separately. However, if the $G \times E$ effects are sizable, then a model that captures these effects would be beneficial.

2.1.2 QTLs and linkage disequilibrium

Loci (locations in the DNA) that regulates quantitative traits are called quantitative trait loci (QTLs). These are the causal genes that affects the expressed phenotype. Identifying QTLs is important for purposes such as disease risk modelling. In genomic prediction we mainly want to model their effects. Usually however, the whole genome is not available for analysis. This is also true for the data studied in this thesis. Only certain genes are recorded, single nucleotide polymorphisms (SNPs). These are only loci on the genome where the population has a certain level of variation, meaning that important QTLs are

not necessary present in the data. This is where the linkage disequilibrium becomes an important principle. Dense marker maps will have markers sufficiently close to any QTL so that they are more likely to be inherited together, and therefore be correlated in the data. Linkage disequilibrium refers to genes not appearing independently of each other, but being correlated based on how they are inherited (Conner and Hartl 2004, p. 157). High linkage disequilibrium means that knowing the genotype at one locus gives information about likely genotypes at nearby locus. This is why markers, such as SNPs, can be used to identify QTLs, even if the markers are not causal themselves. When using these SNPs as features in a statistical model, the linkage disequilibrium introduces the problem of multicollinearity. Multicollinearity is the presence of highly correlated features in the data set. This informed decisions on the pre-selection of SNPs described in Section 3.5.

2.1.3 Gene actions: Dominance and epistatic effects

Gene actions, the mechanism through which a gene influences the phenotype, come in multiple different flavours and is a central reason to why non-parametric machine learning techniques such as neural networks might be suited for genomic prediction. Interactions on the genome happens at two levels, within a locus and between different loci. The the absence of interaction at any of these levels, is referred to as additive gene action. Between different loci, additive effects means that the effect of one locus is independent of the genotype at another locus, making the effects summable. Within a locus, additive effects are present when the heterozygote has an intermediate contribution to the phenotype, between the homozygotes for the alternative alleles (gene variants) (Conner and Hartl 2004, p. 103).

Dominance effects refers to the alternative ways there can be an interaction within a locus (Conner and Hartl 2004, p. 105). These are again organized into sub-categories. The simplest case for illustration might be the case of complete dominance. Complete dominance describes the case when the effect of one allele in a heteroztgote genotype is completely muted by the other. The effect of one allele is therefore dependent on the allele at the same locus, making it non-additive. This motivated the exploration of an alternative way of encoding the SNPs, introduced in Section 2.5.

The final category of gene action, essential for the intentions of this study, is epistatic effects. Epistatic effects are interactions between different loci, meaning that the contribution of a genotype is not known without knowing its interactions. When epistatic effects are present, the effect sizes of each gene cannot simply be summed up without considering the interplay with other loci. Non-additive effects complicates the genomic prediction task and might make approaches such as neural networks suitable, as they are not directly modelled in the more commonly applied linear methods.

2.1.4 Breeding values

The breeding value, often denoted as the *genetic value*, is defined as the additive effects of an individual's genes on the value of the trait in its offspring (Conner and Hartl 2004, p. 111). It is essentially the sum of gene effects across all loci contributing to the trait. In practice, we don't usually know the exact breeding value of an individual, but we can estimate it based on observed phenotypes and genetic data. These estimates are known as estimated breeding values. In genomic prediction, the goal is to estimate these breeding values as accurately as possible, often to guide selection decisions.

2.1.5 Heritability

Heritability is a measure that quantifies the proportion of observed phenotypic variation in a population that can be attributed to genetic variation (Conner and Hartl 2004, p. 112). In the context of genomic prediction, the concept of heritability represents the upper limit of prediction accuracy that can be achieved using genomic data for a given trait in a specific population. Heritability is dependent on the genetic variation and environmental variation seen in a studied population. Importantly, this means that the performance in a genomic prediction task is not transferable to other populations. In the extreme case, if a trait is fixed among all individuals (say the number of eyes in humans), the heritability will be zero. However, this does not mean that it is not genetically determined.

The heritability is often separated into two types, broad-sense heritability (H^2) and narrow-sense heritability (h^2). Broad-sense heritability is defined as $H^2 = \frac{\text{var}(G)}{\text{var}(P)}$, the total genetic varaince as proportion of the phenotypic variance. Narrow-sense heritability is restricted to the proportion of phenotypic variance attributed to the additive genetic variance, defined as $h^2 = \frac{\text{var}(G_a)}{\text{var}(P)}$, where $\text{var}(G_a)$ is the variance of the additive genetic effects. Narrow-sense heritability is often used because of its relation to statistical assumptions in the most used linear models for genomic prediction. It is also an easier quantity to estimate, as complex interactions on the genome is discarded, and only the average effects in the population are included.

2.2 The prediction problem

Linear mixed models predicting breeding values in livestock and plants are often called animal models. For n observations, a simple version of an animal model can be formulated as

$$y_i = \mu + g_i + \varepsilon_i,\ i = 1, 2, ..., n\ , \tag{1}$$

where y_i is the phenotype of individual i, μ is the population mean, g_i is the breeding value of individual i and ε_i is the independent environmental effects. Note that the effects g_i and ϵ are separate terms and no interaction is included. The result is that the variation in phenotype is decomposed as the genetic and environmental variance under the assumption of $\sigma^2_P = \sigma^2_E + \sigma^2_G$. This decomposition of phenotypic variation is often violated, as discussed in Section 2.1. Estimating the genetic effects, or in the case of Equation 1, the breeding values , can be done multiple ways. To provide important context for the application of neural networks to genomic prediction, the two approaches of animal models and marker based regression are briefly introduced. The results of an animal model will serve as the comparison for the neural networks trained in this thesis.

2.2.1 Animal models and relatedness based regression

Animal models are linear mixed models (Kruuk 2004), a model family that includes explanatory variables which are fixed and some which are random. Fixed effects are constants associated with a feature, such as sex, affecting the mean of the distribution. For example, an individual specific random effect can be included in the model. The variance of this random effect would reflect the variance between different individuals not explained by the fixed effects. In equation (1), the vector of breeding values, $\mathbf{g} = [g_1, ..., g_n]^{\mathrm{T}}$ is assumed drawn from a multivariate normal distribution $\mathbf{g} \sim \mathbf{N}(0, \sigma^2_G \cdot \mathbf{G})$. The $n \times n$ relatedness matrix is denoted $\mathbf{G}$ and σ^2_G is the additive genetic variance. The relatedness matrix can

be estimated from a pedigree if it is known. For instance, full siblings have a coefficient of 0.5, indicating that they in expectation would share about half of their genetic material, assuming non-related parents. With the availability of SNP data, the relatedness matrix can also be calculated based on the amount of shared genetic material (Bérénos et al. 2014). From the matrix of markers, $\mathbf{Z}$, the relatedness matrix is defined as

$$\mathbf{G} = \frac{\mathbf{Z}\mathbf{Z}^T}{2\sum p_i(1-p_i)},$$

where p_i is the frequency of the second allele (VanRaden 2008). The matrix $\mathbf{Z}$ has dimensions $n \times p$, where p is the number of markers.

2.2.2 Marker-based regression

Marker-based regression is an alternative to the animal model where each marker, often SNPs, is directly included as an explanatory variable. The total genetic effect for an individual is then the sum of the marker effects. For n samples with p markers the marker based regression can be formulated as

$$\mathbf{y} = \boldsymbol{\mu} + \mathbf{Z}\mathbf{u} + \mathbf{X}\mathbf{b} + \mathbf{W}\mathbf{d} + \boldsymbol{\varepsilon}\ , \tag{2}$$

where $\mathbf{Z}$ is a $n \times p$ matrix of marker codes (encoding the genotypes aa, aA and AA as 0, 1 and 2 respectivly). The vector of responses, $\mathbf{y}$, is $n \times 1$ dimensional. μ is the population mean, $\mathbf{Xb}$ accounts for any fixed effects and $\mathbf{Wd}$ incorporates the random effects included in the model. In the case of $n < p$, this model is not solvable by ordinary least squares (Lande and Thompsont 1990). Different approaches are possible to get an identifiable model, in both the frequentist (e.g., GBLUP, Meuwissen et al. (2001) and VanRaden (2008)) and Bayesian paradigm (Gianola 2013).

In this study, the genetic merit of an individual, $\mathbf{Zu}$ in Equation (2), will be estimated by neural networks. This will allow the modelling of more complex gene actions. As defined in Equation (2), there is no terms for between loci interactions or $G \times E$ effects. With the SNPs encoded as 0, 1 or 2, there is also an assumption of additive effects within a loci, as the genotype Aa, coded as 1, will be given an intermediate effect between aa and AA, coded as 0 and 2. An approach where both the environmental and genetic effects are modelled by a neural network was also explored in this thesis (Section 3.7), which also allows interactions between environmental and genetic variables.

2.3 Neural networks

The first, and most basic neural network was the single-layer perceptron, used for binary classification. It was inspired by the functioning of neurons and neural pathways in the brain, and was first conceptualized by McCulloch and Pitts in 1943 (Mcculloch and Pitts 1943), with the first formulation of the perception by Frank Rosenblatt in 1958 (Rosenblatt 1958) . Since then, modern computing power and innovations in new architectures have made neural networks arguably the most powerful machine learning method, most notably in tasks such as image classification and natural language processing. The meth-

ods developed in these areas of research have found applications in a wide variety of fields, including genetics and biology (e.g. Jumper et al. 2021).

A neural network consists of at least three layers: an input layer, one or more hidden layers, and an output layer. The input layer receives the initial data for the neural network, while the output layer makes the final prediction (or any other type of output such as a decision or generated text). In the networks considered in this thesis, the flow of information is always from the input layer to the output layer. Other network types incorporates recurrence, where the flow of information also flows from a later layer to an earlier layer. Each layer consists of a certain number of nodes (or units). The nodes in each layer is connected forwards and backwards to later and earlier layers. Associated with each connection there is a weight, $w_{i,j}^k$ (weight associated with input from node j to node i in layer k), which the value passed between the nodes is multiplied by. At each node all the incoming connections are summed and a bias is added. Finally an activation function is applied before information is passed on to the later layers. To further describe a neural network we will have to consider a specific type of architecture, as the operations within them vary.

Various architectures have been developed to address specific problems, with the choice of architecture being highly dependent on the data being analyzed. Most methods are applicable to multiple domains, especially if similar structure and dependencies exist in the data. Examples of architectures and areas where they excel include convolutional neural networks for image classification and transformers for natural language processing. Three classes of neural networks are used in this study. These are multi layered perceptrons (MLPs), convolutional neural networks (CNNs) and locally connected neural networks (LCNNs). The specific layer types and their potential benefits for genomic prediction are now introduced.

2.3.1 Multi-layered perceptrons

MLPs (also called fully connected, or dense, feed forward neural networks) are the perhaps the simplest form of neural networks, being a natural extension of the perceptron. Consider Figure 1, where an MLP with two hidden layers and an input vector of length four is illustrated. The characterization of an MLP is that every node in each layer is connected to every other node in both the preceding and proceeding layer, meaning that the whole output of the preceding layer is given as input to each node in the next layer.

Consider the output a_j from a node j in the first hidden layer of an MLP, as illustrated in Figure 1. It is fully connected to the input layer, and therefore has the full feature vector, $\mathbf{X}$, as its input. Every value x_i in the vector $\mathbf{X}$ is multiplied by a weight $w_{i,j}^1$, which can be represented as the vector product $(\mathbf{W}_i^1)^T\mathbf{X}$. In addition to this vector product, a constant β_i^k is added to node i in layer k. Finally, an activation function, ϕ_1, is applied, giving the final expression for the value, a_j, attained at node i in the first hidden layer,

$$a_j = \phi_1\left(x_1 \cdot w_{1,i}^1 + x_2 \cdot w_{2,i}^1 + x_3 \cdot w_{3,i}^1 + x_4 \cdot w_{4,i}^1 + \beta_i^1\right) = \phi\left((\mathbf{W}_i^1)^T\mathbf{X} + \beta_i^1\right) \ , \quad (3)$$

where $\mathbf{W}_i^1 = [w_{1,i}^1, w_{2,i}^1, w_{3,i}^1, w_{4,i}^1]$, that is, every weight for the input to node i in layer 1.

One can also consider the whole operation at a hidden layer as one matrix operation. Define $\mathbf{W}_k = [W_1^k, W_2^k, W_3^k, W_4^k]^T$, the matrix of stacked row vectors containing the input weights. In addition, define the vector of stacked biases, $\boldsymbol{\beta}_k = [\beta_1^k, \beta_2^k, \beta_3^k, \beta_4^k]^T$. For

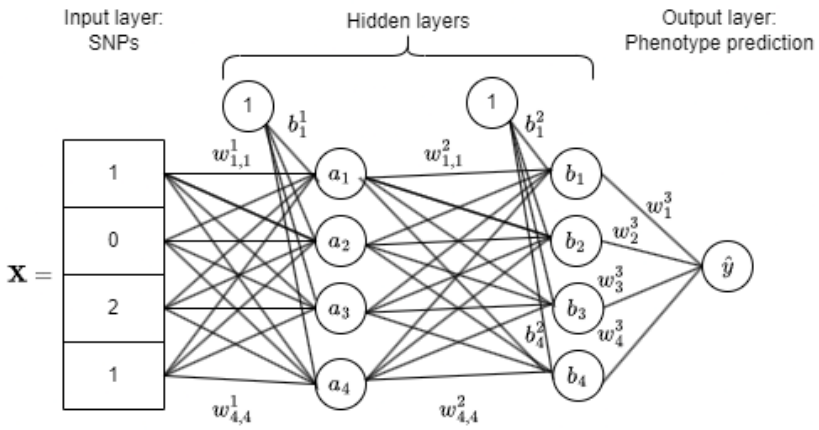


Figure 1: Illustration of a multi-layered perceptron with two hidden layers and SNP vector $\mathbf{X}$, taking values 0, 1 or 2, as input. Information flows from left to right, giving the output $\hat{y}$. The weights $w_{i,j}^k$ and biases b_j^k are fitted during training.

simplicity assume that the same activation function, ϕ, is applied at each layer. The output of the network can then be stated as,

$$\hat{y} = \phi_3 \left(\mathbf{W}_3^T \cdot \phi_2 \left(\mathbf{W}_2^T \cdot \phi_1 \left(\mathbf{W}_1^T \cdot \mathbf{X} + \boldsymbol{\beta}_1 \right) + \boldsymbol{\beta}_2 \right) + \boldsymbol{\beta}_3 \right) \ . \tag{4}$$

Until now, not much have been said on the activation function ϕ. If the activation function was the identity function, then the model as defined in Equation (4), would simply be a linear predictor. The inclusion of non-linear activation functions is what gives neural networks its power to model non-linearity and represent an integral part of the function approximation ability of neural networks. According to the Universal Approximation Theorem, a network with at least one hidden layer containing a finite number of neurons can approximate any continuous function to an arbitrary level of accuracy, provided the activation function is non-linear (Hornik et al. 1989).

The choice of activation function for the output layer is of special importance as it must ensure that the model is giving output on the correct scale, which is decided by the specific task at hand. For example, a sigmoid function is often used when modeling the probabilities of binary events, as it outputs values in the interval $(0, 1)$. The rectified linear unit (ReLu) is a popular choice of activation function in the hidden layers, as it has many desirable properties for the training process of neural networks (Hara et al. 2015). ReLu can be expressed as $\phi_{ReLu}(x) = max(x, 0)$.

2.3.2 Convolutional neural networks

Convolutional neural networks (CNNs) leverage the convolution operation to find effective representations in the data. Desirable properties of CNNs include sparse interactions, parameter sharing and equivariant representations (Ian Goodfellow et al. 2016, Ch. 9), properties which might be beneficial for the genomic prediction task. These will be elaborated upon further, but first let us introduce the basic convolution operation. The convolution, $s(t)$, between two real valued functions v and ω can be stated as

$$s(t) = \int \omega(x)v(t\ -\ x)\ dx\ .$$

Neural networks operate on discrete input data, which is why the discrete concolution is needed. In the context of a 1D vector, a discrete convolution operation can be defined as follows (Ian Goodfellow et al. 2016, Ch. 9). Consider an input vector $\mathbf{X}$ of length N and a filter (or kernel) $\mathbf{A} = [A_1, A_2, ..., A_M]^T$ of length M, where $M \leq N$. The convolution of $\mathbf{X}$ with $\mathbf{A}$ is another vector $\mathbf{a} = [a_1, a_2, ..., a_{N_M+1}]$ of length $N - M + 1$, assuming no padding is applied on the boundaries. Each element of $\mathbf{a}$ is given by the dot product of $\mathbf{A}$ with a subvector of $\mathbf{X}$ of length M. The dot product can be expressed as,

$$a_i = \sum_{j=0}^{M-1} \mathbf{A}_j \cdot x_{i+j}, \quad i = 0, \ldots, N - M\ . \tag{5}$$

The subvector of $\mathbf{X}$ considered for output a_i of the filter will be the vector $[x_i, x_{i+1}, ..., x_{i+N-M}]^T$. The operation of applying a filter of length M to a vector is often thought of as a window covering M values in the input, and being shifted by a certain stride length for each operation (Equation (5) assumes a stride of one). Each placement of the window corresponds to one operation on the input covered by the filter. In the training of a CNN the weights of $\mathbf{A}$ will be tuned to minimize the chosen loss function.

The interactions in a CNN will be sparse compared to an MLP, as long as the kernel is smaller than the input vector. Consider Figure 2. For a filter of width of three applied with a stride of one, every value in the preceding layer is only fed to three different filter operations. Similarly, for every output of the filter, only three values in the preceding layer are considered. In an MLP, the entirety of the output of the preceding layer is given to every node in the next layer.

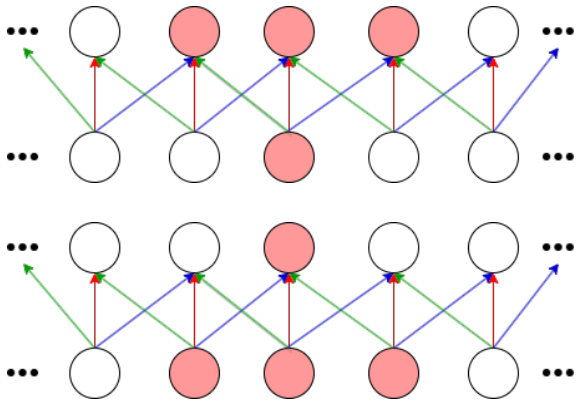


Figure 2: One convolutional filter with a width of three applied with a stride of one. Same colored edges represents the same learned weight, illustrating the parameter sharing in CNNs. Top: Connectivity of a node in the preceding layer to the next layer. Only three nodes in the next layer are passed a value from any given node in the preceding layer. Bottom: Similarly, every filter output is only given input from three nodes in the preceding layer.

Another potentially desirable trait of CNNs is parameter sharing, as illustrated i Figure 2. In an MLP, each weight is applied exactly once for each input vector. This is not the case for CNNs. Each filter is applied along the whole input, with the same weights. In other

words, the filter **A** is constant along the whole sequence. The weights are therefore trained to perform well on the whole sequence simultaneously (although it might not have to, as different filters may learn to capture important features in different regions). A property of CNNs related to parameter sharing is equivariance to translation. If the input is shifted by some value, then the output of the convolution is shifted accordingly. Equivariance to translation is a property well suited for problems with a spatial (or temporal) structure, as the output of the convolution retains a notion of location in the input. Consider a convolutional neural network trained to detect edges in an image. Edges are edges wherever they appear in the image, and the sharing of parameters in CNNs facilitates an effective way of detecting edges by applying the same "edge detection" at all locations, without having to learn it at each location separately. In addition, it will retain a sense of the location of the edge. If an edge is shifted by some value before being passed to the convolutional layer, it will give the same output as if shift was applied to output of the convolutional layer applied to the original image. The notion of location in an image can be especially useful for building higher level representations in later layers.

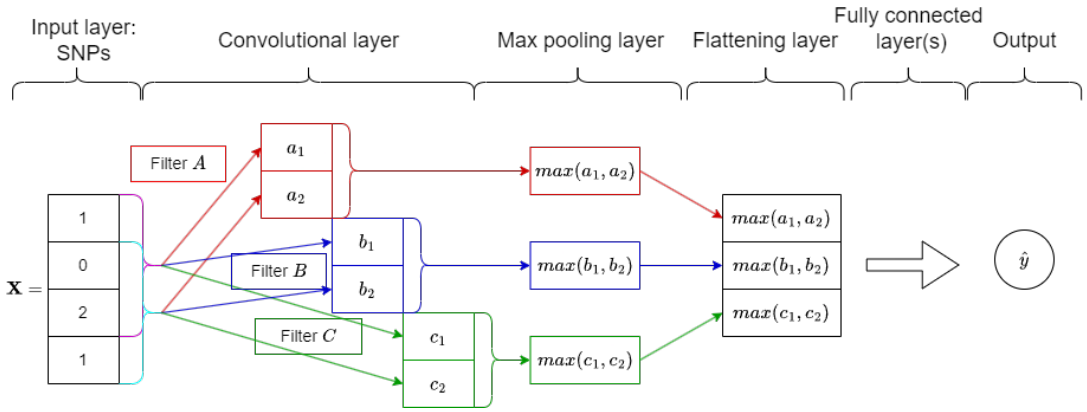


Figure 3: A convolutional neural network applied to a SNP vector. The convolutional layer consists of three filers (A, B and C) with a width of three and a stride of one, resulting in only two passes through each filter. The resulting output is the three vectors of length two, corresponding to each of the filters. A max pooling layer is applied to each of the filter output in its entirety. Generally the pooling layer is a sliding window which takes the maximum over some width of the filter output and is shifted by some stride between each operation. A flattening layer stacks the max pooling output before it is passed subsequent layers.

The convolutional layer in a neural network often consists of multiple filters applied to the input, as illustrated in Figure 3. After the convolutional layer, there is typically a pooling layer, which reduces the dimension of each filter output. A common choice is max pooling (Zhou and Chellappa 1992), which reports the maximum value within a window of each filter output. A typical layer after the pooling layer is the flattening layer, which simply concatenates the output from the different filters into a dimension of suitable shape for the coming layers. The final layers of a CNN usually consists of one or more fully connected layers, before the final output layer.

A CNN can, in principle, be applied to many types of data. However, it is not immediately apparent that SNP data possess a structure that can efficiently be leveraged the properties of CNNs, such as sparse interactions, parameter sharing, and equivariant representations. There is especially no reason to expect the SNP vector to be equivariant to transition, allowing the shared weights of a CNN excel. The same sequence of markers detected by a convolutional filter at two different location in the SNP vector will probably have no

relation to each other in terms of their effect on the phenotype. It is however easier to argue that the sparse interactions of a CNN is suitable for the SNP vector. Genetic material is inherited as longer haplotypes, meaning that closely located SNPs will be highly correlated and possibly encode the same information about greater genetic material. Having a fully connected layer connected to the SNP vector will therefore have a high degree of redundant interactions. The convolution operation might allow the network to recognise haplotypes more efficiently. CNNs have been the best performing network type in multiple studies on genomic prediction (Bellot et al. 2018; Pérez-Enciso and Zingaretti 2019). The best performing architectures are however highly dependent on the type of gene action affecting the phenotype being modeled.

2.3.3 Locally connected neural networks

Locally connected layers can provide the benefits of sparse connectivity, without the assumption of in-variance to translation of the SNP vector (following from shared weights). Locally connected layers may be thought of as a convolutional layer, but without the shared weights (Elizondo and Fiesler 1997). In the same way as in a CNN, for each filter, a set of weights is applied to a window of the input vector and the repeated with a certain stride length.

Locally connected neural networks (LCNN) have been applied to genomic prediction of simulated phenotypes in Pook et al. (2020) and was found to outperform both CNNs and MLPs. LCNNs also outperformed a linear model for large population sizes ($n = 10000$) and high heritability. The LCNN architecture has an especially interesting interpretation when combined one-hot encoded SNPs, introduced later in Section 2.5.

The neural network types introduced in this section might not represent the cutting edge of research in machine learning, but they are still highly relevant. More novel approaches such as transformer neural networks, which uses the attention mechanism, has also been applied to genomic data (Kim et al. 2018; Ng 2017; Romero et al. 2016). The attention mechanism provide a method for weighing the importance of different elements in a sequence, and has become the leading network type for natural language processing (Vaswani et al. 2017). Cahyawijaya et al. (2022) introduces SNP2Vec for genome wide association studies, in which an attention based model is used to encode long genomic sequences (reduce the dimension) for the prediction of Alzheimer's disease.

2.3.4 Training

Modern neural networks are typically trained using backpropagation and stochastic gradient descent (Ian Goodfellow et al. 2016, Ch. 6.5). The training process aims to find the set of trainable parameters, which minimizes the chosen loss function. A common choice of loss function for regression problems is the mean squared error (MSE). Before one iteration of backpropagation and gradient descent is performed on the network weights, more than one of the observation is usually passed through the network. This subset of the training set is called a batch. After one batch is passed through the network, the loss is computed, before backpropagation is used to compute the gradient of the loss with respect to the network's parameters. The model parameters are then improved by a gradient descent step, which is dependent on the hyperparameter learning rate, which is set by the user. For a batch of size N and corresponding set of responses, $\{y_i\}_{i=1}^N$, and predictions, $\{\hat{y}_i\}_{i=1}^N$, the MSE loss is computed as,

$$L_{\text{batch}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \ . \tag{6}$$

A larger batch size provides a more accurate estimate of the gradient of the loss function, but requires more computational resources and may lead to less frequent updates, potentially slowing down the learning process (Ian Goodfellow et al. 2016, Ch. 8). On the other hand, smaller batch size leads to more noise in the gradient estimate, but can sometimes benefit the training process by providing a form of implicit regularization, by helping the optimization escape local minima in the loss function (Wilson and Martinez 2003). It also allows for more frequent updates, potentially leading to faster convergence. The learning rate determines the step size taken in the direction of the negative gradient during each update. A high learning rate leads to larger steps in the parameter space, potentially enabling faster convergence, but at the risk of overshooting the optimal parameters and causing instability in the learning process. Conversely, a lower learning rate allows for more cautious steps, which may increase the likelihood of reaching the optimal parameters, but could also slow down the training process (Ian Goodfellow et al. 2016, Ch. 8). The training process will often consist of passing through the entirety of the training data multiple times. One pass through the training data is called an epoch. Determining a suitable set of hyperparameters (learning rate, batch size, number of epochs) is crucial to model performance.

2.3.5 Regularization

Regularization techniques are essential for preventing overfitting and improving the generalization capabilities of neural networks. Overfitting occurs when a model captures noise in the training data, resulting in poor performance on unseen data. The bias-variance trade-off describes the trade-off between a model's dependency on the training data (high variance), and its bias towards the initial assumptions of the model. More complex models generally lead to higher variance. In the case of polynomial regression, assuming a linear relationship between two variables which in reality has a more complex relationship will give a bias error towards the false assumption of linearity. On the other hand, if one assumes a high degree polynomial for a relationship between two variables which in realty is linear, the remaining degrees of freedom will be fitted to the potential noise in the data, making the regression highly dependent on the specific training data. The bias-variance trade-off can be stated as following (Trevor Hastie et al. 2009, p. 223). Assume the target phenotype, y, is a function of the SNP data, $\mathbf{X}$, through the relation $y = f(\mathbf{X}) + \varepsilon$, where $E(\varepsilon) = 0$ and $Var(\varepsilon) = \sigma_\varepsilon^2$. The squared error of a prediction based on an input SNP vector x_0 using the fitted model $\hat{f}$, which approximates f, is then

$$\begin{aligned}
\text{Squared Error} &= E[(y - \hat{f})^2 | x_0] \\
&= \sigma_\varepsilon^2 + [E\hat{f}(x_0) - f(x_0)]^2 + E[\hat{f}(x_0) - E\hat{f}(x_0)]^2 \\
&= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance} \ .
\end{aligned}$$

Regularization techniques are particularly important for data where the number of features exceed the number of observations, which is often the case in genomic data. Two common regularization techniques are L1 or L2 regularization, which adds a penalty term to the loss function proportional to the sum of the absolute values or squares of the model weights

(Ian Goodfellow et al. 2016, Ch. 7). This penalty encourages the model to learn simpler, less complex functions with. Other techniques include early stopping (Ian Goodfellow et al. 2016, Ch. 8.1.2), which halts training when the validation error starts to increase, and batch normalization (Ioffe and Szegedy 2015), which normalizes the inputs to each layer to maintain a stable distribution.

Another widely used regularization technique is dropout. Dropout is a method specific to neural networks which randomly deactivates a subset of nodes during training and it will be the most prominent regularization method in this thesis. By deactivating a subset of nodes with probability p, the model is forced to rely on a broader range of features and reduces the dependence on individual nodes, promoting a more robust learning process (Srivastava et al. 2014). Dropout resembles an ensemble method, where predictions are averaged over multiple models. Creating an ensemble of models is a very successful method that usually performs better than better than a single model. However, for neural networks this would become computationally expensive since many separate networks must be trained. Dropout addresses this by letting the ensemble of networks be thinned versions of the total network, allowing weight sharing between each thinned network. During training with a dropout layer with rate p, the output from each node is scaled by $\frac{1}{p}$ to compensate for deactivated nodes. The dropout is deactivated when predicting on new data and the weights are therefore not scaled. This results in an approximate averaging over all the sampled thinned networks.

Libraries such as Keras (Chollet et al. 2015), are available for fast and easy training of neural networks with multiple different architectures and regularization techniques. These tools enable users to develop networks without a deep understanding of the underlying algorithms or mathematical principles. However, model performance often depends on choosing the right network architecture and hyperparameters.

2.4 Neural networks for genomic prediction

Neural networks offer several potential benefits over linear models for genomic prediction tasks. The ability of neural networks to capture complex interactions and handle high dimensional data, are two of these benefits. However, with its ability to capture complex interactions comes the problem of low interpretability, which is why neural networks are often referred to as black box models.

2.4.1 Capturing interactions

Genomic data is often characterized by complex interactions between genetic markers, such as epistasis (non-additive interactions between genes) and dominance effects. These interactions can significantly influence the phenotype, and traditional linear models may not effectively capture these non-additive relationships. Neural network models, with their ability to learn hierarchical representations and capture intricate relationships, are inherently better suited for modeling such interactions. This complexity could lead to improved prediction accuracy in comparison to traditional linear models.

However, neural networks have yet to consistently outperform linear models. In Ubbens et al. (2021) the authors suggests that the underwhelming results of neural networks in the field of genomics can, at least in part, be attributed to the effect of shortcut learning. Shortcut learning is a problem encountered not just in the field of machine learning,

but many other types of leaning systems, including humans Geirhos et al. (2020). The phenomenon appears when a learning system is able to perform well on a task by adopting an easier decision rule than the intended solution. The system might pick up on patterns in the data which are highly predictive of what is to be modeled, but which might not generalize to other cases. This might come from poor training data or simply be a feature of the problem. Consider a neural network doing image classification of animals. During training the network is presented with the typical pictures of cows and achieves good performance. If the cows in the training data always appears on a grassy landscape you might have ended up with a network that has taken a shortcut and whenever it is looking for a cow, it is really looking for the color green and the contours of grass. A model which actually detected cows would clearly generalize better.

In Ubbens et al. (2021) the authors shows a case where a neural network that is given access to the value of the genetic markers directly performs no better than a network only given access to locations of matches between markers for pairs of individuals, indicating that the network mainly relies on a measure of relatedness when performing its predictions. This gives reason to believe that the specific marker values of an individual are not essential to the networks when performing predictions, but that the networks are rather modelling something akin to a relatedness matrix, as used in animal models. The relatedness between individuals is therefore used as shortcut, instead of modeling the actual marker effects. This idea is tested in Nazzicari and Biscarini (2022), where a neural network is trained on a tensor (high-dimensional array) consisting of stacked relatedness matrices(additive, dominance, and epistasis kinship matrices) to predict nine simulated phenotypes with different gene actions. The neural networks were found to perform worse than the benchmark linear model on all phenotypes.

Montesinos-López et al. (2021) gives a overview of the performance of neural networks in genomic selection. The authors noticed no significant differences in prediction performance between conventional models and neural network models, as 11 out of 23 studied papers reported neural networks having better prediction performance. Nevertheless, most of the studies where neural networks outperformed traditional genomic selection models used variations of CNNs, showing their promise for genomic prediction.

In Montesinos-López et al. (2018) neural networks shows promise for capturing $G \times E$ interactions when the environment is unknown. MLPs were trained on nine different data sets and compared to two different linear models. Only one of the linear models contained an explicit term for the $G \times E$ interactions. The neural netwoks were superior to the linear model without the $G \times E$ term in six out of nine data sets. This might indicate that neural network models learn better representations than parametrized models when the parametrized model is not correct (the significant $G \times E$ term was not included). When a $G \times E$ was included in the linear model however, the neural networks were outperformed by linear model on eight of the nine data sets.

2.4.2 High-dimensional data ($n \ll p$)

Genomic prediction tasks typically involve data sets where the number of features (p) is significantly larger than the number of samples (n). This high-dimensional nature of the problem can lead to overfitting and poor generalization. Neural network models, especially when combined with appropriate regularization techniques, have demonstrated the ability to cope with high-dimensional data and can potentially provide accurate and robust predictions. It is often the case that the number of parameters in a neural network

model exceeds the number of data points. This is referred to as overparametrization (Zhou et al. 2020) and takes place after the interpolation threshold in deep double descent (Nakkiran et al. 2019). Overparametrized deep neural networks often perform well on unseen test data, even though it according to classical statistical principles should be overfitting to the training data.

In genomic data, this discrepancy between features and samples is often too large to be passed directly to a neural network. We therefore seek to reduce the number of SNPs before applying a neural network to the SNP vector. The paper Pérez-Enciso and Zingaretti (2019) touches on this feature (SNP) selection. They highlight a limitation in dealing with very large genomic datasets, stating, "in our experience, very large genomic data sets, such as over 100k SNPs, cannot be effectively handled due to the enormous number of parameters to be estimated" (Pérez-Enciso and Zingaretti 2019, p. 12).

2.4.3 Interpretability

When developing models for understanding the genome, depending on the purpose, interpretability of the model might be of an importance. In Genome Wide Assosciation Studies (GWAS), the purpose is often to find specific genetic variants that causes some disease or influences a phenotype. The statistical model employed then has to provide important statistics to evaluate questions of such causal relationships. Understanding which variants influences a certain phenotype might be of an importance in genomic selection, but in many applications, such as predicting complex traits in livestock or crops, the priority might be to achieve the best possible accuracy. Such use cases is were the less interpretable "black box" models are best suited. The trade-off between interpretability and accuracy is an inherent challenge to machine learning and complex data. As models increase in complexity, they become less easy to interpret. This happens because an increase in complexity involves adding more parameters, making inference about the effect of any given variable harder. Conversely, simple linear regression models are easy to interpret as the model gives each variable an explicit effect size. However, much work is being done in the field of explainable machine learning (Christoph Molnar 2022), which might make these models viable for tasks requiring more interpretability as well. Multiple approaches for extracting feature importance from neural networks have been developed (Sheehan and Song 2016).

2.5 One-hot encoding

One-hot encoding was explored as an alternative to the default encoding of the SNPs. The way the data is encoded in machine learning and statistical models can greatly impact the resulting performance, as it affects possible interactions and representations that can be learned. The default encoding of the SNPs as the integer values 0, 1 and 2, may be sub-optimal in important ways.

Consider a very simple network with no hidden layers, so that the output is simply a set of weights multiplied by the SNP vector in its default representation. If the homozygote (coded as 2) genotype at a given location is given a weight w_i, its effect on the output will be $2 \cdot w_i$. This will then imply that the heterozygote genotype (coded as 1) at this location will have half the impact on the output, as it contributes by $1 \cdot w_i$. This is also the case for marker-based regression models. The implicit assumptions of the default encoding might potentially negatively impact the performance of the neural network models. The

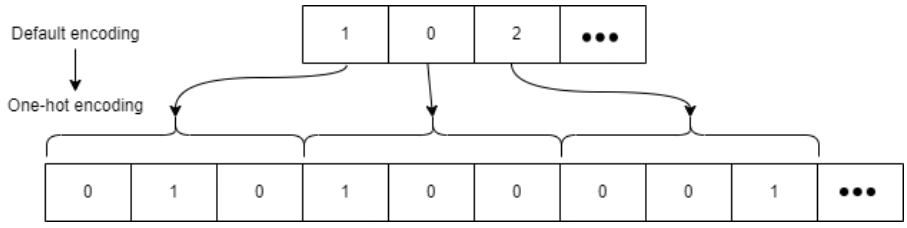


Figure 4: Illustration on how the one-hot encoding procedure transforms the original SNP vector (above), where the integers 0, 1 and 2 codes for the wild type, heterozygote and homozygote genotypes respectively, to the one-hot encoded vector (below). The mapping used is: 0, $[1, 0, 0]$; 1, $[0, 1, 0]$; 2, $[0, 0, 1]$.

output in this case is a prediction of the phenotype, and we know that there are different dominance effects which is not possible to represent in this example network using the default representation. The default representation would only be effective at capturing incomplete dominance and additive effects, as the heterozygote then is an intermediate between the wild type and homozygote. In deeper neural networks this might not be the case. Deeper neural networks can, in theory, form representations of incomplete dominance effects from default encoded SNPs, but it may be hard to learn. One-hot encoding might lend itself more easily to forming representations of these effects.

One-hot encoding transforms a column containing a feature of l categories into l columns, where all entries are zero, except for the column coding for the category observed. The procedure is illustrated in Figure 4. The one-hot encoding procedure thus expands the SNP vector by a factor of three. The increase in the dimension of the input is a potential downside of one-hot encoding, as we are already dealing with the challenge of having more features than observations. An alternative to one-hot encoding in dummy encoding, which is often used in statistical models. Dummy encoding expands the feature to $l - 1$ dummy variables by omitting the reference category. The effects of the included categories are then relative to the reference category. This would mitigate the severeness of the increase in number of features, but is less recommended for neural networks and is therefore not tested in this thesis. One-hot encoding let's each genotype have an effect independent of the effects of the alternative genotypes, removing the assumption of additive effects.

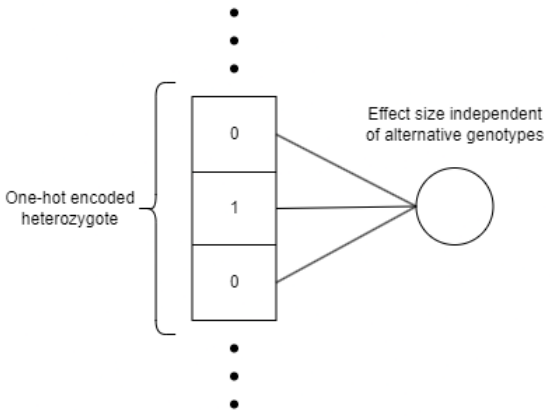


Figure 5: Illustration of a locally connected layer with one-hot encoded SNPs as input. The locally connected layer is flexible enough to accommodate different effects for each genotype, while avoiding a large increase in parameters.

The combination of a locally connected layer with one-hot encoded SNPs is a reoccurring

design in this thesis. Consider a locally connected layer with a single filter and a window width of three, as illustrated in Figure 5. If the input to this layer is one-hot encoded SNPs, then every genotype is given a separate effect size. A standard fully connected layer would give separate effects to all genotypes, but would also include interactions within and between all genotypes. LCNNs might therefore be an efficient way of handling the higher dimension of the input caused by one-hot encoding.

3 Methodology

3.1 Data description

Data for the analysis is provided through a project by the Center for Biodiversity Dynamics (CBD) at NTNU where measurements of an insular house sparrow meta-population in the Helgeland region in Norway have been made annually since 1993 (Lundregan et al. 2018). The meta-population consists of sixteen islands with different environmental conditions, habitat types and population sizes (Muff et al. 2019). There are two data sets considered in the analysis of this thesis, both from this population in Helgeland. These two data sets differ in the number of samples and the SNPs genotyped. The data set with the largest number of samples will be referred to the large data set, while the one with the smaller samples is referred to as the small data set. Note that the small data set contains more genotyped SNPs than the large data set. The small data set contains 182 854 SNPs from 3032 unique individuals. The large data set contains a larger set of 15737 measurements from 6092 unique individuals with 66018 genotyped SNPs. Multiple phenotypes are recorded for each sample. Relevant for this thesis are the phenotypes body mass and tarsus length. The large data set contains 5986 individuals with recorded tarsus length and 6046 with recorded body mass. The small data set contains 1918 unique individuals with recorded body mass and SNPs, which was the only phenotype modeled from the small data set.

3.2 Visualizing high-dimensional data

As a stage in the exploratory data analysis, principle component analysis (PCA) and t-Distributed Stochastic Neighbourhood Embedding (t-SNE) (Van Der Maaten and Hinton 2008) was applied to the SNP data. Both methods were applied using the scikit-learn library in Python (Pedregosa et al. 2011). These two methods are used to investigate to what extent the SNP data shows seperation by hatch island of the samples, which again reflects the presence of the environmental variables in the genetic data. PCA is the most used method for visualizing high-dimensional data, especially in the biology literature. t-SNE has seen more use in the machine learning literature for tasks such as visualizing the learned word embedding of neural networks, although it has also been used on SNP data (Platzer 2013).

t-SNE is a statistical method used for the task of high-dimensional data visualization. Unlike many dimensionality reduction techniques such as PCA, t-SNE is particularly good at maintaining local structures within the data, making it an excellent tool for uncovering clusters or groups. It achieves this by transforming high-dimensional distances between data points into probability representations of similarities. Only the basics of PCA is described in this section, as it is a widely applied and described method. t-SNE is given a more detailed description.

3.2.1 Principle Componen Analysis (PCA)

For data of n dimensions, PCA projects the data onto n new orthogonal axis. The principal components are the eigenvectors of the data's covariance matrix, and the amount of

variance captured by each component is the corresponding eigenvalue. The first principle component is the axis which maximizes the variance of the data on one axis. The second principle component then has the second most variance, etc. Investigating the cumulative variance explained by the components 1 through k is a way to determine the number of principle components to include (k), as for certain purposes we can drop the components after which there is less variance explained. However, there is not always a clear point at which the variance explained drops of. As the procedure returns data in n dimensions, it is not inherently a dimensionality reduction technique, but might serve as one for visualizations, as the first k components might be sufficient to show certain properties of the data. Before applying PCA it is essential to mean and center the data, to avoid bias towards certain features.

3.2.2 t-Distributed Stochastic Neighbour Embedding (t-SNE)

t-SNE can give a projection into any number of dimensions less that the dimension of the original data. It differs from PCA in that the number of dimension is a parameter chosen by the user before the algorithm is carried out. Following is a description of t-SNE for a projection into two dimensions, including some important mathematical concepts and how the resulting plot can be interpreted. A full walk-through of the mathematical details is beyond the scope of this thesis.

Let $\mathbf{x_i} = (x_1, x_2, ..., x_p)$ be one data point in the original p-dimensional data. Through t-SNE, this point is projected into $\mathbf{y_i} = (y_1, y_2)$ in two dimensions. For each point $\mathbf{x_i}$, we define a Gaussian distribution centered at $\mathbf{x_i}$. The conditional probability $p_{j|i}$ is a measure of the similarity of $\mathbf{x_j}$ to $\mathbf{x_i}$, defined as

$$p_{j|i} = \frac{\exp(-\|\mathbf{x_i} - \mathbf{x_j}\|^2/2\sigma_i^2)}{\Sigma_{k\neq i}\exp(-\|\mathbf{x_i} - \mathbf{x_k}\|^2/2\sigma_i^2)},$$

where σ_i is the variance of the distribution centered at $\mathbf{x_i}$, given more attention later. For all points i, we set $p_{i,i} = 0$, as only pairwise comparisons are of interest. The distance metric, $\|\cdot\|$, is chosen by the user. The interpretation of $p_{j|i}$ in the words of VanRaden (2020) is: "The similarity of data point x_j to data point x_i is the conditional probability, $p_{j|i}$, that x_i would pick x_j as its neighbor". A similar distribution can be defined for the low-dimensional points. Let $q_{j|i}$ be the conditional probability of $\mathbf{y_j}$ given the distribution centered at $\mathbf{y_i}$, but in the case of t-SNE, the distribution is chosen to be a student t-distribution and not a Gaussian (as in stochastic neighbourhood embedding (Hinton and Roweis 2002)). Setting the variances of $\mathbf{q}_{j|i}$ to $\frac{1}{\sqrt{2}}$ results in,

$$q_{j|i} = \frac{\exp(-\|\mathbf{y_i} - \mathbf{y_j}\|^2)}{\Sigma_{k\neq i}\exp(-\|\mathbf{y_i} - \mathbf{y_k}\|^2},$$

If the mapping of $\mathbf{x_i}$ into $\mathbf{y_j}$ is good, then the two conditional distributions $q_{j|i}$ and $p_{j|i}$ will be similar. The natural measure for this similarity is the Kullback-Leibler divergence. In the case of t-SNE, the cost function (C) is defined as the Kullback-Leibler divergence of the two joint probability distributions P and Q. The cost function is then

$$C = KL(P|Q) = \Sigma_i\Sigma_j p_{ij}\log\frac{p_{ij}}{q_{ij}},$$

which can be minimized by gradient descent. The details of p_{ij} and q_{ij} can be found in (Van Der Maaten and Hinton 2008). Finally, the variance of the distributions in the high dimensional space, σ^2, is set such that P_i has a perplexity defined by the user. Essentially, the so-called perplexity is the effective number of neighbours considered. For larger perplexity values, the position of a point in the low dimensional space is sensitive to a larger number of neighbours. Typical values for the perplexity falls between 5 and 50. In addition to varying the perplexity parameter, users may choose to use another metric for the high dimensional space than the Euclidean norm, such as the cosine distance.

The Euclidean norm is often the natural choice, but it is worth investigating its relevance for SNP data and how it relates to the amount of shared genetic material between individuals. Because of the SNPs being encoded as 0, 1 or 2, if two individuals have values 0 and 2 at the same location, this contributes more to the Euclidean distance than if the values were 0 and 1. This means that even though the amount of shared SNPs may be the same, the Euclidean distance will be different. Recall that the genotypes aa, Aa and AA are coded as 0, 1 and 2. If two samples differ in a location by having 0 and a 1 versus 0 and a 2, the latter case tells us that the two individuals does not share any alleles. As the Euclidean metric gives a greater distance to the latter case, it might reflect the relatedness of individuals better than just the amount of similar SNPs. The distance measure which only considers the number of identical SNPs would be the Hamming distance, which simply counts the locations at which the two SNP vectors differs. In this thesis, t-SNE is carried out for the Euclidean, cosine and Hamming distance, and with changing perplexity.

t-SNE excels at revealing local patterns and clusters, but it does not always maintain the global structure of the data. That is, the distances between clusters in the t-SNE plot might not reflect their actual relationships in the high-dimensional space. The same goes for the size of a cluster. A bigger cluster does not necessarily reflect a greater genetic diversity. The overlap of clusters might however indicate a greater genetic flow between groups or shared ancestry. Another important thing to note when using t-SNE is that the final representation might be highly dependent on the hyperparameters chosen, especially the perplexity parameter, but also the number of iterations gradient descent performed. It is therefore a good idea to experiment with different hyperparameters. These are some of the downsides of t-SNE when compared to PCA. While PCA operates by quite simple principles, making it easy to interpret, the non-deterministic and more complex behaviour of t-SNE makes it prone to misinterpretations. These pitfalls are visualized and explained in greater detail in Wattenberg et al. (2016).

PCA and t-SNE was in this thesis used to visually explore the SNP data in relation to the hatch island of the samples. Insights into the presence of genetic groups within the data can inform our model training procedure. High levels of clustering by hatch island might suggest that the neural network models can infer the hatch island of the sample from the SNPs, and thereby the environment. It also suggests that the models may become overfitted to dominant clusters and fail to generalize well to smaller clusters. Islands with less than ten observations were removed from this analysis. Very distinct clustering by of an hatch island from the rest of the metapopulation might warrant an investigation into the accuracy of the predictive models on test data from this island (introduced in Section 3.9).

3.3 Adjusted phenotype: Accounting for non-genetic effects

Quantitative traits often vary depending on age, sex, and environmental effects. Traditional linear mixed models can accommodate these factors by including fixed and random effects in the model. Most of the neural networks in this thesis will predict body mass solely based on SNPs. To ensure that the neural networks focus on capturing the genetic variation and not confounding effects, a linear mixed model (LMM) is fitted to the phenotypic data prior to performing genomic prediction with neural networks (e.g Ashraf et al. 2022). This LMM can be stated as

$$\begin{aligned} y_{i,j} = \mu + \beta_1 \cdot \text{sex}_i + \beta_2 \cdot \text{FGRM}_i + \beta_3 \cdot \text{month}_{i,j} + \beta_4 \cdot \text{age}_i \\ + b_{\text{island\_current},\, i} + b_{\text{hatchyear}\ ,i} + b_{\text{ring\_number},\, i} + \varepsilon_{i,j}\ , \end{aligned} \tag{7}$$

where $y_{i,j}$ is the j-th observation of the i-th individual's body mass. The population mean is denoted by μ, and the the fixed effects sex, FGRM (the individual's genomic inbreeding coefficient), month, and age are included, which have corresponding regression coefficients β_1, β_2, β_3, and β_4. The model also incorporates the random effects $\text{b}_{\text{island\_current}}$, $\text{b}_{\text{hatchyear}}$, and b_{ringnr}. Each of these random variables is assumed to be independently distributed, following a normal distribution with a mean of zero and an estimated variance. Finally, $\varepsilon_{i,j}$ is the residual term in the model for observation j of individual i. The new adjusted phenotype, y_i^* for individual i is then defined as,

$$y_i^* = b_{\text{ring\_number},\, i}\ . \tag{8}$$

The ring number effect is defined as the new, adjusted, phenotype. This new adjusted phenotype is then used as the response variable in the neural networks in Section 3.6. Removing variation in the phenotype due to environmental variables leaves a greater proportion of genetic variation. The R code used to define the model and retrieve the adjusted body mass is available in Appendix E.

3.4 Genomic Best Linear Unbiased Predictor

The results of the neural networks trained in this thesis would not be informative without a comparison with one of the widely used statistical methods for genomic prediction. To this end, the results of a Bayesian version of the Genomic Best Linear Unbiased Predictor(GBLUP) model was used as a benchmark when evaluating the viability of the neural network models.

The GBLUP animal model was fitted using the integrated nested Laplace approximations (INLA) approach to fitting Bayesian models (Rue et al. 2009), using the R-INLA package in R version 4.3.2. INLA is used for Bayesian inference in models belonging to the latent Gaussian models class, often having superior performance compared to Markov Chain Monte Carlo approaches. The model was in this case a linear mixed model, with both genetic and non-genetic effects. GBLUP was trained on both body mass and tarsus length. For comparisons with the neural networks predicting the adjusted phenotype (Section 3.3), only the estimated breeding values are extracted. The full predictor of GBLUP serves as comparison for the neural network predicting the raw phenotypes body mass and tarsus, using both genetic and non-genetic variables. (later introduced in 3.7).

The GBLUP model contains the same variables as in Equation 7 , but with the addition of an additive genetic term. Hatch island and the ID effect were again modeled as random effects with Gaussian priors. Finally, a more complex random effect for the genetic effect is included using the relatedness matrix. For the random additive genetic effect, the relatedness matrix defines the covariance structure. In the case of INLA, the inverse of the relatedness matrix was given as the precision. Individuals that are closely related will then have a high covariance for the genetic effect. The run-time of the GBLUP model is highly dependent on the dimensions of the relatedness matrix. As the relatedness matrix is $n \times n$, including more samples can become very computationally expensive. GBLUP was therefore only applied to the small data set, since applying it to the large data set was not feasible. R code for defining the model in INLA and retrieving the breeding value and full predictor is available in Appendix G.

3.5 Feature selection: Pre-selecting SNPs

Genetic data is often characterized by a large number of features (SNPs) compared to the number of samples. This might be caused by financial limitations, as sequencing the genes of more individuals is costly, or the number of individuals from the relevant populations might be limited. By reducing the dimensionality of the data, we can potentially improve model performance, reduce computational complexity, and facilitate the identification of significant SNP-phenotype relationships.

Selecting a subset of the SNPs before training the neural networks was the primary way of how we are dealing with high-dimensional nature of the SNP data here. Three different procedures will be used to select a subset of SNPs. Two of them will be based on the Spearman correlation of the SNP with the response. Spearmans correlation is very much like the more commonly used Pearson correlation, but does not assume a linear relationship between the two variables (Wilcox 2001).

1. **"Random"** selects SNPs with uniform probability. While it is not expected to yield particularly strong results (Pérez-Enciso and Zingaretti 2019), it provides a reference point for assessing the effectiveness of two other approaches.
2. **"Most_correlated"** selects the SNPs by the absolute value of their correlation with the response. This method aims to capture the SNPs with the largest individual effects on the phenotype. By focusing on the strongest correlations, we can potentially improve model performance. However, this approach may not be effective at selecting SNPs with important interactions, as it only considers one SNP at a time. In addition, due to linkage equilibrium, a highly influential QTL might cause multiple SNPs to be included that all correlate with the phenotype due to this QTL. This motivates the final approach for pre-selecting SNPs.
3. **"Best_unif"** selects the p SNPs that are most correlated from with the phenotype from p uniformly sized non-overlapping windows, measured in base pairs. A single marker is selected from each of these windows. This is intended to counter the effect of a highly influential QTL causing multiple SNPs to be included, which would contain the same information and cause multicollinearity. This might also facilitate the effective application of a convolution filter to the data, as it preservers some notion of spatial relatedness between the features as suggested in Bellot et al. (2018) and Pérez-Enciso and Zingaretti (2019). Code for the Best_unif selection scheme is available in Appendix F.

The Best_unif method used in this thesis defines an extreme of this type of SNP selection, in that no window contribute with more than one SNP. A method which allows multiple SNPs from the same window, or which uses overlapping windows, would tend to choose a set of SNPs with a grater overlap with the SNPs selected by the Most_Correlated approach. These nuances of the SNP selection are not investigated in this thesis, but could lead to a better SNP selection method. These methods are applied to the training data, meaning that only the correlation of the SNPs with the response in the training data is used to select SNPs. The intersection of the two sets chosen by Best_unif and Most_correlated is shown in Appendix D.

Similar methods of pre-selecting SNPs have been tested in the literature. A summary is given in Pérez-Enciso and Zingaretti (2019). The authors recommends choosing SNPs only based on their individual p-value, and restrictions on this criterion, such as minimum distance between selected SNPs was not found helpful. Although selecting solely based on the SNPs correlation with the phenotype might not capture epistatic effects, the authors argue that the individual additive effects will capture some fraction of the epistatic effects, and the procedure will therefore indirectly include genes with non-additive behaviours.

3.6 Neural networks predicting the adjusted body mass

The architectures of models fitted on the small and large data set are presented in Table 1 and Table 2 respectively. The MLPs trained on the large data set all had a dropout rate of 0.5 applied to the input layer. The models trained on the adjusted body mass (Section 3.3) are abbreviated as MLP_, LCNN_ and CNN_ by the architecture type, multi layered perceptron, locally connected neural network or convolutional neural network, followed by an index. Models with indexes less than ten are trained on the small data set. Models with indexes larger than ten are trained on the large data set. All neural networks was built using Keras in Python 3.10.9.

Before training, all features and phenotypes were standardized and centered, as is good practice for neural network modeling. The exception was in the case of one-hot encoded SNPs, which was kept binary. The only regularization technique in the final models was dropout, although L1 and L2 were tested in initial model exploration. Dropout rates ranged between 0.3 and 0.5. All MLPs trained on the large data set had dropout applied to the input layer, effectively dropping features randomly. This is an unusual placing of dropout. The idea is to create a model that allows the inclusion of a greater number of SNPs, under the assumption that many of them will be redundant. The redundancy will limit the loss of information by dropping features. For the LCNNs and the CNNs, the dropout is only applied after the locally connected or convolutional layer. All models were trained using a learning schedule, which means that the learning rate at each bathch is not kept constant, but changes for each batch. The specific schedule used was cosine decay, where the learning rate is initialized at a chosen starting value before decaying, following a cosine function, to a final learning rate which is constant until training is done. The decay happens over a number of batches set by the user. Typical values for the cosine decay used in this study were a decay from 0.01 to 0.001 over the span of 200 batches. Early stopping was used in the training on all modes. Early stopping monitored a validation set (10% of training data), and terminated training if no improvement in the MSE was seen over ten epochs. All activation functions were chosen to be ReLu, except for the output layer, which had alinear activation function.

Model name	**SNP_selection_P, encoding**	**Layers**
MLP_random	Random_10k, default	Dense (80), Dropout (0.3), Dense (96)
MLP_0	Most_correlated_1k, default	Dense (112), Dropout (0.35), Dense (80)
MLP_1	Most_correlated_1k, one-hot	Dense (112), Dropout (0.35), Dense (80)
MLP_2	Most_correlated_10k, default	Dense (80), Dropout (0.3), Dense (96)
CNN_0	Most_correlated_1k, default	Conv1D(4,6,1), Dropout (0.3), Dense (96), Dropout (0.3), Dense (96)
CNN_1	Most_correlated_1k, one-hot	Conv1D(4, 18 ,3), Dropout (0.3), Dense (96), Dropout (0.3), Dense (96)
CNN_2	Best_unif_1k, one-hot	Conv1D(4,6,1), Dropout (0.3), Dense (96), Dropout (0.3), Dense (96,)
CNN_3	Most_correlated_10k, default	Conv1D(12,6,2), Dropout (0.4), Dense (54), Dropout (0.3), Dense (54)
LCNN_0	Most_correlated_1k, default	LocallyConnected1D(1,15,5), Dropout (0.4), Dense (96), Dropout (0.3), Dense (64)
LCNN_1	Most_correlated_1k, one-hot	LocallyConnected1D(2,3,3), Dropout (0.4), Dense (96), Dropout (0.3), Dense (64)
LCNN_2	Best_unif_1k, default	LocallyConnected1D(1,15,5), Dropout(0.4), Dense(128), Dropout (0.4), Dense (64)
LCNN_3	Most_correlated_10k, default	LocallyConnected1D(1,15,5), Dropout(0.4), Dense(128), Dropout (0.4), Dense (64)

Table 1: Summary of the 12 neural network models trained for the small data set, predicting the adjusted body mass (Section 3.3). The *Layers* column shows the specific layer types used in the models with their respective parameters. Layer types are stated using the naming from the Keras library: Dropout(rate), dropout regularization layer; Dense(units), fully connected layer; Conv1D(filters, kernel_size, strides), one-dimensional convolutional layer; LocallyConnected1D(filters, kernel_size, strides); one-dimensional locally connected layer. The column *SNP_selection_P, encoding* contains information on the SNP selection scheme, number of SNPs(P) and finally the encoding of those SNPs as either default (0, 1 or 2) or one-hot encoded (Section 2.5).

Model name	**SNP_selection_P, encoding**	**Layers**
MLP_10	Most_correlated_1k, default	Dropout(0.5) Dense(128), Dense (64)
MLP_11	Most_correlated_10k, default	Dropout(0.5) Dense(64), Dense (64)
MLP_12	Most_correlated_10k, one-hot	Dropout(0.5) Dense(64), Dense (64)
MLP_13	Most_correlated_20k, one-hot	Dropout(0.5) Dense(64), Dense (64)
CNN_10	Most_correlated_1k, default	Conv1D(24,5,2), Dense (64), Dense (64)
CNN_11	Most_correlated_1k, one-hot	Conv1D(16,5,2), Dense (64), Dense (64)
CNN_12	Most_correlated_10k, default	Conv1D(12,10,10), Dense (64), Dense (32)
CNN_13	Most_correlated_10k, one-hot	Conv1D(8,15,15), Dense (64), Dense (64)
LCNN_10	Most_correlated_1k, one-hot	LocallyConnected1D (1,3,3), Dense (64), Dense (64)
LCNN_11	Most_correlated_1k, default	LocallyConnected1D 1,15,10), Dense (64, ReLu), Dense (64)
LCNN_12	Most_correlated_10k, default	LocallyConnected1D (1, 15, 10), Dropout(0.35), Dense (64), Dropout(0.35), Dense (64)
LCNN_13	Most_correlated_10k, one-hot	LocallyConnected1D (1, 3, 3), Dropout(0.5), Dense (64), Dense (64)
LCNN_14	Most_correlated_15k, one-hot	LocallyConnected1D (1, 3, 3), Dropout(0.5), Dense (64), Dense (64)

Table 2: Summary of the 13 neural network models trained for the large data, set predicting the adjusted body mass (Section 3.3). The *Layers* column shows the specific layer types used in the models with their respective parameters. Layer types are stated using the naming from the Keras library: Dropout(rate), dropout regularization layer; Dense(units), fully connected layer; Conv1D(filters, kernel_size, strides), one-dimensional convolutional layer; LocallyConnected1D(filters, kernel_size, strides); one-dimensional locally connected layer. The column *SNP_selection_P, encoding* contains information on the SNP selection scheme, number of SNPs(P) and finally the encoding of those SNPs as either default (0, 1 or 2) or one-hot encoded (Section 2.5).

3.7 Parallel neural network: Predicting on environmental and genetic data

As described in the Section 2.1, phenotypes are affected by both genetic and environmental factors. For most of the models trained in this thesis, the environmental effects are accounted for by a linear mixed model as outlined in the Section 2.2. This section proposes an alternative approach, which aims to escape the downsides of modelling the environmental and genetic effects separately.

The environmental variables, which explains much of the phenotypic variance, might be correlated with genetic markers, causing genetic variance to accounted for in the pre-processing step described in Section 3.3. Additionally, the interactions between genetic and environmental variables, $G \times E$, cannot be efficiently captured by constructing two separate models for explaining the genetic and environmental variance. This motivates constructing a neural networks that is trained on both environmental and genetic data simultaneously, which then is capable to capture the variance due to interactions between environmental and genetic variables. This surmounts to a neural network that predicts $\mathbf{y}$ in Equation 2, and not just $\mathbf{Zu}$.

It is the highly flexible nature of neural networks that makes this approach possible. The most naive approach would be to concatenate the genetic and environmental input to form a larger input to the full network processing both types of variables without any more care.. There are multiple reasons to choose a more intricate approach to building this model. Concatenating the inputs leads to a high redundancy in the interactions between the genetic and environmental effects, leading to a model that is hard to train. Intuitively, one might also expect the relatively few non-genetic variables to "drown" in the thousands of genetic markers, when they most likely code for greater effect sizes than any single genetic marker. Choosing an architecture which assumes that the genetic and environmental effects are mostly separate, but allows for a certain degree of interaction might be more suitable. The architecture investigated in this thesis consists of two separate networks, one network with SNP data as input and one with non-genetic variables as input. The output of these two networks are then concatenated and passed to a fully connected layer before the output node, as illustrated in Figure 6. Concatenating the output from genetic and non-genetic variables at a later layer ensures that they are of more comparable dimensions, provided that the final layer of the genetic and non-genetic layers contain a similar number of nodes. This model will be referred to as a *parallel neural network*, and to the best of my knowledge, this is not a design explored in the context of genomic prediction in existing literature. This approach allows the genetic network and non-genetic network to have different architectures and different sizes. The non-genetic network might be a simple one-layer dense network, while the genetic network might be a much deeper convolutional neural network. The final hidden layer, which takes output from both the genetic and non-genetic networks allows for $G \times E$ interactions. The two separate networks can also be pre-trained before being merged to form the parallel neural network. In addition to helping in the training process, the approach of pre-training the networks separately allows the usage of data which does not have both genetic and non-genetic data. If there are individuals where the non-genetic and phenotype data is present, but not the genetic data, these observations might still be used in the pre-training of the non-genetic network.

Before training the non-genetic network, the data involved must be formatted to suitable input for a neural network. In the linear mixed model of Section 3.3 we are able to define

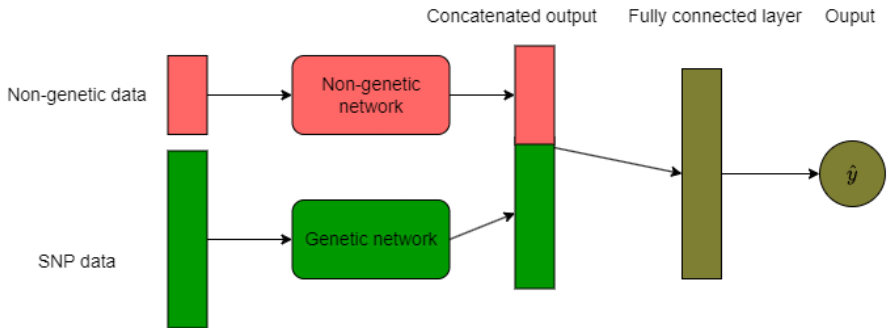


Figure 6: Illustration of the parallel network. Genetic and non-genetic data is given separately as input to two distinct networks before the output is concatenated and passes to a fully connected layer, allowing potential $G \times E$ effects to be captured. The non-genetic sections of the network (red) was pre-trained before the full network is formed by adding the final fully connected and output layer. The internal weights of the non-genetic network was also frozen, meaning that they did change during training of the full model.

random effects for variables, such as hatch island. The neural network cannot handle these variables in the same way. For the neural network with non-genetic variables as input, these was given a one-hot encoding, just as described for the SNP data in Section 2.5. Consider the variable hatch island, which is initially encoded as integers representing each observed hatch island. For the non-genetic neural network, there was one input node for each potential hatch island, all of which are set to zero except for the hatch island which is observed for that sample. The non-genetic variables used were sex, month of sampling, the estimated inbreeding coefficient, age of the bird and sampling and hatch year. The Python code for defining the parallel network is available in Appendix H.

The two parallel neural networks in this thesis predicted the phenotype body mass or tarsus length. The parallel networks are denoted as PNN_mass and PNN_tarsus for the networks predicting the body mass and tarsus length respectively. Both networks had the same architecture. The non-genetic network is consisted of two fully connected layers, both with 12 units. The genetic network resembles MLP_11, using two 64 unit layers and a dropout of with rate 0.5 applied to the input layer. All activation functions were set be ReLu, except for the output node which used a linear activation function. The non-genetic network was pre-trained on the relevant variables (then with the addition of an output node). Additionally, repeated measurements of individuals was used during this pre-training. Before the full model was constructed, the internal layers of the non-genetic network was frozen, meaning that they did not change after pre-training. The comparison for the PNN models was the full predictor of two GBLUP models including both the genetic and non-genetic variables, as described in Section 3.4.

3.8 Model assessment and loss function

The traditional measure of prediction accuracy in genomic prediction tasks is the Pearson correlation between the predicted value and the observed phenotype (Meuwissen et al. 2001). The higher the correlation coefficient, the more variance in the phenotype is explained by the genetic material. The Pearson correlation was therefore used as the metric for assessing the accuracies of all models. Note that the final evaluation metric was not the same as the loss function used during training of the networks. The mean squared error is used as the loss function for training of all models in this thesis. The Pearson correlation

coefficient and the Mean Squared Error provide complementary information about model performance. The Pearson correlation measures the linear relationship between the observed and predicted values, which reflects how well the model predictions align with the true values. However, it does not directly measure the magnitude of the errors.

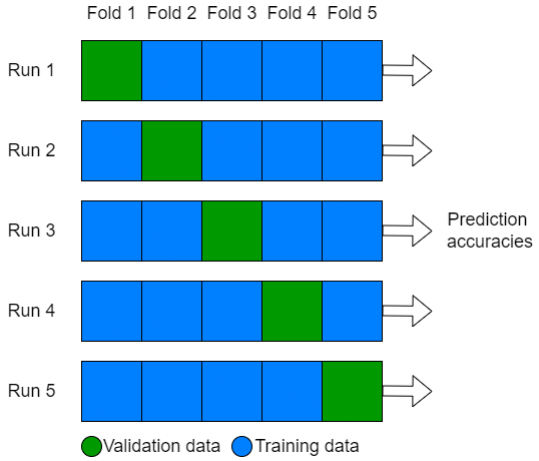


Figure 7: The five-fold cross-validation procedure illustrated. The procedure consists of five training-validation runs, where every data point will be in the validation set for one run.

The accuracy of all models in this thesis was assessed by performing five-fold cross-validation, including the GBLUP models. The data was split into five folds, meaning five distinct subsets of the total data. Each of the five folds was set as the validation set for one run each, meaning that the remaining four folds are combined to form a training set before the prediction was made on the validation set. In five-fold cross validation, the validation data is 20% of the total data in each run. The final performance measure is then the mean over the five correlations from each training and validation run. The standard deviation of the accuracies is a measure of the models variance depending on the data set. The cross validation procedure is illustrated in Figure 7. For the models trained on data including repeated samples (non-genetic part of the PNNs and GBLUP), the folds was split by the unique individuals, meaning that no individual appeared in both the training and test set.

3.9 Testing for difference in model performance by hatch island

Neural networks have not been widely applied to genomic prediction in wild populations. The genetic variation in a wild meta-population might propose additional challenges compared to domestic populations. Data containing samples from widely different distributions often causes poor performance in machine learning models. To investigate whether the neural networks performed worse on samples from certain sub-populations, the test set was separated by hatch island and the accuracy was recorded for each of the subsets. These subsets of samples from different island groups are of widely different sizes. A small set will have a larger probability of having a performance widely different from the test set as a whole that a large set, purely by chance. The differences in set sizes therefore need to be considered when assessing the significance of the differences. Hatch islands with less than 50 samples are omitted from this analysis.

An approximate permutation test was used to quantify the significance of the differences in performance by samples from different islands. The approximate permutation test differs from the ordinary permutation test in that not all possible permutations are iterated through, as there are too many. A Monte Carlo-like simulation was instead used (Nichols and Holmes 2001). The test is performed for each of the set sizes seen after splitting the test set by hatch island. Consider a hatch island with n samples in the test data forming a subset. The performance of the neural network on this subset is recorded as C_{island}. The performance on the whole test set is denoted as C_{meta_pop} Next, a 1000 new random subsets with n samples are defined. The performance of the model was recorded on each of these random subsets, forming $C_i,\ i = 1, 2, ...1000$. The statistic of interest is the difference between the performance on the subset from a specific island with the population as a whole, $T_{island} = C_{meta\_pop} - C_{island}$. The larger the value of this statistic, the worse the model performed on this island compared to the whole population. The same statistic is defined for each of the random subsets, $T_i = C_{meta\_pop} - C_i$. The p-value of the observation T_{island} is then the proportion of T_i greater than T_{island},

$$p\text{-}value = \frac{\sum_{i=1}^{1000}(T_i > T_{island})}{1000}.$$

This p-value corresponds to the null hypothesis that the performance on a hatch island group in the test data is drawn from the same distribution as a random set of that size, meaning that the model performs similarly on that hatch_island compared to the rest. The distribution formed by $\{C_i\}_{i=1}^{1000}$ was used to construct confidence intervals. For a 95% confidence interval, the lower and upper bound is given by the percentiles at 2.5% and 97.5%. The model was trained on the training set containing all hatch islands. Only the test set was split by hatch island. The test and training set split was done randomly, meaning that there was no control over the relative frequency of each hatch island. The permutation test was only carried out for the best performing neural network model (MLP_11) due to computational costs, and does not incorporate cross-validation.

4 Results

t-SNE and PCA

The plots from the PCA and t-SNE are presented in Figure 8. The data points are colored by the hatch island. Plots from t-SNE with different distance metrics and perplexity parameters are available in Appendix A. Figure 8 shows the t-SNE projection with $perplexity = 30$ and the Euclidean distance. The t-SNE projection of the SNP data shows a high degree of grouping by hatch island. A high degree of grouping is also seen in the PCA, although more components are needed distinguish the islands, compared to t-SNE. The first principle component showed especially good separation of the island colored in grey, Aldra. The first 6 PCs are available in Appendix C and the explained variances of each component in Appendix B. The distance measure used was either Euclidean, cosine or the Hamming distance. Grouping by island was similar between all distance measures and perplexity parameters. The relative position of different groupings was also similar between different parameter and metric choices, meaning that islands that appeared grouped close to each other for one configuration, tended to appear close for all other configurations. The t-SNE plots can be compared to the results from Ranke et al. (2021), where the genetic flow between the islands is investigated. Their results are visualized in Figure 9. By comparing this figure with the t-SNE plot, we can get an idea of how the clusters appearing in the t-SNE plot relates to the genetic flow between the islands.

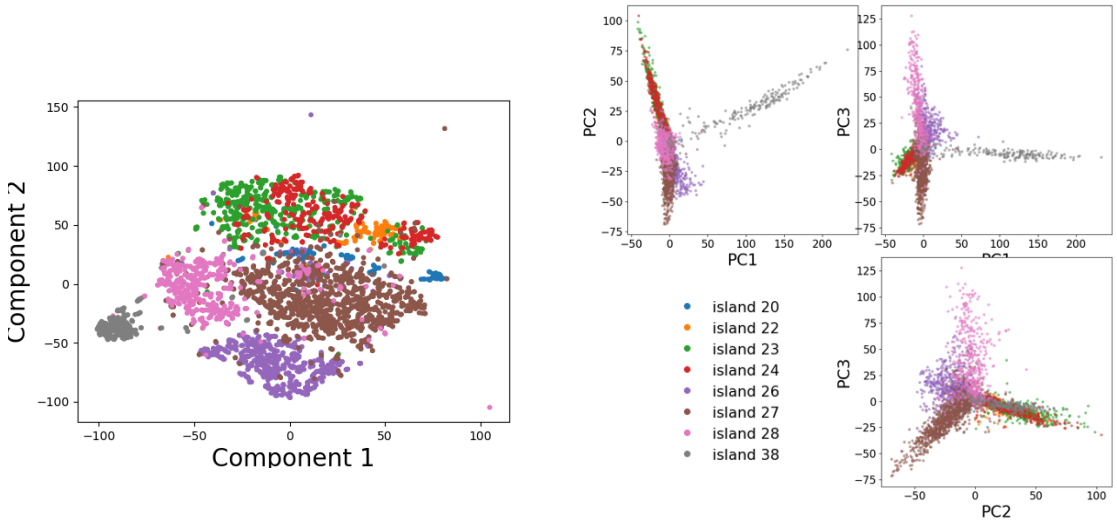


Figure 8: PCA and t-SNE on large data set. Left: t-distributed stochastic neighbor embedding (t-SNE) of the SNP data using the Euclidean distance and perplexity set to 30. Right: The first three principle components of the SNP data. Each observation is colored according to their hatch island. Islands with fewer than ten observations were left out of the analysis, leaving eight islands.

t-SNE mainly resulted in groupings of samples from the same hatch island, but without clearly distinct boundaries. Island 38 (Aldra) had the highest degree of clustering in the t-SNE plot for both data sets. Aldra consistently appeared on the periphery of the islands groupings and with little overlap with the other islands. This observation is mirrored in the PCA plot, as the first component shows high differentiation of Aldra from the rest of the population. From Figure 9 it is apparent that there is a very little genetic flow

between Aldra and the other islands.

Overlapping island groupings in the t-SNE plot did to some degree correspond to islands with greater genetic flow between them, compared to non-overlapping groupings. Island 23 and 24, corresponding to Træna and Selvær, are largely overlapping in the both the t-SNE and the PCA plot. These locations are each others closest neighbour. Figure 9 shows a high degree of genetic flow between these two islands. There was also a separation of what is referred to as "inner" and "outer" islands, colored yellow and blue respectively in Figure 9. The outer islands are colored red, green and orange in the t-SNE plot. The t-SNE plot shows a separation by inner and outer islands. The outer islands appeared largely overlapping. Additionally, the inner most islands, Aldra and Gjerøy (appearing grey and purple respectively in Figure 8), appears furthest from the outer islands. In summary, the t-SNE projection appears to recreate much of the both the local (grouping of islands) and global (relative position of groupings) structure in Figure 9 from the SNP data.

Number	Name
20	Nesøy
22	Myken
23	Træna
24	Selvær
26	Gjerøy
27	Hestmannøy
28	Indre Kvarøy
33	Onøy og Lurøy
34	Lovund
35	Sleneset
38	Aldra

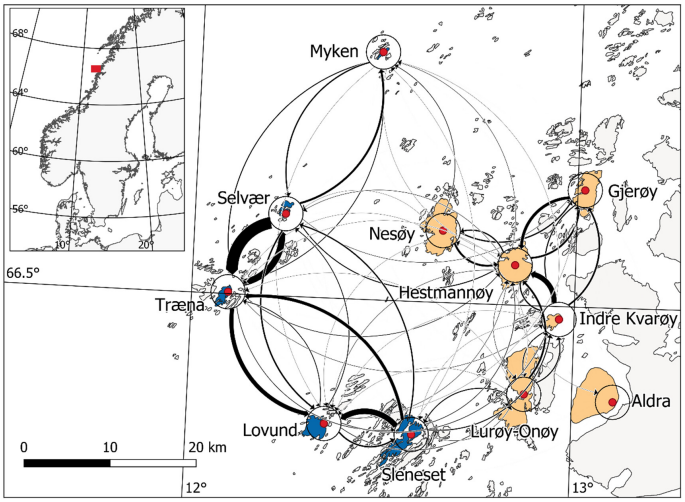


Figure 9: Islands of birth for samples in data set and genetic flow between them. Right figure is borrowed from Ranke et al. (2021). The thickness of the arrows corresponds to the amount of genetic flow (migration) between the respective islands

Neural networks

Model performances are reported by the mean and standard deviation of the Pearson correlation between the predicted and true response from the five-fold cross-validation, denoted as mean (standard deviation). The standard deviations are relatively large compared to the differences in model performances. The successful training of a network was generally characterized by slow learning (many epochs).

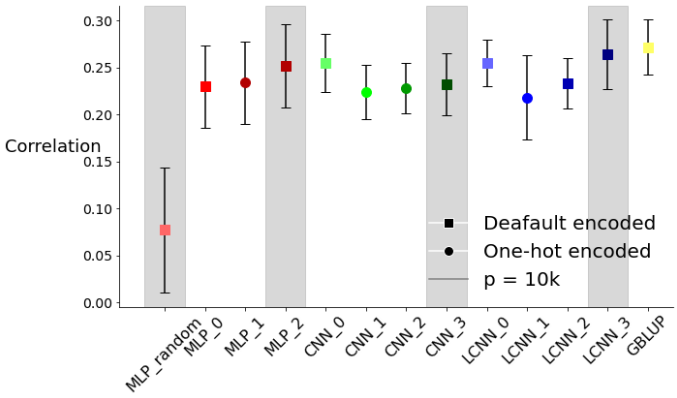
Predicting adjusted body mass on the small data set

The final architectures of the 12 models trained on the small data set ($n = 1884$) are given in Table 1 and their performance is presented in Figure 10. There was a large difference in performance between each fold in the five-fold cross validation. The MLP with 1k randomly selected SNPs performed a lot worse than all other models for any of the considered input sizes, encodings and network architectures. GBLUP, which was only

applied to the small data set, achieved a correlation of 0.272 (0.029). None of the neural network models trained on the small data set performed better than GBLUP.

The best performing neural network model trained on the small data set was LCNN_3, with a correlation of 0.264 (0.037). LCNN_3 was trained on the output from Most_correlated_10k (selecting the 10k SNPs most correlated with the adjusted phenotype) using the default encoding (SNPs encoded as 0, 1 or 2) and had a locally connected layer with one filter, a stride of 15 and window width of 5. Increasing the number of SNPs from 1k to 10k was generally positive for model performance in MLPs and LCNNs. The best CNN did however use 1k SNPs as input.

The Best_unif SNP selection method, which selects SNPs based on their correlation with the adjusted phenotype while hindering multiple SNPs being selected from the same region, did not improve the accuracy of any of the tested model architectures. One-hot encoding of the SNPs was observed to give similar or worse performance for all models when compared to the default encoding. Using 10k one-hot encoded input, resulting in the dimension of the input space being 30k, gave unstable models for all models on the small data set, and is therefore not included.



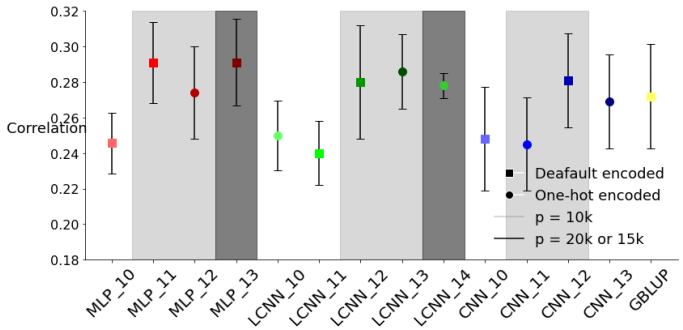
Model	Correlation (std)
MLP_random	0.077 (0.67)
MLP_0	0.230 (0.044)
MLP_1	0.234 (0.044)
MLP_2	0.252 (0.043)
CNN_0	0.255 (0.031)
CNN_1	0.224 (0.029)
CNN_2	0.228 (0.027)
CNN_3	0.232 (0.033)
LCNN_0	0.255 (0.025)
LCNN_1	0.218 (0.045)
LCNN_2	0.233 (0.027)
LCNN_3	0.264 (0.037)
GBLUP	0.272 (0.029)

Figure 10: Model accuracy on the small data set measured as the Pearson correlation coefficient between the predicted and true body mass, after adjusting for non-genetic effects as described in Section 3.3. MLP models are colored shades of red, CNNs are colored shades of green and LCNN are colored shades of blue. The GBLUP result is colored yellow. Square markers represent models with default encoded SNPs, while circular markers are models with one-hot encoded SNPs. The shaded areas indicates models with 10k SNPs as input, while the white area contains models trained on 1k SNPs. All models are trained on 1918 observations. Model architecture, input type and input size is given in Table 1.

Predicting adjusted body mass on the large data set

As the results for the smaller data set indicated that including 10k SNPs was beneficial for performance, some of the networks for the larger data set ($n = 5986$) was trained on 15k and 20k SNPs. The Best_unif selection scheme was abandoned based on the results on the smaller data set, as it did not cause any improvement and requires more computation time than the Most_correlated approach. The results from the large data set are seen in Table 11 and all the model architectures are seen in Table 2. The best performing models were

MLPs with 10k and 20k default encoded SNPs as input, both achieving a correlation of 0.291 with a standard deviation of 0.023 and 0.024 respectively. An MLP was trained on 30k default encoded SNPs, but led to poor performance and failed to converge too often to get a cross validation result, and is therefore not included in the results. All architecture types (MLP, CNN and LCNN) resulted in at least one model that outperformed GBLUP (0.272 (0.0294)), which was trained on the smaller data set.



Model	Correlation (std)
MLP_10	0.246 (0.017)
MLP_11	0.291 (0.023)
MLP_12	0.274 (0.026)
MLP_13	0.291 (0.024)
LCNN_10	0.250 (0.020)
LCNN_11	0.240 (0.018)
LCNN_12	0.280 (0.031)
LCNN_13	0.286 (0.021)
LCNN_14	0.278 (0.007)
CNN_10	0.248 (0.029)
CNN_11	0.245 (0.0262)
CNN_12	0.281 (0.0284)
CNN_13	0.269 (0.0266)

Figure 11: Model accuracy on the large data set measured as the Pearson correlation coefficient between the predicted and true body mass, after adjusting for non-genetic effects as described in Section 3.3. Model architecture, input type and input size is given in Table 1. The GBLUP result is colored yellow, but is the result is from the smaller data set. Square markers represent models with default encoded SNPs, while circular markers are models with one-hot encoded SNPs. The lightly shaded areas indicates models with 10k SNPs as input, while the darker shaded areas contains the models with more than 10k SNPs. The white regions contain models trained on 1k SNPs.

The best LCNN (LCNN_13) did, however, achieve a performance close to that of the best performing MLP with a correlation of 0.286 (0.021), using the one-hot encoded SNPs combined with a locally connected layer with windows covering three inputs (covering a single one-hot encoded SNP) and with a stride of three. By far the most consistently performing model in cross-validation was LCNN_14, which achieved a correlation of 0.278 (0.007). LCNN_15 had 15k one hot encoded SNPs as input, which is the largest input size of all the models. One hot encoding was not beneficial for the MLPs or CNNs. Common to all these were that the input was 10k or more SNPs. For the models trained on 1k SNPs there was a smaller increase in model performance on the large data set compared to the small data set, and none outperformed GBLUP. Models generally achieved a lower variation in performance on the large data set compared to the small data set.

Performance depending on hatch island

To investigate whether the prediction accuracy of the neural network models differs depending on the hatch island of the samples, we test how it differs on the MLP_11 model, the best performing neural network trained on the adjusted body mass from the large data set. The model is given the full training data (containing all hatch islands) during training, while the test data is split by hatch island. The results are shown in Figure 12

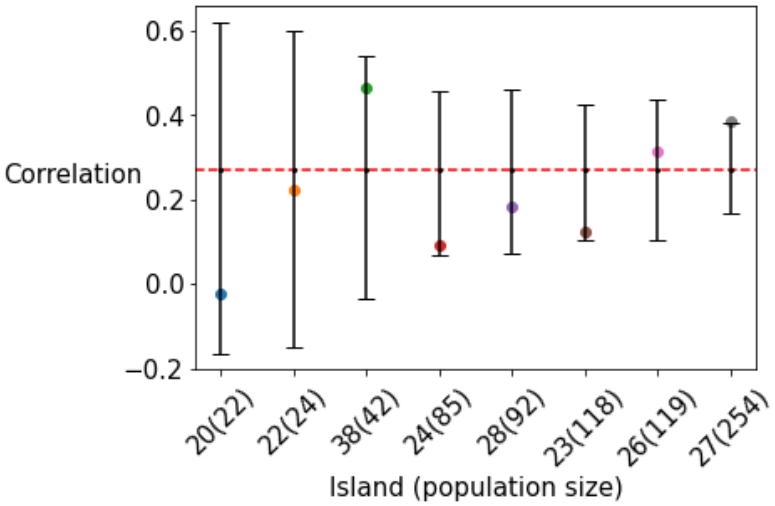


Figure 12: Performance of MLP_11, separated by hatch island. Error bars represent 95% confidence interval from an approximate permutation test. Dotted red line shows the model performance on the complete test set.

with associated 95% confidence intervals from an approximate permutation test. Island 27, Hestmannøy, was the only island with an accuracy outside this confidence interval. This is also the island with the largest number of samples in the test set with 254 samples.

Parallel network

Two neural networks were trained on the large data set using the raw phenotypes and both genetic and non-genetic variables. One model predicted the body mass (PNN_mass) and the other ont the tarsus length (PNN_tarsus). Both neural networks have the same architecture of two parallel fully connected networks processing genetic and non-genetic variable separately, before merging at the second to last layer.

PNN_mass and PNN_tarsus are compared to two GBLUP models as described in Section 3.4, namely GBLUP_mass and GBLUP_tarsus respectively. The GBLUP models are however trained on the smaller data set. The model performances are shown in Figure 13. PNN_mass achieved a correlation of 0.347 (0.0416), slightly outperforming GBLUP which achieved 0.320 (0.0366). PNN_tarsus achieved an accuracy of 0.324 (0.029), which was outperformed by GBLUP_tarsus with an accuracy of 0.385 (0.035).

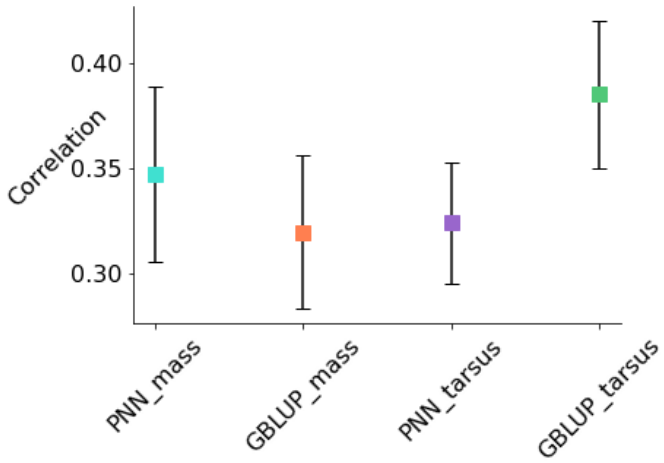


Figure 13: Performance (Pearson correlation) of the parallel neural networks PNN_mass and PNN_tarsus predicting the raw phenotypes body mass and tarsus length using both genetic and non-genetic variables from the large data set. These networks are compared to the results of GBLUP, which is trained on the small data set using the same variables. Error bars are one standard deviation, retrived from the 5-fold cross-validation.

5 Discussion and Conclusion

This study has explored several dimensions of applying neural networks to the task of genomic prediction, including pre-selection of SNPs, SNP encoding, neural network architectures and data visualization. A total of 27 neural networks and two GBLUP models were applied to the two data sets used in this thesis. Of those, 25 networks modelled an adjusted phenotype defined as the individual-specific random effect of an linear mixed model accounting for non-genetic variables, and two networks predicted the raw phenotypes body mass or tarsus length using bot SNP data and non-genetic data.

t-SNE and PCA were used as dimensionality reduction techniques with the goal of visually exploring potential genetic clustering by hatch island. The projections of the SNP data showed a high degree of grouping by the animals island of origin (Figure 8). The fact that t-SNE efficiently visualized the local structure of the data, meaning the relatedness between pairs individuals, might not be a surprise, as this is the strength of t-SNE. However, t-SNE did also create a visualization that remarkably retained the global structure of the data. Islands with a high level of genetic flow between them appeared more overlapped, and islands that are located far away from each other appeared on opposite sides in the t-SNE visualization. t-SNE tends to conserve more global structures for larger perplexity parameters, which is also the case for this data.

The grouping of the genetic data by hatch island may indicate challenges to fitting a model, when not considering these differences. Training data containing samples from distinct clusters may lead to poor generalization. A potential approach is to use pre-training on the total data set, before fine-tuning it on the specific clusters of interest. This approach is called transfer learning (Hosna et al. 2022) and is a widely used technique in the field of machine learning. Here, to test whether the neural networks performed better or worse on samples from certain islands, the test set of MLP_11 was split by hatch island. The result of this analysis was that none but Hestmannøy achieved a performance outside (above) the 95% confidence interval, which was simulated by an approximate permutation

test. Note that this analysis checked performance on eight different islands (essentially testing multiple hypotheses), requiring care when evaluating the significance of the result. Hestmannøy is also the island with the most samples in this test set, supporting the hypothesis that the neural networks performes best on the samples from the group with the largest number of samples in the training data. However, the model performance did not generally increase for increasing population size for the other islands. The best performing subset was the samples from Aldra, island 38, but the result was within the 95% confidence interval. The separation by hatch island done in this analysis might not be accurate enough to show the true differences in performance by genetic clustering and habitat differences, as there is a high degree of genetic flow between certain islands. This genetic flow causes islands to effectively be a part of the same larger sub-population. This analysis can be improved upon by controlling the number of samples from each hatch island in both the test and training set. A similar approach would be to train on all samples except those from one island, which will be the test set (essentially k-fold cross-validation for k islands).

Four types of regularization were tested during the development of the models, namely L1, L2, early stopping and dropout regularization. Dropout and early stopping proved to be efficient in handling overfitting, while weight penalty methods were discounted as the tuning process set them close to zero. Applying dropout layers was an efficient way of regularizing the neural networks. Dropout rates ranged between 0.2 and 0.5. From the limited number of models trained in this thesis, the CNNs was observed to need a lower rate of regularization, which is in line with the sparse properties of CNNs (Ian Goodfellow et al. 2016, Ch. 9). Early stopping was crucial in terminating the learning process before the network overfitted the training data. The optimal number of epochs varied widely, which meant that setting the number of epochs for training manually will likely involve either overfitting or underfitting the data. For the large data set, dropout layers were used before the first fully connected layer, meaning that weights directly associated with SNPs are set to zero. This approach proved efficient when the number of SNPs increased.

One avenue investigated by this thesis is the different SNP selection approaches, the Best_unif scheme and the more straight forward Most_correlated approach. The Most_correlated procedure selected the SNPs most correlated with the response when considering the whole SNP vector. Best_unif selected the one SNP most correlated with the response from uniformly spaced windows (measured in base pairs). The models trained on SNPs selected by the Bes_Unif_1k scheme, CNN_2 and LCNN_2, performed similarly or worse than similar models trained on SNPs selected by Most_correlated_1k. For larger selections of SNPs, the overlap between the two SNP selection methods increases. This increase in overlap would decrease the need for the Best_unif scheme. The overlap between the SNP selection methods is in Appendix D. One intention of the Best_unif approach was to decrease the amount of multicollinearity. The presence of multicollinearity is known to negatively impact the performance of many statistical models, but is not as detrimental to neural networks.

The approach of Best_unif was intended to be especially suited for CNNs, as it conserves some spatial regularity in the input. This notion of spatial consistency could be beneficial for the convolution operation as it allows the model to efficiently capture reoccurring patterns. However, these results might suggest that the spatial equivariance property of CNNs may not fully apply to the SNP vector. Rather than detecting features independent of their spatial location, when applied to genomic data, CNNs can be seen as extracting local contextual information from sequences. Interactions between neighboring genes or SNPs can significantly influence the traits they govern. Therefore, spatial invariance might

not be what makes CNNs valuable for genomic prediction tasks, but their capacity to capture local interactions in the data. The CNNs was however consistently outperformed by the LCNNs, which might suggest that the sparsity is beneficial, but that the shared weights are not.

One of the clear trends of the results was the improvement in performance when increasing the number of SNPs used as model input, up until 20k SNPs. Due to computational costs, too few models were trained with input sizes in the interval 10k to 20k SNPs to draw any conclusion on what the optimal number is.

One-hot encoding did not consistently outperform the default encoding of SNPs, especially for the small data set. On the larger data set, one-hot encoding showed some promise when combined with locally connected layers. A locally connected layer may efficiently handle the increased dimensionality of the input space caused by one-hot encoding, better than the MLPs. In the case of MLPs, one-hot encoding causes a large increase in model parameters, which may impact performance. The LCNN trained on 15k one hot encoded SNPs (LCNN_14) achieved an especially notable result. Even though it did not achieve the best performance overall, it was competitive and with a drastically lower variation between the folds in cross-validation. This result highlights LCNNs as a viable option for handling the large dimensions of the SNPs, especially when using one-hot encoding.

The neural networks predicting the raw phenotypes, using both environmental and genetic data simultaneously (PNN_mass and PNN_tarsus), outperformed GBLUP when predicting body mass, but performed worse when predicting tarsus length. As with many model performances in this study, were the uncertainties a too large to make any strong conclusions. This does however serve as a proof of concept for this network structure. The final architecture of the parallel network was informed by which models performed best when predicting the adjusted body mass and a limited number of alternative design choices were tested. This might have contributed to PNN_tarsus performing worse than GBLUP_tarsus, as the optimal architecture is known to be dependent on the phenotype. As only two models were tested using this approach, both only applying MLPs to default encoded SNPs, there is reason to expect room for improvement on the design choices made in this thesis.

The neural network models in this thesis were competitive with GBLUP when trained on the large data set. However, the implications of the comparison between the GBLUP models and neural networks trained on the large data set are limited as they are trained on different data sets. Including the comparison between GBLUP and the parallel neural network predicting the adjusted phenotype. GBLUPs computation time increases drastically with number of individuals (due to the relatedness matrix being $n \times n$), which is why it was not feasible to apply GBLUP to the large data set in this thesis. Neural networks may for this reason be a more suitable choice for large data set.

Further studies could investigate the effect of the choice of encoding on a simulated data set, allowing control of what genetic effects are most important to the phenotypic variation. This could give greater insight into whether one-hot encoding is necessary to model dominance effects, or if networks are able to represent such effects from the default encoding. What genetic effects are the most important may also affect which neural network architecture is most suitable. The SNP selection can also be better tested and understood by using an artificial data set. A less extreme version of Best_unif which allows some overlap of windows selecting SNPs is a natural method to test. Testing the ability of models to generalize between different populations could be further explored using the approximate permutation, preferably controlling what samples are in the training set, which is not done in this thesis.

Bibliography

Ashraf, Bilal et al. (Dec. 2022). 'Genomic prediction in the wild: A case study in Soay sheep'. In: *Molecular Ecology* 31.24, pp. 6541–6555. ISSN: 1365294X. DOI: 10.1111/mec.16262.

Bellot, Pau, Gustavo de los Campos and Miguel Pérez-Enciso (Nov. 2018). 'Can deep learning improve genomic prediction of complex human traits?' In: *Genetics* 210.3, pp. 809–819. ISSN: 19432631. DOI: 10.1534/genetics.118.301298.

Bérénos, Camillo et al. (July 2014). 'Estimating quantitative genetic parameters in wild populations: a comparison of pedigree and genomic approaches.' In: *Molecular ecology* 23.14, pp. 3434–51. ISSN: 1365-294X. DOI: 10.1111/mec.12827.

Budhlakoti, Neeraj et al. (Feb. 2022). *Genomic Selection: A Tool for Accelerating the Efficiency of Molecular Breeding for Development of Climate-Resilient Crops*. DOI: 10.3389/fgene.2022.832153.

Cahyawijaya, Samuel et al. (Apr. 2022). 'SNP2Vec: Scalable Self-Supervised Pre-Training for Genome-Wide Association Study'. In: URL: http://arxiv.org/abs/2204.06699.

Chollet, François et al. (2015). *Keras*.

Christoph Molnar (2022). *Interpretable Machine Learning*. 2nd ed.

Conner, Jeffrey K and Daniel L Hartl (2004). *A Primer of Ecological Genetics*. Sinauer Associates Incorporated.

Elizondo, D. and E. Fiesler (Oct. 1997). 'A Survey of Partially Connected Neural Networks'. In: *International Journal of Neural Systems* 08.05n06, pp. 535–558. ISSN: 0129-0657. DOI: 10.1142/S0129065797000513.

Geirhos, Robert et al. (Nov. 2020). 'Shortcut learning in deep neural networks'. In: *Nature Machine Intelligence* 2.11, pp. 665–673. ISSN: 25225839. DOI: 10.1038/s42256-020-00257-z.

Gianola, Daniel (July 2013). 'Priors in whole-genome regression: The Bayesian alphabet returns'. In: *Genetics* 194.3, pp. 573–596. ISSN: 00166731. DOI: 10.1534/genetics.113.151753.

Hara, Kazuyuki, Daisuke Saito and Hayaru Shouno (July 2015). 'Analysis of function of rectified linear unit used in deep learning'. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8. ISBN: 978-1-4799-1960-4. DOI: 10.1109/IJCNN.2015.7280578.

Hinton, Geoffrey and Sam Roweis (2002). 'Stochastic Neighbor Embedding'. In: *Proceedings of the 15th International Conference on Neural Information Processing Systems*, pp. 857–864.

Hornik, Kurt, Maxwell Stinchcombe and Halbert White (Jan. 1989). 'Multilayer feedforward networks are universal approximators'. In: *Neural Networks* 2.5, pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8.

Hosna, Asmaul et al. (Dec. 2022). 'Transfer learning: a friendly introduction'. In: *Journal of Big Data* 9.1. ISSN: 21961115. DOI: 10.1186/s40537-022-00652-w.

Hunter, D. C. et al. (2022). 'Using genomic prediction to detect microevolutionary change of a quantitative trait'. In: *Proceedings of the Royal Society B: Biological Sciences* 289.1974. ISSN: 14712954. DOI: 10.1098/rspb.2022.0330.

Ian Goodfellow, Aaron Courville and Yoshua Bengio (2016). *Deep learning*. MIT Press.

Ioffe, Sergey and Christian Szegedy (Feb. 2015). 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: URL: http://arxiv.org/abs/1502.03167.

Jumper, John et al. (Aug. 2021). 'Highly accurate protein structure prediction with AlphaFold'. In: *Nature* 596.7873, pp. 583–589. ISSN: 14764687. DOI: 10.1038/s41586-021-03819-2.

Khan, Asifullah et al. (Dec. 2020). 'A survey of the recent architectures of deep convolutional neural networks'. In: *Artificial Intelligence Review* 53.8, pp. 5455–5516. ISSN: 15737462. DOI: 10.1007/s10462-020-09825-6.

Kim, Sunkyu et al. (Apr. 2018). 'Mut2Vec: Distributed representation of cancerous mutations'. In: *BMC Medical Genomics* 11. ISSN: 17558794. DOI: 10.1186/s12920-018-0349-7.

Kruuk, Loeske E.B. (June 2004). *Estimating genetic parameters in natural populations using the 'animal model'*. DOI: 10.1098/rstb.2003.1437.

Lande, Russell and Robin Thompsont (1990). *Efficiency of Marker-Assisted Selection in the Improvement of Quantitative Traits*. Tech. rep.

Li, Bo et al. (July 2018). 'Genomic prediction of breeding values using a subset of SNPs identified by three machine learning methods'. In: *Frontiers in Genetics* 9.JUL. ISSN: 16648021. DOI: 10.3389/fgene.2018.00237.

Lundregan, Sarah L. et al. (Sept. 2018). 'Inferences of genetic architecture of bill morphology in house sparrow using a high-density SNP array point to a polygenic basis'. In: *Molecular Ecology* 27.17, pp. 3498–3514. ISSN: 1365294X. DOI: 10.1111/mec.14811.

Mcculloch, Warren S and Walter Pitts (1943). 'A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY'. In: *BULLETIN OF MATHEMATICAL BIOPHYSICS* 5.

Meuwissen, T H, B J Hayes and M E Goddard (Apr. 2001). 'Prediction of total genetic value using genome-wide dense marker maps.' In: *Genetics* 157.4, pp. 1819–29. ISSN: 0016-6731. DOI: 10.1093/genetics/157.4.1819.

Meuwissen, Theo, Ben Hayes and Mike Goddard (Jan. 2016). 'Genomic selection: A paradigm shift in animal breeding'. In: *Animal Frontiers* 6.1, pp. 6–14. ISSN: 21606064. DOI: 10.2527/af.2016-0002.

Montesinos-López, Abelardo et al. (Dec. 2018). 'Multi-environment genomic prediction of plant traits using deep learners with dense architecture'. In: *G3: Genes, Genomes, Genetics* 8.12, pp. 3813–3828. ISSN: 21601836. DOI: 10.1534/g3.118.200740.

Montesinos-López, Osval Antonio et al. (Dec. 2021). *A review of deep learning applications for genomic selection*. DOI: 10.1186/s12864-020-07319-x.

Muff, Stefanie et al. (Feb. 2019). 'Animal models with group-specific additive genetic variances: Extending genetic group models'. In: *Genetics Selection Evolution* 51.1. ISSN: 12979686. DOI: 10.1186/s12711-019-0449-7.

Nakkiran, Preetum et al. (Dec. 2019). 'Deep Double Descent: Where Bigger Models and More Data Hurt'. In: URL: http://arxiv.org/abs/1912.02292.

Nazzicari, Nelson and Filippo Biscarini (Nov. 2022). 'Stacked kinship CNN vs. GBLUP for genomic predictions of additive and complex continuous phenotypes'. In: *Scientific Reports* 12.1, p. 19889. ISSN: 2045-2322. DOI: 10.1038/s41598-022-24405-0. URL: https://www.nature.com/articles/s41598-022-24405-0.

Ng, Patrick (Jan. 2017). 'dna2vec: Consistent vector representations of variable-length k-mers'. In: URL: http://arxiv.org/abs/1701.06279.

Nichols, Thomas E and Andrew P Holmes (2001). 'Nonparametric Permutation Tests For Functional Neuroimaging: A Primer with Examples'. In: *Human Brain Mapp.*

OpenAI (Mar. 2023). *GPT-4 Technical Report*. Tech. rep.

Pedregosa, F. et al. (2011). 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Pérez-Enciso, Miguel and Laura M. Zingaretti (July 2019). *A guide for using deep learning for complex trait genomic prediction*. DOI: 10.3390/genes10070553.

Platzer, Alexander (Feb. 2013). 'Visualization of SNPs with t-SNE'. In: *PLoS ONE* 8.2. ISSN: 19326203. DOI: 10.1371/journal.pone.0056883.

Pook, Torsten et al. (Nov. 2020). 'Using Local Convolutional Neural Networks for Genomic Prediction'. In: *Frontiers in Genetics* 11. ISSN: 16648021. DOI: 10.3389/fgene.2020.561497.

Ranke, Peter S. et al. (Dec. 2021). 'Spatial structure and dispersal dynamics in a house sparrow metapopulation'. In: *Journal of Animal Ecology* 90.12, pp. 2767–2781. ISSN: 13652656. DOI: 10.1111/1365-2656.13580.

Romero, Adriana et al. (Nov. 2016). 'Diet Networks: Thin Parameters for Fat Genomics'. In: URL: http://arxiv.org/abs/1611.09340.

Ros-Freixedes, Roger et al. (Dec. 2022). 'Genomic prediction with whole-genome sequence data in intensely selected pig lines'. In: *Genetics Selection Evolution* 54.1. ISSN: 12979686. DOI: 10.1186/s12711-022-00756-0.

Rosenblatt, F (1958). 'THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN 1'. In: *Psychological Review* 65.6, pp. 19–27.

Rue, Håvard, Sara Martino and Nicolas Chopin (Apr. 2009). 'Approximate Bayesian Inference for Latent Gaussian models by using Integrated Nested Laplace Approximations'. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 71.2, pp. 319–392. ISSN: 1369-7412. DOI: 10.1111/j.1467-9868.2008.00700.x.

Sheehan, Sara and Yun S. Song (Mar. 2016). 'Deep Learning for Population Genetic Inference'. In: *PLoS Computational Biology* 12.3. ISSN: 15537358. DOI: 10.1371/journal.pcbi.1004845.

Srivastava, Nitish et al. (2014). 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'. In: *Journal of Machine Learning Research* 15, pp. 1929–1958.

Trevor Hastie, Robert Tibshirani and Jerome Friedman (2009). *The Elements of Statistical Learning*. 2nd ed.

Ubbens, Jordan et al. (Nov. 2021). 'Deep neural networks for genomic prediction do not estimate marker effects'. In: *The Plant Genome* 14.3. ISSN: 1940-3372. DOI: 10.1002/tpg2.20147. URL: https://onlinelibrary.wiley.com/doi/10.1002/tpg2.20147.

Van Der Maaten, Laurens and Geoffrey Hinton (2008). *Visualizing Data using t-SNE*. Tech. rep., pp. 2579–2605.

VanRaden, P. M. (June 2020). *Symposium review: How to implement genomic selection*. DOI: 10.3168/jds.2019-17684.

— (Nov. 2008). 'Efficient Methods to Compute Genomic Predictions'. In: *Journal of Dairy Science* 91.11, pp. 4414–4423. ISSN: 00220302. DOI: 10.3168/jds.2007-0980.

Vaswani, Ashish et al. (June 2017). 'Attention Is All You Need'. In: URL: http://arxiv.org/abs/1706.03762.

Wang, Kelin et al. (Jan. 2022). 'DNNGP, a deep neural network-based method for genomic prediction using multi-omics data in plants'. In: *Molecular Plant*. ISSN: 16742052. DOI: 10.1016/j.molp.2022.11.004.

Wattenberg, Martin, Fernanda Viégas and Ian Johnson (Oct. 2016). 'How to Use t-SNE Effectively'. In: *Distill* 1.10. DOI: 10.23915/distill.00002. URL: https://distill.pub/2016/misread-tsne/.

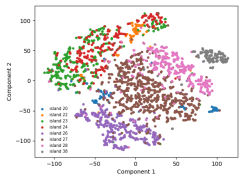
Wilcox, Rand R. (2001). 'Modern Insights about Pearson's Correlation and Least Squares Regression'. In: *International Journal of Selection and Assessment* 9.1-2, pp. 195–205. ISSN: 0965075X. DOI: 10.1111/1468-2389.00172.

Wilson, D. Randall and Tony R. Martinez (2003). 'The general inefficiency of batch training for gradient descent learning'. In: *Neural Networks* 16.10, pp. 1429–1451. ISSN: 08936080. DOI: 10.1016/S0893-6080(03)00138-2.

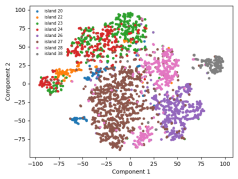
Zhou, Y T and R Chellappa (1992). 'Computation of Optical Flow Using A Neural Network''. In: *IEEE International Conference on Neural Networks*.

Zhou, Zhi-Hua et al. (Sept. 2020). *PERSPECTIVE. Why Over-parameterization of Deep Neural Networks Does Not Overfit?* Tech. rep.

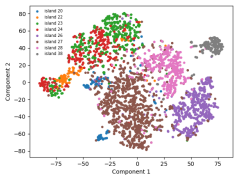
A t-SNE for varying perplexity and metric



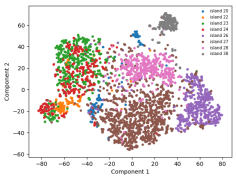
(a) $p = 10$ (Cosine)



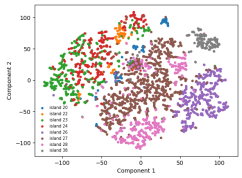
(b) $p = 20$ (Cosine)



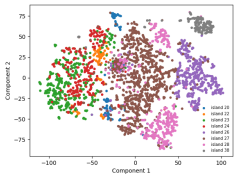
(c) $p = 30$ (Cosine)



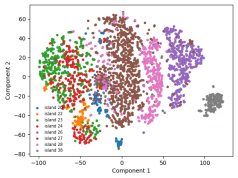
(d) $p = 50$ (Cosine)



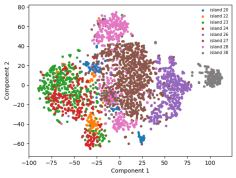
(e) $p = 10$ (Euclidean)



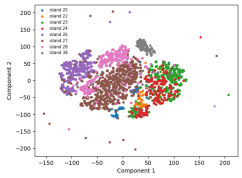
(f) $p = 20$ (Euclidean)



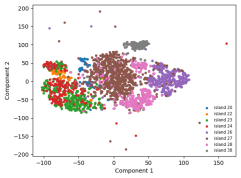
(g) $p = 30$ (Euclidean)



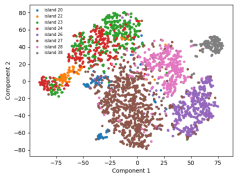
(h) $p = 50$ (Euclidean)



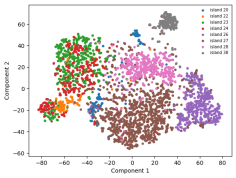
(i) $p = 10$ (Hamming)



(j) $p = 20$ (Hamming)



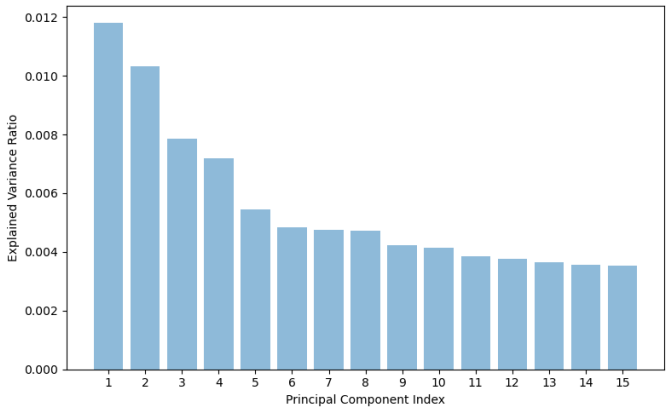
(k) $p = 30$ (Hamming)



(l) $p = 50$ (Hamming)

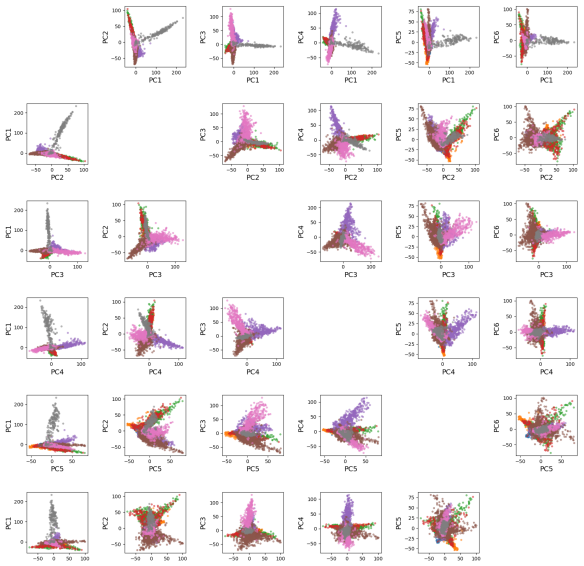
Perplexity (p) and metric used.

B Variance explained in PCA



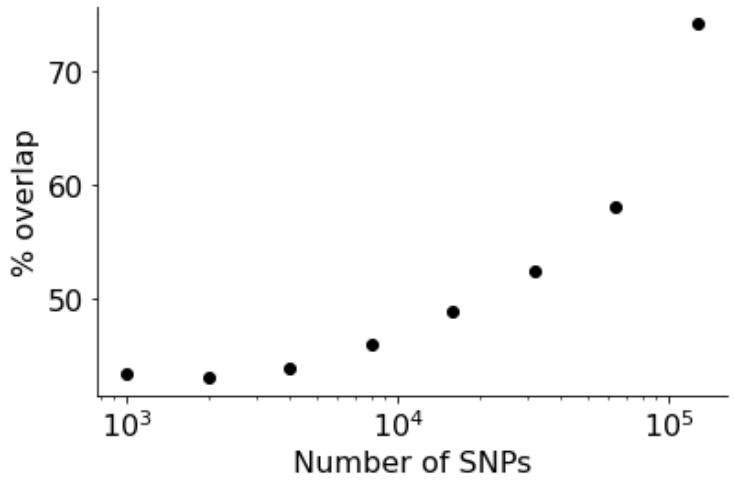
Variance explained by the 15 first principle components.

C Six principle components of the large data set



The first six principle components from the large data set.

D Intersection of SNP selection methods by increasing SNPs



Intersection of the SNP selection Best_unif and Most_correlated for increasing numbers of SNPs selected. X-axis is on a log scale.

E Adjusting body mass for non-genetic variables

Code for fitting an LMM using the non-genetic effects and extracting the adjusted phenotype. Implemented in R.

```
library(lme4)
formula.mass.lmm = mass ~   sex + FGRM + month + age  +
  (1 | island_current) +
  (1 | hatchyear) +
  (1 | ringnr)
r.mass.lmer <- lmer(formula.mass.lmm, data = dd)
# The adjusted phenotype is the ID effect:
ID_effect <- ranef(r.mass.lmer)$ringnr
```

F Best_unif SNP selection

The Best_unif_ selection scheme (Section 3.5) implemented in Python.

```

def Best_Unif_N(X, Y, N, method = 'spearman'):

    """
    Returns a list of about n SNPs. Usually a few more than
       n as there is always one SNP selected from each
       Chromosome. The SNPs
    most correlated with response on each of the n windows
       ()f same width meassured in base pairs) is selected.

```

```
8       Parameters
9       ----------
10      N : Int
11          Number of SNPs to be chosen.
12      X : Pandas data frame
13          Training data.
14      Y: Numpy array or pandas series
15
16      Returns
17      -------
18      output_SNP : list
19          The chosen SNPs based on the Best_unif_n criteria.
20
21      """
22
23      import json
24      import numpy as np
25
26      file_path = "Path to dictionary containing position of
           each SNP"
27      with open(file_path, "r") as json_file:
28          SNP_locations = json.load(json_file)
29
30      file_path = "Path to dictionary of chromosome contents"
31      with open(file_path, "r") as json_file:
32          Chromosome = json.load(json_file)
33
34      correlations = X.corrwith(Y, method=method)
35      top_N_indices_correlations=np.abs(correlations).nlargest(N).index
36
37      output_SNP = []
38      windows= []
39      not_i_ndata_count = 0
40
41      for c in Chromosome.keys():
42
43          nn = int(len(list(Chromosome[c])) /
               len(correlations.index) * N)
44          nn = max(nn, 1)
45
46          bp = []
47          for snp in Chromosome[c]:
48              bp.append(int(SNP_locations[snp][1]))
49          bp_max = max(bp)
50          t = bp_max / nn
51          lower = 0
52
53          current_window = []
54          for snp in Chromosome[c]:
55              if snp in correlations.index:
56                  if int(SNP_locations[snp][1]) - lower > t:
```

```
                    windows.append(current_window)
                    current_window = []
                    current_window.append(snp)
                    lower += t
                else:
                    current_window.append(snp)
            else:
                not_i_ndata_count +=1
        windows.append(current_window)

    for i in range(len(windows)):
        correlations.loc[windows[i]]
           output_SNP.append(np.abs(correlations.loc[windows[i]]).nlargest

    best_unif_final = []
    for w in output_SNP:
        for snp in w:
            if type(snp) == list:
                for snp2 in snp:
                    best_unif_final.append(snp2)
            else:
                best_unif_final.append(snp)

    return best_unif_final
```

G GBLUP using INLA

Defining the GBLUP model in R-INLA and retrieving the breeding values and full predictions.

```
library(INLA)
"""
idx_test: Indexes of test set. The phenotypes and genetic
   effects are set to NA in the dataset passed to the INLA
   model.
Cmatrix: The inverse of the relatedness matrix.
D.morph: Data set
"""
#Defining formula
formula.phenotype <- phenoype ~ sex + FGRM + month + age +
    f(hatchisland, model = "iid", hyper = list(
      prec = list(initial = log(1), prior = "pc.prec", param
         = c(1, 0.05))
    )) +
    f(hatchyear, model = "iid", hyper = list(
      prec = list(initial = log(1), prior = "pc.prec", param
         = c(1, 0.05))
    )) +
    f(IDC, model = "iid", hyper = list(
```

```
      prec = list(initial = log(1), prior = "pc.prec", param
         = c(1, 0.05))
    )) +
    #The genetic effect
    f(IDC2,
      values = idxs, model = "generic0",
      Cmatrix = Cmatrix,
      constr = TRUE,
      hyper = list(
        prec = list(initial = log(0.5), prior = "pc.prec",
           param = c(sqrt(2), 0.05))
      )
    )


# Creating model
model.phenotype <- inla(
    formula = formula.phenotype, family = "gaussian",
    data = d.morph_train,
    control.family = list(hyper = list(theta = list(initial
       = log(0.5), prior = "pc.prec", param = c(sqrt(2),
       0.05)))),
    control.compute = list(dic = F, return.marginals = FALSE)
    # control.compute=list(config = TRUE)
  )
# Full predictor
preds_EG <-
   model.phenotype$summary.fitted.values$mean[idxs_test]
# Get breeding values
preds_G <-
   model.phenotype$summary.random$IDC2$mean[idxs_test]

#Correlation of full predictor with true phenotype
corr_EG <- c(corr_EG, cor(preds_EG, pheno_test_EG, method =
   "pearson"))

#Correlation of breeding value with true phenotype
corr_G <- c(corr_G, cor(preds_G, pheno_test_G, method =
   "pearson"))
```

H Parallel neural network

Defining and training a parallel neural network in as described in Section 3.7.

```
"""
X_env: Data frame containing non-genetic variables
X_gen: Data frame containing genetic variables

Make sure that the samples in both data sets correspond
```

```
"""
#Define learning rate schedule
lr_schedule =
    tf.keras.optimizers.schedules.CosineDecay(initial_learning_rate
    = 0.01, decay_steps = 300, alpha = 0.0001)


# Pre training of environmental model
env_model = Sequential()
env_model.add(Dense(units=12, activation='relu',
    input_shape=(X_env_train.shape[1],)))
env_model.add(Dense(units = 12, activation='relu'))
env_model.add(Dense(1))
env_model.compile(optimizer=Adam(learning_rate=lr_schedule),
                  loss='mse',
                  metrics=['mae'])

early_stopping = EarlyStopping(monitor='val_loss',
    patience=10, restore_best_weights=True)

env_model.fit(x = X_env_train, y = Y_train, verbose = 1,
    epochs = 100,
              batch_size = 32, callbacks=[early_stopping],
              validation_split = 0.1)

# Pre-selecting SNPs
N_SNP = 10000
selected_SNPs = Most_correlated(X_gen_train,
    pd.Series(Y_train), N_SNP)
X_gen_train, X_gen_test = X_gen_train.loc[:, selected_SNPs],
    X_gen_test.loc[:, selected_SNPs]

lr_schedule =
    tf.keras.optimizers.schedules.CosineDecay(initial_learning_rate
    = 0.01, decay_steps = 200, alpha = 0.001)

#GENETIC MODEL
gen_model = Sequential()
gen_model.add(Dropout(0.5,
    input_shape=(X_gen_train.shape[1],)))
gen_model.add(Dense(units = 64, activation='relu'))
#gen_model.add(Dropout(0.4))
gen_model.add(Dense(units = 64, activation='relu'))
#gen_model.add(Dense(1))
gen_model.compile(optimizer=Adam(learning_rate=lr_schedule),
                  loss='mse',
                  metrics=['mae'])



# Freeze layers of the pre-trained models
```

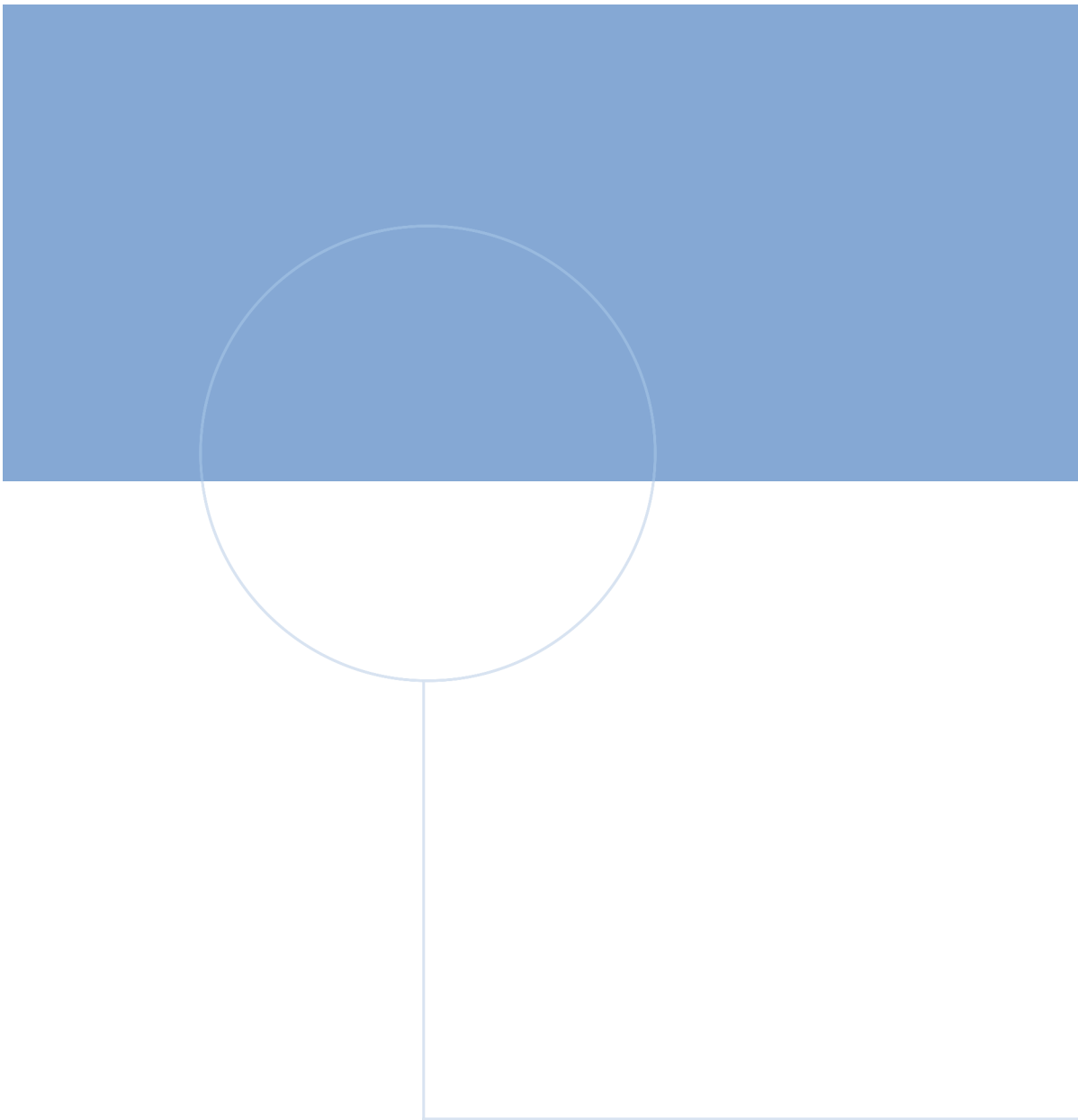
```
for layer in env_model.layers:
    layer.trainable = False

# Now, combine models into the final full model
combined_input = concatenate([gen_model.output,
   env_model.layers[-2].output])
#Add a fully connected layer
x = Dense(64, activation='relu')(combined_input)
output = Dense(1, activation='linear')(x)

final_model = Model(inputs=[gen_model.input,
   env_model.input], outputs=output)
final_model.compile(optimizer='adam',
   loss='mean_squared_error')

#Train the final full model
early_stopping = EarlyStopping(monitor='val_loss',
   patience=10, restore_best_weights=True)
final_model.fit([X_gen_train, X_env_train], Y_train,
   epochs=50, batch_size=32, callbacks=[early_stopping],
   validation_split = 0.1, verbose = 1)

#Predictions of full model
predictions = final_model.predict([X_gen_test, X_env_test])
```



NTNU

Norwegian University of
Science and Technology