Optimisation of autoencoders for prediction of SNPs determining phenotypes in wheat

Karthik Nair

Degree project in bioinformatics, 2021
Examensarbete i bioinformatik 30 hp till masterexamen, 2021
Biology Education Centre and Information Technology Centre, Uppsala University
Supervisor: Carl Nettelblad
Abstract

The increase in demand for food has resulted in increased demand for tools that help streamline plant breeding process in order to create new varieties of crops. Identifying the underlying genetic mechanism of favourable characteristics is essential in order to make the best breeding decisions. In this project we have developed a modified autoencoder model which allows for lateral phenotype injection into the latent layer, in order to identify causal SNPs for phenotypes of interest in wheat. SNP and phenotype data for 500 samples of Lantmännen SW Seed provided by Lantmännen was used to train the network. Artificial phenotype created using a single SNP was used during training instead of real phenotype, since the relationship between the phenotype and SNP is already known. The modified training model with lateral phenotype injection showed significant increase in genotype concordance of the artificial phenotype when compared to the control model without phenotype injection. Causal SNP was successfully identified by using concordance terrain graph, where the difference in concordance of individual SNPs between the modified modified model and control model was plotted against the genomic position of each SNP. The model requires further testing to elucidate its behaviour for phenotypes linked to multiple SNPs.
Identifying genetic variations responsible for interesting phenotypic characteristics: An AI based approach

Popular Science Summary

Karthik Nair

From the dawn of agriculture, farmers have selected and bred plants to isolate and establish favourable traits in a population of crops. This selective breeding has resulted in multiple varieties of the same kind of plants, each with their distinctive characteristics, sometimes so distinct that the domesticated variety looks nothing like the wild plant. A good example of this would be wild cabbage, which humans have bred into common cabbage, cauliflower, broccoli and more. This manual selective breeding process came with a lot of uncertainty. Usually, the process involves a lot of trial and error and for the most part one does not know what the underlying genetic mechanism for a desired phenotype is or if it is present at all in the plants being selected.

With the advent of genome sequencing, it is now possible to identify single point mutations or variations in the genetic code of any organism much faster than before. It is important to note that marker maps which detail such variations have existed before the advent of genome sequencing. But, sequencing technologies have made this process a whole lot faster. These genetic variations called Single Nucleotide Polymorphims or SNP for short, can influence the underlying cellular mechanisms and thereby result in various physical characteristics known as phenotypes. In this project a dataset derived from massive testing of many SNPs at many different genomic position was used.

Some of these phenotypes and thereby their causal SNPs, are beneficial for farmer and hence they would want to fix it in their crops. But it is difficult to know which
SNP is linked to which phenotype as a plant can have many SNPs in its genome, and having this knowledge can help farmers and agricultural scientists identify plants carrying the beneficial SNPs they want and use them in the breeding process. In this project we have developed an AI model which does exactly that i.e., it can identify the causal SNPs for the phenotype we are interested in.

But what makes our model unique, is the way it works, and it is best explained by the following example. Let's say, we have a huge collection of car photos from different brands such as Volkswagen, Suzuki, BMW, Nissan etc, where each brand has at least 1000 photos. We give these photos to two computers which are expected to learn these photos and then draw all of them from memory. In order to do that they will try to identify and learn the most important features of each brand such as wheel design, ground clearance, tyre width etc. This is a repetitive process where the computer constantly corrects itself. Now, we want to identify, based on these photos, which feature contributes to mileage. So while the computers learn these images, we pass a hint to one of the computers, which contains the mileage for each car. Now as the computer which got the hint (called Comp-H), repeatedly learns, draws and corrects itself, it will learn to use the hint to make a better image. At the end of this learning phase, we can compare the drawings made by the two computer and see which features of the image are better drawn better by Comp-H. If the Comp-H recreates tyre width better than the other computer, we can say that tyre width might have something to do with the mileage of the car.

Just like the example mentioned above, our AI models learns the most important features from an SNP dataset of a crop instead of car photos, then uses the information to recreate the original dataset. Furthermore, just like in the example, one of the model allows for phenotype injection, while the other model gets no such information. The recreated datasets are then compared to see which SNPs in the dataset have been recreated better by the model with the phenotype injection. Those SNPs which has the best reconstruction are most likely to be linked to the phenotypes.

As global demand for food continues booming, new robust crop varieties with desirable characteristic need to be developed rapidly. In this race to meet the market demands, the AI model developed in this project, could allow agricultural scien-
tists make better decisions by helping them understand, identify, and isolate the genetic mechanism behind the desirable qualities and thereby help create new and improved crop varieties.
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCE</td>
<td>Categorical Cross Entropy</td>
</tr>
<tr>
<td>CVAE</td>
<td>Convolutional Variational Autoencoder</td>
</tr>
<tr>
<td>IWGSC</td>
<td>International Wheat Genome Sequencing Consortium</td>
</tr>
<tr>
<td>KLD</td>
<td>Kullback–Leibler Divergence</td>
</tr>
<tr>
<td>mRNA</td>
<td>Messenger Ribonucleic Acid</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Squared Error</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>SNP</td>
<td>Single Nucleotide Polymorphism</td>
</tr>
</tbody>
</table>
# Contents

1 Introduction 1

2 Background 2
   2.1 SNPs and Phenotypes 2
   2.2 Neural Networks 3
      2.2.1 Neurons and Layers 4
      2.2.2 Weights and Loss functions 5
      2.2.3 Autoencoders 7

3 Theory 9
   3.1 Deep Learning Neural Networks 9
   3.2 Network Architecture 12
   3.3 Loss Functions 13
      3.3.1 Mean Squared Error 13
      3.3.2 Categorical Cross Entropy 14
      3.3.3 Kullback-Leibler Divergence 14
   3.4 Optimisers 15
   3.5 Hyperparameters 16
      3.5.1 Batch Size 16
      3.5.2 Dropout 17
      3.5.3 Learning Rate 17
      3.5.4 Noise 17
      3.5.5 Regularisation 18
   3.6 Evaluation Parameters 18
      3.6.1 Loss Values 18
      3.6.2 Genotype Concordance 19

4 Materials 19
   4.1 Data 19
   4.2 Hardware 20
   4.3 Software 21
4.4 Version Control .................................................. 21

5 Methods ......................................................... 21
  5.1 Model 1 - Simple Autoencoder: Encoder And Decoder .... 22
  5.2 Model 2 - Trained Encoder With Phenotype Injection And Two
      Untrained Decoders .......................................... 23

6 Result ............................................................. 26
  6.1 Model Performance ........................................... 26
  6.2 SNP Prediction ............................................... 33

7 Discussion ...................................................... 35

8 Conclusion ....................................................... 39

9 Acknowledgement ............................................... 39
1 Introduction

Artificial intelligence in its various forms is revolutionising almost all fields that encompass humanity; from consumer-friendly chatbots on a customer care website, to self-driving Tesla automobiles. Given appropriate computational resources a well trained AI model helps save time and human effort in terms of critical decision making in research. Machine learning is now increasingly being used in life science to automate otherwise manual processes e.g., genome annotation, protein structure prediction etc., and to streamline data intensive research projects, especially in multi-omic research.

The advent of Large Scale Data Analysis and Artificial Intelligence in genomics has allowed researchers to gain new deeper insights into molecular factors that govern cellular and evolutionary processes. The use of deep learning algorithm such as Stacked Denoising Autoencoders have been used for gene function annotation in cancer pathways (Guan et al., 2018).

Similarly, deep learning has also found application in agricultural research as well. Deep learning neural networks have been used to identify diseases in crops (Lu et al., 2017), to discriminate geographic origin of rice (Long et al., 2017), and to reveal genetic variation among plants. (Champigny et al., 2020).

Identification of genetic factors that affect crops can help breed better crops to meet future market and dietary demands, and deep learning algorithms can help build tools and methods to better identify such genetic factors. In this thesis project, we optimise an unsupervised neural network called an Autoencoder, to identify causal Single Nucleotide Polymorphisms (SNPs) responsible for phenotypes for interest in wheat. For this purpose, we use SNP data from 500 samples of Lantmännen SW Seed wheat provided by Lantmännen. The dataset also contains phenotypic information for the samples. The standard autoencoder architecture has been modified to allow for injection of phenotypic value into the middle of the network in order to increase the performance of the autoencoder. This will allow the observed genotype background and the phenotype to interact during the reconstruction of the genotype and help study the relationship between he genotype and phenotype.
We compared its performance of a modified autoencoder architecture to that of a standard autoencoder and assessed it’s performance in terms of loss values and genotype concordance.

2 Background

2.1 SNPs and Phenotypes

Single nucleotide polymorphisms or SNPs are single base (or single ”letter”) variations in a nucleotide sequence that occur at specific positions among individuals in a population. These variations usually occur due to substitution of one nucleotide at a certain position by another. These substitutions can influence the activity of one or more genes, depending on where substitutions occur in the genome.

The alternative bases that may occur at a given SNP position are called alleles. Usually there are only two alleles in a population; the most common allele is called the major alleles and the others are called minor alleles. The alleles are present in an individual as pairs since autosomal chromosomes exist as two copies, and the combination of a pair defines the genotype of the individual. Individuals who carry a pair of major allele or a pair of minor allele are referred to as homozygous, whereas individuals who carry one major allele and one minor allele are called heterozygous.

SNPs can influence gene expression, mRNA stability etc., (Shastry, 2009) and can also result in completely different coding sequence. As a result SNPs can influence phenotypic expression in organisms. It is important to note that while there might be many SNPs in a genomic region, it does not necessarily mean that all of them are causal SNPs. Furthermore, due to linkage disequilibrium, there can be SNPs in close proximity to the causal variant which might seem to correlate directly with the phenotype. In other words not all SNPs affect the phenotype, only the causal ones do.

Identification of phenotypic characteristics is essential in agricultural breeding
programs. Phenotypes such as higher grain volume, longer shelf life, shorter harvest period etc impact both the market availability of the crop, and the resources spent by the farmers in growing, tending and selling the crops. With the rise in global population, it is essential to improve existing crop varieties to meet the rising demand. Moreover, global warming associated climate change is expected to bring significant changes to climate patterns globally and thereby affect the growing seasons, rainfall, local temperature etc. Therefore characteristics deemed beneficial in terms of resilience, nutrition and/or economics are sought out by farmers, plant breeders, agricultural researchers, and agri-tech companies in order to create new varieties of a crop and optimise food supply.

Many a times while phenotypes are known, the exact underlying genetic changes that give rise to these phenotypes remain a mystery. Some of these phenotypic characteristics occur due to individual SNPs or a group of SNPs working together (Amos, 2010). Identification of these causal SNPs will allow agricultural researchers to better understand the genetic mechanism behind crop phenotypes and use this understanding to optimise existing crop breeds and develop new ones.

2.2 Neural Networks

Neural networks are computational algorithms inspired by the network of neurons such as those found in animal brains. Despite being inspired by biological neural networks, they work quite differently from their natural counterparts. Between the two, the similarity lies in their ability to learn, infer, and then predict, classify and sometimes even reproduce data (McCulloch & Pitts, 1943).

A standard neural network architecture consists of several nodes and layers, and use mathematical loss functions to constantly correct themselves (further explained in section 2.2.1 and 2.2.2) in order to make their results more accurate.

Neural networks need to be trained before they can be used for any practical purposes. An untrained neural network is like a newborn baby, it is beautiful but it cannot make any sense of the information it is exposed to. The network needs to be trained using a dataset known as the training data so that it can “learn” to make sense of the data in order to perform a specific task, which in our case might be
classification, prediction, or regression. The larger the dataset, the more examples the network has to learn from, the better it gets at its task.

During the training, the neural network learns or “understands” the important features of the data that helps it do the task at hand. Typically, in case of a supervised network three dataset are used for training and validating the model - training set, validation set, and test set. Training set is the dataset that is used for training the model. The validation set is used for validating the training of the model and check if the model has learned important features. Finally, the behaviour of the fully trained network is verified using the test set. All the three datasets are similar to each other, but the samples in the dataset are mutually exclusive. In this project though, only test and training set were sufficient as we have are implementing an unsupervised neural network model with high dimensionality.

2.2.1 Neurons and Layers

Now that we have a fair understanding as to what a neural network does, it is equally important to understand how and why it works. Every neural network consists of several nodes called neurons, and these nodes are arranged into multiple layers in order to create a neural network.

Each node is capable of receiving an input, processing it and transmitting to other nodes. These nodes are further arranged into many layers, and a typical neural network consists of at least three layers: input layer, hidden layer or processing layer, and output layer (see Fig. 1). The input layer consists of the nodes for data input, the hidden layers contain the nodes where data is processed, and the output layer provides the processed results from the network.
Neural networks with multiple hidden layers are called deep neural networks and network width is described by the number of nodes in each layer. Deep learning neural networks can extract more information from a given dataset, since the deeper network is able to do process more pieces of information as compared to a simple neural network (Najafabadi et al., 2015). It allows for the identification of complex non-linear relationships between multiple parameters, but this comes at the cost of slower training and in an increased risk of overfitting (Zhu et al., 2018). Another issue with deep networks is vanishing gradients. Vanishing gradient refers to the problem where the gradient being back-propagated diminishes to an ineffectual level by the time it reaches the input layer (Tan & Lim, 2019).

2.2.2 Weights and Loss functions
When a neural network is initialised, the connections between the nodes in each layer have random weights. Weights are mathematical values associated with the connections between two or more nodes, which determine how much influence
each node has on the final result. Neural networks constantly correct themselves by adjusting the weights during the learning process, in order to get the best possible results. To do this, it needs a measure of how correct it is.

For this purpose, mathematical functions called Loss Functions are used. These loss functions calculate the amount of error in each training run and the network then uses it to adjust the weights. The error signal is then propagated back through the network all the way back to the input layers. This is called backpropagation. This helps the network make better changes in the weight so that the calculated error is lower than the previous run. But being a complex set of interaction, this may not be the case always, as sometimes, the loss can increase over epochs. This is sometimes beneficial, as it allows the network to get out of a local minima, but is definitely problematic if it keeps increasing for all future epochs. The entire learning process can be described as a process of continuous minimisation of the loss function.

There are many different kinds of loss functions, but the two main categories used in the course of this project are Mean Errors, and Entropy-based loss functions. Examples of Mean Errors include Means Squared Error, Mean Absolute Error, and Mean Squared Logarithmic Error. All mean error loss functions take the average of the sum of all errors in the network or the average of the sum of squares of all errors. Mean errors are conventionally used for regression, as most real world data follows a Gaussian distribution, and mean errors are better suited for continuous value prediction in such scenarios.

Entropy-based loss functions are probabilistic in nature, and are conventionally used for classification problems. Entropy based loss functions measure the entropy of a the network i.e., it measures the probability distribution for a specific sample with respect to the different classification labels. In entropy-based functions, higher values indicate higher uncertainty and vice versa (Goodfellow et al., 2016). Due to the way entropy based loss functions work mathematically, they penalise wrong classifications which have very high confidence, much more severely than wrong classification with very low confidence. In essence, bigger errors lead to larger penalties. Two common entropy based functions are Binary Cross Entropy and Categorical Cross Entropy.
Cross Entropy are used for two-class classification problems and multi-class classification problems respectively.

During the learning process, the weights of the connections between the nodes, are constantly adjusted in order to minimise the loss in each training step. By the end of the training, the nodes and connections have been optimised and refined for the classification or prediction task at hand.

2.2.3 Autoencoders

An autoencoder is a type of unsupervised neural network which can learn efficient data representation known as data encoding from the data that it is being trained on (Kramer, 1991). Essentially, an autoencoder takes input data, learns only the most important features of the data, and then recreates the data. By doing this, the neural network learns the bare minimum representation required to recreate the entire dataset, and this is where it differs from a simple copy & paste technique. Due to their ability to learn only the most important features of a dataset, autoencoders are generally used for data compression, and noise reduction tasks.

In a standard autoencoder, the network gets narrower (has fewer nodes) with each successive layer until the middle layer, also know as the latent layer. The latent is the most narrow part of the network. The network then starts widening again, and the last layer or the output layer has the same number of nodes as the input layer. This makes the overall network similar to an hourglass (see Fig. 2).
Figure 2: An Autoencoder architecture contains three parts: encoder, latent layer, and decoder

An autoencoder consists of three parts (Goodfellow et al., 2016):

- **Encoder:** The encoder contains nodes for data input and other layers used for data encoding. Data encoding in this context refers to producing the most compact representation devised by the network. Each layer in this part of the network is smaller than its preceding layer. The reduction in the number of nodes causes the data to undergo dimensional reduction, and forces the network to extract the most relevant features and ignore the noise present in the data.

- **Latent Layer:** The latent layer is the narrowest part of the network. This layer is crucial to the network as it learns and holds the bare minimum rep-
representation or the encoded form of the entire dataset.

- **Decoder:** Decoder is where the encoded data is decoded. In other words, the original data is reconstructed starting out from the representation from the latent layer. Each successive layer in this part of the network is wider than its preceding layer. In the decoding process, with each layer, the reconstruction gets closer to the final representation. The final layer or the output layer is always the same size as the input layer, as it is the layer which provides the final reconstruction.

### 3 Theory

#### 3.1 Deep Learning Neural Networks

Adding to the description *Background* section, Deep Learning Neural Networks can be further elaborated as multi-layered non-linear information-processing models capable of learning features using successive and higher layers of abstraction (Deng & Yu, 2014).

Among the three basic layers of a neural network, hidden layers calculate intermediate states of data between the input and output layers. It is essential to keep in mind that during the actual training, it is impossible to know exactly occurs in the hidden layers. Therefore, hidden layers are essentially a black box. While data is certainly present in these layers, it may not be human interpretable. This also depends on the network architecture, for example it is possible to visualise the latent layer of an autoencoder, provided it is 2 dimensional, in a way similar to PCA, making it a bit more interpretable (Ausmees & Nettelblad, 2020).

Each node in a layer is connected to one or more nodes in other layers of the network. Each of these connections have an adjustable weight associated with them. Weights are numeric values which represent the strength of the connection between two nodes. When data is passed from one node to another, it gets multiplied by the weight of that connection. A weight with a high magnitude implies a strong
connection, and vice versa. It is important to note that weights can have negative magnitude, which might imply an inverse relationship between the nodes.

Once a node receives the data from other nodes, the data from each node is multiplied by the weight of each connection and summed, following which it is transformed into a signal using a mathematical function called an activation function. Activation functions are non-linear mathematical functions that produce a signal which decides if a node is ON or OFF. This behavior is inspired by the firing of biological neurons in animals. The signal may be continuous or discrete depending on the kind of activation function used in the node, but the discrete activation functions are much rarer as one wants a network which is capable of discerning even subtle changes in the data. In a deep learning neural network, each layer extracts a disparate set of features using the data passed to it from the preceding layer. Each layer builds up on the information from the previous layer, extracting successively more complex features (Bengio, 2012).

For example, consider a dataset of images consisting of human faces to be used for a clustering task where the neural network has to correctly cluster the different characters. The first hidden layer of the network learns to identify the contrast between different sets of pixels. The second layer would then use the contrasting pixel data to learn to identify lines. The next layer might use this information to identify curves and straight lines in the images. Eventually, some layer downstream, building up on the information from previous layer, will be able to identify features like eyes, nose etc., and finally the last layer would be able to build up on all the complex features and then correctly classify the image (see Fig. 3).

This series or hierarchy of successively complex features is known as Feature Hierarchy. Feature hierarchy is one of the key aspects of deep learning neural networks and helps explain how complex high-dimensional datasets can be handled by neural networks with relative ease, despite each only applying simple non-linear mathematical functions (Najafabadi et al., 2015).
Figure 3: A typical example of feature hierarchy: The first level learns pixel contrasts, the second level builds on the information from the first and learns line and curves, the third layer level builds on the third to learn facial features and, the fourth layer learns faces using the facial features. Adapted from (Lee et al., 2011, p. 100-101)
3.2 Network Architecture

Just like all other neural networks, deep learning autoencoders consist of multiple layers arranged in a manner that ensures that the output dimensions match the input. This arrangement of layers and their relationship with each other is called the network architecture. The autoencoder model used in this project is called a convolutional variational autoencoder (CVAE) and these models are usually employed for pattern finding, anomaly detection etc., as they are good at finding non-linear patterns in the data (Chen et al., 2018). One of the most common applications areas for CVAE is noise reduction in image data. A CVAE which is well trained on similar images would be able leverage the pattern finding capability to identify and learn only the most relevant features for image reconstruction and as a result, ignore the noise. In the context of this project, this pattern-finding capability would allow for better reconstruction of causal SNPs whereas the reconstruction of other SNPs stay unaffected.

CVAEs such as the one implemented in this project derive their name from the convolutional layers used in the network. Convolutional layers use filters of defined size that “slide” across the data, and output a dot product for each step the filter takes. These steps are known as strides. This “sliding” operation creates a feature map. Depending on the requirement, the filter and stride of the layer can be adjusted. On the encoder side, another important layer is max pooling. Max-pooling layers usually follow convolutional layers and in this layer, the feature maps undergo dimensionality reduction, where only the most relevant feature values are kept. This can also be viewed as a form of data compression. After these layers, the penultimate layer before the latent layer is usually a dense layer. Dense layers are fully connected layers i.e., all nodes in a dense layer are connected to all other nodes in the adjacent layer. The latent layer is also dense a layer, and is the narrowest part of the network. The data is at maximum compression in this layer.

On the decoder side, the layers are similar in function and use similar mathematical operations as the encoder to decode the data and reconstruct it. The biggest difference is the presence of the up-sampling layer instead of max-pooling, where the data undergoes an increase in dimensionality. Finally, the data is reconstructed
at the output layer, which has the exact same dimensions as the input layer. An example CVAE has been illustrated in Figure 4.

![Figure 4: Architecture of a typical convolutional autoencoder - - Light red blocks represent convolutional operations, dark blue blocks represent max-pooling, and purple blocks represent up-sampling. Thick dashed lines indicate dense connections.](image)

### 3.3 Loss Functions

Loss functions are used to measure the correctness of the prediction given by the neural network. The lower the loss values, the better the prediction. Hence during the course of a training run, the network tries to minimise the loss function, and thereby get better predictions. In other words, neural network training is an optimisation problem, where the aim is to minimise the loss function.

For the purpose of this project, three loss functions were considered: Mean Squared Error, Categorical Cross Entropy, and Kullback-Leibler Divergence.

#### 3.3.1 Mean Squared Error

Mean Squared Error (MSE) is one of the simplest loss function used in machine learning, and is primarily used for regression. It calculates average of the sum of squares of the difference between the true value and the predicted value.
\[ MSE(y, \bar{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \bar{y}_i)^2 \]  

Mathematically, MSE is represented by Equation 1, where \( i \) is an SNP from the dataset, \( N \) is the total number of SNPs in the given dataset, \( y_i \) is the true value of the SNP and \( \bar{y}_i \) is the predicted value of the SNP. MSE only measures the magnitude of difference and not the correctness. For example, a SNP prediction of 1.5 is equally different/correct as a prediction of 0.5 when compared to the true SNP value of 1.0, even though an SNP value greater than 1.0 is impossible.

### 3.3.2 Categorical Cross Entropy

Categorical Cross Entropy (CCE) (West & O’Shea, 2017) is an entropy-based function used for multi-class classification problems. CCE calculates the negative log likelihood for a given prediction. In simple terms, it can be said that CCE penalises less wrong predictions less, and more wrong predictions more.

\[ CCE(y, \bar{y}) = -\frac{1}{M} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \cdot \log(\bar{y}_{ij}) \]  

CCE is described by Equation 2, where \( y_{ij} \) is the probability distribution of the true value, \( \bar{y}_{ij} \) is the probability distribution of predicted value, \( i \) is the class, \( j \) is the SNP, \( N \) is the total number of classes, and \( M \) is the total number of SNPs.

### 3.3.3 Kullback-Leibler Divergence

Kullback-Leibler Divergence or KL Divergence, also called relative entropy, is another entropy based function which calculates how divergent the predicted probability distribution is as compared to the probability distribution of the reference(true value).

\[ D_{KL}(y||\bar{y}) = \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \cdot \log \left( \frac{y_{ij}}{\bar{y}_{ij}} \right) \]  

KL Divergence is described in equation 3, where \( D_{KL} \) is the KL Divergence, \( y_{ij} \) is the probability distribution of the true value, \( \bar{y}_{ij} \) is the probability distribution of
predicted value, \(i\) is the class, \(j\) is the SNP, \(N\) is the total number of classes, and \(M\) is the total number of SNPs.

It is important to note that KL Divergence is an asymmetric function i.e., \(D_{KL}(y||\tilde{y}) \neq D_{KL}(\tilde{y}||y)\).

### 3.4 Optimisers

After each iteration of a training run, loss value is calculated, and this information is then relayed back to the network using back-propagation (refer Section 2.2.2). The network then uses this information to adjust the weights in the network. The loss gradient is calculated after each iteration, which allows the network to identify which weights need to be adjusted and by what magnitude. This computation of the gradient is what is referred to as back-propagation. But, the gradient is usually not applied directly to the network, instead an Optimiser is used to apply the calculated gradient.

Optimisers are algorithms which help apply the calculated gradient and then adjust the weights of the network in order to minimise the loss function value. Optimisation is essentially a hunt for the global minima of the loss function, and it is easy for the optimiser to get stuck in a local minima. Therefore, if the optimiser makes weight adjustments which are too big, it can overshoot the minima, and if it is too small, then it will never find the minima and can potentially get stuck in a local minima. Different optimisers use different methods to adjust the weights of the network. Common optimisers include Standard Gradient Descent, RMSProp, and Adam. For the purpose of this project Adam (Kingma & Ba, 2017) was used as the optimiser.

In a nutshell, one could say that the optimiser continuously solves the optimisation problem of training by adjusting the weights in the network in order to minimise the loss value.
3.5 Hyperparameters

At the beginning of the training, the network along with the optimiser will try to find the best solution to the loss function. To help find the solution, some parameters can be tuned to find the solution more quickly or efficiently. These parameters are called hyper-parameters. Hyperparameters are usually tuned manually, and help in more efficient training. While hyperparameters vary from model to model, in the context of this project, the hyperparameters that were used are Batch Size, Dropout, Learning Rate, Noise, Regularisation.

3.5.1 Batch Size

Batch size is the maximum number of samples from the training set that is propagated through a network. This hyperparameter is extremely relevant when working with genomic data, which tends to be large. With large datasets it is possible that all of the data points for all samples may not fit within the RAM of the computer. It is important to note that, it is far more important that the activation in all the layers of the network need to be stored for every sample in the batch in order for back-propagation to work. Even if the dataset does not fit the RAM, these stored activations have to fit on the processing unit(CPU/GPU) irrespective of the RAM size.

This requires the data to be split in batches. Batch size defines how big the batches should be. For example, if a dataset contains 560 training samples and the batch size is set to 100, the model will train 5 batches of 100 samples, and the last batch would consist of 60 samples. Once all the batches have passed through the network, an epoch of training is completed.

While smaller batch sizes can reduce the speed of the training, the repeated selection of small batches promotes well performing weights. Also, smaller batches also introduce more noise, but the noise added in this manner can help in achieving weights which are perform consistently well over all samples. This, in combination with optimiser, normalisation and loss function, affects the performance of the training and not just the speed.
3.5.2 Dropout
Dropout is a technique where a certain fraction of nodes in a layer are removed during training. The nodes being removed during each training run are selected at random. Thus the network is forced to learn features using multiple weights and nodes, and not just use specific nodes to get the output. This forces the network to generalise the model and hence reduces the chances of overfitting. If the dropout for a layer is set at 0.3, 30% of the nodes will be randomly ignored during each training run. Given enough period of time, the model will be capable of utilising any 70% set of nodes in that layer to produce generalised data in a manner that minimises the loss function.

3.5.3 Learning Rate
Learning rate is the magnitude by which the weights in the network are adjusted in order to minimise the loss. A more intuitive way to look at it, would be the length of each step the model takes during its search for the minima of the model. If the learning rate is too small, the model can take a really long time to converge and could get stuck in a local minima, whereas a large learning rate could make the network completely overshoot the minima, and if each iteration overshoot, the network can become divergent instead of being convergent. It is one of the most important hyperparameters in machine learning as it defines how fast a network learns.

3.5.4 Noise
In the case of an autoencoder, high-dimensional data is encoded in a low dimensional form in the latent layer. This forces the network to retain as much of the original information as possible within the constraints of the layer, essentially forming a bottleneck. Given the low dimensional space, it becomes easier for the network to overfit, as it might learn specific patterns within this layer. Adding Gaussian noise to the latent layer makes it difficult for the network to learn specific patterns but also forces the network to generalise better. This in turn, prevents overfitting of the data.
3.5.5 Regularisation

As the model trains over the data set, some weights and/or activations in the network can get very large in magnitude. This can cause the model to be too complex in an effort to capture all the information, resulting in overfitting, producing an output which is too good to be true. This results in the model having a variance higher than that of noise. Regularisation penalises high magnitude weights or high magnitude activation functions of the model in an effort to keep the weights or activations reasonable reasonably small. When the regularisation is applied to activation functions, it is referred to as activation regularisation and in case of weights, it is called weight regularisation.

L2 regularisation is one of the most common regularisation functions used. It penalises the heavier weights and/or activations by taking the L2(euclidean) norm of the weights multiplies it by a user defined regularisation factor and adds it to the loss function. In the scope of this project, regularisation refers to activation regularisation.

3.6 Evaluation Parameters

The performance of the network in terms of the mathematical model, and the actual output must be validated in order to understand how well the networks has learned and performed. For this purpose Loss values, and genotype concordance have been used in this project. In the case of this project, loss values and genotype concordance provide internal and external validation respectively. To be more specific, loss value helps us validate how well we solve the optimisation problem, and genotype concordance helps validate how well we solve genotype reconstruction problem.

3.6.1 Loss Values

Loss values is the measure of how well the model has predict compared to the actual information at hand, and thereby gives an indication if how well the model performs. Loss values are calculated by loss functions during each epoch. During the course of the training, the loss values initially drop drastically as they get minimised and should plateau out to a reasonably stable value over time. Any long
term upward deviation in the model might indicate issues which might need to be investigated further.

3.6.2 Genotype Concordance

While the loss function gives an idea how well the model has learned, it is not easy to tell if the model accurately predicts the SNP values. This demands the need for a metric which uses actual genotype values instead of probabilities i.e., if the three possible genotype values are 0, 1 and 2 and their respective predicted probabilities are 15%, 12% and 73% then the metric would take the SNP value to be 2. This would then be compared against the true SNP values to check whether the prediction was right or not. Genotype concordance is one such metric. Furthermore, while evaluating any AI model, it can be hazardous to use the training metric (in this case the loss function) for evaluation as well, even when using a different dataset. The model could have optimised itself for that specific metric i.e., one used in the training, resulting in good metrics but bad results. In other words, a generalised model should perform well on both the training metric (loss function) and an independent evaluation metric i.e., in this case, genotype concordance.

Genotype concordance measures the fraction of SNPs, which have been correctly labelled, in a range from 0 to 1. For example, out of a 1000 genotypes, if the model correctly predicts 850 of them, then the genotype concordance is 85%.

4 Materials

4.1 Data

The data for this project was provided by Lantmännen, and consisted of two Excel files. The first file contained 9 phenotypic and 16200 SNP values for 500 wheat samples. The second file consisted of the 16128 SNP names, the chromosome they are located on and their position on the chromosome. Since the number of SNPs were not equal in both files, only those SNPs common to both files were to be used. Since using Excel files was not the most memory efficient approach, the files
needed to be converted into .ped and .map which are compatible with PLINK1.9 (Chang et al., 2015). PLINK is a software for genetic association studies which is also used to create memory efficient genomic files. In order to achieve both the goals a custom Python script was written which read the Excel files, extracted the common SNPs and then converted them into .ped and .map files. The .ped file contains the information regarding the SNP values and .map file contains the SNP name and their location in the genome. Both .ped and .map files are header-less files i.e., there are no column names in the file and therefore, the SNPs are sorted according their genomic position in both the files.

The .ped and .map files were given as input to PLINK1.9, which converted them into .bed, .bim and .fam files. The .bed file contains the raw SNP data for each SNP and sample in a binary format assuming that there are reference and alternative alleles at each position i.e., 00,10,11,01, wherein 00 signifies that both alleles are reference alleles, 10 indicates that there is one of each allele present at the position, 11 indicates both alleles are alternative and, 01 indicates a missing genotype. The .bim file stores the name and position of each SNP contained in the .bed file. The .fam file contains the metadata of each individual such as which family they belong to and how they are related to each other within the family and outside the family.

The data was further normalised to fit them within the range of 0 to 1. After normalisation, 0 indicates homozygous for reference alleles, 0.5 indicates heterozygosity, and 1 indicates homozygous for alternate alleles.

4.2 Hardware

Computational resources on the Rackham and Snowy clusters on UPPMAX (located in Uppsala) and Kebnekaise on HPC2N (located in Umeå) were allocated for the purpose of this project. Rackham provides only CPUs for processing, whereas Kebnekaise and Snowy have access to both CPUs and GPUs. The model file conversion and the model training were done primarily on Rackham. Snowy was utilised when Rackham cores were not available.
4.3 Software

Tensorflow 2 (Abadi et al., 2015) and Keras (Chollet et al., 2015) were used to implement and train the neural network model. Tensorflow is a high performance machine learning package for Python3.5 and higher. Keras is a deep learning package for Python which runs on top of the Tensorflow module. Despite being implemented in Python, the core modules of Tensorflow are written in C and C++ in order to improve performance efficiency. As C and C++ are compiled languages, they are faster and more efficient at runtime as compared to Python, which is an interpreted language, making the former faster and more efficient at number crunching.

The code base for this project was forked from the existing code for Convolutional Autoencoder for SNP data models at the Nettelblad group. The pre-existing code was designed for dimensionality reduction and therefore needed to be adapted for the purpose of this project.

4.4 Version Control

Version control for this project was done using GitHub. Since the code for the autoencoder models of the Nettelblad group was already available on GitHub, the code base for this project was created from a fork of the pre-existing code. GitHub also made it easy to keep track of code modifications and bug fixes safely within the group.

5 Methods

The primary objective of the project is to identify causal SNPs for the phenotype of interest in wheat and altering the standard autoencoder architecture to optimise the network for this purpose. Initially only two models were planned for the purpose of the project - a standard autoencoder model as control and, an altered model with lateral phenotype injection and untrained decoders. But due to insufficient data
points in the actual phenotype and to have a better understanding and control over the behaviour of the network, a third model was created which uses an artificial phenotype generated by mathematically manipulating SNP values.

5.1 Model 1 - Simple Autoencoder: Encoder And Decoder

The first model involved training a standard convolutional variational autoencoder (CVAE) with SNP data. This was done in order to create a control model for the project and also to refine the architecture to achieve acceptable performance in terms of reconstruction error. This model served as the base for the development of the other two models.

The original model architecture developed at the Nettelblad group consisted of the entire autoencoder i.e., encoder and decoder as a single monocoque unit. For the purpose of this project the encoder and decoder had to be decoupled into separate units (refer Fig. 5). This decoupling allows for the modification to be done to the latent layer, which is required in the future models. This also help keep the basic architecture consistent across the models.

Figure 5: Architecture of Model 1 - Blue boxes denote the encoder and decoder sections of the autoencoder. The red box represents the latent layer, and the yellow boxes represent the original and reconstructed dataset
Once a working model was developed, and adapted, different architectural and parametric changes were done in order to identify factors which may contribute improvement or depreciation of the model performance. The parameters thus changed included changes to batch sizes, learning rates, learning rate schemes, loss functions, noise, and regularisation. Architectural changes consisted of increasing the dimension of the latent layer, dropouts rate for layers in the network, and the number of dense layers. The changes were tested using the wheat dataset.

5.2 Model 2 - Trained Encoder With Phenotype Injection And Two Untrained Decoders

The second model is derived from the first model but has been significantly altered in terms of workflow and architecture. This model follows a two step training process, the first step of which involves training a standard autoencoder as described in Model 1. In the second step, the trained encoder from model is decoupled and two new decoders are initialised. The first decoder is coupled with the encoder with the latent layer unmodified. Whereas the second decoder is coupled with the encoder from model but with an extra phenotype dimension added to the latent layer (refer Fig. 6).

The two step training process was designed to better understand the impact of injecting the phenotype into the network. Running a completely untrained model with phenotype injection, while doable, brings more variables that might affect model reconstruction, making it difficult to precisely control the experimental conditions. This includes the initial weights of the model and the effects of backpropagation throughout the model, making it difficult to isolate the effects of the phenotype injection.

This issue was solved by training an autoencoder as described in Model 1 and then extracting the trained encoder. This encoder provides a standardised initial condition to better isolate and understand the effects of phenotype injection on data reconstruction. Then, two untrained decoders are initialised, in which the gradient propagation to the encoder side is disabled. These decoders provide blank slates which are not subject to any past training dynamics. As mentioned before, one of
Figure 6: Architecture of Model 2 - The model undergoes two training steps. (a) The first training step. The model being trained in this step is identical to Model 1. (b) This model consists of the encoder and latent layer (red bordered box) from Model 1 coupled with an untrained encoder (purple box). The orange box represents the reconstructed dataset from this model. (c) This model consists of the encoder and latent layer (red bordered box) from Model 1 coupled to an untrained decoder (purple box) but with the phenotype values (pale green boxes) being injected to the latent layer. The green box represents the reconstructed dataset from model (c). In the second training step the encoder and latent layer are decoupled and then coupled to two untrained encoders as shown in (b) and (c) and trained in parallel. The dot-dashed arrow represents the disabled backpropagation from decoder to encoder.
the decoders is coupled with the encoder without any modifications (referred to as New Trained Model). The second decoder is coupled with the same encoder but has been modified programatically to inject phenotype values into the latent layer during the training (referred to as Modified Training Model).

Since the idea of the project is to identify the causal SNPs for a phenotype, it is essential to map the effects of phenotype injection to genotype reconstruction. Using a natural phenotype would have been a black box approach as we do not know which SNPs are actually responsible for the phenotype and therefore, make it difficult to identify which of the reconstructed SNPs are actually the ones responsible. Therefore, artificial phenotypes were generated by multiplying the SNP values of a specific SNP, in the dataset by 100. This resulted in a direct one to one relationship between the SNP and the phenotype. By using artificial phenotype, we know the exact SNP linked to the phenotype. This allows us to compare the reconstruction of the phenotype linked SNP with other SNPs, thereby helping us understand the effects of phenotype injection.

The dataset does provide natural phenotypes, one of which was Bread Volume. Initially this phenotype was to be used for training the model, but the fact that only 275 out of the 500 samples had associated bread volume values also strengthened the case for the use of artificial phenotypes.

Once a working model was developed, and adapted, different architectural and parametric changes were done in order to all the models, in order to identify variables that improve model performance. The parameters thus changed included changes to batch sizes, learning rates, learning rate schemes, loss functions, noise, and regularisation. Architectural changes consisted of increasing the dimension of the latent layer(excluding the phenotype), dropouts rate for layers in the network, and the number of dense layers. The summary of the final hyper-parameters are given in the Table 1 and the architecture parameters are described in Table 2.
Table 1: Hyperparameter summary for the final model

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial learning rate</td>
<td>2.75e-04</td>
</tr>
<tr>
<td>Decay rate</td>
<td>0.999</td>
</tr>
<tr>
<td>Batch Size</td>
<td>20</td>
</tr>
<tr>
<td>Standard Noise</td>
<td>0.25</td>
</tr>
</tbody>
</table>

6 Result

The models worked well in terms of reconstruction and prediction. Genotype concordance was used as the performance metric. Percentage increase in concordance was used as a measure for SNP prediction. Baseline genotype concordance was established by assuming that all samples of each SNP in the dataset belonged to the most common genotype of that particular SNP. This was then used as reference for the other concordance measures. By exceeding the baseline concordance, the model showed that its predictions had outdone simple guessing.

6.1 Model Performance

One of the key requirement of this project was the improved performance of the modified model with phenotype injection in comparison to the standard autoencoder model. On plotting the losses for the final version of the models i.e., Model 1, New Training Model, and Modified Training Model, the Modified Training Model showed the lowest stable loss value, followed by New Training and Model 1 (Fig. 7). Furthermore, while the total concordance of all the models surpassed the baseline, Modified Training Model had the highest concordance, again followed by New Training and Model 1 (Fig. 8). These two observations confirms that the modified model performs better both in terms of learning and genotype reconstruction.
Table 2: Architecture Table for the final model

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Layer</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x</td>
<td>BiasWeightLayer</td>
<td>-</td>
</tr>
<tr>
<td>1x</td>
<td>Conv1D</td>
<td>filters: 10, kernel size: 5, padding: same, strides: 1, activation: elu</td>
</tr>
<tr>
<td>3x</td>
<td>BatchNormalization</td>
<td>-</td>
</tr>
<tr>
<td>1x</td>
<td>MaxPool1D</td>
<td>pool size: 5, strides: 3, padding: same</td>
</tr>
<tr>
<td>1x</td>
<td>Conv1D</td>
<td>filters: 10, kernel size: 5, padding: same, activation: elu</td>
</tr>
<tr>
<td>1x</td>
<td>Flatten</td>
<td>-</td>
</tr>
<tr>
<td>4x</td>
<td>Dropout</td>
<td>rate: 0.01</td>
</tr>
<tr>
<td>4x</td>
<td>Dense</td>
<td>units: 20, activation: elu</td>
</tr>
<tr>
<td>1x</td>
<td>Dense</td>
<td>units: 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decoder</th>
<th>Layer</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x</td>
<td>Dense</td>
<td>units: 20, activation: elu</td>
</tr>
<tr>
<td>4x</td>
<td>Dropout</td>
<td>rate: 0.01</td>
</tr>
<tr>
<td>1x</td>
<td>Dense</td>
<td>units: 10, activation: elu</td>
</tr>
<tr>
<td>1x</td>
<td>RepeatVector</td>
<td>n: ns[1]</td>
</tr>
<tr>
<td>1x</td>
<td>Conv1D</td>
<td>filters: 13, kernel size: 1, padding: same, activation: elu</td>
</tr>
<tr>
<td>4x</td>
<td>BatchNormalization</td>
<td>-</td>
</tr>
<tr>
<td>1x</td>
<td>Reshape</td>
<td>target shape: (ns[1],1,13)</td>
</tr>
<tr>
<td>1x</td>
<td>Conv2DTranspose</td>
<td>filters: 10, strides: (3,1), kernel size: (5,1), padding: same, activation: elu</td>
</tr>
<tr>
<td>1x</td>
<td>Reshape</td>
<td>target shape: (ns[1]*3,10)</td>
</tr>
<tr>
<td>1x</td>
<td>Conv1D</td>
<td>filters: 10, kernel size: 5, padding: same, activation: elu</td>
</tr>
<tr>
<td>3x</td>
<td>Conv1D</td>
<td>filters: 10, kernel size: 1, padding: same, activation: elu</td>
</tr>
<tr>
<td>1x</td>
<td>BiasWeightLayer</td>
<td>-</td>
</tr>
<tr>
<td>1x</td>
<td>Conv1D</td>
<td>filters: 4, kernel size: 1, padding: same</td>
</tr>
</tbody>
</table>
Loss function was one of the most important training parameters for the model. Initially CCE was considered as the loss function for the model, and it was found that for a given constant learning rate ($\approx 10^{-4}$), test models running CCE were able to achieve concordance only slightly higher than the baseline. Following this KLD was implemented as a part of the CVAE logic and it was found that for the same parameters, models running KLD as loss function with constant learning rate, were consistently able to cross the baseline concordance, while also exhibiting better convergence.

As the dataset consisted of only 500 samples, it was important to fix the initial learning rate, since using a higher learning rate would make the training faster but result in inadequate convergence, whereas an extremely low learning rate might result in model not converging fast enough. During the training test of the models with learning rate of $\approx 10^{-4}$ and $\approx 10^{-5}$, it was found that while both learning rates achieved convergence for all models in less than 100 epochs, only the former achieved baseline concordance, whereas the latter displayed genotype concordance below the baseline value. It was therefore more favourable to keep the learning rate $\leq 10^{-4}$ as it gave acceptable performance in terms of convergence, concordance, and time. Furthermore to reduce the risk of overfitting, and to keep the training dynamic, the final model used exponential decay as the learning scheme instead of a constant learning rate, although the initial learning rate was kept the same. The final models, using this scheme converged for all three cases - Model 1, New Training Model, and Modified Training Model. All the three models also surpassed the baseline concordance values.

Batch size had significant impact on the performance of the model due to the small sample size of the dataset used for this project. Initially while training Model 1, a batch size of 100 was used for the model. While this resulted in low training times, it also displayed inadequate convergence in the tests along with low genotype concordance. Following this, a batch size of 20 was picked. This increased the training time for Model 1 as there were now five times the number of batches in each training iteration as when compared to the previous batch size, but this also resulted in model convergence being achieved in the test and a slightly better genotype concordance. All other models inherited the same batch size and per-
formed satisfactorily in all test cases and was used in the final model as well. In short, smaller batch size was found to be beneficial in terms of convergence and concordance, albeit with increased training time.

Varying regularisation factors of the weights were tested alongside different loss functions and learning rates but no significant difference in performance in all test cases with respect to regularisation. Also, increasing the depth of the model had a significant effect on the model performance, with model concordances matching and crossing the baseline. Among the models running the deeper architecture, the ones using BiasWeightLayers in the initial layers demonstrated improved performance in terms of genotype concordance as both Modified Training Model and New Training Model had near identical performance above the baseline. In contrast, for models using LocallyConnected1D in the initial layers, Modified Training Model showed concordance values significantly below those of New Training Model and Model 1.
The final version of Model 2 - with New Training Model and Modified Training Model exceeded the baseline value concordance significantly and also had their respective losses converge. The final model architecture (refer Table 2) and hyperparameters (refer Table 1) achieved all the goals of this project as intended. The Modified Training Model, achieved convergence at a lower loss value than New Training Model and Model 1, as shown in Figure 7. This showed that injecting phenotype (in this case artificial phenotype) values into the latent layer optimises the model in terms of convergence and allows it hit lower stable loss values as compared to standard models.
To identify the effects of phenotype injection on genotypic reconstruction, concordance of the SNP linked to the artificial phenotype, referred to as SNP-specific concordance, was also tracked for all three models along with the overall model concordance. By the end of the training, the Modified Training Model, New Training Model, and Model 1 had achieved a concordance of 0.8412, 0.8384, and 0.8365 respectively. This showed that the phenotype injection in the Modified Training Model had resulted in a slightly better overall genotype reconstruction in comparison to both the New Training Model, and Model 1. The SNP-specific concordance for the phenotype-linked SNP (referred to as SNP-Art) for the Modified Training Model was 0.972, which was higher than the overall model concordance, and also
outperformed SNP-specific concordance of New Training Model and Model 1, both of which showed SNP-Art concordance below the baseline (refer Fig. 8). This indicated that phenotype injection implemented in the Modified Training Model drastically improved the genotype reconstruction of the SNP to which the phenotype was linked, in comparison to the traditional autoencoder models i.e., New Training Model and Model 1.

![Genotype concordance: SNP Specific](image.png)

*Figure 9: SNP-specific concordance graph - SNPs from Model 1 (unmodified), New Training Model and Modified Training Model are represented by orange, blue, and green lines respectively. Solid lines correspond to phenotype-linked SNP (SNP-Art), dotted lines represent SNP with similar distribution as SNP-Art (SNP-similar) and dashed lines represent a random SNP (RandomSNP).*

The concordances of two other SNPs were also tracked for all three models (refer Fig. 9), along with the phenotype-linked SNP. One of these was picked for hav-
ing similar allelic distribution as the phenotype linked SNP (referred to as SNP-
similar), while the other was picked at random (referred to as RandomSNP). This
was done to better isolate and understand the effects of phenotype injection on the
model. As seen in Figure 9, for the Modified Training Model, SNP-Art demon-
strated significantly higher concordance as compared to the similar and random
SNP. Furthermore, the concordances of SNP-similar and RandomSNP in the Mod-
ified Training Model were comparable to their counterparts in the other two mod-
els. This showed that phenotype injection only affected the reconstruction that
SNP which is linked to the phenotype and did not significantly affect reconstruc-
tion other unrelated SNPs even if they had similar distributions.

6.2 SNP Prediction

As some SNPs in the data set have very little allele variation, they attain high con-
cordance rates, with some even achieving 100% concordance early on in the first
training step and stabilise at those values. Some of these SNPs show no significant
change in concordance even in the second training step. This creates a situation
where high concordance values for certain SNPs may not necessarily indicate any
relationship with the phenotype, even though phenotype injection does increase
concordance for the phenotype linked SNP.

This problem was circumvented by measuring the percentage increase in concor-
dance between Modified Training Model and New Training Model for each SNP.
Since the two models are identical in all aspects, except for the phenotype injection
in the latter, measuring the increase in concordance allows us to observe the rela-
tive change in concordance of the SNPs between the two models. This is a better
measure than absolute concordance values in order to identify the causal SNPs, as
the output is not obscured by stable and high concordance values. To better visu-
alise this, a concordance terrain plot was created, wherein the SNPs are ordered
along the x-axis based on their genomic position and the percentage concordance
increase along the y-axis.

The highest peak in the concordance terrain plot (see Fig. 10) was that of the 4434th
SNP in the dataset. The number corresponds to the index of the SNP used to create
Figure 10: Concordance Terrain - Wheat is allopolyploid, hexaploid and has 7 sets of chromosomes. Each chromosome is represented by the chromosome number (1-7) and an alphabet (A, B, and D). The alphabets, A, B, and D correspond to the copy of the chromosome. In this graph the chromosomes are arranged from 1A to 7D. The chromosomes are represented by alternating colours where each colour spans a full chromosome.
the artificial phenotype which was used to train the Modified Training Model. This shows that the model has predicted/identified the causal SNP successfully, thereby achieving one of the key goals of the project.

7 Discussion

As seen in Figure 7 the losses for all three models converge and the modified model has the lowest loss value. Furthermore, there is greater difference between the losses of Modified Training Model and New Training Model, than that between New Training Model and Model 1. This trend is further validated by overall the genotype concordance values of the models (refer Section 6.1 and Fig. 8), wherein the difference in concordance values of between New Training Model and Model 1 is less than the difference between Modified Training Model and New Training Model.

This can be explained by the fact that using the trained encoder from Model 1 in the second training step, gives a significant edge to the New Training Model as the encoder does not need to be trained from scratch, and therefore simply improves upon it. Also, it is important to note that architecturally, there is very little difference between Model 1 and New Training Model, whereas Modified Training Model adds extra information into the latent layer in the form of phenotype injection, in effect adding an extra node to the latent layer which results new connections with the adjacent layers fundamentally affecting the networks dynamics. The Modified Training Model also inherits the same advantage as the New Training Model from the pre-trained encoder from Model 1 and further improves upon this inherent advantage since phenotype injection provides additional information to the model which helps in better learning, leading to significant improvement in loss convergence and genotype concordance. This indicates that there is a tangible improvement in data reconstruction due to phenotype injection, as compared to the traditional models.

It was observed that phenotype injection has had a strong effect on the overall model performance. While, the overall concordance of the modified model was
only slightly higher than the other two models, the SNP-specific concordance of the phenotype-linked SNP for the Modified Training Model was markedly higher, when compared to those from Model 1 and New Training Model. This indicated that the phenotype injection might have affected the reconstruction of only the phenotype linked SNP. In the plot described in Figure 9, it can be seen that SNP-similar does not show any significant difference in concordance when compared to its counterparts from Model 1 and New Training Model. This is in sharp contrast to the SNP-Art, which, as mentioned before, shows significant improvement to genotype concordance in the case of Modified Training Model as can be seen in both Figure 8 and Figure 9. The aforementioned observations thus indicated that phenotype injection affected only that SNP to which the phenotype is related and did not affect unrelated SNPs.

In the concordance terrain plot (see Fig. 10) while the highest peak corresponded to the index of the phenotype linked SNP, it was found that a block of SNPs adjacent to the highest peak also showed a marked increase in their concordances. The concordance increase of adjacent SNP blocks indicate that the phenotype linked SNP has intra-chromosome linkage with other SNPs on chromosome 2B. This is expected since this is an artificially selected variety of wheat and hence, some of the samples are related to each other within a few generations. As the SNPs in that block are on the same chromosome (2B), it is possible that linkage with the phenotype linked SNP improves the reconstruction of these adjacent SNPs, even though the magnitude of improvement is significantly less than the highest peak. This also demonstrates a key strength of the model as it is able to capture other SNPs whose genotypic distributions have some sort of association with the phenotype.

It is important to note that the SNPs in this dataset are not evenly spaced out. Therefore, SNPs of adjacent indices do not necessarily mean that they are immediately adjacent in the genome. Due to the lack of time, the linkage hypothesis could not be tested. This can be further explored by mapping the physical distances to mapping distances and analysing the effects this has on concordance increase. The linkage effects cannot be ruled out without further studies since the improvement in concordance is too pronounced to ignore. This would help isolate the effects of
linkage on concordance increases, and help identify other biological factors that might influence the results.

It was also noted that the peaks on chromosome 2A and 2D (depicted on either side of 2B) followed a pattern similar to that observed on chromosome 2B with respect to the phenotype linked SNP. Both 2A and 2D exhibited peaks which showed as much as 30% increase in concordance, along with significant improvements in the concordance of the SNPs on either side of the peaks. One possible explanation for the peaks in adjacent chromosomes of 2B could be that the SNPs associated with peaks on 2A, 2B, and 2D are present on paralogous genes. It is possible that these paralogous genes follow similar allelic distribution patterns due to selection in this population which might be promoting such paralogs. This kind of allelic distribution patterns could explain the improvement in the concordance of the SNPs which were not consciously linked with the phenotype.

Although, IWGSC Reference Wheat Genome browser (Consortium (IWGSC) et al., 2018) with the Chinese Spring variety of wheat as the reference genome was used to check if the paralogs were associated with the genomic positions of interest, no conclusive evidence was found for the same. This could be because the cultivar used in this project is not the same as the one in the genome browser.

Another possible explanation for the peaks on the adjacent chromosome could be the presence of non-paralogous co-varying SNPs. It is possible that the SNP used for creating the artificial phenotype shows co-variance with the SNPs on the adjacent chromosomes (2A and 2D). This could occur if these genes interact with each other over inter-chromosomal distances or if they are the part of the same cell signalling network. It is possible selection promotes such co-varying SNPs and the resulting patterns of allelic distribution related to that of the 4434th SNP (2B) could have resulted in the improvement in the concordance of the peaked SNPs on 2A and 2D.

While this proof of concept model works, it is important to note that it was trained on a simple single SNP-Artificial Phenotype relationship, where the phenotype values are pegged to the genotype values. As a result, we are not sure how the peaks on the concordance terrain would manifest in scenarios where more com-
plicated SNP-Phenotype relationship exists, such as those where a cluster of SNPs determine a phenotype. As of now, one can be sure that the highest peak corresponds to the SNP linked to the phenotype as long there is a one-to-one SNP-Phenotype relationship, whereas significance of other peaks in the model can only be speculated at best. A possible improvement to the model would be training it on a real world dataset, where the Phenotype-SNP relationship is already known. This would help better understand how the concordance terrain is affected in complex SNP-Phenotype relationship. This would also help validate the models performance in real world scenario against standard techniques such differential expression analysis.

When studying which genes or alleles influence a trait, only those associations are studied which are variable in the given population. There could be other possible genetic variants affecting the traits which are not present in the population. This bit of information needs to be kept in mind, especially in cases where the causal SNP of the phenotype has low variance. Such causal SNPs may not have the room to improve in their concordance during the training, due to their low variance. As a result, it is difficult to underpin the relationship between low variance SNPs and a phenotype. As mentioned before in section 6.2, some SNPs in the dataset showed very little allelic variation, resulting in the model learning their reconstruction perfectly within the first few epochs of the first training step. This issue could be alleviated by using dataset with larger sample size and more diverse allele distribution.

Any future extensions of this study must optimise the hyperparameters further. As this project was a first of its kind of proof-of-concept, more importance was given to developing a functional model which meets the basic goals of this project. Further improvements to the model can be done by analysing the effects of regularisation on the model. Although model variants with different levels of regularisation were planned, they could not be tested due to the lack of time.

Further improvement can be made to the model by enabling gradient propagation in New Training Model, and Modified Training Model. As of now gradient propagation to the encoder is disabled in both the models to better control the test scenario. Enabling complete gradient propagation could results in significant im-
provement in results as this would allow optimisation of weights in the encoder and further optimise the weights in the decoders as well.

8 Conclusion

While this project was not the first one to explore SNP-Phenotype relationships, it is the first machine learning based study which tried to predict causal genotypes using phenotype values. Prior to this study, all studies such as those by Das Choudhury (2020), Valenzuela et al. (2010), and Drouin et al. (2019) have attempted to use machine learning models and genotype data to identify/predict the resultant phenotype.

This project on the other hand, turned the question on its head, and successfully so. This project also builds the case for the use genomic autoencoders models in bioinformatics. With the ever increasing availability of genomic data, machine learning models such as neural network models may provide a more efficient method of converting the data to meaningful information and help broaden our understanding of biological systems. Hopefully, future studies will develop more robust and improved models, and contribute to the rise of machine learning in bioinformatics and practical plant and animal breeding.

9 Acknowledgement

I would like to express my gratitude to my supervisor Carl Nettelblad for bearing with me through this challenging and very extended thesis. I don’t think I would have learnt as much as I have without his constant support and input regarding what might be breaking the model, and as he proved time and again, the devil is in the details. I would also like to thank Kristiina Ausmees for patiently answering the most basic questions and helping me understand my own work.

I would like to thank Bengt Sennblad, the subject reader, for asking me questions
at every step of the project, and pointing out the smallest of details, without which I wouldn’t have been able to refine my ideas. I would also like to thank you for looking out for me during this pandemic.

I would be remiss if I do not mention Alfred Andersson, for being my partner in the journey into the uncharted niches of AI, the bittersweet relationship with UPP-MAX, and also for absurd memes that accurately captured the experience. I would like to thank Felix Morsbach for giving me company in lunch and in stupidity.

I would like to thank Shrija Srinivasan for being a constant source of support throughout the course of this degree and for making COVID isolation far far bearable with the daily chai breaks. I would also like to thank Ylva Jondelius for helping me navigate through the Swedish system and also for giving me the opportunity to teach you coding. Your programming problems doubled as welcome break for me and also gave me confidence to possibly start an alternate career on YouTube. Holly Halstead, I would like to thank you for introducing me to new restaurants and for giving company during Christmas.

I would also like to thank everyone who made these two years in Sweden memorable, I am sorry if I have missed out your names but trust me I am really thankful you exist.

Lastly but most importantly, I would like to thank my parents for sacrificing a lot more than I can imagine. Without their undying support, trust, and motivation, I would not have been able to explore, learn, and achieve as much as I have.
References


McCulloch WS, Pitts W. 1943. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics 5: 115–133.


