Introduction of Autoencoder And How We Utilize it in Our Study (Probably introduction and method)

Autoencoder is an artificial neural network that learns a representation of input data and reconstructs output equal to input in an unsupervised manner. To broadly divide data processing into two parts, there are encoding process and decoding process. The encoding part is where the input goes through dimension reduction through hidden layers, and reach the representation layer, usually called the latent space, which would have the least amount of neurons having the "compact summary" of the input. To be more specific, how input moves forward through each layer is by constantly updating the weight matrix and by the activation function. The decoding part is where backpropagation is done from the latent space, which in final creates output, the reconstructed input.

When "autoencoder reconstructs the input", the reconstructed input does not perfectly equal the original input. Loss is created in the encoding and decoding process, and the goal of this autoencoder is to minimize loss. The more equal the reconstructed output is with the original input, we can say that the better autoencoder learned the representation of the input. In other words, the autoencoder wants the output equal to the input by minimizing the loss from dimension reduction.

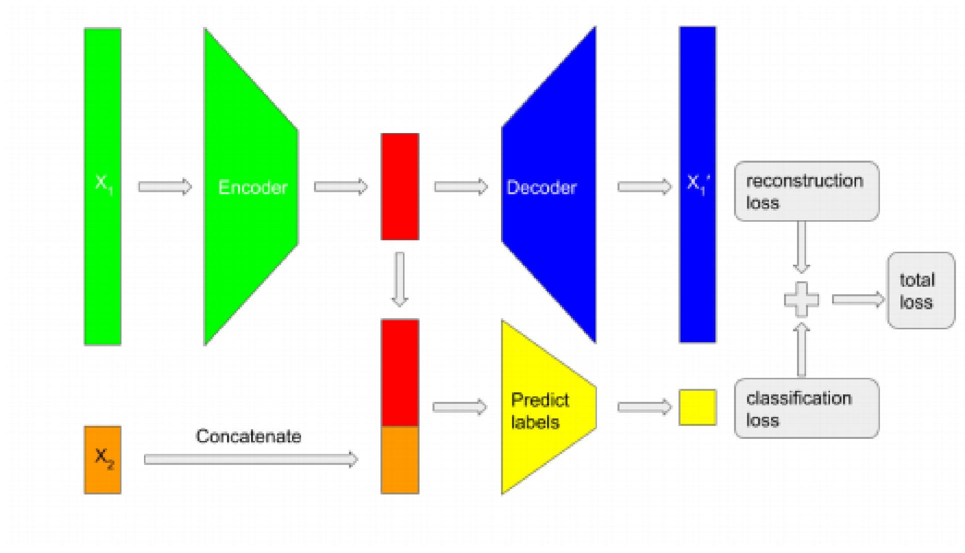

figure1. the architecture of our experiment

The general goal of our study is to predict human traits using genotype, phenotype, and polygenic risk score data with the implementation of cutting-edge machine learning techniques. The architecture of our experiment is well shown in figure 1. We first process genotype data, x1, through autoencoder and concatenate genotype data with PRS/phenotype data, x2. We then use the concatenated data as input to a classification network of the random-forest model. After the prediction, under the premise that we run both autoencoder and random forest model, the total loss in our case would be a combination of cross-entropy loss from random forest model and MSE(Mean Squared Error) loss from autoencoder. Through analyzing the loss, we verify how well our model predicts labels or traits we have chosen.

Our Progress (In regards to autoencoder specifically)

Before we insert the fullscale data set, we decided to try out with pruned data which is only 10 percent of our fullscale data. Pruned genotype data, which would be x1 in figure1, was a merged data of .bim, .bed and .fam files, comprised of 4567 people and about 500000 SNP information. The data that gets concatenated, which would be x2 in figure 1, was comprised of 4567 people's phenotype, PRS score, and PC(Principle Component). Since they were not the full data that we ultimately aim for, we adjusted CONSTANTS in the code which includes in_channel size(x1 size), number of samples, and each_fold_size. We worked in the ubuntu server with three GPUs because our code divided the input(in_channel) into two sections, each one using one GPU because of the vast size of genotype data.

In the process of running the autoencoder with pruned genotype data, we faced several problems. To begin with, we first struggled with a size mismatch error in concatenating data. The error looked like "RuntimeError:size mismatch, m1: [50 x 524497], m2: [524503 x 64] at /opt/conda /conda-bld/pytorch_1579022034529/work/aten/src/THC/generic/THCTensorMathBlas.cu: 290in_channel size" However, through close examination of input files and what m1, m2 meant, we were able to identify that the first 6 columns needed to be excluded in order to concatenate reducted genotype data with PRS/phenotype/pc data. The First 6 columns had data that were not relevant to concatenation such as subject key information.

Secondly, despite pruned data was reduced data, computation time was longer than we expected and we wanted to reuse the models. So we added saving and loading parameters in arguments. They allow saving the model as a path and loading the model that we have previously trained. Preserving models in this way would not only reduce computation time but allow us to take advantage of well-optimized models. Especially when we insert actual genotype data, loading previous models would efficiently save time.

Thirdly, the code we had only was involved with inner cross-validation. Cross-validation is indeed a useful tool to evaluate the capability of a machine learning model on unseen data. Yet, our code conducted 10-fold cross-validation iterating and designating index as a test set in each fold. In other words, a test set was used as a train set after its iteration, which may hinder finding the optimal model and make test set unpure. Also, it can become problematic if a totally new data set gets in because the accuracy will get attenuated by a new dataset. In order to solve this issue, we are planning to do outer cross-validation by splitting the train and test set with a percentage of 80% and 20%. With the 80% train set, we will run the code and save the model trained with the train set. Then, we would validify the model's predictive accuracy with the leftover 20% pure data.

Lastly, in the process of concatenation, there were null values inside some of the phenotype data columns. Approximately 400 out of 4567 were NULL and the code we initially had filled empty columns with mean value, which may have impaired the accuracy of our prediction. So we modified the code to delete subjects that had no matching phenotype data for our labels.

Further Improvements To Be Made (...Discussion part?)

Through processing the data with a sample number from the full scale data, we realized some limitations that could be developed upon.

1. While the genotype data gets processed, our code does not efficiently use GPUs. > very slow > may be solved using data loader from PyTorch(not sure)
2. What is the best middle dimension, classification of hidden layers, batch size, epoch size that gives us the best result?
3. More consideration in how we choose the label/ desired prediction output?
4. Is autoencoder the finest machine learning model to get to compact genotype data?
5. Are there additional ways to improve the accuracy of processing multi-ancestry data other than adding principle components to our PRS and phenotype data?