HapMap3 Autoencoder Lab

Marcin Kierczak

4/5/2019

Background

The purpose of this lab is to evaluate the possibility of using autoencoder as a replacement/complement to the linear dimensionality reduction methods such as PCA or MDS that are commonly used for visualising population structure in genetics. One of the main motivations is that when infering genomic kinship from a large number of markers M (large enough to capture population structure at fine level), one necesserily introduces correlations between variables, here genetic markers. This is predominantly due to the linkage disequilibrium, but also due to a large M that by pure chance introduces correlated variables to the data. This correlation structure introduces non-linearity that, in turn, makes the data not suitable for PCA/MDS since both approaches rely on computing kinship matrix determinants that, for a lot of highly correlated variables, become 0 and prevent us from computing exact solutions (division by zero is undefined).

Here, the working hypotheses is that by choosing non-linear activation functions, e.g. ReLU, one can circumvent this problem and use autoencoder approach to reduce the dimensionality by embedding kinship data in a low dimensional latent representation space that, in turn, can easily be visualised. The idea emerged during the EMBL conference Reconstructing the Human Past, Heidelberg, April 2019 in a number of discussions with Nikolay Oskolkov and other conference participants.
## Data Data comes from the HapMap phase 3 project. Here, for computational feasibility, we will be using smaller dataset. I have pre-selected 5,000 autosomal markers with call rate of 100%. We will not be dealing with missing data here although autoencoders, in contrast to PCA/MDS can.

HapMap 3 populations:

- ASW – African ancestry in Southwest USA
- CEU – Utah residents with Northern and Western European ancestry from the CEPH collection
- CHB – Han Chinese in Beijing, China
- CHD – Chinese in Metropolitan Denver, Colorado
- GIH – Gujarati Indians in Houston, Texas
- JPT – Japanese in Tokyo, Japan
- LWK – Luhya in Webuye, Kenya
- MEX – Mexican ancestry in Los Angeles, California
- MKK – Maasai in Kinyawa, Kenya
- TSI – Toscans in Italy
- YRI – Yoruba in Ibadan, Nigeria

Preparations

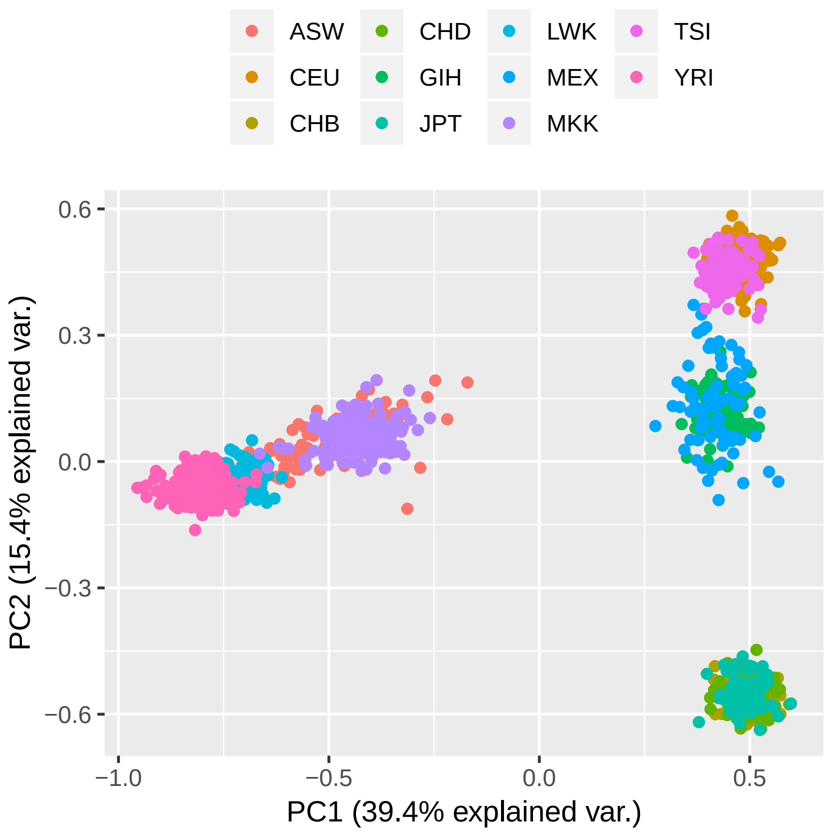
First, we need to download the dataset `autosomal_5k.rdat` R data object which is a `GenABEL` object consisting of 1184 individuals, each genotyped at 5000 loci, randomly spread across autosomes. The commented-out code was used to generate this subset of the original dataset.

Start by setting your working directory and loading necessary packages. You may need to install these packages. Some of them, like `ggbiplot` are on GitHub. You will need to Google them first and than use `devtools::install_github()` function.

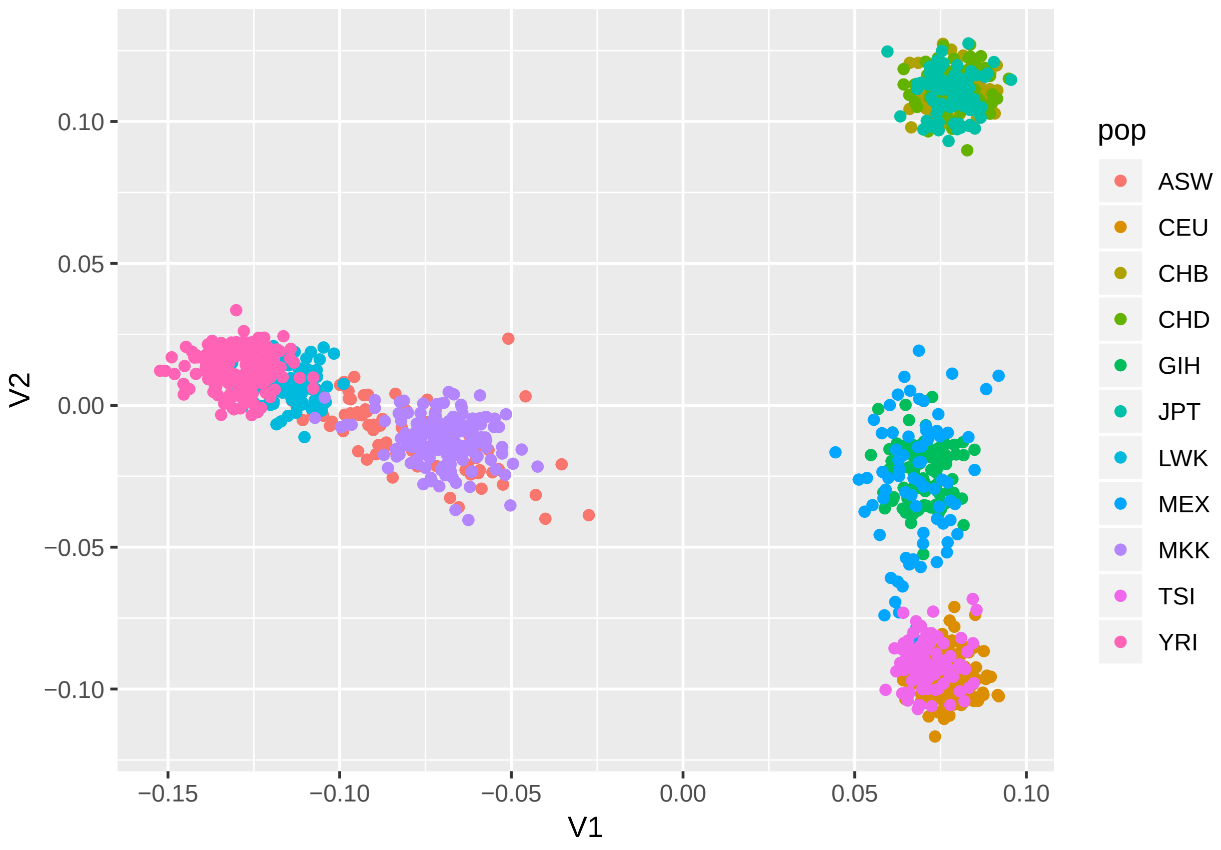
PCA and MDS

First, to have a benchmark, we will do classical PCA and MDS on the genomic kinship matrix to visualise the data.

```
# Compute genomic kinship-based distance
gkin <- ibs(data_autosomal, weight = 'freq')
dm <- as.dist(.5 - gkin) # Normalize it

# PCA
pca <- prcomp(dm)
g <- ggbiplot(pca, obs.scale = 1, var.scale = 1,
              groups = data_autosomal@phdata$population, ellipse = F,
              circle = TRUE, var.axes = F)
g <- g + scale_color_discrete(name = '')
g <- g + theme(legend.direction = 'horizontal',
               legend.position = 'top')
print(g)
```



```
# MDS
ibs <- as.data.frame(cmdscale(dm))
ibs <- cbind(ibs, pop = data_autosomal@phdata$population)
ggplot(ibs, mapping = aes(x=V1, y=V2, col=pop)) + geom_point()
```



Autoencoder

Model parameters

Below, we define model parameters: loss function set to the mean squared error and activation layer set to ReLU.

```
loss_fn <- 'binary_crossentropy'
act <- 'relu'
```

Prepare input

Input data is first normalized so that: * homozygotes AA are set to 1 * heterozygotes to 0.5 * homozygotes aa to 0 Next, the data are split into the validation (20%) and the training (80%) set.

```
# Encode genotypes
geno_matrix <- as.double(data_autosomal)
geno_tensor <- geno_matrix/2 #keras::to_categorical(geno_matrix)

# Split into the training and the validation set
n_rows <- dim(geno_tensor)[1]
train_idx <- sample(1:n_rows, size = 0.8 * n_rows, replace = F)
train_data <- geno_tensor[train_idx, ]
valid_data <- geno_tensor[-train_idx, ]
```

Define the architecture

Here, we define the architecture of the autoencoder. The autoencoder is symmetrical, i.e. decoder is the reversal of the encoder, symmetrical about the 2D latent representation layer. Some dropout layers were added for regularization to prevent overfitting.

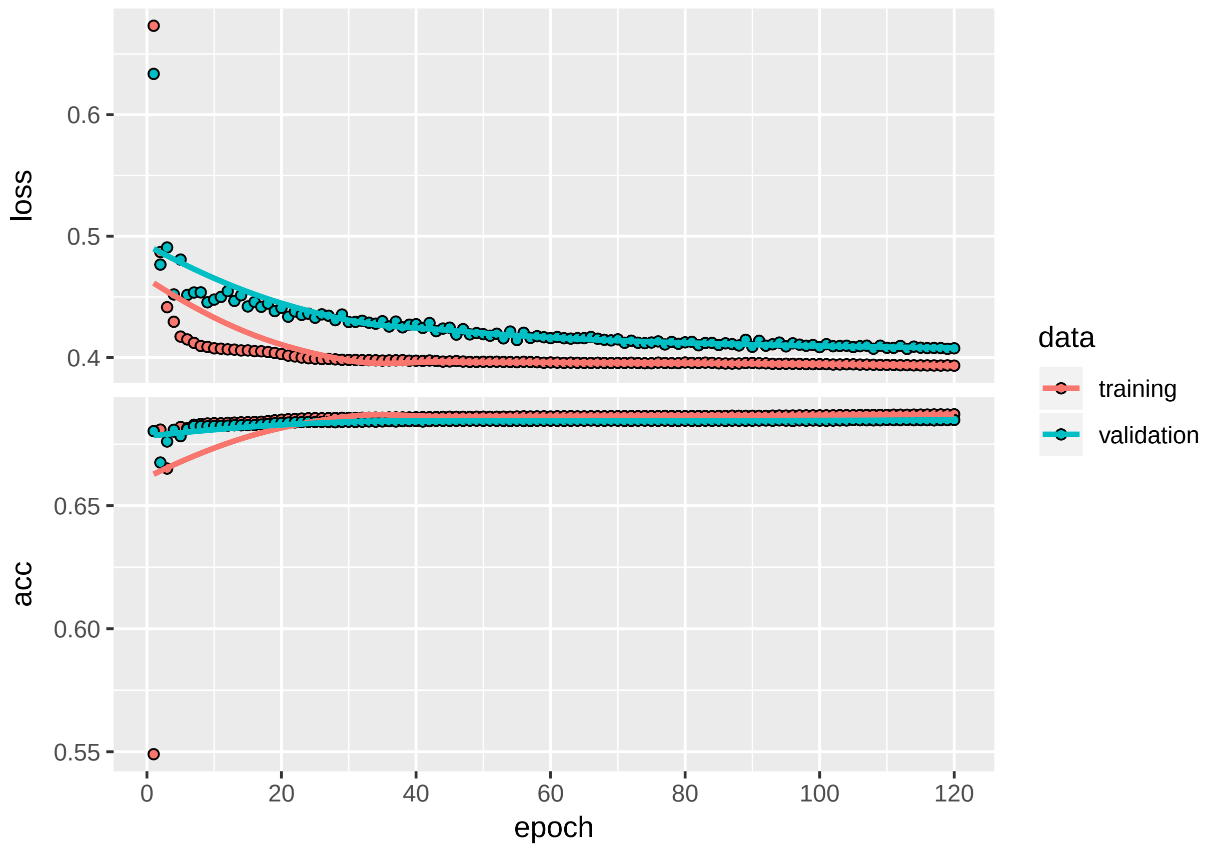
```r
input_layer <- layer_input(shape = dim(train_data)[2])
encoder <-
  input_layer %>%
  layer_dense(units = 1500, activation = act) %>%
  layer_batch_normalization() %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 500, activation = act) %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 25, activation = act) %>%
  layer_dense(units = 2) # bottleneck

decoder <-
  encoder %>%
  layer_dense(units = 25, activation = act) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 500, activation = act) %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 1500, activation = act) %>%
  layer_dense(units = dim(train_data)[2], activation = "sigmoid")

autoencoder_model <- keras_model(inputs = input_layer, outputs = decoder)

autoencoder_model %>% compile(
  loss = loss_fn,
  optimizer = 'adam',
  metrics = c('accuracy')
)

summary(autoencoder_model)
```

```
## ___________________________________________________________________________
## Layer (type)                      Output Shape                  Param #
## ===========================================================================
## input_1 (InputLayer)              (None, 5000)                  0
## ___________________________________________________________________________
## dense (Dense)                     (None, 1500)                  7501500
## ___________________________________________________________________________
## batch_normalization (BatchNormal (None, 1500)                  6000
## ___________________________________________________________________________
## dropout (Dropout)                 (None, 1500)                  0
## ___________________________________________________________________________
## dense_1 (Dense)                   (None, 500)                   750500
## ___________________________________________________________________________
## dropout_1 (Dropout)               (None, 500)                   0
## ___________________________________________________________________________
## dense_2 (Dense)                   (None, 25)                    12525
## ___________________________________________________________________________
## dense_3 (Dense)                   (None, 2)                     52
## ___________________________________________________________________________
## dense_4 (Dense)                  (None, 25)                    75
## ___________________________________________________________________________
## dropout_2 (Dropout)              (None, 25)                    0
## ___________________________________________________________________________
## dense_5 (Dense)                  (None, 500)                   13000
## ___________________________________________________________________________
## dropout_3 (Dropout)              (None, 500)                   0
## ___________________________________________________________________________
## dense_6 (Dense)                  (None, 1500)                  751500
## ___________________________________________________________________________
## dense_7 (Dense)                  (None, 5000)                  7505000
## ===========================================================================
## Total params: 16,540,152
## Trainable params: 16,537,152
## Non-trainable params: 3,000
## ___________________________________________________________________________
```

Training phase

Now the model is trained, loss and accuracy are evaluated on both the training and the external validation set. However, one should remember that with every epoch, there is som leakage of the data from the validation set!

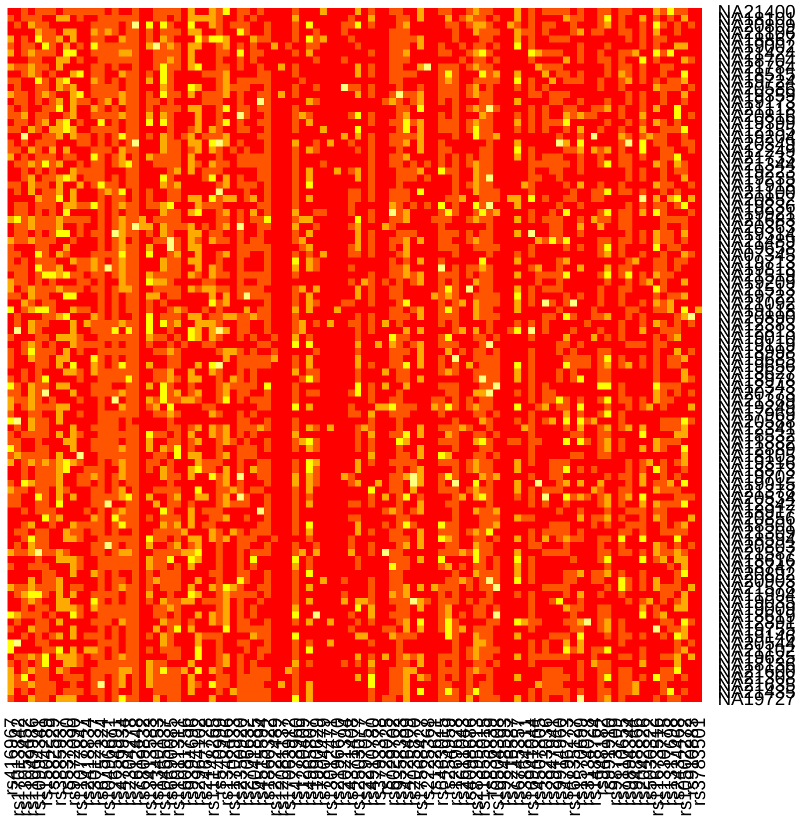
```
history <- autoencoder_model %>% fit(
  x = train_data,
  y = train_data,
  epochs = 120,
  shuffle = T,
  batch_size = 256,
  validation_data = list(valid_data, valid_data)
  #callbacks = list(checkpoint, early_stopping)
)
plot(history)
```



Visualize the embeddings

Here we visualise the difference between the original data and the reconstructed points. We visualise only the top upper fragment of the data.

```
reconstructed_points <-
  keras::predict_on_batch(autoencoder_model, x = train_data)

delta <- abs(train_data - reconstructed_points)
heatmap(delta[1:100, 1:100], Rowv = NA, Colv =  NA,
        col=heat.colors(5), scale = 'none')
```



Encoder

Following the training phase, we will build the encoder.

```r
autoencoder_weights <- autoencoder_model %>% keras::get_weights()
keras::save_model_weights_hdf5(object = autoencoder_model,
                               filepath = './autoencoder_weights.hdf5',
                               overwrite = TRUE)

encoder_model <- keras_model(inputs = input_layer, outputs = encoder)
encoder_model %>% keras::load_model_weights_hdf5(filepath = "./autoencoder_weights.hdf5",
                                                 skip_mismatch = TRUE,
                                                 by_name = F)

encoder_model %>% compile(
  loss = loss_fn,
  optimizer = 'adam',
  metrics = c('accuracy')
)
```

Embedding original data

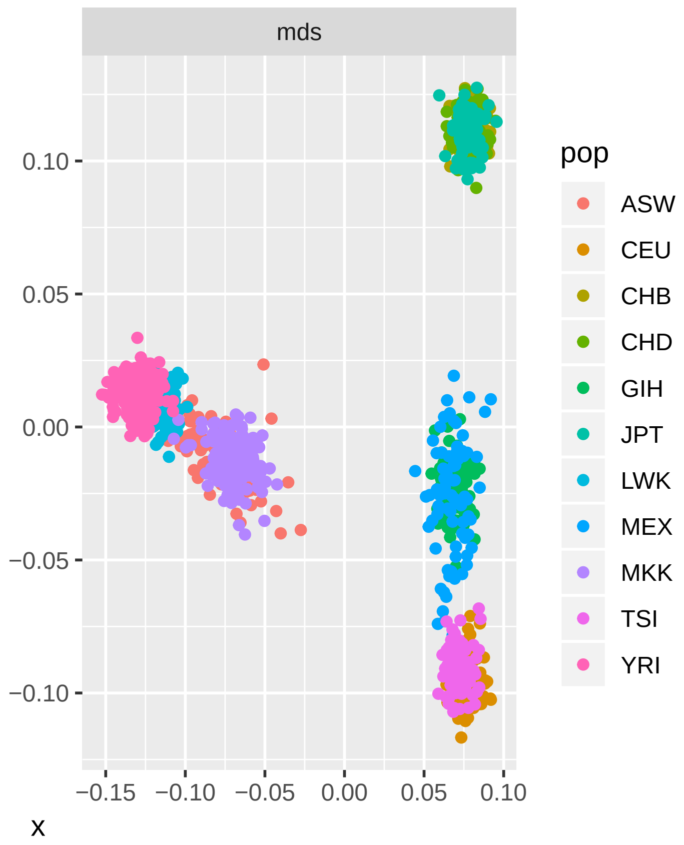
Now, original data can be embedded in the low-dimensional space using the encoder.

```r
embeded_points <-
  encoder_model %>%
  keras::predict_on_batch(x = geno_tensor)
```

Final results

Now, we can see how the embeddings compare with the MDS approach.

```r
embedded <- data.frame(embeded_points[,1:2],
                       pop = data_autosomal@phdata$population,
                       type='emb')
mds <- cbind(ibs, type='mds')
colnames(mds) <- c('x', 'y', 'pop', 'type')
colnames(embedded) <- c('x', 'y', 'pop', 'type')
dat <- rbind(embedded, mds)
dat %>% ggplot(mapping = aes(x=x, y=y, col=pop)) +
  geom_point() +
  facet_wrap(~type, scales = "free")
```



```r
ggsave('embedded.png')
```

```
## Saving 6.5 x 4.5 in image
```

```r
# checkpoint <- callback_model_checkpoint(
#   filepath = "model.hdf5",
#   save_best_only = TRUE,
#   period = 1,
#   verbose = 1
# )
#
# early_stopping <- callback_early_stopping(patience = 5)
```