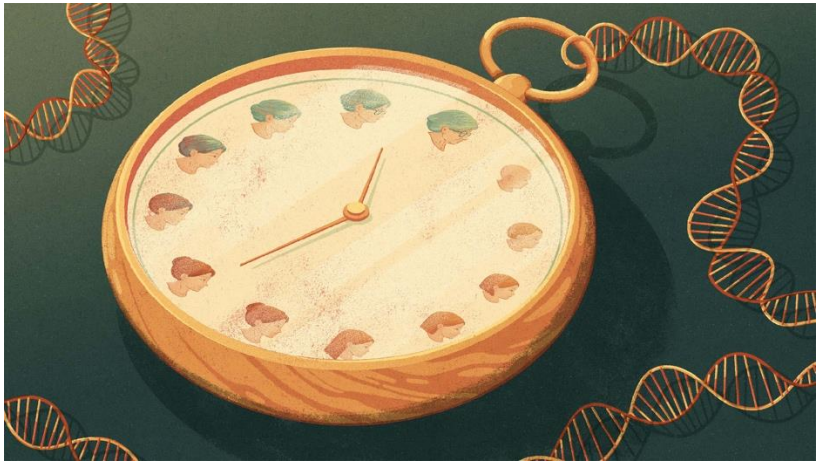
Leveraging Deep Learning with toRch for Next-Generation Epigenetic Clocks

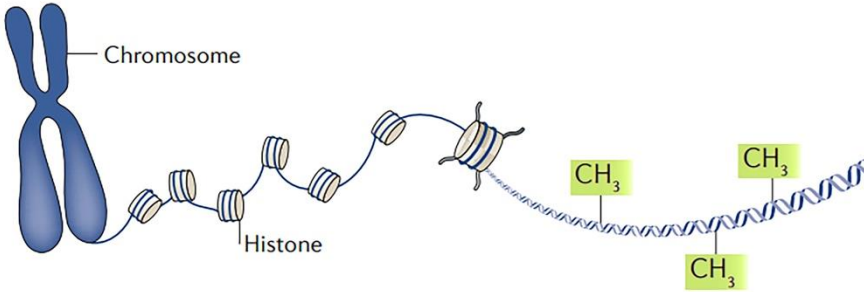


github.com/BacZemin

2025-06-24

Jamie Park (PhD candidate at VAI)

The first epigenetic clock based on DNA methylation



Using Machine learning, Methylomes can predict chronological age.

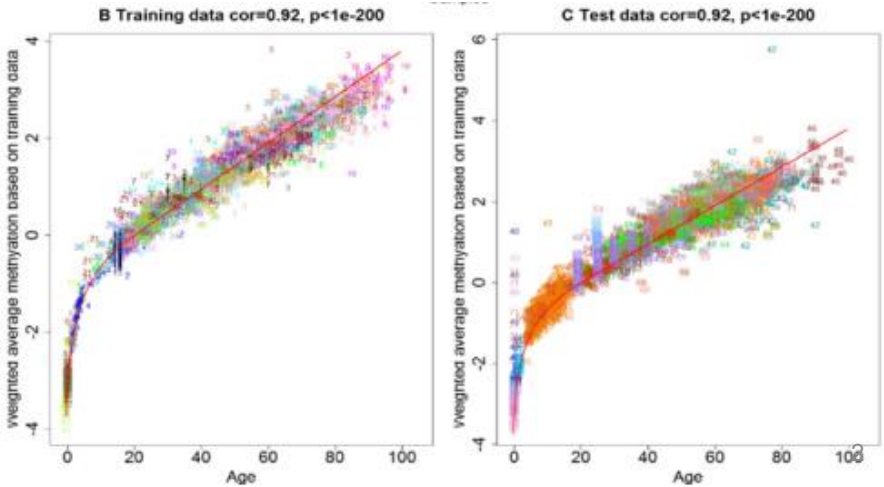
Horvath *Genome Biology*, **14**:R115
http://genomebiology.com//14/10/R115

Genome Biology

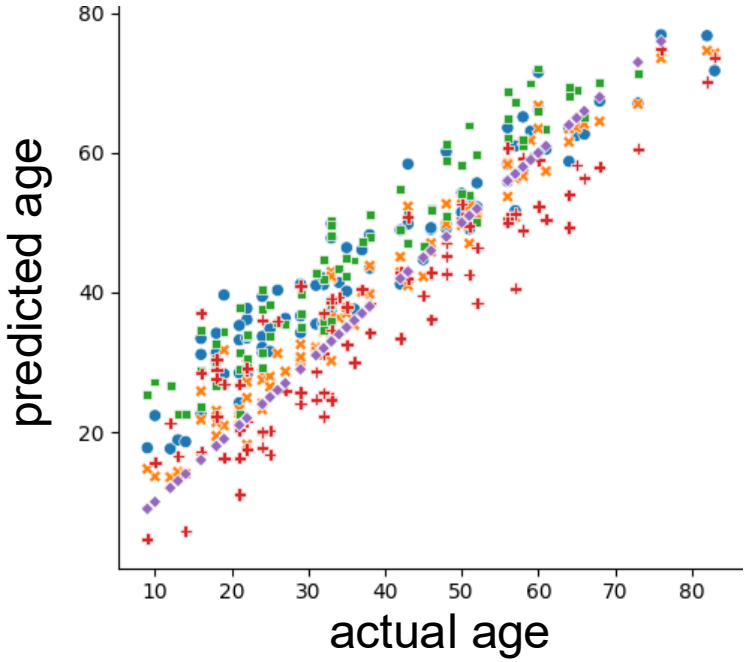
RESEARCH Open Access

DNA methylation age of human tissues and cell types

Steve Horvath[1,2,3]



The big 3 epigenetic clocks



Clock	MAE (years)	Pearson's r
Horvath Clock v1	7.4751	0.9510
Horvath Clock v2	3.1560	0.9791
Hannum Clock	8.8177	0.9679
PhenoAge Clock	6.2097	0.9158

why `elastic net` is useful for DNA methylation clocks

nature reviews genetics https://doi.org/10.1038/s41576-024-00807-w

Review article Check for updates

Epigenetic ageing clocks: statistical methods and emerging computational challenges

Andrew E. Teschendorff[1] & Steve Horvath[2]

Table 1 | DNA methylation clocks for humans

Name	Tissue	Programme	Method
Chronological age clocks			
Horvath	Multi-tissue	R	Elastic net
Hannum	Whole blood	R	Elastic net
PCHorvath1/2 PCHannum	Multi-tissue, whole blood	R	PC-based regression
Zhang	Whole blood	R	Elastic net
SkinBlood	Skin, whole blood	R	Elastic net
AltumAge	Multi-tissue	Python	Deep neural net
IntrinClock	Multi-tissue	R	Elastic net
MEAT	Skeletal muscle	R	Elastic net
Cortical clock	Brain cortex	R	Elastic net
PedBE	Buccal swabs	R	Elastic net

- elastic net is Ridge (L2) + Lasso (L1)
- these are ‘regularization’ ML methods to improve prediction.

- Usually, there’s WAY more probes (850k) than we have samples.
- Fitting a line when # parameters > # sample size doesn’t work.
- To fix this, we can add penalization (lambda) to each parameter

Ridge Lasso

$$\hat{\beta} \equiv \underset{\beta}{\operatorname{argmin}}(\|y - X\beta\|^2 + \lambda_2 \|\beta\|^2 + \lambda_1 \|\beta\|_1).$$

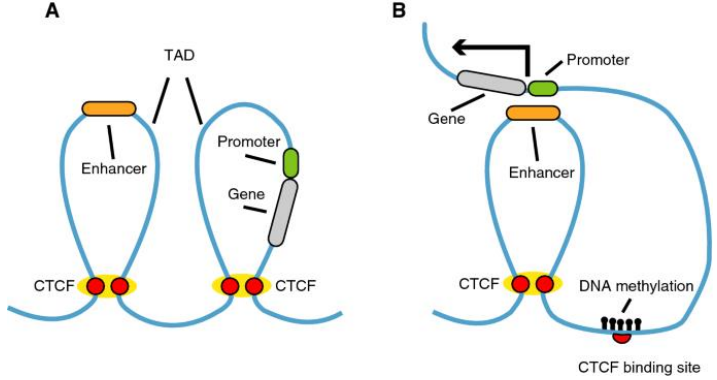
but elastic net has limits

- 1. Linearity Assumption

biological ageing is probably not that simple

- 2. Ignores Interaction between CpG

genome is a set of tangled strings in 3D



- 3. multicollinearity based feature selection

Lasso results in random selection of similar behaving probes


Article

https://doi.org/10.1038/s41467-024-47316-2

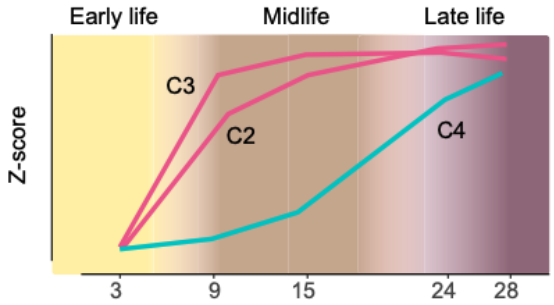
Nonlinear DNA methylation trajectories in aging male mice

Received: 10 July 2023

Accepted: 25 March 2024

Published online: 09 April 2024

Maja Olecka[1,5], Alena van Bömmel[1,5], Lena Best[2], Madlen Haase[3], Silke Foerste[1], Konstantin Riege[1], Thomas Dost[2], Stefano Flor[2], Otto W. Witte[3], Sören Franzenburg[4], Marco Groth[1], Björn von Eyss[1], Christoph Kaleta[2,6], Christiane Frahm[3,6] & Steve Hoffmann[1,6]



Recent advent of ‘deep-learning epigenetic clocks’

npj | aging

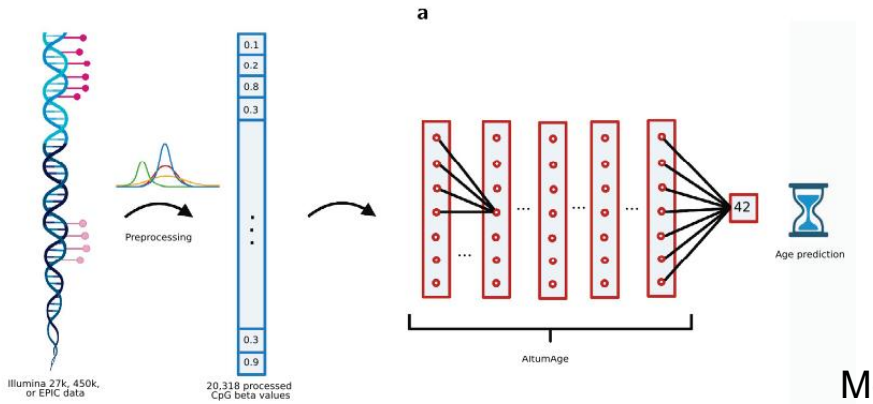
www.nature.com/npjamd

ARTICLE OPEN

Check for updates

A pan-tissue DNA-methylation epigenetic clock based on deep learning

Lucas Paulo de Lima Camillo[1], Louis R. Lapierre[2] and Ritambhara Singh[1,3]

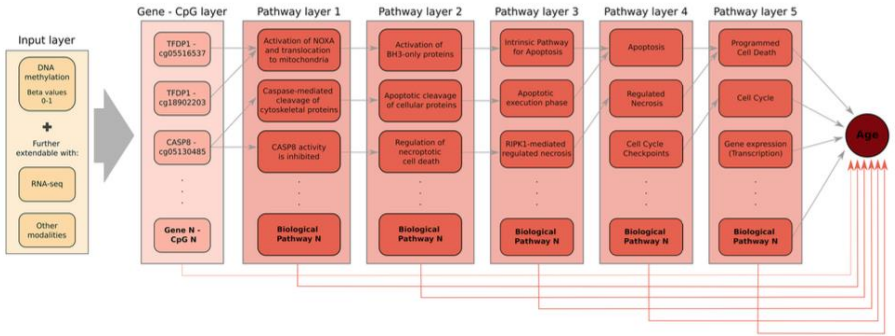


Check for updates

OPEN

Biologically informed deep learning for explainable epigenetic clocks

Aurel Prosz[1], Orsolya Pipek[2], Judit Börcsök[1,3], Gergely Palla[4,5], Zoltan Szallasi[1], Sandor Spisak[6] & István Csabai[2]



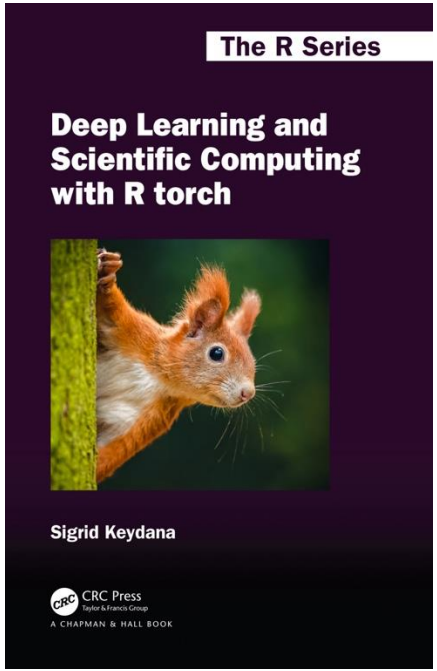
Most of these pipelines utilize `pytorch` or `tensorflow`

using torch; R interface to PyTorch

```{r check if torch is online}
library(torch)

# --- Device Configuration ---
if (cuda_is_available()) {
  device <- torch_device("cuda")
  current_gpu_index <- cuda_current_device() # Get the index of the active GPU
  num_gpus <- cuda_device_count()           # Get the total number of GPUs available
  print(paste0("✅ CUDA (GPU) is available. Set device to 'cuda'."))
  print(paste0("   Currently active GPU index: ", current_gpu_index))
  print(paste0("   Total CUDA devices found: ", num_gpus))
} else {
  device <- torch_device("cpu")
  print("⚠️ CUDA (GPU) not found. Using device: CPU")
}

seed <- 42
set.seed(seed)
torch::torch_manual_seed(seed)
```

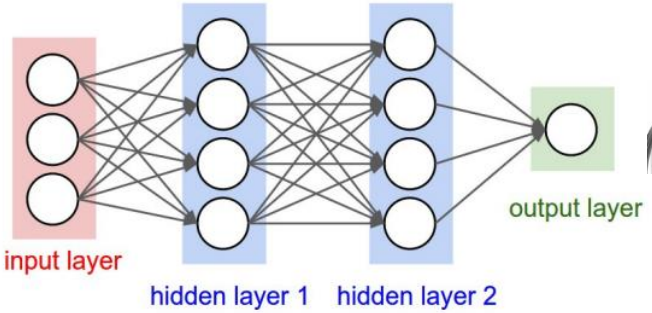


great tutorials by Sigrid Keydana
https://skeydan.github.io/Deep-Learning-and-Scientific-Computing-with-R-torch/

3-layer MLP architecture with GPU (4x NVIDIA L40S)

```{r define_mlp}
# --- Define the MLP Model ---
age_predictor_mlp <- nn_module(
  "AgePredictorMLP",
  initialize = function(input_size,
                        hidden1_size = 512,
                        hidden2_size = 256,
                        dropout_rate = 0.4) {
    self$network <- nn_sequential(
      # Layer 1
      nn_linear(input_size, hidden1_size),
      nn_batch_norm1d(hidden1_size),
      nn_relu(),
      nn_dropout(dropout_rate),
      # Layer 2
      nn_linear(hidden1_size, hidden2_size),
      nn_batch_norm1d(hidden2_size),
      nn_relu(),
      nn_dropout(dropout_rate),
      # Output Layer
      nn_linear(hidden2_size, 1)
    )
  },
  forward = function(x) {
    self$network(x)
  }
)

print("AgePredictorMLP nn_module defined.")
```



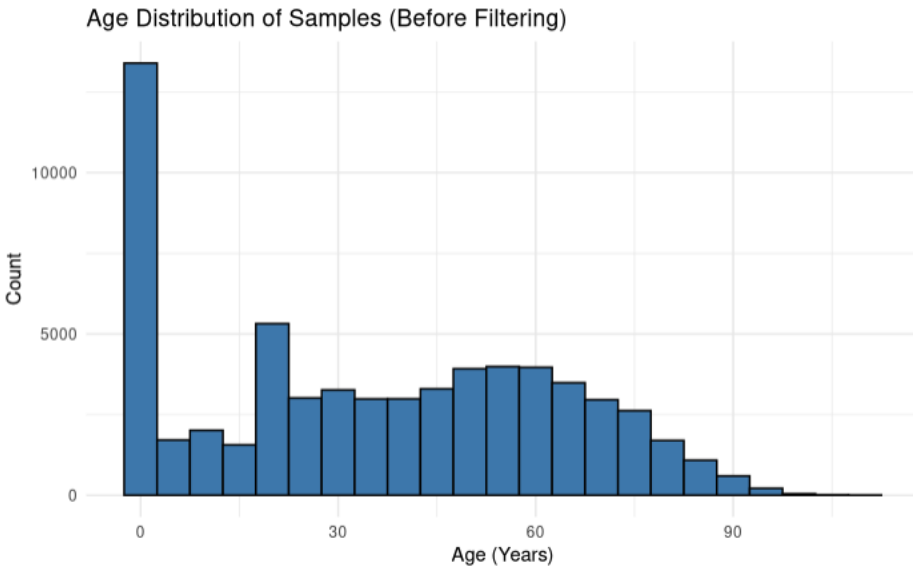

```{r}
# --- Training Hyperparameters ---
num_epochs <- 100
batch_size <- 128
learning_rate <- 1e-4
weight_decay <- 1e-5
patience <- 15
dropout_rate <- 0.4
split_ratios <- list(train = 0.70, val = 0.15, test = 0.15)
num_workers <- 0 # Set based on your system capabilities
```

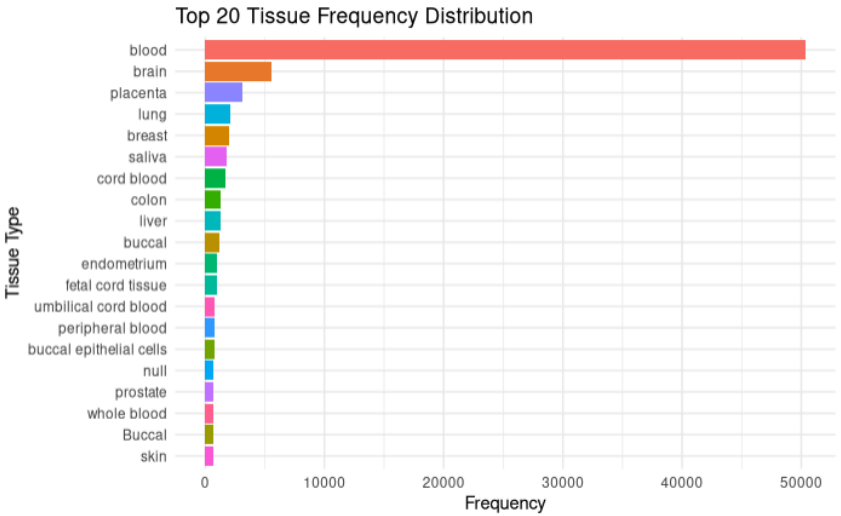
Training data: Illumina EPIC methylation microarrays from blood samples

- curated ~9000 EPIC array methylation data with metadata (age, sex, diseased)
- from the biomarkers of aging platform
  - https://bio-learn.github.io/

BIOMARKERS OF AGING CHALLENGE

BIOMARKERS OF AGING CONSORTIUM · Methuselah Foundation · VoLoFoundation





Main Training Loop

```{r}
results_summary <- list()

# Iterate through each defined probe set
for (probe_set_name in names(probe_sets_to_run)) {

  cat(paste0("\n--- Starting Experiment: ", probe_set_name, " ---\n"))

  # 1. DATA LOADING AND PREPROCESSING
  cat("1. Loading and preprocessing data...\n")
  X_full         <- py_to_r(np$load(beta_matrix_path))
  all_probe_ids  <- as.character(py_to_r(np$load(probe_ids_path,
allow_pickle=TRUE)))
  all_sample_ids <-
toupper(trimws(as.character(py_to_r(np$load(sample_ids_path,
allow_pickle=TRUE)))))
  meta_df        <- arrow::read_feather(metadata_path)

  # Align methylation data with metadata
  X_full         <- t(X_full)
  meta_dt        <- as.data.table(meta_df)[, .(gsm =
toupper(trimws(as.character(gsm))), age_years)]
  samples_dt     <- data.table(gsm = all_sample_ids, original_order =
1:length(all_sample_ids))
  meta_aligned   <- meta_dt[samples_dt, on = "gsm", nomatch=0]
[order(original_order)]
```

1. load and initialize the dataset.
   lightweight dataformat (feather, arrow or npy)
   record time point.

2. Prepare dataset
   train:validation:test = 70:15:15
   feature scaling (beta standardization)

3. Create `tensor`
   convert R matrices in to pytorch tensors

Main Training Loop

```
# 2. DATA SPLITTING & SCALING
cat("2. Splitting and scaling data...\n")
# Split data into training, validation, and test sets
split_obj      <- initial_split(data.frame(idx = 1:nrow(X_filtered)), prop =
split_ratios$train)
train_indices  <- training(split_obj)$idx
temp_indices   <- testing(split_obj)$idx
val_test_split <- initial_split(data.frame(idx = temp_indices), prop = split_ratios$val /
(split_ratios$val + split_ratios$test))
val_indices    <- training(val_test_split)$idx
test_indices   <- testing(val_test_split)$idx

# Scale features based on the training set
train_means    <- colMeans(X_filtered[train_indices, ], na.rm = TRUE)
train_sds      <- apply(X_filtered[train_indices, ], 2, sd, na.rm = TRUE)
train_sds[is.na(train_sds) | train_sds == 0] <- 1 # Avoid division by zero

X_train_scaled <- scale(X_filtered[train_indices, ], center = train_means, scale =
train_sds)
X_val_scaled   <- scale(X_filtered[val_indices, ], center = train_means, scale = train_sds)
X_test_scaled  <- scale(X_filtered[test_indices, ], center = train_means, scale = train_sds)

y_train <- ages[train_indices]; y_val <- ages[val_indices]; y_test <- ages[test_indices]
```

1. load and initialize the dataset.
   - lightweight dataformat (feather, arrow or npy)
   - record time point.

2. Prepare dataset
   - train:validation:test = 70:15:15
   - feature scaling (beta standardization)

3. Create `tensor`
   - convert R matrices in to pytorch tensors

Main Training Loop

```
# 3. DATALOADER CREATION
cat("3. Creating PyTorch dataloaders...\n")
train_dataset <- tensor_dataset(
  torch_tensor(X_train_scaled, dtype = torch_float32()),
  torch_tensor(y_train, dtype = torch_float32())$unsqueeze(2)
)
val_dataset <- tensor_dataset(
  torch_tensor(X_val_scaled, dtype = torch_float32()),
  torch_tensor(y_val, dtype = torch_float32())$unsqueeze(2)
)
test_dataset <- tensor_dataset(
  torch_tensor(X_test_scaled, dtype = torch_float32()),
  torch_tensor(y_test, dtype = torch_float32())$unsqueeze(2)
)

train_loader <- dataloader(train_dataset, batch_size = batch_size, shuffle = TRUE,
num_workers = num_workers)
val_loader   <- dataloader(val_dataset, batch_size = batch_size, shuffle = FALSE,
num_workers = num_workers)
test_loader  <- dataloader(test_dataset, batch_size = batch_size, shuffle = FALSE,
num_workers = num_workers)
```

1. load and initialize the dataset.
   lightweight dataformat (feather, arrow or npy)
   record time point.

2. Prepare dataset
   train:validation:test = 70:15:15
   feature scaling (beta standardization)

3. Create `tensor`
   convert R matrices in to pytorch tensors

Main Training Loop

```r
# 4. MODEL INITIALIZATION AND TRAINING
cat("4. Initializing and training the model...\n")
model <- age_predictor_mlp(input_size = n_features, dropout_rate = dropout_rate)$to(device = device)
optimizer <- optim_adamw(model$parameters, lr = learning_rate, weight_decay = weight_decay)
scheduler <- lr_reduce_on_plateau(optimizer, patience = 5, factor = 0.2)

best_val_loss <- Inf
epochs_no_improve <- 0
history <- data.frame()

for (epoch in 1:num_epochs) {
  # Training phase
  model$train()
  train_loss <- 0
  coro::loop(for (b in train_loader) {
    optimizer$zero_grad()
    outputs <- model(b[[1]]$to(device = device))
    loss <- nnf_smooth_l1_loss(outputs, b[[2]]$to(device = device))
    loss$backward()
    optimizer$step()
    train_loss <- train_loss + loss$item()
  })

  # Validation phase
  model$eval()
  val_loss <- 0
  all_preds <- c(); all_labels <- c()
  with_no_grad({
    coro::loop(for (b in val_loader) {
      outputs <- model(b[[1]]$to(device = device))
      loss <- nnf_smooth_l1_loss(outputs, b[[2]]$to(device = device))
      val_loss <- val_loss + loss$item()
      all_preds <- c(all_preds, as.numeric(outputs$cpu()))
      all_labels <- c(all_labels, as.numeric(b[[2]]$cpu()))
    })
  })
```

4. Actual train the model

   Training phase

   Validation phase

```
--- Starting Experiment: all_probes ---
1. Loading and preprocessing data...
  Prepared data with 437810 features.
2. Splitting and scaling data...
3. Creating PyTorch dataloaders...
4. Initializing and training the model...
  Epoch 01: Val Loss: 35.9808, Val MAE: 35.8958
  Epoch 02: Val Loss: 35.4621, Val MAE: 35.3992
  Epoch 03: Val Loss: 34.9516, Val MAE: 34.9020
  Epoch 04: Val Loss: 34.3778, Val MAE: 34.3547
  Epoch 05: Val Loss: 33.7785, Val MAE: 33.7739
  Epoch 06: Val Loss: 33.3495, Val MAE: 33.3548
```

~45 min

```
  Epoch 97: Val Loss: 2.0099, Val MAE: 2.4054
  Epoch 98: Val Loss: 2.0611, Val MAE: 2.4506
  Epoch 99: Val Loss: 1.9592, Val MAE: 2.3639
  Epoch 100: Val Loss: 2.0218, Val MAE: 2.4205
```

5. try the best model on test set and save

Main Training Loop

```r
# 5. FINAL EVALUATION ON TEST SET
cat("5. Evaluating on the test set...\n")
# Load best model and evaluate
best_model <- age_predictor_mlp(input_size = n_features, dropout_rate = dropout_rate)
best_model$load_state_dict(torch_load("best_model.pt"))
best_model$to(device = device)$eval()

test_preds <- c()
with_no_grad({
  coro::loop(for (b in test_loader) {
    outputs <- best_model(b[[1]]$to(device = device))
    test_preds <- c(test_preds, as.numeric(outputs$cpu()))
  })
})
test_preds_corr <- pmax(0, test_preds)

final_mae <- mae_vec(truth = y_test, estimate = test_preds_corr)
final_rsq <- rsq_vec(truth = y_test, estimate = test_preds_corr)
```

4. Actual train the model

   Training phase

   Validation phase

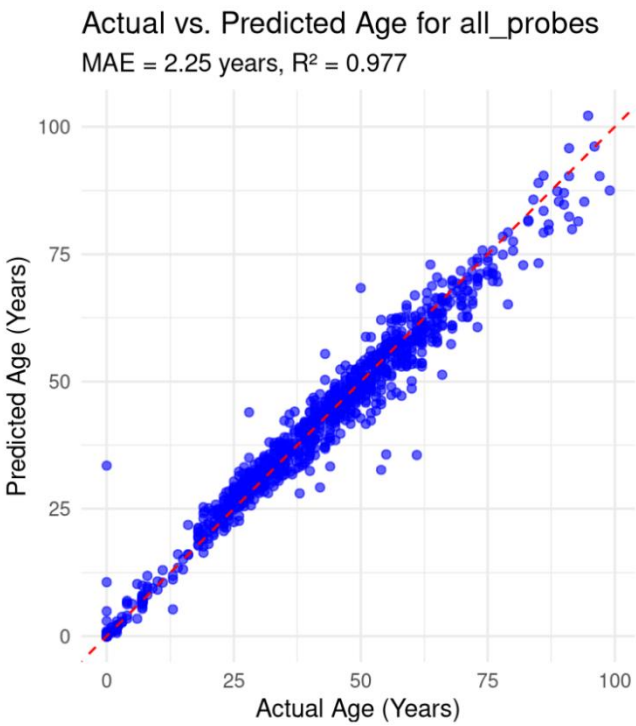
```
--- Starting Experiment: all_probes ---
1. Loading and preprocessing data...
  Prepared data with 437810 features.
2. Splitting and scaling data...
3. Creating PyTorch dataloaders...
4. Initializing and training the model...
  Epoch 01: Val Loss: 35.9808, Val MAE: 35.8958
  Epoch 02: Val Loss: 35.4621, Val MAE: 35.3992
  Epoch 03: Val Loss: 34.9516, Val MAE: 34.9020
  Epoch 04: Val Loss: 34.3778, Val MAE: 34.3547
  Epoch 05: Val Loss: 33.7785, Val MAE: 33.7739
  Epoch 06: Val Loss: 33.3495, Val MAE: 33.3548
```

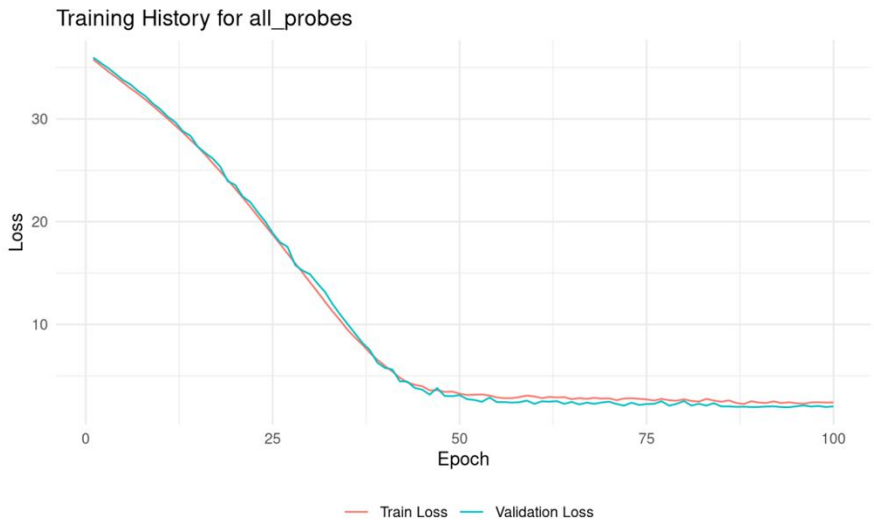
~45 min

```
Epoch 97: Val Loss: 2.0099, Val MAE: 2.4054
Epoch 98: Val Loss: 2.0611, Val MAE: 2.4506
Epoch 99: Val Loss: 1.9592, Val MAE: 2.3639
Epoch 100: Val Loss: 2.0218, Val MAE: 2.4205
```

5. try the best model on test set and save

Results (using all-probe set)





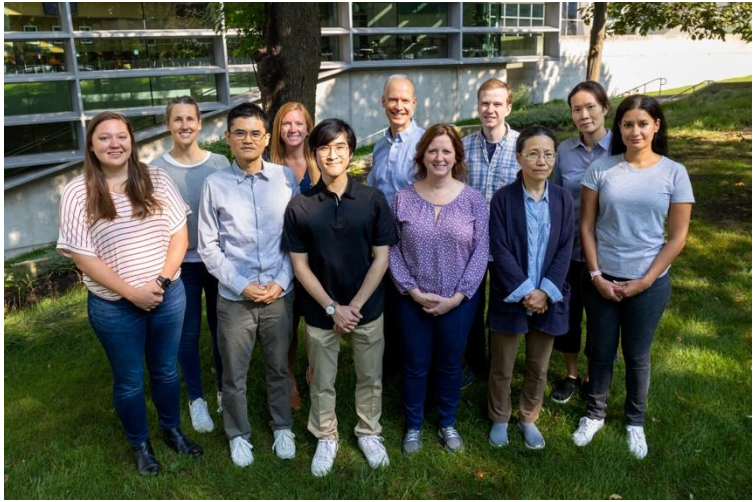
Testing set shows a very well trained model

MAE = 2.25, R^2 = 0.977

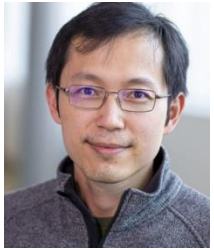
tl;dr

- R-torch can be a powerful tool for deep learning.
- deep-learned epigenetic clocks can predict age more accurately than elastic net clocks.

Acknowledgements



Laird Lab
Peter W. Laird
Nathan Spix
Toshinori Hinoue
Manpreet Kalkat
Felicia Ebot Ojong
David Sokol
Liang Kang
Ava Jensen
Ashlin Slanger
Paula Nolte
Christy
Kelly Foy
Amy Nuffesse



Special Thanks

Zachary Debruine, GVSU
Zack Ramjan, HPC
Kin Lau, BBC

VAN ANDEL INSTITUTE®

NIH National Institute on Aging

github.com/BacZemin

check my knitted qmd at github!