**Evan Domingos**

**MATH 398 Machine Learning**

**Protein Prediction Project 2**

**Prof. Kahn**

**Prediction of CASP 5.9 Model Protein Structure Root Mean Squared Deviation**

**Index**

**Abstract**

Introduction

Preliminary Discussion

Methods

Results

Discussion

Conclusion

Appendix

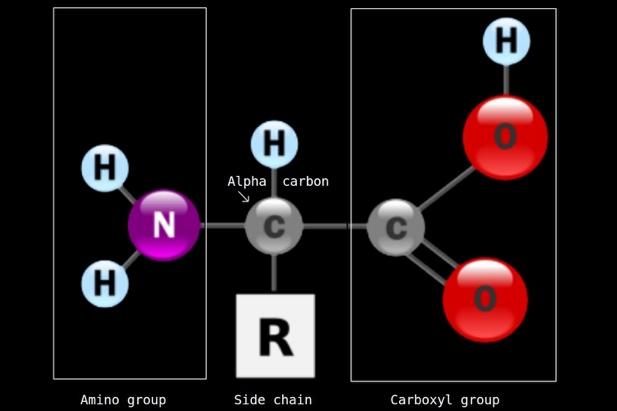
**Abstract**

In this paper, we present a comparison of regression trees, random forest, and deep neural network models for the prediction of the root-mean-square deviation of atomic positions (RMSD) for protein structure predictions from the 2002 CASP 5 competition. The dataset includes 45730 scored predictions for individual amino acids along with physiochemical characteristics. [To be Continued]

**Introduction**

**Basics of Protein Structure & Protein Folding**

Proteins are large biomolecules vital for performing functions within organisms like the replication of DNA, metabolizing enzymes, and structuring of cells. Proteins are formed from long chains of organic compounds called amino acids. Amino acids are formed around a single carbon atom called an alpha-carbon. Carbon has a valence of four so it may form four single bonds. In an amino acid, the alpha carbon is bonded to an amino group (NH2), a carboxyl group (COOH), a single hydrogen atom, and a variable element called the “R” group.



Two amino acids can form a peptide bond between the carboxyl group of one amino acid and the amino group of the other. A single water molecule (H2O) is formed from the expelled (H-) and (H0+), leaving a hydrophobic bond between the two amino acids called a peptide bond. The combined amino acids are called a polypeptide.

Shape

Description automatically generated with low confidence

Any number of amino acids can bond to that structure, aligning the carboxyl group to the amino group of each successive amino acid resulting in a chain referred to as the primary structure of a protein. Two or more combined amino acids forming a peptide are also called a residue. There are 20 organically common amino acids that can be categorized by the chemical properties of their R-groups; polarity and electrical charge. The physical interaction of the R-group and the hydrogen bonds linking the amino acids influences the angle of the bond, contorting the flat primary structure into either a helical structure or a pleated sheet, named alpha helices and beta sheets respectively. This is referred to as the secondary structure. The tertiary structure is determined by the interaction of R-groups within the polypeptide chain.

These interactions include hydrogen bonding, ionic bonding, and hydrophobic interactions. The R-groups of amino acids that are hydrophilic (polar) tend to be located on the protein's surface, while those that are hydrophobic (non-polar) are generally buried in the structure's interior, away from the water. These physical interactions between the R groups of amino acids determine the protein's final three-dimensional shape, the tertiary structure which is ultimately responsible for determining the function of the resulting protein.

Primary Structure: Sequence of bonded amino acids (residues) forming a polypeptide

Secondary Structure: 3D structure from the interaction of R-groups and hydrogen bonds

Tertiary Structure: Contortions from the interaction between R-groups

Given the importance of protein function and structure in biochemistry for the development of drugs among other applications, the prediction of structure from amino acid sequences is of immense interest. Since the structures of proteins are determined by their amino sequences, protein structure prediction has shown to be a powerful use-case for machine learning techniques and neural networks in particular. The use of machine learning algorithms in protein prediction has significantly reduced the time and cost required for the experimental determination of protein structures, making it a valuable tool in biochemistry research. Just 200,000 protein structures have been verified in the 130 years since x-ray crystallography was invented (Source: Protein Data Bank, 2023). In a single year, a neural network developed by Google’s DeepMind was able to produce and publish predictions for over 200 million known amino sequences with impressive results on verified structures.

This model named Alpha was developed for the Critical Assessment of Structure Prediction (CASP) which is a biennial worldwide experiment to evaluate the efficacy of computational protein structure prediction methods developed by academia and industry teams. CASP allows researchers from around the world to test their prediction methods on a set of protein sequences whose structures have not been determined experimentally or whose verified structures have not been released.

The scoring data from the 2002 CASP 5 competition was published to the UCI ML Database which was used for this project. The data contains the scored residue predictions for models submitted for the competition, measuring the deviation in position between the predicted structure and the aligned verified protein structure.

**Structural Alignment and the Calculation of RMSD**

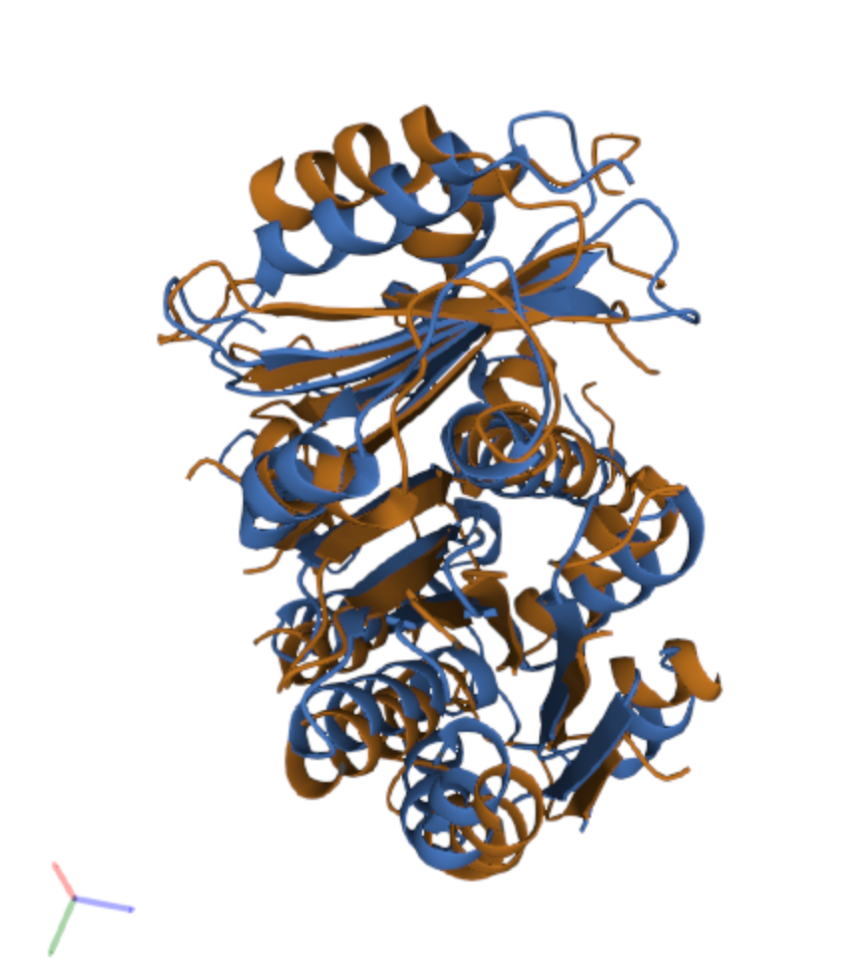
Protein prediction models are evaluated on the residuals of the predicted structure and the verified structure. Results in the CASP 5.9 dataset are evaluated using root mean squared deviation of atomic position (RMSD).

Structures, verified and modeled, are indexed by the amino sequence. For the purpose of prediction scoring, the sequences are identical. The alpha carbons of each residue are considered as the location and each residue in the model sequence has XYZ coordinates set considering the first amino acid as the zero. Root mean squared deviation is calculated at each residue for the difference between the predicted and verified locations of the alpha carbon.

The formula for RMSD of atomic position in an XYZ coordinate system is:

where n is the number of points in the measurement where v is the verified location and w is the predicted location. In the context of this paper, residues are scored individually but in reference to the first amino acid such that n = 1 for all measurements. The overall score for the prediction is the RMSD for all n amino acids in the sequence given there is one alpha-carbon per amino acid. RMSD is reported in Angstroms (A) where 1A = 10^-10m.





**CASP 5.9 Dataset**

The CASP 5.9 Dataset contains 45730 observations of scored results from the 2002 CASP 5 competition. 67 newly validated proteins where used to score structure predictions of the submitted models. It is not clear the number of unique residues contained within the score (multiple models making predictions for the same protein) but the CASP 5 validation set contains total of 14882 residues, suggesting some residues must be repeated.

The dataset has 9 features and the label RMSD. The 9 features are all continuous numerical variables and are defined as the following:

F1 - Total surface area

F2 - Non-polar exposed area

F3 - Fractional area of exposed non-polar residue.

F4 - Fractional area of exposed non-polar part of residue.

F5 - Molecular mass weighted exposed area.

F6 - Average deviation from the standard exposed area of residue.

F7 - Euclidean distance.

F8 - Secondary structure penalty.

F9 - Spatial Distribution constraints (N,K Value).

**Preliminary Discussion**

The

**Methods**

**Pytorch Deep Neural Network Models**

All models use the RMSprop optimizer and are evaluated using the mean absolute error (MAE) metric. After training, the models are used to predict RMSD values for a test set, and the MAE is calculated and plotted against the actual values.

The first neural network model, "modnn," has a simple architecture of a single hidden layer with 50 neurons and uses a rectified linear unit (ReLU) activation function. It also has a dropout layer for regularization where the dropout . This model was trained for 10 epochs using the mean squared error (MSE) loss function.

The second model "modnn2," is an iteration on the first model with the addition of a second hidden layer with 50 neurons that uses the ELU activation function. This model was trained for 100 epochs.

The "modnn3” model is identical to the second but with the addition of a hidden layer that uses the continuously differentiable exponential linear unit (CELU) activation function of the form:

CELU(x)=max(0,x)+min(0,α∗(exp(x/α)−1))

*Source: pytorch documentation*

This model was also trained for 100 epochs.

The fourth model “modnn4,” again iterated on the architecture of modnn2 with the addition of two more hidden layers with ReLU and ELU activation functions respectively; identical to the existing two layers. The hidden layers have 50 neurons each and use different activation functions: the first uses ReLU, the second uses ELU, the third uses ReLU, and the fourth uses ELU. The output layer has one neuron and uses a linear activation function. The model uses the MSE as the loss function, the RMSprop optimizer for optimization, and the MAE as the evaluation metric. The dropout rate is set to 0.4. The input size is determined by the number of features in the input data.

The fifth model is a neural network with twelve hidden layers, alternating in ReLU and ELU activation fucntions. The output layer has one neuron and uses a linear activation function. The presented model uses the MSE as the loss function, the RMSprop optimizer for optimization, and the MAE as the evaluation metric. A variation utilizing data loader and the SGD optimization algorithm was implemented but was not used to produce the presented results.

The dropout rates for the hidden layers are set to 0.5 for all layers. Multiple dropout rates were evaluated. A dropout rate of 0.4-0.5 resulted in optimal convergence.

The input size is determined by the number of features in the input data.

**Results**

Timeline

Description automatically generated

Splits:

Fractional area of exposed non-polar residue.

Average deviation from the standard exposed area of residue.

MAE: 7.69

Chart, scatter chart

Description automatically generated

Discussion

Chart, line chart, histogram

Description automatically generated

Chart, scatter chart

Description automatically generated

100 Epochs

Summary

Appendix

**Initial Neural Network with Relu Activation Function**

modnn <- nn\_module(

initialize = function(input\_size) {

self$hidden <- nn\_linear(input\_size, 50)

self$activation1 <- nn\_relu()

self$dropout <- nn\_dropout(0.4)

self$output <- nn\_linear(50, 1)

},

forward = function(x) {

## x %>%

## self$hidden() %>%

## self$activation() %>%

## self$dropout() %>%

## self$output() %>%

self$output( self$dropout( self$activation1( self$hidden(x) ) ) )

}

)

#set hyperparameters for nn\_module

# MSE loss function

modnn <- set\_hparams( setup(modnn ,

loss = nn\_mse\_loss(),

optimizer = optim\_rmsprop,

metrics = list(luz\_metric\_mae())),

input\_size = ncol(x))

#training

#takes roughly 6 hours to train at 100 epochs, 210 sec each

fitted <- fit(modnn,

data = list(x[-testid, ],

matrix(y[-testid], ncol = 1)),

valid\_data = list(x[testid, ],

matrix(y[testid], ncol = 1)),

epochs = 10)

**Addition of ELU Activation Function**

#2nd Model, Add elu hidden layer

modnn2 <- nn\_module(

initialize = function(input\_size) {

self$hidden1 <- nn\_linear(input\_size, 50)

self$activation1 <- nn\_relu()

self$dropout1 <- nn\_dropout(0.4)

self$hidden2 <- nn\_linear(50, 50)

self$activation2 <- nn\_elu()

self$dropout2 <- nn\_dropout(0.4)

# self$hidden3 <- nn\_linear()

self$output <- nn\_linear(50, 1)

},

forward = function(x) {

x %>%

self$hidden1() %>%

self$activation1() %>%

self$dropout1() %>%

self$hidden2() %>%

self$activation2() %>%

self$dropout2() %>%

self$output()

}

)

modnn2 <- set\_hparams( setup(modnn2 ,

loss = nn\_mse\_loss(),

optimizer = optim\_rmsprop,

metrics = list(luz\_metric\_mae())),

input\_size = ncol(x))

#training

#210sec per epoch

fitted2 <- fit(modnn2,

data = list(x[-testid, ],

matrix(y[-testid], ncol = 1)),

valid\_data = list(x[testid, ],

matrix(y[testid], ncol = 1)),

epochs = 100)

library(tidyverse)

library(caret)

library(tensorflow)

library(ranger) #random forest

library(nnet) #neural net

library(keras)

library(gtable)

library(torch)

library(luz) # high-level interface for torch

library(torchvision) # for datasets and image transformation

library(torchdatasets) # for datasets we are going to use

library(zeallot)

library(rpart)

library(rpart.plot)

library(torch)

library(torchvision)

#torch deep learning neural network

#load CASP 5.9 Set

dataset <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/00265/CASP.csv", header = TRUE, stringsAsFactors = FALSE)

#remove missing data (none in CASP dataset), create protein set

protein <- na.omit(dataset)

#n number of protein residues: 45730 obs

n <- nrow(protein)

#for reproduction

set.seed(13)

#number of test samples: 1/3 obs

ntest <- trunc(n / 3)

#randomly select test IDs from

testid <- sample(1:n, ntest)

#set torch seed

torch\_manual\_seed(13)

#scale variables, seperate x labels ,y target var. Remove intercepts

x <- scale(model.matrix(protein$RMSD ~ . - 1, data = protein))

y <- protein$RMSD

#very easy to produce good looking tree, extremely overfitted though

#RF regression tree

library(randomForest)

#Split the data into training and testing sets

#train\_id <- sample(1:n, floor(n \* 0.6), replace = FALSE)

#valid\_id <- sample(setdiff(1:n, train\_id), floor(n \* 0.2), replace = FALSE)

#test\_id <- setdiff(1:n, union(train\_id, valid\_id))

#train\_x <- x[train\_id, ]

#train\_y <- y[train\_id]

#valid\_x <- x[valid\_id, ]

#valid\_y <- y[valid\_id]

#test\_x <- x[test\_id, ]

#test\_y <- y[test\_id]

# Train the random forest regression model

rf\_model <- randomForest(train\_x, train\_y, ntree = 500)

#Predict RMSD for test set using the trained model

pred\_valid <- predict(rf\_model, valid\_x)

pred\_test <- predict(rf\_model, test\_x)

#Calculate mean squared error on the test set

mae\_valid <- mean(abs(pred\_valid - test\_y))

mae\_test <- mean(abs(pred\_test-test\_y))

#2.535

mae\_test

#6.148

mae\_valid

#

protein\_tree1 <- rpart(RMSD ~ ., data = protein, control = rpart.control(minsplit = 20))

pred\_protein\_tree1 <- predict(protein\_tree1, newdata = protein[testid, ])

pred\_protein\_tree1

mae\_protein\_tree1 <- mean(abs(pred\_protein\_tree1 - y[testid]))

mae\_protein\_tree1

rpart.plot(protein\_tree1)

#create NN with four layers, input, 50 node hidden layer, Relu activation, dropout layer 0.4

modnn <- nn\_module(

initialize = function(input\_size) {

self$hidden <- nn\_linear(input\_size, 50)

self$activation1 <- nn\_relu()

self$dropout <- nn\_dropout(0.4)

self$output <- nn\_linear(50, 1)

},

forward = function(x) {

## x %>%

## self$hidden() %>%

## self$activation() %>%

## self$dropout() %>%

## self$output() %>%

self$output( self$dropout( self$activation1( self$hidden(x) ) ) )

}

)

#set hyperparameters for nn\_module

# MSE loss function

modnn <- set\_hparams( setup(modnn ,

loss = nn\_mse\_loss(),

optimizer = optim\_rmsprop,

metrics = list(luz\_metric\_mae())),

input\_size = ncol(x))

#training

#takes roughly 6 hours to train at 100 epochs, 210 sec each

fitted <- fit(modnn,

data = list(x[-testid, ],

matrix(y[-testid], ncol = 1)),

valid\_data = list(x[testid, ],

matrix(y[testid], ncol = 1)),

epochs = 10)

plot(fitted)

summary(fitted)

fitted$ctx

#Predict RMSD values for test set

test\_pred <- predict(fitted, x[testid, ])

#calc MSE

mse <- mean((test\_pred-y[testid])^2)

mse

#50.7419 for 100 epoch model

# scatterplot of predicted vs actual RMSD

plot(y[testid], test\_pred, xlab = "Actual RMSD (Angstrom)", ylab = "Predicted RMSD (Angstrom)", main = "RELU HL Nueral Network")

abline(a = 0, b = 1, col = "purple")

#best model

#2nd Model, Add elu hidden layer

modnn2 <- nn\_module(

initialize = function(input\_size) {

self$hidden1 <- nn\_linear(input\_size, 50)

self$activation1 <- nn\_relu()

self$dropout1 <- nn\_dropout(0.4)

self$hidden2 <- nn\_linear(50, 50)

self$activation2 <- nn\_elu()

self$dropout2 <- nn\_dropout(0.4)

# self$hidden3 <- nn\_linear()

self$output <- nn\_linear(50, 1)

},

forward = function(x) {

x %>%

self$hidden1() %>%

self$activation1() %>%

self$dropout1() %>%

self$hidden2() %>%

self$activation2() %>%

self$dropout2() %>%

self$output()

}

)

modnn2 <- set\_hparams( setup(modnn2 ,

loss = nn\_mse\_loss(),

optimizer = optim\_rmsprop,

metrics = list(luz\_metric\_mae())),

input\_size = ncol(x))

#training

#210sec per epoch

fitted2 <- fit(modnn2,

data = list(x[-testid, ],

matrix(y[-testid], ncol = 1)),

valid\_data = list(x[testid, ],

matrix(y[testid], ncol = 1)),

epochs = 100)

plot(fitted2)

summary(fitted2)

fitted2$ctx

#predict RMSD values for test set

test\_pred2 <- predict(fitted2, x[testid, ])

mae\_nn2 <- mean(abs(test\_pred2 - y))

mae\_nn2

plot(y[testid], test\_pred2, xlab = "Actual RMSD (Angstrom)", ylab = "Predicted RMSD (Angstrom)", main = "NN2: 2 Layer RELU/ELU")

abline(a = 0, b = 1, col = "purple")

#ModNN3

#add CELU hidden layer

modnn3 <- nn\_module(

initialize = function(input\_size) {

self$hidden1 <- nn\_linear(input\_size, 50)

self$activation1 <- nn\_relu()

self$dropout1 <- nn\_dropout(0.4)

self$hidden2 <- nn\_linear(50, 50)

self$activation2 <- nn\_elu()

self$dropout2 <- nn\_dropout(0.4)

self$hidden3 <- nn\_linear(50, 50)

self$activation3 <- nn\_celu()

self$dropout3 <- nn\_dropout(0.4)

# self$hidden3 <- nn\_linear()

self$output <- nn\_linear(50, 1)

},

forward = function(x) {

x %>%

self$hidden1() %>%

self$activation1() %>%

self$dropout1() %>%

self$hidden2() %>%

self$activation2() %>%

self$dropout2() %>%

self$hidden3() %>%

self$activation3() %>%

self$dropout3() %>%

self$output()

}

)

nn\_get\_parameters(modnn3)

modnn3 <- set\_hparams( setup(modnn3 ,

loss = nn\_mse\_loss(),

optimizer = optim\_rmsprop,

metrics = list(luz\_metric\_mae())),

input\_size = ncol(x))

#training

#210sec per epoch

fitted3 <- fit(modnn3,

data = list(x[-testid, ],

matrix(y[-testid], ncol = 1)),

valid\_data = list(x[testid, ],

matrix(y[testid], ncol = 1)),

epochs = 100)

matrix(y[-testid], ncol = 1)

list(x[-testid, ],

matrix(y[-testid], ncol = 1))

plot(fitted3)

summary(fitted3)

fitted3$model

fitted3$ctx

# Predict RMSD values for test set

test\_pred3 <- predict(fitted3, x[testid, ])

test\_pred3 <- predict(fitted3, x[testid, ])

plot(y[testid], test\_pred3, xlab = "Actual RMSD (Angstrom)", ylab = "Predicted RMSD (Angstrom)", main = "NN3: 3 Layer RELU/ELU/CELU")

abline(a = 0, b = 1, col = "purple")

#NN3 MAE calc

length(testid)

mae\_nn3 <- mean(abs(test\_pred3 - y[testid]))

mae\_nn3

# Add binary step (Threshold) Hidden Layer

modnn4 <- nn\_module(

initialize = function(input\_size) {

self$hidden1 <- nn\_linear(input\_size, 50)

self$activation1 <- nn\_relu()

self$dropout1 <- nn\_dropout(0.4)

self$hidden2 <- nn\_linear(50, 50)

self$activation2 <- nn\_elu()

self$dropout2 <- nn\_dropout(0.4)

self$hidden3 <- nn\_linear(50, 50)

self$activation3 <- nn\_relu()#nn\_threshold ( value = 15, inplace = FALSE, threshold = 15)

self$dropout3 <- nn\_dropout(0.4)

self$hidden4 <- nn\_linear(50, 50)

self$activation4 <- nn\_elu()

self$dropout4 <- nn\_dropout(0.4)

# self$hidden3 <- nn\_linear()

self$output <- nn\_linear(50, 1)

},

forward = function(x) {

x %>%

self$hidden1() %>%

self$activation1() %>%

self$dropout1() %>%

self$hidden2() %>%

self$activation2() %>%

self$dropout2() %>%

self$hidden3() %>%

self$activation3() %>%

self$dropout3() %>%

self$hidden4() %>%

self$activation4() %>%

self$dropout4() %>%

self$output()

}

)

modnn4 <- set\_hparams( setup(modnn4 ,

loss = nn\_mse\_loss(),

optimizer = optim\_rmsprop,

metrics = list(luz\_metric\_mae())),

input\_size = ncol(x))

#training

#215sec per epoch

fitted4 <- fit(modnn4,

data = list(x[-testid, ],

matrix(y[-testid], ncol = 1)),

valid\_data = list(x[testid, ],

matrix(y[testid], ncol = 1)),

epochs = 200)

plot(fitted4)

summary(fitted4)

xfitted4$ctx

#Predict RMSD values for test set

test\_pred4 <- predict(fitted4, x[testid, ])

test\_pred4 <- predict(fitted4, x[testid, ])

plot(y[testid], test\_pred4, xlab = "Actual RMSD (Angstrom)", ylab = "Predicted RMSD (Angstrom)", main = "3 Layer RELU/ELU Nueral Network")

abline(a = 0, b = 1, col = "purple")

#stack on a bunch of hidden layers

modnn5 <- nn\_module(

initialize = function(input\_size) {

self$hidden1 <- nn\_linear(input\_size, 50)

self$activation1 <- nn\_relu()

self$dropout1 <- nn\_dropout(0.5)

self$hidden2 <- nn\_linear(50, 50)

self$activation2 <- nn\_elu()

self$dropout2 <- nn\_dropout(0.5)

self$hidden3 <- nn\_linear(50, 50)

self$activation3 <- nn\_relu()

self$dropout3 <- nn\_dropout(0.5)

self$hidden4 <- nn\_linear(50, 50)

self$activation4 <- nn\_elu()

self$dropout4 <- nn\_dropout(0.5)

self$hidden5 <- nn\_linear(50, 50)

self$activation5 <- nn\_relu()

self$dropout5 <- nn\_dropout(0.5)

self$hidden6 <- nn\_linear(50, 50)

self$activation6 <- nn\_relu()

self$dropout6 <- nn\_dropout(0.5)

self$hidden7 <- nn\_linear(50, 50)

self$activation7 <- nn\_elu()

self$dropout7 <- nn\_dropout(0.25)

self$hidden8 <- nn\_linear(50, 50)

self$activation8 <- nn\_elu()

self$dropout8 <- nn\_dropout(0.5)

self$hidden9 <- nn\_linear(50, 50)

self$activation9 <- nn\_elu()

self$dropout9 <- nn\_dropout(0.5)

self$hidden10 <- nn\_linear(50, 50)

self$activation10 <- nn\_relu()

self$dropout10 <- nn\_dropout(0.5)

self$hidden11 <- nn\_linear(50, 50)

self$activation11 <- nn\_relu()

self$dropout11 <- nn\_dropout(0.5)

self$hidden12 <- nn\_linear(50, 50)

self$activation12 <- nn\_relu()

self$dropout12 <- nn\_dropout(0.5)

self$output <- nn\_linear(50, 1)

},

forward = function(x) {

x %>%

self$hidden1() %>%

self$activation1() %>%

self$dropout1() %>%

self$hidden2() %>%

self$activation2() %>%

self$dropout2() %>%

self$hidden3() %>%

self$activation3() %>%

self$dropout3() %>%

self$hidden4() %>%

self$activation4() %>%

self$dropout4() %>%

self$hidden5() %>%

self$activation5() %>%

self$dropout5() %>%

self$hidden6() %>%

self$activation6() %>%

self$dropout6() %>%

self$hidden7() %>%

self$activation7() %>%

self$dropout7() %>%

self$hidden8() %>%

self$activation8() %>%

self$dropout8() %>%

self$hidden9() %>%

self$activation9() %>%

self$dropout9() %>%

self$hidden10() %>%

self$activation10() %>%

self$dropout10() %>%

self$hidden11() %>%

self$activation11() %>%

self$dropout11() %>%

self$hidden12() %>%

self$activation12() %>%

self$dropout12() %>%

self$output()

}

)

modnn5 <- set\_hparams( setup(modnn5 ,

loss = nn\_mse\_loss(),

optimizer = optim\_rmsprop,

metrics = list(luz\_metric\_mae())),

input\_size = ncol(x))

#training

#210sec per epoch

fitted5 <- fit(modnn5,

data = list(x[-testid, ],

matrix(y[-testid], ncol = 1)),

valid\_data = list(x[testid, ],

matrix(y[testid], ncol = 1)),

epochs = 200)

plot(fitted5)

summary(fitted5)

xfitted2$ctx

# Predict RMSD values for test set

test\_pred3 <- predict(fitted3, x[testid, ])

test\_pred3 <- predict(fitted3, x[testid, ])

plot(y[testid], test\_pred3, xlab = "Actual RMSD (Angstrom)", ylab = "Predicted RMSD (Angstrom)")

abline(a = 0, b = 1, col = "purple")

modnn4 <- nn\_module(

initialize = function(input\_size) {

self$hidden1 <- nn\_linear(input\_size, 50)

# Use different activation functions for each feature

self$activation1 <- nn\_relu() # F1, F2, F3, F5, F6, F7, F9

self$activation2 <- nn\_sigmoid() # F4

self$activation3 <- nn\_linear() # F8

self$dropout1 <- nn\_dropout(0.4)

self$hidden2 <- nn\_linear(50, 50)

self$dropout2 <- nn\_dropout(0.4)

self$output <- nn\_linear(50, 1)

},

forward = function(x) {

# Split the input tensor based on the feature indices

x1 <- x[, 1:7] # F1, F2, F3, F4, F5, F6, F7

x2 <- x[, 8] # F8

x3 <- x[, 9] # F9

out1 <- x1 %>%

self$hidden1() %>%

self$activation1() %>%

self$dropout1() %>%

self$hidden2() %>%

self$activation2() %>%

self$dropout2()

out2 <- x2 %>%

self$hidden1() %>%

self$activation3() # Use linear activation function for F8

out3 <- x3 %>%

self$hidden1() %>%

self$activation1() # Use ReLU activation function for F9

torch$cat(list(out1, out2, out3), dim = 2) %>%

self$output()

}

)

modnn4\_setup <- setup(modnn4 ,

loss = nn\_mse\_loss(),

optimizer = optim\_rmsprop,

metrics = list(luz\_metric\_mae()))

modnn4 <- set\_hparams(modnn4\_setup, input\_size = ncol(x))

#training

#210sec per epoch

fitted4 <- fit(modnn4,

data = list(x[-testid, ],

matrix(y[-testid], ncol = 1)),

valid\_data = list(x[testid, ],

matrix(y[testid], ncol = 1)),

epochs = 100)

**Plots**

vars <- names(protein)

combinations <- combn(vars, 2)

#scatter plot each variables

par(mfrow = c(1, 1))

par(ask = TRUE)

for (i in 1:ncol(combinations)) {

plot(protein[, combinations[1, i]], protein[, combinations[2, i]],

xlab = combinations[1, i], ylab = combinations[2, i],

main = paste0("Scatter plot of ", combinations[1, i], " vs. ", combinations[2, i]))

}

# sort by increasing RMSD values

protein\_sorted <- protein[order(protein$RMSD),]

# plot RMSD in increasing order

plot(protein\_sorted$RMSD, type = "l", xlab = "Residue", ylab = "RMSD", main = "RMSD Sorted in Increasing Order")

abline(a = 0, b = 20.99/45730, col = "purple")

**Resources**

1. <https://predictioncenter.org/casp12/doc/help.html>