# Abstract

We used 45730 observations from the 2002 Critical Assessment of Structure Prediction 5 Physiochemical Properties data set to train an optimized random forest regression model and deep neural network regression models using properties of amino acid residues to predict root-mean-square deviation of atomic positions (RMSD) for structural models. We investigate which physiochemical characteristics of amino acid residues are linked to higher error in the predicted tertiary structure and discuss the order of prediction dependence inherent to RMSD of structural models. Given the lack of order in the dataset, we found gradient boosted tree methods to most accurately predicted structural error which heavily weight polar surface area of residues.

Prediction of RMSD of Atomic Position in CASP-5 Protein STructure Models

Comparison of Gradient Boosted Regression Tree, Feedforward , and GMDH Deep Neural Network Methods

EE

Table of Contents

Abstract [Cover](#_Abstract)

Introduction [1](#_Introduction)

Problem Statement

Basics of Protein Structure & Protein Folding [5](#_Basics_of_Protein)

Calculation of RMSD [5](#_Structural_Alignment_and)

Pairwise Structural Alignment [5](#_Structural_Alignment_and)

CASP 5.9 Dataset [5](#_CASP_5.9_Dataset)

Preliminary Discussion [4](#_Preliminary_Discussion)

Methods [4](#_Methods)

Random Forest Regression Tree [5](#_Basics_of_Protein)

Feedforward Neural Networks [5](#_Basics_of_Protein)

MIA GMDH Neural Network [5](#_Basics_of_Protein)

Gradient Boosted Regression Tree [5](#_Basics_of_Protein)

Results [4](#_Results)

Random Forest Regression Tree [5](#_Basics_of_Protein)

Feedforward Neural Networks [5](#_Basics_of_Protein)

MIA GMDH Neural Network [5](#_Basics_of_Protein)

Gradient Boosted Regression Tree [5](#_Basics_of_Protein)

Discussion [4](#_Results)

Conclusion [4](#_Conclusion)

Appendix [4](#_Appendix)

Bibliography [4](#_Appendix)

# Introduction

## Problem Statement

Proteins are vital to the functions of human cells, and they are responsible for performing a variety of tasks. There are tens of thousands of unique proteins within a typical human cell. Proteins fold into functional shapes and their different structures allow them to perform diverse roles. However, predicting the three-dimensional structure of a protein from its amino acid sequence remains a challenging task. Computational methods like machine learning algorithms have proven extremely well fitted for the task of protein folding predictions but it is crucial to identify and discard predicted models with structural errors, as relying on incorrect models can lead to erroneous conclusions about protein function and hinder the development of effective therapeutics. Some predictive models include predicted structural error factors to adjust predictions, similar to gradient boosting. We aim to construct a model to predict the structural errors in protein folding models from the 2002 CASP-5 competition.

## 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.

A screenshot of a video game

Description automatically generated with medium confidence

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.

A picture containing chart

Description automatically generated

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 of 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 or protein folding 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 residue prediction scores for models submitted for the competition, measuring the deviation in position between the predicted structure and the aligned verified protein structure.

## 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).

Both modeled and verified structures have the same spatial geometry. The structure is indexed by the amino sequence. For the purpose of prediction scoring, the sequences are identical.

Example:



Source: *Protein Data Bank*

The alpha carbon for a given residue is treated as the location for that amino acid and each residue in the structural model has XYZ coordinates. Generally speaking, the XYZ coordinates treat the center of the structure or sequence as the origin. The position of following amino acids are dependent on the position of the preceding residue in the sequence.

Root mean squared deviation is calculated at each residue for the difference between the predicted and verified locations of the alpha carbon.

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

where n is the number of points in the measurement (residues in the structure where v is the verified location and w is the predicted location. In the context of this paper, observations are scores for single residues thus we can consider that n = 1 for all measurements and the formula for RMSD is equivalent to the distance formula in a 3-dimensional cartesian coordinate system.

Again, v is the verified alpha carbon location and w is the predicted location. RMSD is reported in Angstroms (A) where 1A = 10E-10 meters.

## Pairwise Structural Alignment

Protein structures are compared via pairwise structural alignment. In the context of model validation, the verified structure of the protein is suspended in a 3-dimensional coordinate system. The predicted structure is superimposed in the space. In rigid body alignment which was used in CASP-5, the relative orientations and positions of the alpha carbons in each structure remain fixed through the alignment process. (Strucural Alignment) The structural model of each protein is considered as a matrix for which the Kabsch algorithm (or an estimation of the algorithm) is employed to calculate the optimal rotation matrix to minimize the distance or RMSD between the two. The algorithm first translates each structure so that the geometric center or the centroid is positioned at the origin of the 3-dimensional coordinate system. The algorithm then calculates the rotation matrix so as to minimize the objective function of RMSD. (Kabsch)

**Structural geometry in a 2-dimensional coordinate system**



**Pairwise Structural Alignment**



v is the verified structure centered at the origin and w is the superimposed predicted structure

**Calculation of RMSD for the ith residue in the aligned structures**



## 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. This dataset is ***not*** the structural predictions themselves. Each observation represents a prediction for a single amino acid and contains physiochemical properties from the prediction as well as RMSD. No information for the specific protein, amino acid, or predictive model is provided in the data set.

Further, it is not clear the number of unique residues contained within the score (multiple models making predictions for the same protein). The CASP-5 validation set (used in the competition, not the UCI data set) contained a total of 14882 residues for 67 different proteins. The UCI dataset contains scores for 45730 residues so at a minimum, the dataset contains predictions from multiple models for the same amino residues.

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

F1 - Total surface area (Angstroms

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.

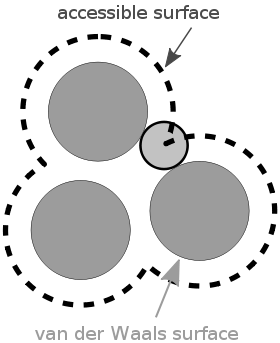
F7 - Euclidean distance.

F8 - Secondary structure penalty.

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

Exposed or accessible surface area (ASA) is the surface area of the protein that is accessible to a solvent, water in this application. One method of calculating ASA is the Shrake-Rupley algorithm which draws a mesh of points around each atom of the residue with a radius equal to the Van der Waals radius (conceptually the surface of the atom) plus the radius of the solvent (water is 1.4Å) [Shrake Rupley Algorithm]. In essence, the algorithm rolls a ball the size of a water molecule along the surface of the residue.

**Visual of Accessible Surface Area Calculation**



# Preliminary Discussion

There is no visible structure between RMSD and any of the predictors. To assess the relative importance of each feature we utilized a random forest regression tree model on all features. The architecture of the model is discussed in the Methods section. Below is the feature importance of the trained model.

Table

Description automatically generated

F4 and F3 are Fractional area of exposed non-polar part of residue and Fractional area of exposed non-polar residue respectively. The importance of these features was consistent with all interpretable models. In the context of protein folding physical principles, the importance is explained in that non-polar regions are hydrophobic. Water is a chemical product of protein folding and the repulsive interaction between water and the hydrophobic surface causes the protein to contort so that the non-polar residues are inward facing. In the context of modeling structure, more complicated physicochemical interactions would be positively correlated with error.

Chart

Description automatically generated Chart, scatter chart

Description automatically generated

*Note the clustering in the plot of RMSD vs. F4 and the loose positively correlation between F1 and RMSD*

F8, the Secondary structure penalty is not clearly defined in the UCI dataset but appears to be related to the dehydration process (production of water) and the resulting formation of a hydrogen bond which is also non-polar. F2 is the non-polar exposed area, similar to the above features. (secondary structure)

Chart, scatter chart

Description automatically generated

*Note the clustering in the plot of RMSD vs. F8, similar to that of F4*

Features pertaining to total surface area (F1), mass-weighted exposed area (F5), and Euclidean distance (F7) are highly colinear and seem to correlate with the size of the residue. There is no apparent conceptual argument for any relationship between the size of the residue and the predicted aligned error.

# Models & Results

*All models in this analysis are written in R. All R code as well as some Python rewrites can be found in the appendix or at https://github.com/evandomingos/CASP-Protein-Prediction*

## Random Forest Regression Tree

The first method analyzed was a random forest regression model. A different setup for training and validation sets was use for this model where 60% of the cleaned protein dataset was included in the training and the remaining 40% was broken up into two validation sets. The objective of this model was to overfit in an attempt to minimize error on the training set. The features and label are not scaled.

Random Forest model

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

The model was trained with ntree = 500. No tangible gain in accuracy on the test set was seen for n>500.

## Feedforward Neural Network

Several carryforward neural networks were developed with PyTorch, all of which are contained in the appendix. The input data for all neural networks was scaled and the intercept removed as follows:

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

y <- protein$RMSD

torch\_manual\_seed was set = 7 for all trained models.

**modNN**

The first model tested, modNN1, had a simple architecture of a single hidden layer with 50 neurons and used a rectified linear unit (ReLU) activation function. Changing the number of neurons in the hidden layer did not have a tangible impact on the convergence speed or trained model. The model also contained a single dropout layer for regularization where the dropout rate was set to 0.4. Changing the dropout rate to <0.3 or >0.5 resulted in a model that failed to converge in the allowed training period. This model was trained for 10 epochs using the mean squared error (MSE) loss 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)

**modNN2**

Chart, scatter chart

Description automatically generated

**modNN3**

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.

**modNN5**

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.

## MIA GMDH Neural Network

modNN7 is a Multilayered Iterative (MIA) Group Method of Data Handling (GMDH) Neural Network, implemented using the gmdh.mia() function from the GMDHreg library in R.

The GMDH (Group Method of Data Handling) algorithm is a neural network architecture that involves iteratively generating candidate models and selecting the best one based on some criterion to create a model of models. In this case, the MIA-GMDH algorithm is used, which generates candidate models and selects the best model based on the predicted residual error of sum squares (PRESS). Nodes are pruned from each layer and the top k models are selected.

Prune (an arugment in the gmdh.mia() function) is the selected number of neurons from layer i to layer i+1. The resulting layer i+1 has prune(prune-1)/2 neurons. The selected models are then used to generate a set of candidate models for the second layer, where each candidate model consists of the combination of the inputs from the selected models in the first layer. This process is repeated for each subsequent layer until the final layer is reached.

The output of one layer is used as input to the subsequent layer, resulting in a hierarchical stack of models.

.

For modnn7, the input data for the model is identical to the previous carryforward neural networks but must be treated as a matrix object of predictor variables (x) and a vector of the response variable (y).

The prune argument specifies the maximum number of predictors to include in each layer of the model, while the criteria argument specifies the criterion to use for selecting the best model. In this model, prune is specified as ncol(x) or 9, as per recommended in the GMDHreg documentation.

In this case, the PRESS (Predicted Residual Error Sum of Squares) criterion is used, which considers all information in the data sample and is computed without recalculation at each test point. “test” is an estimation RMSE and is more computationally efficient than “PRESS”. PRESS is used as the stopping criterion in the training process and can be thought about as ridge regression in that it adds a penalty for additional complexity to the model. The algorithm continuously adds hidden layers until this stopping criterion is met.

**modNN7**

# Fit GMDH NN model

modnn7 <- gmdh.mia(X = as.matrix(protein[-testid, 2:10]),

y = as.matrix(protein[-testid, 1]),

prune = ncol(protein[-testid, 2:10]),

criteria = "PRESS")

The trained modNN7 model using set.seed(13) produced a 136 hidden layer model. Notably, this model was a significantly faster learner than the strict carryforward neural networks.

When making predictions on the validation set, the model outputted 27 missing (NA) values. These values were removed using row indexing. Finally, the mean absolute error (MAE) is calculated on the test set (protein.test[,1]) using the cleaned predicted values.

## Gradient Boosted Regression Tree

The final method evaluated was a gradient-boosted regression tree using the XGBoost R library. Gradient boosted trees are essentially an ensemble model made of successive decision trees starting by building a simple regression tree. The algorithm then calculates the residuals on each training data point and trains a second tree to predict those residuals from the first tree. This process is iterated for n-rounds where each successive tree predicts the residuals from the previous. After n-rounds, the final model is the ensemble of all trees. An outputted prediction is the sum of the prediction from all n-trees in the model.

Where is the predicted value from the initial tree and is the predicted residual from the nth tree

The objective argument of the gradient-boosted regression algorithm determines how the residuals are calculated and how the algorithm tries to minimize them. For this model, the objective function used is the mean absolute error (reg:absoluteerror) which is minimized via gradient descent.

# Results

## Random Forest Regression Tree

## Feedforward Neural Network

**modNN**

Chart, scatter chart

Description automatically generated

Training MAE: 3.89

Validation MAE: 4.56

**modNN2**

Chart, scatter chart

Description automatically generated

Training MAE: 3.89

Validation MAE: 4.56

**modNN3**

Chart, scatter chart

Description automatically generated

**modNN5**

Chart, scatter chart

Description automatically generated

## MIA GMDH Neural Network

Chart, scatter chart

Description automatically generated

## Gradient Boosted Regression Tree

Chart, scatter chart

Description automatically generated

Chart

Description automatically generated

# Discussion

Chart, line chart, histogram

Description automatically generated

# Conclusion

# Appendix

**Plots**

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

library(ggplot2)

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

protein <- na.omit(dataset)

names(protein)

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")

par(ask = FALSE)

**Models**

#Various libraries

library(tidyverse)

library(caret)

library(tensorflow)

library(ranger) # Random forest with Ranger (more efficient than rf library for higher dimensional data)

library(randomForest) # Random Forest

library(xgboost) # For Gradient Boosted Tree

library(nnet) # Neural net

library(keras)

library(gtable)

library(zeallot)

library(rpart)

library(rpart.plot)

library(torch)

library(luz) # High-level interface for torch

library(torchvision) # For datasets and image transformation

library(GMDHreg) # Library for Group Method of Data Handling (GMDH) Nueral Network

# Load CASP 5.9 Set from

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(7)

# Number of test samples = 1/3 observations

ntest <- trunc(n / 3)

#randomly select test IDs from

testid <- sample(1:n, ntest)

# Set PyTorch seed

torch\_manual\_seed(7)

# In this segment, only Gradient Boosted tree uses

# Scale variables, separate x labels ,y target var. Remove intercepts

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

length(x)

y <- protein$RMSD

# First Method

# Random Forest regression tree with the obejctive of plotting feature important and purposely overfitting a model

# Very easy to produce good looking tree, extremely overfitted and validation MAE explodes

library(randomForest)

# Split the data into training and testing sets

# \*\*\* Un-comment these blocks to run the Random Forest Model. Commented out so the train\_id object does not affect Nueral Networks \*\*\*

# For reproducibility

set.seed(13)

#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))

# Create extra holdout set, "valid"

#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)

# Plot feature importance

varImpPlot(rf\_model, main = "Feature Importance")

# 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 and validation (extra holdout data)

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

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

# MAE 2.5555 on test set

# After purposely overfitting as much as possible (try nrounds >>500) this MAE of 2.55 appears to be the absolute minimum error achievable

mae\_test

# MAE 6.161 on validation, second holdout set

mae\_valid

# Conventional Regression Tree for the purpose of displaying featurew importance

protein\_tree1 <- rpart(RMSD ~ ., data = protein[train\_id,], control = rpart.control(minsplit = 1))

# Predict on Training set

train\_pred\_protein\_tree1 <- predict(protein\_tree1, protein[train\_id,])

# Calculate MAE on training set

train\_tree1\_mae <- mean(abs(train\_pred\_protein\_tree1-protein$RMSD[train\_id]))

# Train MAE 5.49

train\_tree1\_mae

# Predict on Test Set, Calculate MAE

test\_pred\_protein\_tree1 <- predict(protein\_tree1, newdata = protein[test\_id, ])

# Calculate MAE for test set

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

# Test MAE 5.49

mae\_protein\_tree1

# Plot Fitted Tree

rpart.plot(protein\_tree1)

#Gradient Boosted Tree

# Gradient Boosted Regression Tree with MAE Objective Function

gdboosteds\_protein <- xgboost(data = x[-testid,], label = y[-testid], nrounds = 10000, objective = "reg:absoluteerror")

gdboost\_test\_pred <- predict(gdboosteds\_protein, x[testid, ])

# For set.seed(13), nrounds = 10,000, minsplit = DEFAULT, trained MAE = 2.628

# Calculate MSE

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

# 19.3254 MSE for set.seed(13), nround = 10,000

mse

# Calculate MAE on test set

mae <- mean(abs(gdboost\_test\_pred-y[testid]))

# 2.979239 MAE for set.seed(7), nround = 10000

mae

importance\_matrix <- xgb.importance(model = gdboosteds\_protein)

importance\_matrix

xgb.plot.importance(importance\_matrix, col = rgb(124/256,148/256,198/256), xlab = "Importance (0-1)",

ylab = "Feature")

# Load CASP 5.9 Set from

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(7)

# Number of test samples = 1/3 observations

ntest <- trunc(n / 3)

#randomly select test IDs from

testid <- sample(1:n, ntest)

# Set PyTorch seed

torch\_manual\_seed(7)

# Scale variables, separate x labels ,y target var. Remove intercepts

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

y <- protein$RMSD

# PYTORCH NEURAL NETWORKS

#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, ])

# Calculate MSE on test set

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

mse

# MSE of 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")

# 2nd Model evalauted, Addition of 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 for ModNN2

#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 = 1)

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 for ModNN3 Neural Network

# 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 on the 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

# Addition of Binary Step (Threshold) Hidden Layer

# Threshold of 15 Angstroms

# Intended to ebtter predict RMSD hta appears to be steeped at the 15 Angstrom level

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

# 215 seconds 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 successive RELU ELU 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")

# Separation of features Nueral Network

# Attempt to isloate Polar surface areas which appear to be important for RMSD prediction, likely a function of van der Waals force

modnn6 <- 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, F5, F6, F7, F9

self$activation2 <- nn\_linear() # F4 F3 (Polar Surface Areas)

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

# Forward pass for each feature with the corresponding activation function

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

# Concatenate the output tensors from each feature and pass through the output layer

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

self$output()

}

)

modnn6\_setup <- setup(modnn6 ,

loss = nn\_mse\_loss(),

optimizer = optim\_rmsprop,

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

modnn6 <- set\_hparams(modnn6\_setup, input\_size = ncol(x))

# Training ModNN6 Separation of Features Nueral Network

# Approximately 240 seconds for each epoch

fitted6 <- fit(modnn6,

data = list(x[-testid, ],

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

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

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

epochs = 100)

# Group Method of Data Handling Neural Network (GMDH)

# Define the base models for the GMDH NN

# GMDH function is from GMDHreg library

# Training Multilayered Iterative (MIA) GMDH Neural Network

# Fit GMDH NN model

modnn7 <- gmdh.mia(X = as.matrix(protein[-testid, 2:10]),

y = as.matrix(protein[-testid, 1]),

prune = ncol(protein[-testid, 2:10]),

criteria = "PRESS")

# PRESS criteria: Predicted Residual Error Sum of Squares takes into account all information in data sample and it is computed without recalculation each test point.

# "test" is alternative argument, estimation of RMSE and is computationally more efficient

summary(modnn7)

# Predict on validation set for GMDH NN

pred.modnn7 <- predict(modnn7, as.matrix(protein[testid, 2:10]))

sum(is.na(pred.modnn7[,1]))

# For set.seed(13), there are 27 NA values

# Get row numbers for NA vlaues

missing\_rows <- which(is.na(pred.modnn7[,1]))

missing\_rows

pred.modnn7\_clean <- pred.modnn7[-missing\_rows, ]

# Should be 0

sum(is.na(pred.modnn7\_clean))

# Remove missing rows from the test set

protein.test <- protein[testid,][-missing\_rows,]

# Calculate MAE on test set

mae <- mean(abs(pred.modnn7\_clean - protein.test[,1]))

# With set.seed(13), MAE is 4.359865

mae

# Plot predicted RMSD against actual RMSD

plot(protein.test[,1], pred.modnn7\_clean, xlab = "Actual RMSD (Angstrom)", ylab = "Predicted RMSD (Angstrom)", main = "NN7: 136 Layer MIA GMDH NN")

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

# Fit GMDH GIA NN model

modnn8 <- gmdh.gia(X = as.matrix(protein[-testid, 2:10]),

y = as.matrix(protein[-testid, 1]),

prune = 9\*2,

criteria = "test")

ncol(protein[-testid, 2:10])

# Predict on validation set for GMDH NN

pred.modnn8 <- predict(modnn8, as.matrix(protein[testid, 2:10]))

sum(is.na(pred.modnn8[,1]))

# For set.seed(13), there are 27 NA values

# Get row numbers for NA vlaues

missing\_rows <- which(is.na(pred.modnn7[,1]))

missing\_rows

pred.modnn7\_clean <- pred.modnn7[-missing\_rows, ]

# Should be 0

sum(is.na(pred.modnn7\_clean))

# Remove missing rows from the test set

protein.test <- protein[testid,][-missing\_rows,]

# Calculate MAE on test set

mae <- mean(abs(pred.modnn7\_clean - protein.test[,1]))

# With set.seed(13), MAE is 4.36

mae