**Phylogenetic Tree Assembly**

Itzik Vainsenker 302280144

Oz Granit 203202684

The Problem:

A phylogenetic tree, describes the evolutionary relationship among a set of organisms,

genes, or genomes. Since there is a vast amount of possible trees that can be assembled from a given set of sequences ,All the techniques for inferring the most likely tree have to compromise on either accuracy or runtime.

Goal: Reconstruct the tree which best explains the evolutionary history of this gene.

The Data:

21,152 datasets, each dataset contains ~10-~50 different species.

Each dataset is a set of N aligned sequences (genes, a string made from ‘atcg’ characters of length ~200-~5000) from N different species.

Performance Metrics:

We use log-likelihood analysis of phylogenetic trees as our performance metric, the higher the score, the better the resulting tree.

Likelihood is a common optimization criteria in phylogenetic (Felsenstein 1981)

Baseline Approach:

A standard approach in phylogenetic tree inference is to apply hill-climb heuristic: given the current tree, test all trees that are similar in topology (i.e., neighboring trees) and move to the tree with the highest likelihood.

In the hill-climb approach, our challenge is to determine the method that the next step in the “climb” will be decided by. Essentially, our method will follow the pattern:

* Choose an initial tree
* Extract relevant features from the tree
* Usea **Machine Learning trained model** to predict a set of neighboring trees with the highest likelihood
* Calculate the likelihood of the trees from the predicted set and “climb” to the one with the maximum likelihood
* Repeat

The process ends when none of the set of neighboring trees has a score higher than the current one.

In the description above the definition of a neighbor and the likelihood of a tree remained vague.

The possible neighbors of a tree are chosen by a process called pruning and regrafting (SPR). An SPR neighbor is obtained by pruning a subtree from the main tree and regrafting it to the remaining tree.

We take the SPR neighbor tree space and log-likelihood calculations as given from the biology domain.

Machine Learning Improvements:

For reference we use the baseline method described above, and as a basic machine learning model we use a Random Forest.

We will try to improve results by:

* Usingdifferent Machine Learning models, replacing random forest (for e.g. a NN)

Using deep-learning we can insert as many features as we wish, even a tensor representing the whole tree and have the neural-network learn the right features.

Using that whole tree might not give us good results therefore we thought about implementing a network inspired by CNN with filters that cover nodes and their neighbors, also looking into GNN.

* Choosing an initial tree Usinga Machine Learning model, by which possibly reducing the hill-climb runtime.

Roll-out SPR neighbor:

* Usinga Machine Learning model to predict the best 2 or 3 step neighbor, not just one step neighbor, or even having the neighbor degree per step as a parameter.

Reinforcement learning attempt:

* Train Double Deep Q Network (model-free, off-policy):
  + Current Q-network w is used to select actions
  + Older Q-network w is used to evaluate actions
* End-to-end learning of values Q(s, a) from features s (current tree and clipped data)
* Input state s is stack of raw MSAs (=multiple sequence alignment) and current tree
* Output is Q(s, a) for a finite number of move-type('prune', 'rgft' or 'merged')Xtree\_node positions
* Reward is change in likelihood for that step

We consider a method preferable if it has a better accuracy OR runtime.

Regression-Model:

This part is a description of our attempt to build a linear-NN model to predict the log-likelihood value of a given tree, based on a list of features, replacing the need to calculate said value.

When taking into consideration that the hill-climbing approach requires calculating the log-likelihood value of all SPR neighbors (O(n^2) of tree nodes) of a given tree on-every step of the hill climb, this estimation saves a great deal of calculations.

Features used as input for the model:

FEATURE\_LIST = ['edge\_length\_prune', 'longest\_branch', 'ntaxa\_prunned\_prune', 'pdist\_average\_pruned\_prune',  
 'tbl\_pruned\_prune', 'parsimony\_pruned\_prune',  
 'longest\_pruned\_prune', 'ntaxa\_remaining\_prune', 'pdist\_average\_remaining\_prune', 'tbl\_remaining\_prune',  
 'parsimony\_remaining\_prune',  
 'longest\_remaining\_prune', 'orig\_ds\_tbl', 'edge\_length\_rgft', 'ntaxa\_prunned\_rgft',  
 'pdist\_average\_pruned\_rgft', 'tbl\_pruned\_rgft',  
 'parsimony\_pruned\_rgft', 'longest\_pruned\_rgft', 'ntaxa\_remaining\_rgft', 'pdist\_average\_remaining\_rgft',  
 'tbl\_remaining\_rgft',  
 'parsimony\_remaining\_rgft', 'longest\_remaining\_rgft', 'topology\_dist\_between\_rgft',  
 'tbl\_dist\_between\_rgft',  
 'res\_tree\_edge\_length\_rgft', 'res\_tree\_tbl\_rgft']

The data was formatted as a large csv file(12~Gb), each row represents a tree, the columns contain the features detailed above and the correct label (LL).

As a first step we divided the file into test and train file, where each row was assigned independently, at a ratio of 20-80.

To train the model, each step we sample a batch from the Train file and train the model according to the batch sampled and its Gradient.

At every plot – the x-Axis represents the number of those steps taken.

We are aware that usually one would go over the entire Train file in an ‘epoch’ and divide it into batches, however we were looking to avoid holding the Train file in main memory as it is large.

The test file was used to evaluate test-loos, which is calculated as the average loss on the entire test file. Calculating the test-loss is very time consuming, so we limited ourselves to testing every 0.5M steps or so. The main goal of testing throughout training was to notice when we are nearing overfitting, as the test loss will grow.

We used different net models and different learning rates and batch sizes, all of our models for the regression problem were using the ‘Relu’ function and 1-5 layers deep.

The models are detailed and numbered under ‘regression\_model.py’.

To get a first basic assessment of our models we compared them to a ‘Naïve-model’ which predicted the averaged label of the Train file as the label for all inputs.

We used two of Dana’s indexes to rank the results:

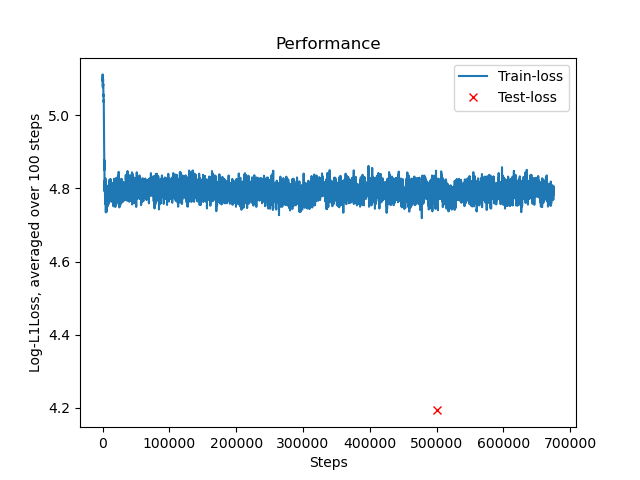
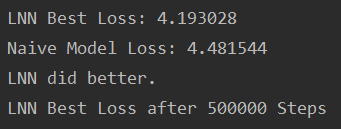
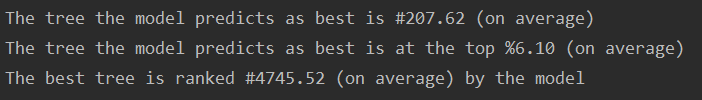
* “best\_predicted\_ranking" - # returns the true ranking of the tree that was predicted 1st by the model"
* “best\_empirically\_ranking" - # returns the ranking the model predicted for the (true) best tree

For “best predicted ranking" we also added the index as a percentile.

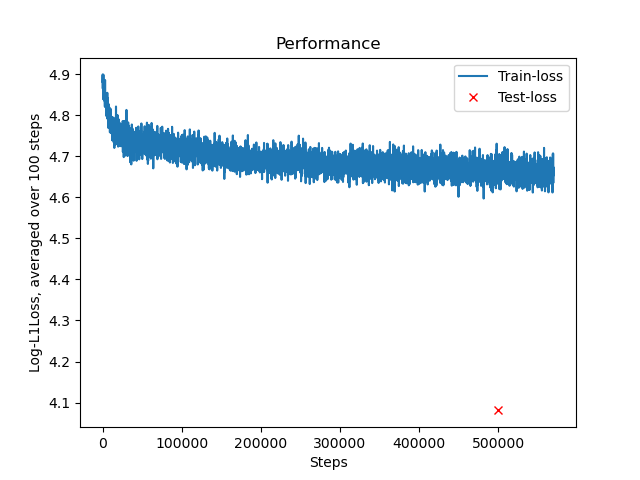
As a loss function we initially chose MSE, which caused exploding gradients, later we switched to L1.

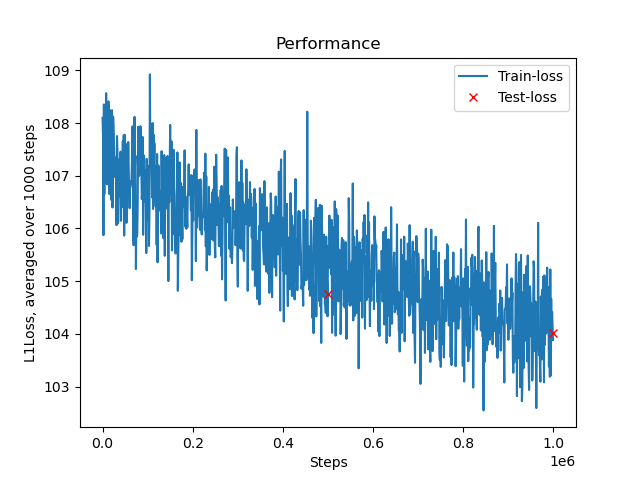
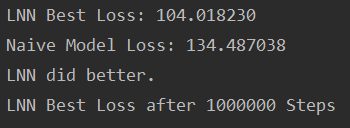
Below are detailed the results of some of the models tested.

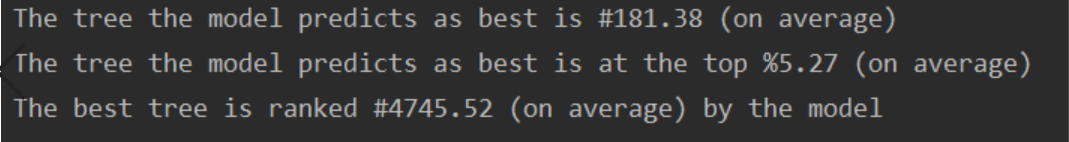
Some of our experiments used Adam optimizer or a decreasing learning rate, those delivered same or poor results – so are not shown here.

**model#3, lr=1e-4, bs=64:**  

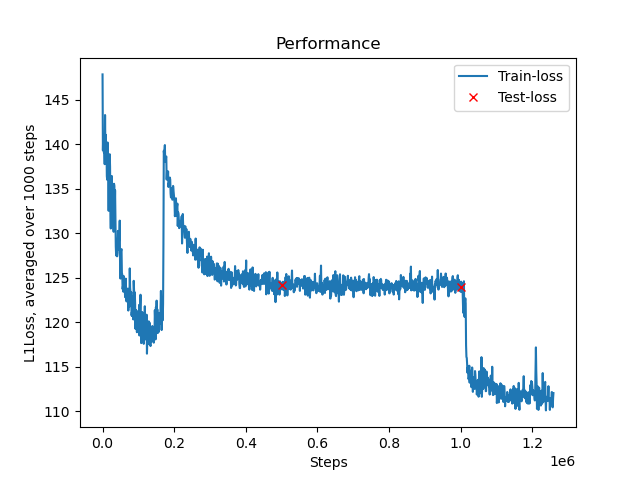
**model#2, lr=1e-5, bs=64:**



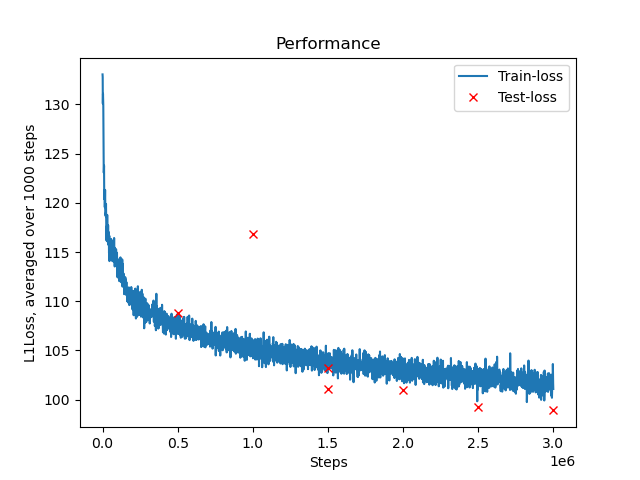
continued run from 0.5M: 

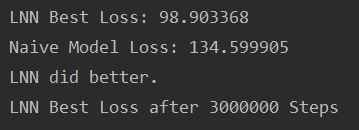


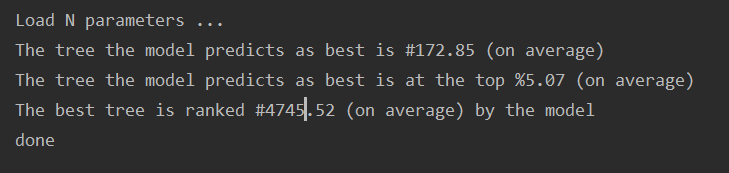
**model#4, lr=1e-3, bs=64:**



**model#4, lr=1e-5, bs=64:**







Classification-Model:

Taking a modified approach on the prediction problem – we trained a model to predict which tree is best instead of trying to predict a score for each tree and then taking the highest score.

The dataset had to be modified- so that the labels contained the rank (1 to #of neighbor trees) of each tree.

Loss function was cross-entropy.

Two different approaches were taken, the first defined the classes as 1-100, and every rank over 100 was treated as 100, since we only care for the top ranked trees and the ability to separate between them.

The second divided the ranks of neighbor trees to percentiles (2.5% apart) to use as classes.