*Bioinformatics*, 2018, 5–9

doi: 10.1093/bioinformatics/xxxxx

Advance Access Publication Date: DD Month YYYY

Phylogenetics

|  |
| --- |
| Phylogenetics  Finding Parsimonious Phylogeny With Neural Nets By Converting Into Continuous Space  Jesse Galef  Abstract  **Motivation:** While the question of scoring the parsimony of a phylogenetic tree is simple once a tree is defined, finding the most parsimonious tree is a challenging problem.  This project aims to design a technique to convert the problem from discrete, binary values into a continuous space that a neural net could be trained on.  **Results:** On simple test cases of five species and six species, the neural net was able to generate a tree as parsimonious as the Nearest Neighbor Interchange technique.  **Availability:**  **Contact:** jesse.galef@columbia.edu  **Supplementary information:** Supplementary files are available at *github.com/jessegalef/ProbabalisticTrees* online. |

# Introduction

While the question of scoring the parsimony of a phylogenetic tree is simple once a tree is defined, finding the most parsimonious tree is a challenging problem. The number of possible trees increases dramatically with the number of species included, going from three possible trees when there are four species to over 2 million when there are 10 species. An exhaustive search of that "tree-space" is infeasible, and current techniques rely on heuristics and branch-and-bound search algorithms.

This project aims to design a technique and the infrastructure to transfer the problem from discrete, binary values into a continuous space that a neural net could be trained on. Such a neural net could be very flexible and able to incorporate a diverse set of data – both aligned genome sequences and geographic location of the DNA source, for example – in finding a parsimonious tree more quickly.

# Methods

The central change in creating a phylogeny in continuous space was to frame the tree as a directed graph, with each species and most recent common ancestor acting as a node. The edges of this directed graph were encoded as a matrix of probabilities representing how likely each node was to be another node's immediate parent. (See fig. 2 for an example.)

Resulting components created were:

* A TreeLayer class
* A TreeRegularizer class
* A probabilistic parsimony loss function
* A neural net architecture

These components were implemented with a combination of Python’s Keras package with further extensions written in TensorFlow.

A probabilistic tree with N species would have N-1 MRCA nodes, and be encoded as a (2N-1 x N-1) matrix of edges. The TreeLayer construction takes a flat tensor of values as input and applied trainable weight and bias kernel values to create this matrix as output. These edge strength/parenthood probabilities can then be scored and optimized for with a custom parsimony function taking into account the true values of the data.

## Constraints in Constructing the Tree

In a valid tree, there are a number of additional constraints to the parenthood probability matrix the TreeLayer attempted to address.

First, edges between nodes must be in one direction or the other, so the TreeLayer subtracts the parentage matrix from its transpose and setting any negative numbers to 0. This has the added benefit of setting the diagonal to zero - no node can be its own parent.

Second, the parenthood probabilities of each node must sum to 1, and each parent’s probabilities of having a specific child must sum to 2.

Cappellini et al (2009) investigated the generation of doubly stochastic matrices and suggested two algorithms. The TreeLayer implements the first algorithm each call, iteratively normalizing rows and columns until the matrix converged on a doubly stochastic matrix to any level of precision desired. Additional regularization was then applied to encourage values closer to 0 or 1.

A further problem was that of the graph forming loops – allowing species to be their own grandparents with some non-zero probability. The solution takes advantage of the fact that the tree can be modeled as an absorbing Markov chain in which the root is the absorbing state.

Absorbing Markov chains have a “fundamental matrix”, denoting how many times an agent starting at one is expected to visit each other state before being absorbed. Values along the diagonal indicate how many times an agent is expected to return to a state it has already visited. Penalizing the sum of these diagonal terms encourages a well-formed tree in which all nodes can reach the root efficiently without cycles

## 2.2 Scoring with a Probabilistic Parsimony Loss Function

Scoring a standard tree according to Fitch's algorithm is a dynamic programming exercise relying on discrete values - taking different actions based on whether the intersection of two sets is empty or not. This is a non-starter for probabilistic trees mid-training, which might be comparing several different nodes each of which has a distribution across nucleotides.

Instead, the probabilistic parsimony loss function uses the known nucleotide values to approximate the best assignments for each MRCA node in an iterative assignment process. The error is the mean absolute difference between each node and its parent, weighted by how likely that node was to be its immediate parent.

This technique was run on the 879-nucleotide-long gene NM\_008288 from the UC Santa Cruz golden alignment of vertebrate species. Five species were used in the initial test run: Human, Chimpanzee, Rat, Mouse, and Lizard. The similarity between Humans and Chimpanzees, and between Rats and Mice, was expected to make for a simple test of the algorithm.

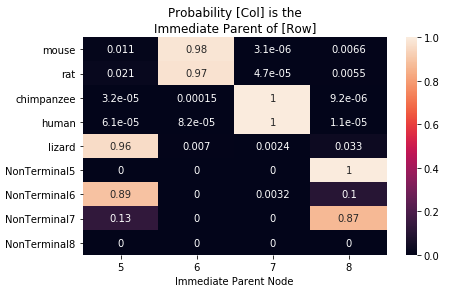
The aligned sequences were one-hot encoded and converted into a five-by-five matrix each position in the alignment indicating whether each pair of species shared the same nucleotide at that location. This binary matrix was passed in batches of 64 sequence locations through a series of increasingly-large dense layers with rectified linear unit activation functions before being fed into the regularized TreeLayer.

# Results

After 100-300 training epochs, the neural net was converging on a well-defined tree in which each non-root node had a single parent and achieved the desired bifurcating structure. It correctly grouped Humans and Chimpanzees, Mice and Rats, and Lizards last (fig. 1).

To serve as the comparison tree, I used BioPython’s implementation of the Nearest Neighbor Interchange (NNI) algorithm. I scored the trees with BioPython’s implementation of the Fitch algorithm, counting the total number of mutations the given tree would require to produce the species’ true genomic sequences.

After converting the probabilistic tree into discrete values, the tree produced by the neural net matched the Fitch parsimony score of the Nearest Neighbor Interchange-produced tree (fig 2). Three random initial starting points were chosen, and the same tree was produced each time.



Because it was possible for the neural net to be “bumped” out of a better solution to a local optimum, a Keras callback was added to keep the best tree during training (fig. 3).

The next test was adding a sixth species – I chose armadillos. After training the net for 500 epochs, it was still capable of converging on a tree that got the same parsimony score as the tree the Nearest Neighbor Interchange produced (see additional Figure 4,5).

Further testing should be done on more species, more genes, and with different architectures.

## Flexibility of the Framework

Probabilistic tree encoding allows users to design any infrastructure they choose around the tree encoding, incorporating any information or structure they think will help the neural net find a parsimonious tree.

We don’t know how robust the framework is. If the space of possibilities is large and unexplored but the range of structures that produce good results is narrow (which we don’t know), the entire framework could be too fragile to use.

It was often difficult to know whether a part of implementation was sound or not because of the number of hyperparameters and decisions that could influence the resulting neural net. For example, the out-of-the-box adadelta optimizer had trouble setting the appropriate learning rate, and were prone to diverge if the parameters were poorly set.

However, these results look like a promising start and offer several avenues of exploration.

Acknowledgements

I would like to thank Professor Pe’er and TA Tyler Joseph for their guidance and patience as this project bounced around.

I would also like to thank my girlfriend Kate for her support and patience even as I drew graphs and matrices on every available surface in the apartment.

References

Valerio Cappellini1, Hans-Jurgen Sommers, Wojciech Bruzda and Karol Zyczkowski, (2009) Random bistochastic matrices. JOURNAL OF PHYSICS A: MATHEMATICAL AND THEORETICAL

Figure 1: Parentage Matrix produced by neural net & the resulting phylogeny

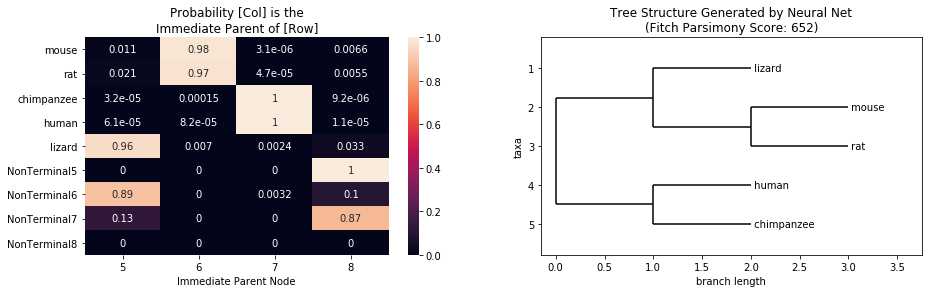


Figure 2: Phylogeny generated on the same five species by BioPython’s   
Nearest Neighbor Interchange method

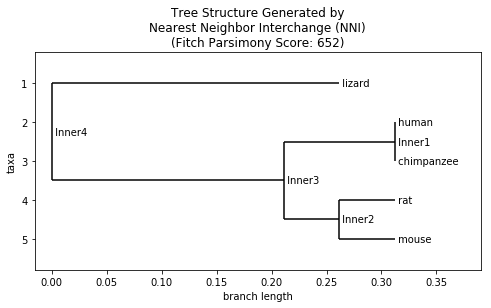


Figure 3: Training loss per epoch on five species test, demonstrating being “bumped” from a better local optimum

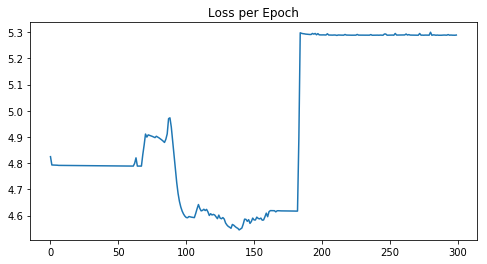


Figure 4: Parentage Matrix and resulting Phylogeny from the six-species trial

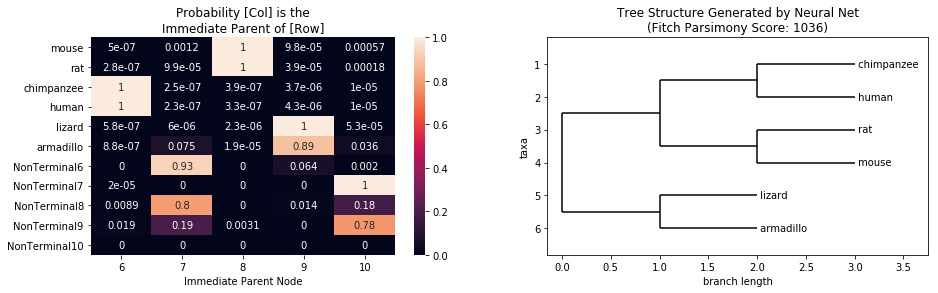


Figure 5: Phylogeny generated by BioPython’s Nearest Neighbor Interchange method on the same six species

