Tumor Phylogeny Topology Inference via Deep Learning

Erfan Sadeqi Azer[1,†], Mohammad Haghir Ebrahimabadi[1,2,†], Salem Malikić[1], Roni Khardon[1], and S. Cenk Sahinalp[2,*]

[1]Department of Computer Science, Indiana University, Bloomington, IN 47405, USA
[2]Cancer Data Science Laboratory, Center for Cancer Research, National Cancer Institute, National Institutes of Health, Bethesda, MD 20892, USA

Abstract

Motivation: Principled computational approaches for tumor phylogeny reconstruction via single cell sequencing (SCS) typically aim to identify the most likely perfect phylogeny tree through combinatorial optimization or Bayesian inference. Because of the limitations of SCS technologies, such as frequent allele dropout and variable sequence coverage, a noise reduction/elimination process may become necessary to infer a tumor phylogeny. Such noise reduction processes may aim to correct for the most likely/parsimonious set of false negative/false positive variant calls so as to construct a perfect phylogeny. Since these problems are NP-hard, available principled approaches for tumor phylogeny reconstruction are limited in their ability to scale up for handling emergent SCS datasets. In fact, even when the goal is to infer basic topological features of the tumor phylogeny rather than reconstructing it entirely, available techniques may be prohibitively slow. As a result, fast techniques to deduce, e.g. (i) whether the most likely tree has a linear (chain) or branching topology, or (ii) whether a perfect phylogeny is feasible from single-cell genotype matrix, without explicitly testing for the three gametes rule, are highly desirable.

Results: In this paper we introduce deep-learning solutions to the above mentioned problems for studying tumor evolution from SCS data. After training with sufficiently many examples: (1) our fully connected neural network for differentiating linear vs branching topologies, can improve the running time of the fastest combinatorial tumor phylogeny reconstruction methods by a factor of $\geq$ 1000, while achieving an accuracy of $\sim$ 98% on simulated data including 100 cells and 100 mutations with realistic noise levels (leading to mostly false negatives) of $10-15\%$; (2) similarly, our fully connected neural network for checking whether the input data permits a perfect phylogeny, achieves an accuracy of $\sim$ 90% on simulated data including 10 cells and 10 mutations, with similar noise levels; (3) finally, our reinforcement learning approach for tumor phylogeny reconstruction can actually eliminate noise and obtain the PP, when false negative/false positive rate $\leq$ 2%, for a large fraction of evaluation data sets with varying number of cells and mutations, even when trained with fixed size data sets of only 10 cells and 10 mutations - this may be useful for future clinical applications that would employ emerging SCS technologies with lower noise levels.

Availability: `https://github.com/algo-cancer/PhyloM`

Contact: cenk.sahinalp@nih.gov

[†]Joint First Authors
[*]Corresponding Author

1 Introduction

Cancer is an evolutionary disease characterized by progressive accumulation of somatic mutations in tumor cells. Deciphering evolutionary history of a given tumor represents an important problem in studies of cancer and can help us in better understanding of several clinically important aspects of the tumor including progression, metastatic spread, the existence of divergent subclones evolving simultaneously and many others.

Due to the importance of the problem, there has been rapid developments in the design of principled computational methods for tumor phylogeny inference. Many of these methods use bulk sequencing data where DNA from millions of tumor (and normal) cells are sequenced together. Tree inference from this type of data is typically based on the use of cancer cell fraction of detected variants - in particular single-nucleotide variants [36, 6, 11, 25, 28, 8, 12, 31, 27, 19], but also copy number (e.g., [39]) and structural variants (e.g., [9, 29]). While being cost-effective, the low resolution of bulk sequencing data is a limiting factor in tumor evolution modeling. In particular, bulk sequencing data from a single tumor sample typically admits a linear topology [15] as an optimal solution under common tree-scoring models. However, inferring whether the underlying tumor includes divergent subclones which evolve through distinct branches of the tumor phylogeny represents an important step towards better understanding tumor progression and improving treatment design.

Recent technological developments have enabled researchers to perform single-cell sequencing (SCS) experiments, where DNA from an individual cell is extracted, amplified and sequenced. SCS provides high-resolution data for studying tumor evolution at unprecedented detail, e.g., it offers the possibility to identify branching topologies, or to solve the general problem of inferring the complete history of tumor evolution. Early tools for inferring tumor evolutionary history from SCS data include SCITE [20], OncoNEM [30] and SiFit [41] (and its improved version SiCloneFit [40]), probabilistic methods for learning tumor phylogenies, as well as SPhyR [10] a combinatorial optimization based approach, all under the *infinite sites assumption* (ISA), i.e. each mutational gain can occur only once during the evolution of a tumor and is never lost (even though SiFit has a provision for possible loss and concordant gain of mutations).

When both SCS and bulk sequencing data for a tumor sample are available, two of the latest methods, namely B-SCITE [24] and PhISCS [26], can provide more accurate tumor phylogenies. These methods can also be applied solely to single-cell data. B-SCITE is a probabilistic method that integrates SCITE [20] and CITUP [25], whereas PhISCS is based on the use of combinatorial optimization and has two distinct implementations: PhISCS-I uses integer-linear programming while PhISCS-B formulates the same problem as a boolean constraint satisfaction program and solves it via available methods for weighted max-SAT. Both approaches can also be applied to SCS data only.

As summarized above, available methods for tumor phylogeny reconstruction, especially by the use of SCS data have important limitations. First many of these methods all employ ISA (even though some offer a provision for limited loss and concordant gain of mutations) and assume a uniform noise level (false negative as well as a false positive rate) - both subject to change with advances in our understanding of tumor evolution and SCS technology. More importantly, these methods aim to infer the most likely tumor phylogeny, and for that eliminate noise (due to allele dropout or low sequence coverage) with a maximum likelihood/parsimony approach. As such, they all aim to solve an NP-hard problem from scratch, and thus cannot scale up to handle large SCS datasets. Even when the goal is to infer basic topological features of the tumor phylogeny rather than reconstructing it entirely, these methods cannot easily handle SCS data involving a few hundred mutations and cells. As a result, fast techniques for inferring key features of a tumor

phylogeny, e.g. those that can discriminate linear from branching topologies, especially for SCS data sets with high noise levels (as per the current practice) are in high demand. Similarly, it is desirable to quickly infer whether any noise elimination is necessary for constructing a perfect phylogeny. Finally each of these tools have required a great deal of human effort in algorithmic design and implementation, as each technological advance in data generation necessitated the development of completely novel methodologies. It is thus highly desirable to have a general computational approach that can adopt to technological change, simply through training it with new data, without the need for explicit objective or noise profile modeling.

It may be possible to address these limitations via a *data-driven*, machine learning approach that considers a general set of functions and choose one that best fits a training dataset - which could be simulated or obtained through real world measurements. Such an approach could not only reduce the inaccuracies in noise profile modeling but also identify implicit underlying patterns in the data or problem towards developing more realistic objectives. Recent advances in deep learning have demonstrated remarkable generalization of formulations for solving many problems (e.g., mastery over games such as Chess, Go[34], and Poker, or handling natural language tasks via BERT [7] and most recently RoBERTa [23]) and it is possible that a single deep learning architecture may succeed in inferring distinct properties of tumor phylogenies when properly trained on sufficient number of datasets.

In recent years, many computing applications have experienced a shift to data-driven approaches, from deciphering handwritten text (e.g., [5] for digit recognition) to natural language processing (NLP), e.g., [7, 23]. Problems with poorly understood/formulated objectives such as those in structural biology (e.g. *AlphaFold* [33] for inferring the 3D structure of a protein sequence) seem to benefit considerably from deep learning methods. However we are not aware of any data driven approach for tumor phylogeny inference.

In this work we offer the first data-driven tumor phylogeny reconstruction methods to address the limitations of existing strategies. We have used SCS data with deep neural networks and reinforcement learning to infer topological features of a tumor phylogeny, as well as the most likely evolutionary history of a tumor. In order to achieve this we had to overcome several distinct challenges: (1) The neural network should ideally be designed so as to handle varying number of cells and mutations. Alternatively, for models with fixed sized inputs, it is desirable to use our domain knowledge to prepare the data in a manner that will facilitate success in predictions.

(2) Given the use of neural networks, one needs a large number of samples for proper training. Unfortunately, the number of publicly available tumor SCS datasets are not sufficiently large to train deep learning models, thus we needed to produce a large number of simulated SCS data. (3) Errors/noise in SCS data add further complexity to the problem and the proposed deep learning framework had to be evaluated in terms of noise tolerance. (4) The chosen architecture also expects a specific type of supervision which we must be able to compute. In order to perform noise reduction/elimination in the input *genotype matrix* (see below for a formal definition) extracted from SCS data, it is possible to offer supervision in the form of a dataset of noisy inputs along with their denoised versions. An alternative and cheaper supervision is offered by a feedback mechanism for determining whether a candidate output of the neural network is successfully denoised. A third alternative is given by a cost function that indirectly helps supervise a reinforcement learning process.

Inspired by novel deep learning approaches to combinatorial problems such as the *reinforce policy gradient algorithm* [38] for the traveling salesperson (TSP) and the knapSack problems [2], and, to a limited degree the *NeuroSAT* approach [32] for the satisfiability (SAT) problem using a single bit supervision, we established a computational framework to successfully address all the above challenges as follows. (i) We employed a Reinforcement Learning approach to train a denoising

model without a need for the ground truth. The cost function we used is novel and problem specific. (See Section 5). (ii) We devised a novel method to transform the input matrix (obtained from noisy SCS data) to feed to the neural network. This method incorporates the noise rates as well as coordinates and values of entries in the given matrix (See section 5.1). This representation also supports training and testing instances of different sizes and encourages robustness to permutation of rows and columns. (iii) We also present an example for incorporating domain knowledge to improve the performance of a machine learning model: this was achieved by sorting the columns of the input genotype matrix in a preprocessing step. (See Section 4.) (iv) Finally we introduced a new method to construct simulated input data with a given number of mutations and cells. This method along with the simulation tools from previous work allows us to construct *unbiased* datasets to train our models.

Results. We study two major problems. First, given a noisy SCS dataset in the form of a genotype matrix, we consider the problem of inferring whether the tumor phylogeny topology is linear or branching, i.e. contains at least one branch or not. The method we devised for this problem succeeds to learn and differentiate the two topology types in $\sim 98\%$ of randomly chosen noisy genotype matrices of size 100×100. The trained model runs at least 1000 times faster than the fastest available baseline techniques (see Table 6.2), which need to first denoise the input and then examine the tree. The trained model is noise tolerant, scalable, and can analyze real datasets (see Section 6.2). Moreover, having an efficient prediction function allows us to systematically study the relationship between the amount and type of noise in the input and the information preserved in the data to enable correct prediction.

For the more general problem of fully reconstructing the full tumor phylogeny, we introduce a Reinforcement Learning based approach, which finds a solution for, e.g. 92% of the randomly generated genotype matrices of size 10×10. Furthermore, our trained model on genotype matrices of size 10×10 successfully solves a notable proportion of larger genotype matrices, demonstrating the *generalizability* of our approach.

We also describe results on the decision version of this problem, i.e., whether a given matrix is noise free and thus admits a perfect phylogeny, or does not satisfy the so called three-gametes rule. Our deep learning approach for this binary classification problem offers notably more accurate results in comparison to support vector machines. Crucially, sorting the columns of the input genotype matrix according to their binary representation as a preprocessing step further improves the accuracy of the trained models by 13-27% (See Table 2).

2 Preliminaries

Given a positive integer z, let $[z]$ denote $\{1, 2, \ldots, z\}$. For a given matrix M, we denote by $|M|_0$ the number of nonzero entries in M. For any boolean proposition P we use Iverson bracket notation as

$$[\mathsf{P}] = \begin{cases} 1 & \text{if } \mathsf{P} \text{ is true} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Let the binary genotype matrix that represents the ground truth for the evolution of a tumor be denoted $A : \{0, 1\}^{n,m}$; here n and m correspond to the number of cells and mutations in the input SCS data respectively. Let $a_{i,j}$ for $i \in [n] \wedge j \in [m]$ denote an entry of A, for which $a_{i,j} = 1$ indicates the presence of mutation j in cell i, and $a_{i,j} = 0$ indicates its absence. It is often not possible to infer A from SCS data perfectly. Instead, suppose we are given A', an *approximate/noisy* version

of A, obtained through SCS data processing, where $a'_{i,j}$ for $i \in [n] \wedge j \in [m]$ denotes whether cell i is inferred to include mutation j.

Matrix A' is said to represent a *perfect phylogeny* (PP) if it satisfies the *three-gametes* rule, i.e.

$$\forall i,j,k \in [n] \wedge g,h \in [m], \begin{vmatrix} a'_{i,g} & a'_{i,h} \\ a'_{j,g} & a'_{j,h} \\ a'_{k,g} & a'_{k,h} \end{vmatrix} \neq \begin{vmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{vmatrix}.$$

Notice that the rows and columns in the definition above can be in any order. In case of an equality, the column-triplet (i,j,k) and row-pair (g,h) is said to have a *conflict* or a *three-gametes violation*.

We used the simulator described in [17] to generate PP matrices representing instances of A. To obtain instances of A', we *added noise* to A by choosing to *flip* each entry independently with a probability depending on the value of the entry - with predefined false positive (α) and false negative (β) rates. To ensure correctness of the labels, we check each generated matrix to verify that the values flipped generated at least one *conflict* and otherwise remove the matrix from the dataset.

Assuming an independent and identically distributed (i.i.d.) noise process explained above, the conditional probability that a (likely PP) matrix B (with entries denoted by $b_{i,j}$) generates A' is:

$$P(A'|A = B, \alpha, \beta) = \alpha^{N_{10}} \cdot \beta^{N_{01}} \cdot (1-\alpha)^{N_{00}} \cdot (1-\beta)^{N_{11}} \tag{2}$$

where N_{pq} for any $p,q \in \{0,1\}$ is defined as

$$N_{pq} = \sum_{(i,j) \in [n] \times [m]} [b_{i,j} = q] \wedge [a'_{i,j} = p]. \tag{3}$$

Throughout the paper we refer to this probability as the *likelihood* of B, for any (potentially PP) matrix B that is the result of a specified denoising process on A'.

2.1 Branching Inference

As mentioned in Section 1, the problem of inferring whether a tumor contains divergent subclones evolving through distinct branches of its phylogeny has potential applications in improving treatment design. Given a binary genotype matrix A' as the input, we consider here the problem of inferring whether the most likely denoised version of A' implies a chain (i.e. linear) or a branching topology.

Let us start with an important observation. We say that a binary matrix $C_{n \times m}$ has the *staircase* property, if $\forall i,j \in [n-1] \times [m-1]$, $c_{i,j} \leq c_{i+1,j}$ and $c_{i,j} \leq c_{i,j+1}$. In other words, no entry in C with value one is a right-neighbor or above-neighbor of another entry with value zero value. Observe that a (denoised) genotype matrix C implies a linear topology, i.e. has the *no-branching* property, if there exists a permutation on rows and permutation of columns of C such that the resulting matrix has staircase property.

Given C, we define binary function $\mathsf{Sing}(C) : \{0,1\}^{n,m} \rightarrow \{0,1\}$ to have value 1 if C has the no-branching property, i.e. implies a linear topology, and 0 otherwise. Our goal for the branching topology problem, as will be described in Section 3, is to train a machine learning approach to compute a function $f(C) : \{0,1\}^{n,m} \rightarrow \{0,1\}$ which agrees with $\mathsf{Sing}(C)$ on as many potential input matrices much as possible.

2.2 Noise Inference and Elimination Problems

The problem of denoising the binary matrix obtained from SCS data so as to obtain a PP is well studied in the literature [4, 26, 20, 24]. We consider two variants of this problem below: (1) The decision version, which we call the noise_inference problem problem, can help investigate whether neural network models can successfully check whether the three-gametes rule is satisfied by the input matrix. (2) The general denoising problem, which we call the noise_elimination problem problem, considers a noisy input matrix A' and considering the noise parameters, outputs a matrix B with the goal of maximizing the likelihood of B, i.e. $P(A'|A = B, \alpha, \beta)$.

noise_inference problem. The objective here is to decide whether the input A' is conflict-free. Let the function $\mathsf{Icf}(A') : \{0,1\}^{n,m} \rightarrow \{0,1\}$ have value 1 if A' contains at least one conflict, and 0 if it is conflict-free. Given a family of functions, our goal is to pick function $f(A') : \{0,1\}^{n,m} \rightarrow \{0,1\}$ that approximates Icf with the highest possible accuracy, where the accuracy of f is defined as the proportion of potential input matrices A' where $f(A') = \mathsf{Icf}(A')$.

noise_elimination problem. Given the input A', and noise parameters α and β, the objective here is to compute an output matrix B with the highest likelihood. The noise_elimination problemis the most computationally challenging problem considered in this paper since it cannot be easily formulated as a classification, regression or clustering problem which are better suited for deep learning approaches. More importantly the problem is NP-hard [4] and available principled solvers deploy sophisticated methods such as MILP, CSP, MCMC towards its solution for practical instances [4, 26, 20, 24]. Finally, since the problem is computationally intractable, it is not possible to construct large matrices with known optimal solutions to be used for training a neural network. This issue alone makes it impossible to use several successful machine learning techniques for our purposes.

3 Solutions to the Branching Inference Problem

Training Dataset. For this problem, we first generate a set of PP matrices A with $\mathsf{Sing}(A) = 0$, i.e. have a branching tree topology, through the procedure explained in Section 2. In order to generate a second, complementary set of PP matrices, for each matrix A from the first set, we describe below how a new random matrix A^L can be constructed with the same number of ones, but $\mathsf{Sing}(A^L) = 1$, i.e. satisfies the no-branching property. This not only guarantees the the union of these two sets have a balanced number of instances of branching and non-branching topologies, but also eliminates any *artificial* distinguishing features (e.g., based the number of ones in the matrix).

Briefly, given a matrix A with l ones, in order to generate A^L as described above, we produce a random non-decreasing sequence of n positive integers that add up to l (n is the number of rows in A). For the i^{th} value of this sequence, say k, we set the rightmost k entries of the i^{th} row of A^L, i.e. $A^L[i, m-k+1..m]$ to one and $A^L[i, m-k]$ to zero, satisfying the *staircase* property. This is followed by a random permutation of rows and columns of the matrix to finalize A^L, ensuring that A^L is a PP with the same number of ones with A, but is topologically different.

We combined the two sets of matrices and used them for training without adding noise. However we evaluated the trained model on both noisy and noise-free inputs.

Network Structure. A two-layer fully connected artificial neural network is used with either 10 or 100 hidden neurons. The input layer corresponds to an arbitrary-size batch of vectors with

length $n \cdot m$. We used tanh as the activation function. To avoid overfitting, a drop-out rate of 0.9 was used. Note that the architecture had a moderate level of sensitivity to the hyperparameter settings (i.e. the number of neurons and the drop-out rate).

4 Solutions to the Noise Inference Problem

In order to solve the decision version of the noise_inference problem, we formulated it as a classification task through supervised learning.

Training Dataset. We generated a set of PP matrices A, i.e., $\mathsf{lcf}(A) = 0$ as described in Section 2. In addition, for each matrix A we obtained its noisy version A' with predetermined noise parameters α and β so that $\mathsf{lcf}(A') = 1$. The combined set of matrices A and A' form a balanced binary dataset of noisy and noise-free matrices as required for classification purposes.

For some of our experiments we preprocessed each input matrix as per [16] by sorting its columns based on the binary number each represents (read from top to bottom). E.g. this would move column $\begin{bmatrix}0\\1\\1\end{bmatrix}$ to the left of column $\begin{bmatrix}1\\0\\0\end{bmatrix}$ as $(011)_2 = 3 < (100)_2 = 4$. Sorting columns was shown to help checking whether a matrix forms a PP or not in non-ML settings [16]. We show that it also helps ML strategies (see Section 6).

Network Structure. For this problem we use a two-layer fully connected network with 100 neurons in each layer. The input layer corresponds to an arbitrary-size batch of vectors with length $n \cdot m$. The Sigmoid function was used as the activation function for both layers. We used a drop-out rate of 0.2 for each layer for preventing our network from overfitting.

5 Solutions to the Noise Elimination Problem

This section presents our approach for solving the noise_elimination problem and thus contains our major contributions from a methodological standpoint.

5.1 Input Format

Recall that input to this problem is given as a binary matrix A'. We transform A' to a new matrix A'' as shown below, before feeding it to the neural network. Each row of A'' corresponds to exactly one of the entries of A'. The first two columns of a given row in A'' respectively represent the row and the column of the corresponding entry in A'. The third column represents the noise level and depends on the actual value of the entry - which is represented in the last column. In the simplest case, the value of the third column is either α - if the entry has value one, or is β - if the entry has value zero. In the more general case the user can specify a distinct false positive or false negative rate for each entry of the input matrix A'. As depicted below, this transformation is key to our ability for training the neural network with matrices of varying shapes/dimensions:

$$\underbrace{\begin{bmatrix} a'_{0,0} & a'_{0,1} & \dots & a'_{0,j} & \dots & a'_{0,m} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a'_{i,0} & a'_{i,1} & \dots & a'_{i,j} & \dots & a'_{i,m} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a'_{n,0} & a'_{n,1} & \dots & a'_{n,j} & \dots & a'_{n,m} \end{bmatrix}}_{\boldsymbol{A'_{n\times m}}} \rightarrow \underbrace{\begin{bmatrix} 0 & 0 & f(a'_{0,0}) & a'_{0,0} \\ 0 & 1 & f(a'_{0,1}) & a'_{0,1} \\ \vdots & \vdots & \vdots & \vdots \\ i & j & f(a'_{i,j}) & a'_{i,j} \\ \vdots & \vdots & \vdots & \vdots \\ n & m & f(a'_{n,m}) & a'_{n,m} \end{bmatrix}}_{\boldsymbol{A''_{n\cdot m\times 4}}}$$

where,

$$f(a'_{i,j}) = \begin{cases} \alpha & \text{if } a'_{i,j} = 1 \\ \beta & \text{if } a'_{i,j} = 0. \end{cases} \tag{4}$$

Note that the values of α and β can be estimated through the SCS data [21]. For example, β has been observed to be as high as 0.3 while $\alpha \approx 2 \times 10^{-4}$.

5.2 Method

Our approach is similar to the model-free policy-based Reinforcement Learning algorithm in [2], which was developed to solve TSP and knapsack problems. This framework requires a *cost function* to be defined on the space of potential outputs of the network. In the training phase, the optimization algorithm tries to minimize this cost function. Notice that this will be the only source of supervision we provide to our model for training purposes. Importantly, we do not need to know or calculate through alternative methods the ground truth A, which is more expensive to figure out in comparison to calculating our cost function as described below.

The cost function we came up with is defined for a given input matrix A' and a potential output matrix X, with entries $x_{i,j}$, and has two terms. The first term, $\mathsf{NumberOfConflicts}$ corresponds to the number of (3×2) submatrices violating the three-gametes rule. As such it will have value 0 if A' forms a PP:

$$\begin{aligned} \sum_{c_1\in[m],c_2\in[m],c_1\neq c_2} \Big(& \sum_{r_1\in[n]} [[x_{r_1,c_1}, x_{r_1,c_2}] = [0,1]] \cdot \\ & \sum_{r_2\in[n]} [[x_{r_2,c_1}, x_{r_2,c_2}] = [1,0]] \cdot \\ & \sum_{r_3\in[n]} [[x_{r_3,c_1}, x_{r_3,c_2}] = [1,1]] \Big) \end{aligned} \tag{5}$$

In addition to the $\mathsf{NumberOfConflicts}$, we use the negative log-likelihood of X as the second term of the cost function, i.e. $\mathsf{Cost}(X) : \{0,1\}^{n,m} \rightarrow R^+$ is

$$\mathsf{Cost}(X) = \mathsf{NumberOfConflicts} - \gamma \cdot \mathsf{ln}(P(A'|X = A, \alpha, \beta)). \tag{6}$$

where γ is a hyper-parameter for establishing a trade-off between the two terms. Recall that likelihood $P(A'|X = A, \alpha, \beta)$ is calculated using Equation 2. This cost function is aimed to capture the objective of our optimization problem, since among all matrices with $\mathsf{NumberOfConflicts} = 0$, our goal is to compute the one with the highest likelihood.

The core architecture we use here, within the Reinforcement Learning framework, is a *pointer* network [2]. Figure1 shows the components of the pointer network and the flow of information (when it is used for evaluation - there are additional connections used for training) - as summarized below.

The embedding layer embeds the input array of shape $(n \cdot m) \times 4$ into an array of shape $(n \cdot m) \times d$, where d is the embedding length used in the network. The encoder consists of q layers of convolution [13]. Each one of the $(n \cdot m)$ rows (with length d) of the embedded array goes through the q convolution layers, to produce a length d vector placed in the buffer as a new row. In our experiments, the values of d and q were set to 8 and 2, respectively. The decoder is a long short-term memory (LSTM) [18] layer and the attention layer [1] is a network of fixed size. Together they compute the output of the neural network. Specifically, the attention layer iteratively learns a real valued *score* function on d dimensional vectors. In each iteration, this function is applied to each row of the buffer to obtain its score, which is then used to pick a single row from the buffer (corresponding to a row of the input to be flipped), to be fed to the LSTM. The output of the LSTM is another vector of length d which is used to update its memory and is then fed to the attention layer, to be used to update the score function. The cost function we described earlier is employed both in updating the LSTM as well as the score function of the attention layer. Note that the score function is not trivially applied only to the rows of the buffer that have not been picked earlier. Those that have been picked are automatically assigned a fixed low score. A soft-max function is then used on these scores to get a probability distribution over the rows of the buffer. In the next iteration, the row to be fed to LSTM is sampled according to this distribution.

This neural architecture is used to decide and evaluate a set of proposed flips through the cost function. More specifically, the buffer row selected by the attention mechanism at iteration is considered to be the proposed flip location. We combine all the proposed flips to A' to get the output matrix B. Then the value of $\mathsf{Cost}(B)$ is calculated and used as a feedback for the network. Since the architecture is trained via policy gradients, the total cost function is sufficient as the feedback signal, and the learned policy, implicit in the network and attention mechanism, will attempt to minimize the cost of the output B.

6 Experiments

In this section we discuss our experimental setup and our results. The codes for the construction of datasets using [17] and running the training and evaluation for all of the networks described in Sections 3, 4, and 5 are available at `https://github.com/algo-cancer/PhyloM`.

6.1 Experimental Setup

All experiments presented here were performed using deep learning (DL) nodes in *Carbonate*, a computation cluster at Indiana University [35]. The specifications of these nodes, along with the version numbers of the packages used, are provided in the README.md file of our repository to ensure reproducibility of our results.

We note that the accuracy values in Figures 2, 4, and 5 are higher for the evaluation phase than the training phase. This reflects the fact that the error during training was measured in a noisy manner using dropout, which is necessarily worse than the true error rate of the network.

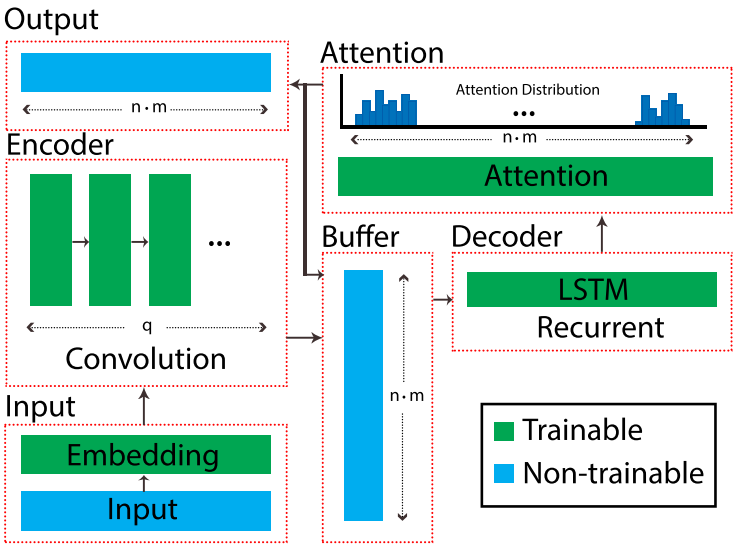


Figure 1: The architecture of the network used for the noise_elimination problem. The roles for the inner-components of the network including embedding layer, encoder, decoder and attention mechanism interacting with the buffer, are explained in the text. In each training epoch, a batch of input matrices is fed to the network. Each input matrix $A'_{n\times m}$ in a batch, is preprocessed to an $(n \cdot m) \times 4$ matrix as described in Section 5.1. The output of the attention layer, right before final output layer, is a function that forms a distribution over the $n \cdot m$ entries of the input matrix A' (after having been processed through the embedding layer and the encoder). The attention layer iteratively forms the final **output** of the network which is a set of entries of A' to flip.

6.2 Branching Inference

Recall the no_branching problem, where the goal is to identify whether the tumor phylogeny has a linear or a branching topology.

Learning curves. In the first set of experiments for this problem, we split the dataset to training and evaluation sets with a ratio of 9 to 1. We designed several experiments with different settings and all of them show success in predictions. We include results for three settings in Figure 2: (a,b) a dataset of 2000 matrices of size 100×100 and a hidden layer of size 10, (c,d) same data set, now with a hidden layer of size 100, and (e,f) a dataset of 10000 matrices of size 500×500 and hidden layer of size 100. In each of these experiments we run the training process for 200 epochs. Note that the model distinguishes up to 98% of randomly chosen instances of size 100×100 correctly - see for panel (c).

Comparison of Running Time for the no_branching problem. We performed an experiment to compare the running time between our approach and a baseline for the no_branching problem. The baseline first tries to denoise the input matrix and then detect if the resulting tree has a linear structure. For the denoising phase, we use the ILP-based method from [26]. We note that unlike our approach, the baseline's running time is affected by the presence of noise. Especially, there is an approximately monotonic relation between the baseline's running time and the amount of introduced noise.

To show superiority of our approach, we choose the worst running time among 50 runs for each input regarding our approach, while we consider only the best cases for the baseline which correspond to the inputs without noise. The superiority of our approach in running time is observed to be at least 1000 times (See Table 6.2).

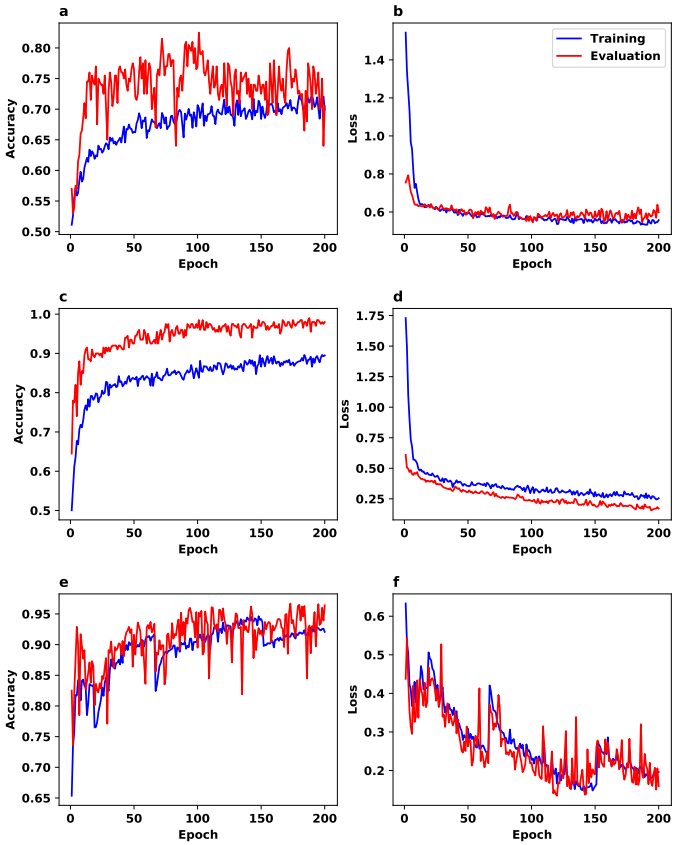


Figure 2: The accuracy (left column) and binary cross entropy loss (right column) plots corresponding to the no_branching problem and the architecture described in Section 3. We used a dataset of 2000 matrices with dimension 100×100 and hidden layer of size 10 in (a, b), the same the dataset as in (a) and (b) with hidden size 100 in (c, d), and a dataset of 10000 matrices with dimension 500×500 and hidden layer of size 100 in (e, f).

Input Dimensions	100×100	500×500
Time: Our approach	0.0372 sec	0.0537 sec
Time: PhISCS	> 80 sec	> 3772 sec

Table 1: A comparison of the running times between our approach and the baseline for the no_branching problem. For our approach, we tested 50 noisy cases ($\alpha = 0.002$, $\beta = 0.2$) for both matrix sizes, and report the maximum running time we observed. For the baseline, we report the best case running time for noise free input matrices (adding noise increases the running time substantially). All times are in seconds.

Evaluation of noise-tolerance for the no_branching problem. In another experiment, we tested the noise tolerance of our model, on three different values for β/α (the false negative to

false positive rate ratio). For each value, we plotted the percentage of inputs whose topologies are correctly distinguished as a function of the false negative rate β as shown in Figure 3. Especially when the false positive value is relatively low ($\beta/\alpha = 10$) the accuracy of the model remains close to perfect across all (realistic) values of the false negative rate β.

Notice that the speedup shown in Section 6.2 enabled us to run an extensive number of experiments to obtain the data points in Figure 3. The plots in this figure suggest a new understanding of relationship between the amount of preserved information that could be captured by our neural network in the presence of different noise levels. For example, for settings where false negative and false positive rates are equal, a false negative rate of ≤ 0.15 preserves the deduced topological outlook fairly consistently. The drop in accuracy after this point seems more steep.

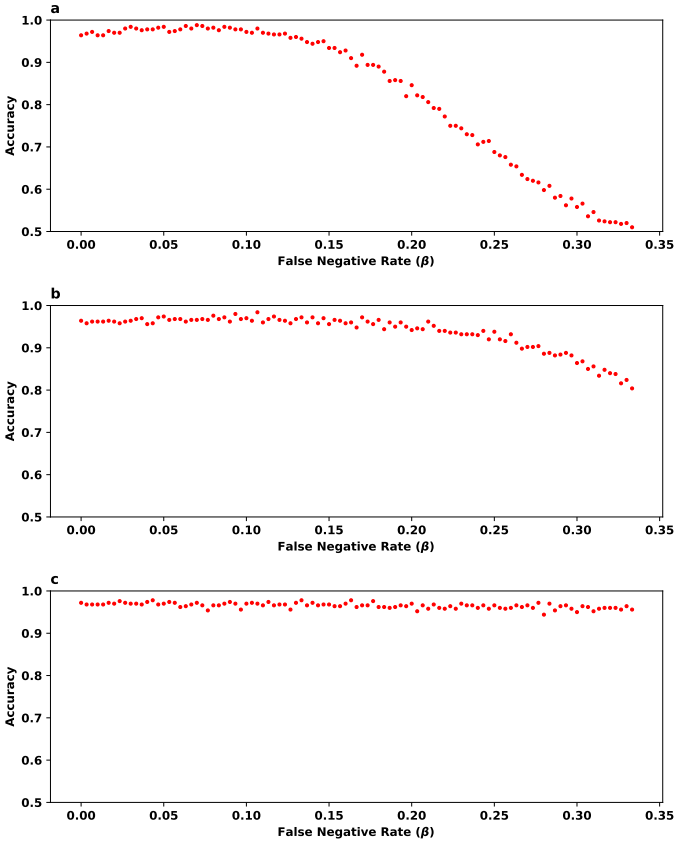


Figure 3: The evaluation of noise tolerance for our model for the no_branching problem. In each panel, the y-axis (accuracy) represents the fraction of instances classified correctly from 500 randomly chosen instances, not seen during training, with equal number of instances from each topology type (See Section 3). The trained model consists of two layers with size 100 and 0.9 drop out rate. This model was trained using 2000 matrices of size 100×100 with no noise for 200 epochs. The three panels differ in the ratio of false negative to false positive rates: (a) $\alpha = \beta$, (b) $\alpha = \beta/2$ and (c) $\alpha = \beta/10$. Note that the noise tolerance of the method gets more robust as the relative value of α decreases.

6.3 Noise Inference

We trained and evaluated our proposed neural network for the noise_inference problem introduced in Section 4, on multiple datasets. We consider two sizes for the input matrices: 10×10 and

25×25. For each size, we experimented with two distinct datasets that differed with respect to the amount of noise introduced. Notice that in the context of this problem, unlike both the no_branching problem and the noise_elimination problem, the lower the noise level, the harder the decision problem becomes. In order to assess the full capabilities of our model, we used lower values for α and β than typical false positive and false negative rates observed in real SCS datasets.

In Figure 4, panels a and b show accuracy and binary cross entropy loss[3] for training and evaluation datasets, respectively comprised of $2M$ and $2K$ matrices, each of size 10×10, where half of each set is noisy with $\alpha = 0.002$ and $\beta = 0.1$ (with the remaining half being PP). Panels c and d show results for $\alpha = 4 \times 10^{-4}$ and $\beta = 0.02$ (again accompanied by equal number of PP matrices, both in training and evaluation). We observe an accuracy of 90% in the dataset with a higher noise level. We confirm that differentiating noisy matrices from those representing PP gets harder as the noise level decreases.

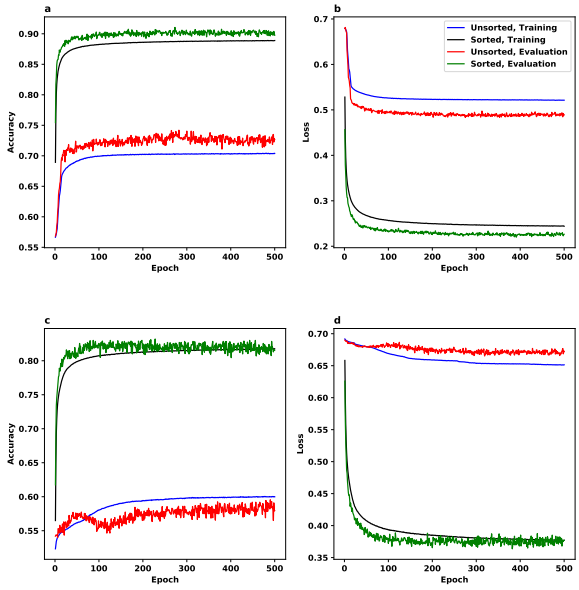


Figure 4: The accuracy and loss plots for the noise_inference problem using the neural network described in Section 4. The datasets in these experiments include matrices of size 10×10. The colors of the plots correspond to whether column-wise sorted or unsorted datasets were used - as well as whether the plot corresponds to the training phase or evaluation. We use $\alpha = 0.002$ and $\beta = 0.1$ in panels (a) and (b) and $\alpha = 4 \times 10^{-4}$ and $\beta = 0.02$ in panels (c) and (d). Note that panels (a) and (c) depict accuracy whereas panels (b) and (d) depict binary cross entropy loss.

Results for matrices of size 25×25 are given in Figure 5. We used $\alpha = 3.2 \times 10^{-4}$ and $\beta = 0.016$ in panels a and b and $\alpha = 4 \times 10^{-4}$ and $\beta = 0.02$ in panels c and d.

As mentioned in Section 2.2, sorting the (binary values represented by the) columns of the input genotype matrix as a preprocessing step is expected to improve the prediction accuracy. As summarized in Table 2 this is confirmed by the results in Figure 4 and Figure 5.

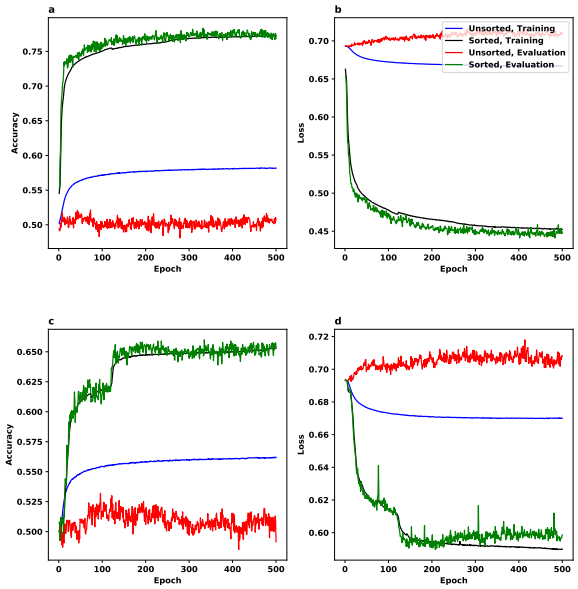


Figure 5: The accuracy and loss plots for the noise_inference problem using the architecture described in Section 4 on 25×25 matrices. The plots for column-wise sorted and unsorted genotype matrices as well as those for training and evaluation phases are depicted in distinct colors. The noise settings for this figure are $\alpha = 3.2 \times 10^{-4}$ and $\beta = 0.016$ for panels (a) and (b), and $\alpha = 6.4 \times 10^{-5}$ and $\beta = 0.0032$ for panels (c) and (d). Note that panels (a) and (c) depict accuracy, while panels (b) and (d) depict binary cross entropy loss.

Dim.	α	β	Unsorted Acc.	Sorted Acc.	Figure
10	0.002	0.1	72	90	Fig. 4 (a)
10	4×10^{-4}	0.02	60	81	Fig. 4 (b)
25	3.2×10^{-4}	0.016	50	77	Fig. 5 (a)
25	6.4×10^{-5}	0.0032	52	65	Fig. 5 (b)

Table 2: The impact of the preprocessing the input genotype matrix by sorting its columns described in Section 4, on the accuracy for the noise_inference problem. The input matrices were square matrices with sizes 10×10 or 25×25 (first column). The accuracy figures for unsorted (fourth column) and column-sorted genotype matrices (fifth column) are reported as percentages.

In order to evaluate the relative advantage of neural networks against other machine learning techniques for the noise_inference problem we explored the use of support vector machines (SVM) [3] for the same task. Figure 6 shows the training and evaluation accuracy for (column-sorted) 10×10 matrices using SVM. The x-axis in this figure represents the training sample size. Since SVM is not memory efficient, the maximum number of input instances that can be used in training is limited by our computational resources. That is why, the training procedure is stopped before the accuracy plot has converged. We use radial basis function (RBF) [3] as the kernel in all of our experiments with SVM. RBF is the most commonly used kernel in practice. In our experiments it consistently showed better performance than other kernels (e.g., polynomial).

Observe that the accuracy of the neural network model on the evaluation data is notably higher than that for SVM. E.g. the final evaluation accuracy values in Figure 4 vs. Figure 6 is 90% vs 73% in the first setting and 81% vs 64% in the second.

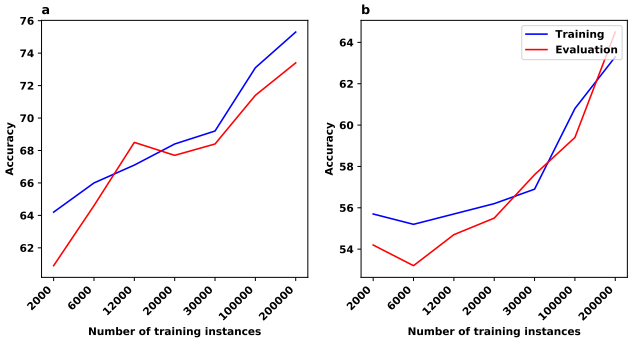


Figure 6: The training and evaluation accuracy in the noise_inference problem for SVM. All matrices for this experiment are of size 10×10 and are column-sorted. The x-axis corresponds to the training sample size. (a) Training and evaluation accuracy for $\alpha = 0.002$ and $\beta = 0.1$ (for the noisy matrices). (b) Training and evaluation accuracy for $\alpha = 4 \times 10^{-4}$ and $\beta = 0.02$ (for the noisy matrices).

6.4 Noise Elimination

We train the network shown in Figure 1 using the Reinforcement Learning framework described in Section 5 on matrices of size 10×10. We constructed an evaluation dataset of 100 matrices with the α and β values identical to that of the training dataset. We made sure that the training and evaluation datasets had no overlaps. Our trained model computed a noise-free output for 92% of the evaluation matrices.

We represent by r_1 and r_2 the number of flips made by the model on the noisy input matrix A' respectively from 1 to 0 and 0 to 1, to produce a noise-free output matrix B. Also, we represent by o_1 and o_2, the number of flips from 0 to 1 and respectively 1 to 0 we introduced to a ground truth PP matrix A to produce A'. Figure 7 shows the histogram of $r_1 - o_1$ (left side) and $r_2 - o_2$ (right side) for the 92 instances of the problem for which our model produced a noise-free matrix. Importantly, in the majority of cases, $r_1 \leq o_1$ and $r_2 \leq o_2$, indicating that the number of flips used by our approach is at most that of the number of flips introduced as noise (in some of the cases, there are solutions that result in a PP with fewer flips than that added as noise).

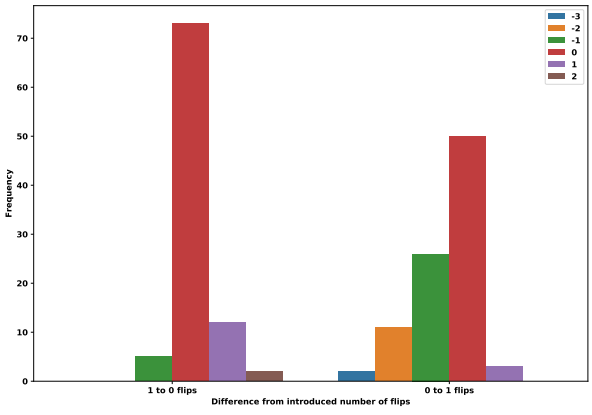


Figure 7: The difference between the number of flips made by our approach and the flips introduced as noise (to the ground truth, for obtaining the input genotype matrices) for each flip type (1 to 0 on the left, and 0 to 1 in the right) in the noise_elimination problem.

We introduce a few definitions to quantify how well our approach's distribution of flip types compare to those flips introduced as noise. Let $\mathsf{ratio}_{\mathsf{rl}} = r_2/(r_1 + r_2)$ and $\mathsf{ratio}_{\mathsf{original}} = o_2/(o_1 + o_2)$. The plot in Figure 8 represents $\mathsf{ratio}_{\mathsf{rl}}$ in the x-axis and $\mathsf{ratio}_{\mathsf{original}}$ in the y-axis. Observe that most of the evaluation cases are on the line $\mathsf{ratio}_{\mathsf{rl}} = \mathsf{ratio}_{\mathsf{original}}$.

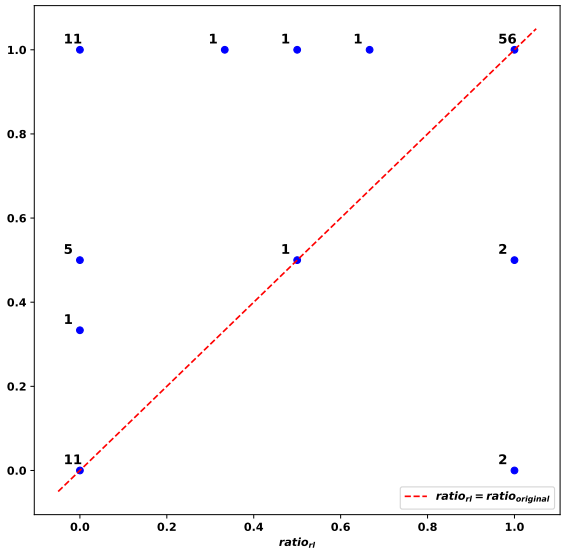


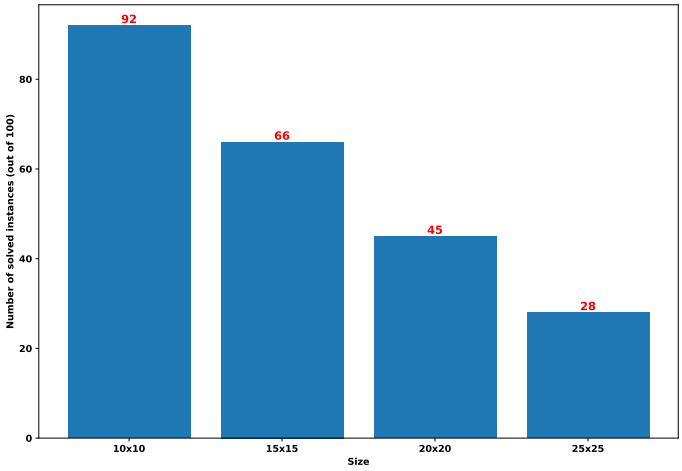
Figure 8: A quantified vizualization of how flips made by our approach correspond to those added as noise for the noise_elimination problem. The plot compares the $\mathsf{ratio}_{\mathsf{rl}}$ with $\mathsf{ratio}_{\mathsf{original}}$ for matrices of size 10×10. Each blue dot is annotated by the number of respective evaluation cases. The red dotted line represents $\mathsf{ratio}_{\mathsf{rl}} = \mathsf{ratio}_{\mathsf{original}}$. The closeness of the majority of blue dots to the red doted line implies the recovery of the ratio of introduced flip types by our approach.

Finally, we evaluated how well our trained models generalize to input matrices with varying dimensionality. In Figure 9 we used a model trained on 10×10 matrices for this purpose. Observe that this trained model can solve a notable fraction of matrices with larger dimensionality. As expected, our success rate decreases when the dimensionality increases.

6.5 Application to real data

We also applied our methods to two real datasets for which previous phylogenetic analyses reported highly concordant trees of tumor evolution. The first dataset consists of 16 single cells from a Triple Negative Breast Cancer (TNBC) patient. Originally it was made available in [37] and here we focused on the analysis of 18 mutations previously used in [24]. Evolutionary history of these mutations was first (indirectly) reported in the original study and the same results were later obtained by the use of B-SCITE [24]. As shown in [24], the reported tree has high support from both single-cell and bulk data.

Second, we analyzed an Acute Lymphoblastic Leukemia (ALL) Patient 6 dataset from [14], where 146 single cells were sequenced and 10 mutations reported. Similar to TNBC dataset, the evolutionary history of these mutations was thoroughly studied in the original study and later in [22].



.

Figure 9: The evaluation of generalizability across different values for matrix dimensions in the noise_elimination problem. The model was trained on 10×10 matrices and was evaluated on larger matrices. The number of solved instances for each matrix dimension relative to that corresponding to 10×10 matrices can be thought as a measure of generalizability.

In order to apply our approach for the no_branching problem on these datasets, we trained two separate models with the same architecture described in Section 3. The dimensions of the input was set to 16×18 for the first dataset and 146×10 for the second dataset.

In the training phase, only simulated instances (described in Section 3) were used. Note that one of drawbacks for the model based on feed-forward neural network without any recurrent layer, described in Section 3, is that input dimensions are fixed. One can use padding or copying techniques to adjust the dimensions to any number of dimensions that a pre-trained model requires, as long as the number of dimensions of the input are smaller than that required by the model. However, this may have an adverse impact in the accuracy; training with the same number of dimensions typically leads to better results.

According to our models, the tree corresponding to the first dataset has at least one additional branch, i.e., does not satisfy the no-branching property (defined in Section 2.1, with probability 0.97. This probability is only 0.21 for the second dataset, implying a non-branching topology. These results are consistent with previously reported trees for these datasets [14, 37, 24, 22]. Small but non-zero branching probability of 0.21 for the ALL patient can be explained by the evidence for recurrent mutation in gene *SUSD2* presented in [22], which may potentially be due to a local branching event (involving only two mutations at the leaf level).

Acknowledgements

This work is supported in part by the Intramural Research Program of the National Institutes of Health, National Cancer Institute. This work was also supported in part by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute [35]. E.S.A. and S.C.S. were supported in part by NSF grant AF-1619081, R.K. was supported in part by NSF grant IIS-1906694, and M.H.E. was supported in part by Indiana U. Grand Challenges Precision Health Initiative.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[3] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] D. Chen, O. Eulenstein, D. Fernández-Baca, and M. J. Sanderson. Minimum-flip supertrees: Complexity and algorithms. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 3(2):165–173, 2006.

[5] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

[6] A. G. Deshwar, S. Vembu, C. K. Yung, G. H. Jang, L. Stein, and Q. Morris. Phylowgs: reconstructing subclonal composition and evolution from whole-genome sequencing of tumors. *Genome biology*, 16(1):35, 2015.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[8] N. Donmez, S. Malikic, A. W. Wyatt, M. E. Gleave, C. Collins, and S. C. Sahinalp. Clonality inference from single tumor samples using low-coverage sequence data. *Journal of Computational Biology*, 24(6):515–523, 2017.

[9] J. Eaton, J. Wang, and R. Schwartz. Deconvolution and phylogeny inference of structural variations in tumor genomic samples. *Bioinformatics*, 34(13):i357–i365, 2018.

[10] M. El-Kebir. Sphyr: tumor phylogeny estimation from single-cell sequencing data under loss and error. *Bioinformatics*, 34(17):i671–i679, 2018.

[11] M. El-Kebir, L. Oesper, H. Acheson-Field, and B. J. Raphael. Reconstruction of clonal trees and tumor composition from multi-sample sequencing data. *Bioinformatics*, 31(12):i62–i70, 2015.

[12] M. El-Kebir, G. Satas, L. Oesper, and B. J. Raphael. Inferring the mutational history of a tumor using multi-state perfect phylogeny mixtures. *Cell systems*, 3(1):43–53, 2016.

[13] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[14] C. Gawad, W. Koh, and S. R. Quake. Dissecting the clonal origins of childhood acute lymphoblastic leukemia by single-cell genomics. *Proceedings of the National Academy of Sciences*, 111(50):17947–17952, 2014.

[15] M. Gerstung, C. Jolly, I. Leshchiner, S. C. Dentro, S. Gonzalez, D. Rosebrock, T. J. Mitchell, Y. Rubanova, P. Anur, K. Yu, M. Tarabichi, et al. The evolutionary history of 2,658 cancers. *Nature*, 578(7793):122–128, 2020.

[16] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991.

[17] G. Hellenthal and M. Stephens. mshot: modifying hudson's ms simulator to incorporate crossover and gene conversion hotspots. *Bioinformatics*, 23(4), 2007.

[18] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] E. Husić, X. Li, A. Hujdurović, M. Mehine, R. Rizzi, V. Mäkinen, M. Milanič, and A. I. Tomescu. Mipup: minimum perfect unmixed phylogenies for multi-sampled tumors via branchings and ilp. *Bioinformatics*, 35(5):769–777, 2019.

[20] K. Jahn, J. Kuipers, and N. Beerenwinkel. Tree inference for single-cell data. *Genome Biology*, 17(1):86, May 2016.

[21] Y.-H. Kim, Y. Song, J.-K. Kim, T.-M. Kim, H. W. Sim, H.-L. Kim, H. Jang, Y.-W. Kim, and K.-M. Hong. False-negative errors in next-generation sequencing contribute substantially to inconsistency of mutation databases. *PloS one*, 14(9), 2019.

[22] J. Kuipers, K. Jahn, B. J. Raphael, and N. Beerenwinkel. Single-cell sequencing data reveal widespread recurrence and loss of mutational hits in the life histories of tumors. *Genome research*, 2017.

[23] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[24] S. Malikic, K. Jahn, J. Kuipers, S. C. Sahinalp, and N. Beerenwinkel. Integrative inference of subclonal tumour evolution from single-cell and bulk sequencing data. *Nature communications*, 10(1):2750, 2019.

[25] S. Malikic, A. W. McPherson, N. Donmez, and C. S. Sahinalp. Clonality inference in multiple tumor samples using phylogeny. *Bioinformatics*, 31(9):1349–1356, 2015.

[26] S. Malikic, F. R. Mehrabadi, S. Ciccolella, M. K. Rahman, C. Ricketts, E. Haghshenas, D. Seidman, F. Hach, I. Hajirasouliha, and S. C. Sahinalp. Phiscs: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data. *Genome research*, 29(11):1860–1877, 2019.

[27] M. A. Myers, G. Satas, and B. J. Raphael. Calder: Inferring phylogenetic trees from longitudinal tumor samples. *Cell systems*, 8(6):514–522, 2019.

[28] V. Popic, R. Salari, I. Hajirasouliha, D. Kashef-Haghighi, R. B. West, and S. Batzoglou. Fast and scalable inference of multi-sample cancer lineages. *Genome biology*, 16(1):91, 2015.

[29] C. Ricketts, D. Seidman, V. Popic, F. Hormozdiari, S. Batzoglou, and I. Hajirasouliha. Meltos: multi-sample tumor phylogeny reconstruction for structural variants. *Bioinformatics*, 2019.

[30] E. M. Ross and F. Markowetz. Onconem: inferring tumor evolution from single-cell sequencing data. *Genome Biology*, 17(1):69, Apr 2016.

[31] G. Satas and B. J. Raphael. Tumor phylogeny inference using tree-constrained importance sampling. *Bioinformatics*, 33(14):i152–i160, 2017.

[32] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

[33] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, pages 1–5, 2020.

[34] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[35] C. A. Stewart, V. Welch, B. Plale, G. Fox, M. Pierce, and T. Sterling. Indiana university pervasive technology institute, 2017.

[36] F. Strino, F. Parisi, M. Micsinai, and Y. Kluger. Trap: a tree approach for fingerprinting subclonal tumor composition. *Nucleic acids research*, 41(17):e165–e165, 2013.

[37] Y. Wang, J. Waters, M. L. Leung, A. Unruh, W. Roh, X. Shi, K. Chen, P. Scheet, S. Vattathil, H. Liang, et al. Clonal evolution in breast cancer revealed by single nucleus genome sequencing. *Nature*, 512(7513):155–160, 2014.

[38] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[39] S. Zaccaria, M. El-Kebir, G. W. Klau, and B. J. Raphael. The copy-number tree mixture deconvolution problem and applications to multi-sample bulk sequencing tumor data. In *International Conference on Research in Computational Molecular Biology*, pages 318–335. Springer, 2017.

[40] H. Zafar, N. Navin, K. Chen, and L. Nakhleh. Siclonefit: Bayesian inference of population structure, genotype, and phylogeny of tumor clones from single-cell genome sequencing data. *Genome research*, 29(11):1847–1859, 2019.

[41] H. Zafar, A. Tzen, N. Navin, K. Chen, and L. Nakhleh. Sifit: inferring tumor trees from single-cell sequencing data under finite-sites models. *Genome Biology*, 18(1):178, Sep 2017.