Generative Adversarial Networks for the Simulation of DNA Sequence Evolution

Sean MacRae

School of Computer Science

McGill University, Montreal

December 2022

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Science in Computer Science

Abstract

Motivation: Sequence evolution models are at the heart of bioinformatics. They play a crucial role in many of its fundamental problems, including sequence alignment, phylogenetic inference, and ancestral genome reconstruction. While mutation rates at a sequence position are known to depend on its flanking positions, accurately incorporating these context dependencies into realistic sequence evolution models remains challenging.

Results: We propose the first generative adversarial network (GAN) approach to automatically learn, in an unsupervised manner, the parameters and weights involved in modeling context-dependent DNA sequence evolution. We exploit a long short-term memory network architecture for both the generator and critic, trained within the framework of a conditional Wasserstein GAN with gradient penalty. We show that the model captures contextual sequence information using various small context sizes. Different strategies to stabilize and accelerate training are discussed. We believe these results open the door for the exploration of more complex network architectures that leverage the state-of-the-art in both GAN and natural language processing research.

Abrégé

Motivation : Les modèles d'évolution de séquences sont au cœur de la bioinformatique. Ils jouent un rôle essentiel dans bon nombre de ses problèmes fondamentaux, notamment l'alignement de séquences, l'inférence phylogénétique et la reconstruction du génome ancestral. Alors qu'il est connu que les taux de mutation à une position de séquence dépendent de ses positions adjacentes, l'intégration exacte de ces dépendances contextuelles dans des modèles réalistes d'évolution de séquences reste difficile.

Résultats : Nous proposons la première méthode de réseau antagoniste génératif (GAN) pour apprendre automatiquement les paramètres et les poids impliqués dans la modélisation de l'évolution des séquences d'ADN, et ce, de manière contextuelle et non supervisée. Nous exploitons une architecture de réseau de longue mémoire à court terme pour le générateur ainsi que le critique, le tout optimisé dans le cadre d'un Wasserstein GAN conditionnel avec pénalité de gradient. Nous montrons, à l'aide de diverses petites tailles de contexte, que le modèle réussit à intégrer des informations contextuelles de séquence. Des différentes stratégies pour stabiliser et accélérer l'apprentissage sont examinées. Nous croyons que ces résultats ouvrent la porte à l'exploration d'architectures de réseau plus complexes à la fine pointe de la recherche sur les GANs et le traitement automatique du langage naturel.

Acknowledgements

I am deeply grateful to Prof. Mathieu Blanchette for his extraordinary mentorship, flexibility, and encouragement throughout this research project. There was no lack of theoretical and implementation challenges, but he always had the right words and ideas to help us overcome the obstacles. His experience in pushing new and tricky boundaries of science was nowhere more appreciated than in this work.

I am also thankful to my family and friends who kept me sane through this most recent pandemic. In particular, I am indebted to my parents and sister for providing constant support at home so that I could focus my efforts on school. Thank you to little Jax for his daily affection and openness to belly rubs. Finally, a special thank you to Alice for being the stubborn source of motivation that allowed me, at long last, to reach the finish line.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Evolutionary Biology

A = Adenine, **C** = Cytosine, **G** = Guanine, **T** = Thymine

bp = base pair

DNA = DeoxyriboNucleic Acid

MSA = Multiple Sequence Alignment

RNA = RiboNucleic Acid

Machine Learning

Adam = Adaptive moment estimation

ANN = Artificial Neural Network

BPTT = BackPropagation Through Time

CNN = Convolutional Neural Network

GAN = Generative Adversarial Network

KL = Kullback-Leibler

LSTM = Long Short-Term Memory

MCMC = Markov Chain Monte Carlo

ML = Machine Learning

MLP = MultiLayer Perceptron

NaN = Not a Number

RNN = Recurrent Neural Network

WGAN = Wasserstein GAN

WGAN-GP = WGAN with Gradient Penalty

Chapter 1

Introduction

Deoxyribonucleic acid (DNA) sequence evolution is the process of deriving from a given ancestral DNA sequence the corresponding descendant sequence. Simulating this process faithfully is a fundamental challenge in bioinformatics. At this time, computational biologists mainly use sequence evolution simulation tools whose parameters and weights have been hard-coded using subject matter expertise. There is one major complication: the mutation probabilities of a given nucleotide are known to depend on its flanking nucleotides, or sequence context. Unfortunately, the cost and sample sizes required to compute the mutation probabilities of a specific nucleotide given a context grow exponentially with context size. As a result, these probabilistic models quickly become intractable for useful context sizes. A context-dependent evolution simulator with automatic feature learning and parameter optimization remains desirable.

This thesis examines the application of a modern machine learning approach to the task of simulating DNA sequence evolution. The objective is to develop a framework to learn, in an unsupervised manner, the correct parameters and weights while automatically capturing longer-range sequence context. This chapter presents the key background to understand the scope of the work. More precisely, relevant concepts from evolutionary biology and machine learning are introduced in Section 1.1 and Section 1.2, respectively.

In evolutionary biology, we review the fundamental models of sequence evolution that both ignore (1.1.1) and consider (1.1.2) contextual sequence information. We then describe the progression of publicly available evolution simulation tools from early models to the current gold standard (1.1.3). Lastly, we summarize three problems in bioinformatics that strongly benefit from a robust sequence evolution model: multiple sequence alignment (1.1.4), phylogenetic inference (1.1.5), and ancestral genome reconstruction (1.1.6).

In machine learning, we conduct a high-level review of artificial neural networks (1.2.1), touching base on their common types and the concept of deep learning. We delve into recurrent neural networks (1.2.2) and in particular the long short-term memory architecture. Next, we examine generative adversarial networks (1.2.3), the modern framework used in this study, including conditional and Wasserstein variants. Finally, we give an overview of important training strategies (1.2.4) and considerations, namely in optimizer choice, hyperparameter selection, and use of curriculum learning. For each subject, applications to bioinformatics are mentioned where appropriate.

1.1 Evolutionary Biology

We begin with three brief definitions.

A nucleotide in DNA can have one of four bases: adenine (A), cytosine (C), guanine (G), or thymine (T). A and G are purines and are two-ringed structures, while C and T are pyrimidines and consist of one ring. DNA sequencing is the process of determining the order of nucleotides within a DNA molecule. One of the main goals of the Human Genome Project, which ran successfully from 1990 to 2003, was to determine the sequence of the three billion base pairs that make up human DNA. Since then, new technologies and automation capabilities have emerged to make DNA sequencing orders of magnitude faster and cheaper [1].

The neutral theory of molecular evolution states that most variation occurs at the molecular level, and that most differences between and within species evolved by selectively neutral mechanisms and not natural selection [2]. The primary method of molecular evolution is genetic drift, where the random sampling of reproducing individuals, or rather their gametes, causes allele frequencies within a population to shift [3]. The effect of genetic drift holds regardless of population size, but is more evident with small populations. The neutral theory of molecular evolution is compatible with the Darwinian theory of phenotypic evolution by natural selection. In particular, they agree that beneficial mutations undergo positive selection in adaptation to the environment and that deleterious mutations encounter negative selection for swift removal from the population, the latter thus contributing little to variation between and within species [3]. However, there is disagreement in the scientific community on the evolutionary importance of neutral mutations, which do not affect the fitness of individuals. The neutral theory of molecular evolution asserts that allele diversity, or gene polymorphism, is mainly due to neutral mutations and not natural selection acting on beneficial mutations. To test this hypothesis, two mutation cases known to be largely free from selection were investigated: mutations in non-coding regions and synonymous mutations in coding regions. Analysis of DNA sequence evolution data belonging to several species revealed a high number of mutations for both of these neutral types. This result strongly supports the neutral theory over the Darwinian theory as the dominant driver of evolutionary change [4].

Sequence alignment is the process of arranging two or more sequences such that similar regions can be identified. It is called a pairwise alignment for two sequences, and a multiple sequence alignment for three or more. When sequences are not of the same length, gap characters must be inserted into the shorter sequences until all sequence lengths match, so as to account for insertion and deletion events in the affected sequences. A scoring scheme for a candidate alignment of nucleotides will consider the number of matches versus mismatches and gap regions created versus extended. Algorithms for producing optimal sequence alignments have been applied to DNA, ribonucleic acid (RNA), and amino acid sequences, as well as natural language strings. The dynamic pro-

gramming based Needleman-Wunsch algorithm is guaranteed to find all global pairwise alignments with the optimal score in polynomial time and space [5]. A variant called the Smith-Waterman algorithm will accomplish the same for local pairwise alignments [6]. Improved algorithms also exist for both cases of pairwise alignments [7]. In contrast, the multiple sequence alignment problem is known to be NP-complete for nearly all of its practical formulations [8]. More discussion on it will follow in Section 1.1.4.

In this work, we assume the neutral theory of molecular evolution and restrict our attention to DNA.

1.1.1 Context-Independent Models of Sequence Evolution

Most works in the field of sequence evolution modeling assume the neutral theory of molecular evolution. Thus, they do not model the effect of natural selection when computing the rates of three different mutation types: base substitutions, insertions, and deletions. Biologically, these mutations may arise from DNA replication and repair errors, spontaneously, or from exposure to mutagens [9]. Substitutions can be either transitions, where a purine replaces another purine or a pyrimidine replaces another pyrimidine ($A \leftrightarrow G,\ C \leftrightarrow T$), or transversions, where a purine exchanges with a pyrimidine and vice-versa ($A \leftrightarrow C,\ A \leftrightarrow T,\ G \leftrightarrow C,\ G \leftrightarrow T$). Given the four DNA nucleotide bases consisting of two purines and two pyrimidines, there are four possible transitions and eight possible transversions. However, exchanging a base for a similarly structured base, which is the case in all transitions, is more likely than the dissimilar exchanges between single-ringed and double-ringed structures in transversions. Transitions are therefore more frequent than transversions, despite there being half as many possibilities. Furthermore, transitions at the third position of codons are more likely than transversions to encode the same amino acid, so they cause more synonymous mutations that tend to remain in the genome [10]. Indeed, there are many stochastic processes to capture just on the level of base substitutions.

Early models of DNA sequence evolution use a simple alphabet $\Sigma = \{A, C, G, T\}$, where each character corresponds to its related base, to tackle this challenge [11, 12]. In addition, nearly all models assume the Markov, or memoryless, property: the future is independent of the past given the present. In other words, each nucleotide in a DNA sequence evolves independently of its history. Context-independent models make three further simplifying assumptions about DNA sequences: there are no interactions between sites in a sequence, substitution rates are the same at different sites in a sequence, and substitution rates do not vary with time. We now review five representative models, though several more exist in the literature.

The Jukes-Cantor (1969) model is the simplest context-independent model of sequence evolution [11]. It assumes all base equilibrium frequencies are equal ($\pi_A = \pi_C = \pi_G = \pi_T = 0.25$) and that all base substitutions occur at some equal probability μ. Consequently, the probability that a given base mutates is 3μ and that it remains unchanged is $1 - 3\mu$. We summarize the overall substitution probabilities below, using the alphabetical base ordering for the rows and columns both here and throughout this work. Rows denote the current base and columns denote the next base.

$$\begin{bmatrix} 1-3\mu & \mu & \mu & \mu \\ \mu & 1-3\mu & \mu & \mu \\ \mu & \mu & 1-3\mu & \mu \\ \mu & \mu & \mu & 1-3\mu \end{bmatrix}$$

The Kimura Two-Parameter (1980) model relaxes the Jukes-Cantor model's assumption that substitution probabilities are all equal [12]. It targets transitions and transversions by providing a different base substitution probability, α and β respectively, for each. However, base equilibrium frequencies are unchanged with $\pi_A = \pi_C = \pi_G = \pi_T = 0.25$.

The overall substitution probabilities are summarized below.

$$\begin{bmatrix} 1-\alpha-2\beta & \beta & \alpha & \beta \\ \beta & 1-\alpha-2\beta & \beta & \alpha \\ \alpha & \beta & 1-\alpha-2\beta & \beta \\ \beta & \alpha & \beta & 1-\alpha-2\beta \end{bmatrix}$$

On the other hand, the Felsenstein (1981) model eases the Jukes-Cantor model's assumption that base equilibrium frequencies are equal [13]. Now, $\pi_A \neq \pi_C \neq \pi_G \neq \pi_T$, but all base substitution probabilities remain equal to μ. The overall substitution probabilities are summarized below. For readability, the elements on the diagonal are the following: $a = 1 - \mu(\pi_C + \pi_G + \pi_T),\ c = 1 - \mu(\pi_A + \pi_G + \pi_T),\ g = 1 - \mu(\pi_A + \pi_C + \pi_T),\ t = 1 - \mu(\pi_A + \pi_C + \pi_G)$.

$$\begin{bmatrix} a & \mu\pi_C & \mu\pi_G & \mu\pi_T \\ \mu\pi_A & c & \mu\pi_G & \mu\pi_T \\ \mu\pi_A & \mu\pi_C & g & \mu\pi_T \\ \mu\pi_A & \mu\pi_C & \mu\pi_G & t \end{bmatrix}$$

The Hasegawa-Kishino-Yano (1985) model combines the advancements made in the two previous models: $\pi_A \neq \pi_C \neq \pi_G \neq \pi_T$ and we have different probabilities, α and β, for transitions and transversions, respectively [14]. The overall substitution probabilities are summarized below. For readability, the elements on the diagonal are the following: $a = 1 - \alpha\pi_G - \beta(\pi_C + \pi_T),\ c = 1 - \alpha\pi_T - \beta(\pi_A + \pi_G),\ g = 1 - \alpha\pi_A - \beta(\pi_C + \pi_T),\ t = 1 - \alpha\pi_C - \beta(\pi_A + \pi_G)$.

$$\begin{bmatrix} a & \beta\pi_C & \alpha\pi_G & \beta\pi_T \\ \beta\pi_A & c & \beta\pi_G & \alpha\pi_T \\ \alpha\pi_A & \beta\pi_C & g & \beta\pi_T \\ \beta\pi_A & \alpha\pi_C & \beta\pi_G & t \end{bmatrix}$$

Lastly, the General Time-Reversible (1986) model is the most general model possible to preserve time reversibility, or symmetry with respect to a change in the sign of time, while

remaining context-independent [15]. It permits different probabilities for each type of base substitution, as well as unequal base equilibrium frequencies ($\pi_A \neq \pi_C \neq \pi_G \neq \pi_T$). Since the model is time-reversible, a $C \rightarrow T$ substitution for instance has the same probability as a $T \rightarrow C$ substitution. The overall substitution probabilities are summarized below. For readability, we use the following elements denoting specific substitution probabilities: $\alpha = A \leftrightarrow C,\ \beta = A \leftrightarrow G,\ \gamma = A \leftrightarrow T,\ \delta = C \leftrightarrow G,\ \epsilon = C \leftrightarrow T,\ \eta = G \leftrightarrow T$. We also use the following elements on the diagonal: $a = 1 - \alpha\pi_C - \beta\pi_G - \gamma\pi_T,\ c = 1 - \alpha\pi_A - \delta\pi_G - \epsilon\pi_T,\ g = 1 - \beta\pi_A - \delta\pi_C - \eta\pi_T,\ t = 1 - \gamma\pi_A - \epsilon\pi_C - \eta\pi_G$.

$$\begin{bmatrix} a & \alpha\pi_C & \beta\pi_G & \gamma\pi_T \\ \alpha\pi_A & c & \delta\pi_G & \epsilon\pi_T \\ \beta\pi_A & \delta\pi_C & g & \eta\pi_T \\ \gamma\pi_A & \epsilon\pi_C & \eta\pi_G & t \end{bmatrix}$$

In sequence alignment, base substitutions are easily handled since they do not affect sequence length. However, insertion and deletion events in a genome's evolutionary history cause gaps to appear when aligning its sequences. Suppose, in fact, we have two sequences A and B belonging to two evolutionary distant individuals. Unless the phylogenetic direction of the sequences is known, an insertion event in sequence $A \rightarrow B$ could be just as plausibly interpreted as a deletion event in sequence $B \rightarrow A$. The term indel is therefore used to contain both possibilities, as well as being a convenient abbreviation. Models of sequence evolution that wish to incorporate indels extend the alphabet to consider gaps with $\Sigma = \{A, C, G, T, -\}$ [16–18].

Context-independent models for DNA sequences containing gaps hold the same general assumptions as the models for sequences without gaps. In addition, they assume that gaps at adjacent positions occur independently. For gaps only one position long, this presumption has no effect. The Thorne-Kishino-Felsenstein (1991) model presents a statistical approach to estimating the rates for base substitutions and indels of size one [16].

Counts of each character, whether base or gap, observed in the input DNA sequences are used to form a likelihood function, which is then maximized.

It is in fact not too difficult to also extend the previously described models of sequence evolution to consider gaps. We show a common way to do it for the Jukes-Cantor and Kimura Two-Parameter models. Recall that character equilibrium frequencies are equal in these two models: $\pi_A = \pi_C = \pi_G = \pi_T = \pi_- = 0.2$. For the Jukes-Cantor model, base substitutions and indels occur at some equal probability μ [18]. The α and β probability parameters for transitions and transversions, respectively, in the Kimura Two-Parameter model are retained with the addition of a third probability parameter γ for indel events. To be precise, deletions occur at a probability γ, while insertions are assumed to take place at an equal probability $\frac{\gamma}{4}$ for each base [19]. The overall mutation probabilities are summarized below for each model. The last row and column correspond to the gap character.

$$\begin{bmatrix} 1-4\mu & \mu & \mu & \mu & \mu \\ \mu & 1-4\mu & \mu & \mu & \mu \\ \mu & \mu & 1-4\mu & \mu & \mu \\ \mu & \mu & \mu & 1-4\mu & \mu \\ \mu & \mu & \mu & \mu & 1-4\mu \end{bmatrix}$$

$$\begin{bmatrix} 1-\alpha-2\beta-\gamma & \beta & \alpha & \beta & \gamma \\ \beta & 1-\alpha-2\beta-\gamma & \beta & \alpha & \gamma \\ \alpha & \beta & 1-\alpha-2\beta-\gamma & \beta & \gamma \\ \beta & \alpha & \beta & 1-\alpha-2\beta-\gamma & \gamma \\ \frac{\gamma}{4} & \frac{\gamma}{4} & \frac{\gamma}{4} & \frac{\gamma}{4} & 1-\gamma \end{bmatrix}$$

Because of difficulties arising in the contiguous gap case, some models contest the assumption that adjacent gaps be treated independently. The advantages and disadvantages of continuing to code gaps as the fifth symbol in an alphabet versus considering gaps entirely separately from bases, via an extra presence or absence character, are debated. In particular, these models advocate for treating contiguous gaps as stemming

from a single indel event, arguing that it is the most parsimonious option, avoids over-weighting gaps, and accounts for the fact that such an event often involves more than one consecutive base [17]. It remains unclear if the added complexities of this approach can be justified in practice.

For all of the sequence evolution models described above, it is possible given only the substitution rates to derive the associated substitution probabilities. If Q is the instantaneous substitution rate matrix from row character to column character, then the substitution probability matrix as a function of time t is $P(t) = e^{Qt}$, where e is matrix exponentiation and t is expressed in units of expected number of substitutions per position. For instance, $p_{CT}(t)$ is the probability of observing, at a given position, base C at time 0 and, at the same position, base T at time t. $P(t)$ can be computed explicitly by solving, for each of its elements, the differential equation describing the change in that element of $P(t)$ over continuous time t. We omit here the equations of this form for each individual model, though they may be found in their respective literature [11–15].

1.1.2 Context-Dependent Models of Sequence Evolution

In models of sequence evolution examined so far, attempts are made to address the different types of base substitutions. The Kimura Two-Parameter model, for example, targets the difference in substitution probability between transitions and transversions. In a broader case, the General Time-Reversible model proposes a separate substitution probability that is symmetric in the reverse direction for each base substitution. Nonetheless, these models fail to incorporate sequence context. The literature is rich to describe at least three classes of biological processes that depend on contextual information: spontaneous mutations, DNA repair mechanisms, and indel events [20].

Concerning spontaneous mutations, a common example is the unusually high substitution rate of methylated CpG sites. These CpG sites are regions in a DNA sequence where cytosine is immediately followed by guanine. The 'p' denotes the phosphate link

between the two nucleotides on the same DNA strand. It avoids confusion with the alternative case of cytosine on one strand forming a Watson-Crick base pair with guanine on the other via hydrogen bonds [21]. Spontaneous deamination of methylated cytosine results in thymine, which explains the elevated methylated $C \rightarrow T$ substitutions found at CpG sites. Other small nucleotide contexts with lower yet still unexpectedly high substitution rates exist, but unfortunately do not seem to have a mechanism of action as clear as the one for CpG sites [22].

Damaged DNA is precisely removed and replaced through specialized repair mechanisms. One such process is called nucleotide excision repair, and is known to occur faster in genes being actively transcribed, or copied to RNA, through a specific pathway called transcription-coupled repair. It has been shown that transcription-coupled repair localizes preferentially to chromosomes dense in genes and to chromosome domains rich in guanine-cytosine base pairs [23]. Furthermore, the nature of the gene being transcribed greatly affects the speed of transcription-coupled repair in the Chinese hamster genome [24]. In both cases, the efficiency of DNA repair depends on sequence context, and it is this dependence that impacts the persistence of mutations.

Insertion and deletion events are also known to occur more frequently in certain DNA regions. Sequences containing tandem repeats, where the same DNA patterns are repeated directly adjacent to each other, are particularly susceptible. During DNA replication, the DNA polymerase enzyme may temporarily slip off tandem repeat regions of the template strand and then reattach. Possible reattachment somewhere shortly upstream of the enzyme's original position produces repeat insertions in the daughter strand. Nucleotide excision repair then occurs at the site of replication slippage. The end result is an insertion or deletion event on the template strand, the daughter strand, or both [25]. In primates and flies, indels can occur in certain sequence contexts at rates over 100 times greater than in sequence contexts with the lowest indel tendency [26]. High indel frequencies then impart a bias in the appearance of particular sequence patterns.

Clearly, there is a need to capture mutational context dependencies. Successful models should consider the interactions between different sites in a sequence, incorporate the different substitution rates at various sites, and ideally address the variation in substitution rates over time. To tackle this challenge, several types of context-dependent models of sequence evolution have been proposed. We now review an extensive selection of those present in the literature.

Most context-dependent models restrict their attention to only base substitutions for simplicity. The Jensen-Pedersen (2000) model uses a Markov chain Monte Carlo (MCMC) method to infer the substitution rates in coding DNA regions [27]. It relaxes the independence assumption by considering the two positions flanking any nucleotide of interest, for a context size of three. As previously stated, Markov processes permit the future to be calculated independent of the past given the present. A Markov chain, therefore, is a sequence or chain of states where the probability of changing to any next state only depends on the current state. By itself, a Monte Carlo method simply draws independent samples from a given probability distribution. Together, Markov chain Monte Carlo breaks this independence by making the next sample to be drawn depend on the previous sample drawn, according to the Markov chain. The construction of the Markov chain is done carefully so as to obtain the desired base equilibrium frequencies. From there, an MCMC method is developed to estimate the base substitution probabilities from one DNA sequence to another.

The Arndt-Burge-Hwa (2003) model presents an analytic solution to the case of dinucleotide contexts in non-coding DNA regions [28]. It exploits a maximum likelihood approach to infer the substitution rates given the observed single- and dinucleotide substitution frequencies. In an extension, the Siepel-Haussler (2004) model introduces an expectation maximization algorithm to estimate the most likely substitution rates in both coding and non-coding DNA regions [29]. It considers trinucleotide contexts and integrates three distinct improvements. First, an explicit definition of the model is given for arbitrary context size N, where any sequence of N nucleotides is termed an N-tuple, though in practice $N \leq 3$ is demonstrated. Second, the model allows for overlapping N-

tuples, which results in the capture of context dependencies between all adjacent pairs of sites. Third, a parameter-rich substitution process is used to enhance estimation accuracy.

The Aggarwala-Voight (2016) model proposes a statistical approach that considers the four or six flanking positions of a given nucleotide for a resulting context size of five or seven, respectively [30]. Counts of each observed nucleotide context are used to form a likelihood function, which is then maximized to find the substitution rates. The Zhu-et-al. (2017) model uses a log-linear method to examine substitution frequencies in context sizes of five [31]. It provides the flexibility to select any desired linear combination of parameters for a specific model configuration, which in turn permits the identification of the most important, or highly contributing, parameters. The Ling-et-al. (2020) model presents a sparse Bayesian approach operating under the assumption that, of all the possible sequence contexts of size five, only a few contribute significantly to substitution rates [32]. Subsequently, these rates are inferred using an MCMC method. Moderating the strength of Bayesian shrinkage, or the level of sparsity, is shown to be important for optimizing accuracy scores.

With a context size of three, five, or seven, inferring the 64 (4^3), 1,024 (4^5), or 16,384 (4^7) different conditional probabilities that underlie these models is feasible using current data availability. In fact, some argue that consideration of the immediate flanking bases in this manner is sufficient for the majority of human DNA sequences. Recent work, however, shows that longer-range local context still contributes meaningfully to mutation variability [33]. Unfortunately, the number of probabilities grows exponentially with context size, as does the number of samples required to observe the increasingly precise and thus rare substitutions. The task is ultimately made intractable from both a computational cost and sample size requirement perspective. Consequently, the latest challenge is to avoid these probabilistic computations and circumvent the exponential blow-up to enable capture of longer-range context dependencies.

To date, the Lim-Blanchette (2020) model has made demonstrated progress towards this problem [20]. It leverages recent advances in machine learning and natural language

processing, in particular sequence-to-sequence modeling, to bypass explicit computation of any conditional probabilities. Furthermore, insertions of size two and deletions of arbitrary size are supported. More discussion on this model will follow in Section 1.2.2.

1.1.3 Evolution Simulation Tools

Evolution simulation tools capable of generating fake but highly realistic DNA sequences have several fundamental uses: evaluating multiple sequence alignment methods, predicting relationships for the reconstruction of phylogenetic trees, and validating various substitution models. In a more applied context, the ability to evolve any given sequence faithfully has important clinical considerations: for instance, in keeping ahead of quickly mutating pathogens and cancers [34]. It can also serve an educational role for the non-scientific community given a sufficiently user-friendly design [35]. We now review three representative tools, though several more exist in the literature.

Rose (1998), or random model of sequence evolution, is a tool designed to generate evolutionarily related sequences, or sequence families [36]. Based on the previously described Hasegawa-Kishino-Yano (1985) substitution model, it is one of the first tools to incorporate indels as well as the usual base substitutions. Rose will evolve a starting ancestral sequence according to a guiding evolutionary tree. Consequently, the tree topology provided directly affects the sequences produced. The substitutions rates are freely set by the user.

INDELible (2009) is a tool that proposes a more realistic model for indel formation [37]. Insertions and deletions are treated as two distinct processes whose sizes are also modeled according to two separate distributions. INDELible supports a wide variety of substitution models beyond those mentioned so far, including some where substitution rates and base equilibrium frequencies vary with time.

The current gold standard is Evolver, a powerful tool for whole genome sequence simulation [38]. In contrast to its predecessors, it permits mutation events to occur over

all theoretical length scales, from single base substitutions to entire chromosomal fission and fusion. Annotations of the ancestral sequence, which provide the location of all the genes and their functions, can be simulated in the evolved descendant sequences as well.

Despite all this work, there is room for improvement. One disadvantage to these existing tools is that they require fixed parameter input from the user. Human experts exploit many heuristics to settle on what appears best qualitatively, but this practice is clearly a suboptimality. A second drawback is that these tools assume context-independence. We have shown the need to consider mutational context dependencies, without which evolution cannot be accurately modeled.

1.1.4 Application to Multiple Sequence Alignment

The multiple sequence alignment (MSA) problem involves finding the optimal alignment of three or more DNA sequences. It is NP-complete for arbitrary sequence length and number, so no polynomial-time algorithm is known to exist for solving it [8]. In fact, if one were found, then the famous $\mathbf{P}$ versus $\mathbf{NP}$ problem would have the unexpected answer that $\mathbf{P} = \mathbf{NP}$, which would deeply impact society at large. Since an exact algorithm is currently intractable, many heuristic methods sacrificing optimality for computational speed have been developed. A discussion of each of these is out of scope here. See Figure 1.1 for a simple example of an MSA.

```
Seq1: ACCAAGCCAATC
Seq2: A-CCAACCATTC
Seq3: AGCCAGCCAATC
Seq4: AGGCCGC-AATC
Seq5: AGGCCAC-AATC
```

Figure 1.1: A toy example of a multiple sequence alignment.

The most important element of an MSA is the scoring scheme, or function, which dictates the quality of the alignment of a set of sequences. It must consider both substitutions and gaps, usually as explicit parameters. However, even the MSA maximizing the scoring scheme often fails to produce a biologically relevant result [39]. Indeed, this outcome is observed in MSAs generated by both exact and heuristic algorithms. It is due to the use of context-independent models of sequence evolution to inform the scoring scheme, yielding inaccurate substitution and indel rates. Unfortunately, the obvious solution of incorporating context-dependent models is not straightforward because calculating maximum likelihood alignments under them is a computational challenge, especially with indels [20].

The Hickey-Blanchette (2011) model tries to address this problem by introducing a pair Tree-Adjoining Grammar to recognize tandem repeats. Consequently, it is able to capture the context effect of short indels on a given nucleotide's substitution rate [40]. A grammar is a formal description of how to formulate valid strings from a language's alphabet. Production rules are provided for how to rewrite a symbol as a string of other symbols. A Tree-Adjoining Grammar replaces symbols with trees, such that trees are generated instead of strings and nodes of trees can be rewritten as other trees. However, the model is limited to aligning only pairs of relatively small DNA regions.

The challenge of obtaining a biologically-optimized scoring scheme has led to the development of compromise solutions. In particular, it has given rise to alignment methods that do not depend on the use of any specific scoring scheme. For instance, the Ramakrishnan-Singh-Blanchette (2018) model uses a reinforcement learning approach to learn the characteristics of a good alignment and treats the scoring scheme as a black box [41]. It supports sequences of moderate length and proposes various heuristics to handle longer ones.

In all situations, an accurate model of sequence evolution permits the alignment of increasingly diverged sequences. The result is a faithful alignment process that is better able to identify homologous, or evolutionarily similar, DNA sequences [20].

1.1.5 Application to Phylogenetic Inference

The phylogenetic inference problem involves reconstructing the evolutionary history of a given set of species, usually in the form of a phylogenetic tree. Multiple sequence alignment is the fundamental process for determining the level of homology between these sequences. By optimally arranging the DNA sequences, it becomes easy to identify identical and similar regions. When sequence similarity is very high at many sites, evidence is strong that the sequences are related by a recent common ancestor. Genome sequences that are conserved, or relatively unchanged, across the set of species tend to be functionally important and often constitute regions of interest for the MSA.

The phylogenetic tree with the best fit to the MSA is frequently determined using maximum parsimony, where the simplest explanation is preferred. In other words, the tree with the fewest evolutionary events to result in the MSA is chosen. Finding the most parsimonious tree for an arbitrary number of DNA sequences is in fact NP-hard [42]. Another common method is to use maximum likelihood to determine the phylogenetic tree with the best fit to the MSA. The advantage of this approach is the ability to specify a desired sequence evolution model under which to compute the probability of observing a particular tree. Unfortunately, finding the most likely phylogenetic tree is also NP-hard [43]. These computational challenges have prompted the development of heuristic methods.

Once inferred, the phylogenetic tree can be rooted or, more frequently, unrooted. A rooted tree is directed such that its root is the most recent common ancestor of all the sequences located at the leaf nodes. In contrast, an unrooted tree does not assume a directed relationship. In both cases, the distance between leaf nodes is informative of genetic distance. See Figure 1.2 for an example of both types of phylogenetic trees.

The closeness of the constructed phylogenetic tree to the true evolutionary history depends on the accuracy of multiple sequence alignment, which we have in turn established to rely on a robust model of sequence evolution.

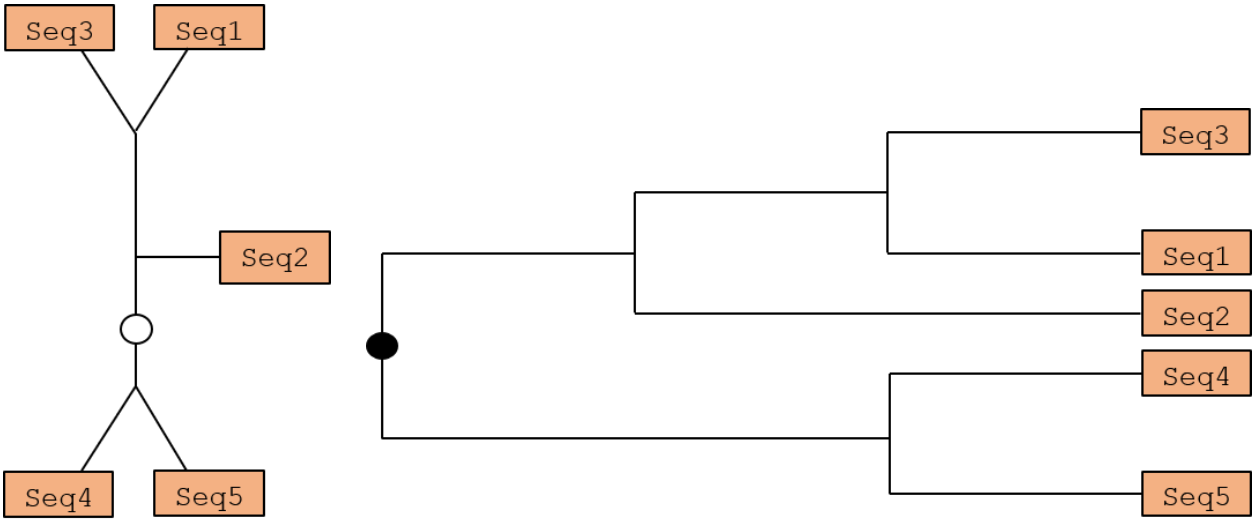

Figure 1.2: Left: an unrooted phylogenetic tree for the MSA given in Figure 1.1. Right: a rooted phylogenetic tree for the same sequences using midpoint rooting. The black node denotes the root. The white node shows the corresponding root placement to obtain the rooted tree from the unrooted one. Branch lengths are not drawn to scale.

1.1.6 Application to Ancestral Genome Reconstruction

The ancestral genome reconstruction problem involves predicting the DNA sequence of all ancestral species in a phylogenetic tree. Provided as input are a set of evolutionarily related extant DNA sequences and the phylogenetic tree describing their relationships [44]. The suitability of the tree topology clearly has a direct effect on the faithfulness of the reconstructed ancestral genomes. See Figure 1.3 for a visual example of this importance.

A maximum likelihood approach is commonly used to generate ancestral sequences. The most likely character, whether base or gap, at each position is given by the position weight matrix calculated from the input DNA sequences.

Again, robust models of sequence evolution are important for securing an accurate multiple sequence alignment and phylogenetic tree.

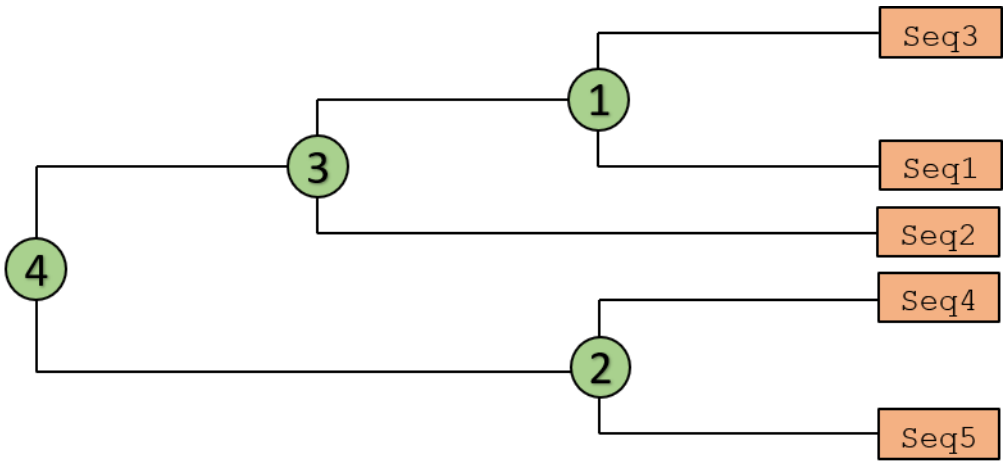

Figure 1.3: Placement visualization of all ancestral sequences for the rooted phylogenetic tree given in Figure 1.2 and based on the MSA given in Figure 1.1. The green nodes denote the ancestral sequences for the extant sequences in orange. The reconstruction accuracy of more distant ancestors depends on that of more recent ancestors.

1.2 Machine Learning

Machine learning (ML) is the use and development of artificial intelligence systems that learn from experience by detecting patterns in data. The textbook *Deep Learning*, published in 2016 by Ian Goodfellow, Yoshua Bengio, and Aaron Courville, is the primary authority for the theory presented in this section [45].

The rise in popularity of ML approaches over the last decade has led to their application in many fields, including computer vision, natural language processing, and bioinformatics. While novel ML algorithms are often developed for image collections or text corpora, working with biological sequence data is particularly challenging due to its hybrid structure. Aligned DNA sequences, with their discrete series of characters over the alphabet $\Sigma = \{A, C, G, T, -\}$ and dependence on contextual clues, bear a striking resemblance to natural language strings. At the same time, context patterns interspersed regularly throughout noisy backgrounds make DNA sequences characteristically similar to images. This mixed structure can lead to unexpected results when applying ML algorithms tested only separately on either images or text.

In this work, we assume a basic understanding of the mathematical and statistical underpinnings of ML. We also assume knowledge of multilayer perceptrons and, to a lesser degree, the other classical ML algorithms. Essential background is reviewed at a high level for context before advancing to specialized theory.

1.2.1 Artificial Neural Networks

First developed in the mid-twentieth century, artificial neural networks (ANNs) are loosely inspired by biological neural networks [46]. The nodes or units in an ANN are termed neurons. At minimum, they can be organized into input and output layers, but there are often one or more intermediate layers. Since the latter are not directly observable, they are commonly called hidden layers. Each layer in an ANN, considered as a whole, receives an input vector and produces an output vector. Each individual neuron in a layer, after computing its nonlinear activation function over the weighted sum of its input values and an optional bias term, sends its scalar output to one or more neurons in the next layer. These synapse-like connections form the edges of the ANN, and their relative strength can be increased or decreased by applying various weights. Connection weights between every two layers are organized into a distinct matrix [45].

Since an ANN is a function, feeding it an input example will produce an observable output. Training an ANN involves fine-tuning the values in each weight matrix to minimize the network error, or the difference between the network's final output and the target output, over many input examples. Specific user-controlled parameters, or hyperparameters, such as the learning rate and the number of neurons recruited for each layer, are important for smooth training progression [45]. More discussion on hyperparameters will follow in Section 1.2.4.

The analogy between neurons in an ANN and those in the brain does not extend much further. For instance, the loss functions used to measure the severity of network error, like cross-entropy and mean squared error, do not seem to have a biological basis. The biggest

culprit is backpropagation, the prevalent method by which network weights are adjusted. This algorithm will efficiently compute, using the chain rule, the gradient of the specified loss function with respect to the weights. It is unknown if these same mathematical operations occur during human learning, though it is strongly suspected not to be the case [45]. More discussion on using gradients for training optimization will follow in Section 1.2.4.

While the width of an ANN is defined as the maximum number of neurons in one of its network layers, the depth is conventionally measured by the total number of network layers minus the input layer. The term deep learning refers to the training of ANNs with depth at least two, meaning there is at least one hidden layer. In practice, deep networks generally have two or more hidden layers. The universal approximation theorem states that an ANN with one hidden layer of unbounded width, using a nonlinear activation function, can approximate any continuous function. Examples of such activation functions include the sigmoid (σ), hyperbolic tangent ($\tanh$), and rectified linear unit (ReLU). However, the theorem gives no clue as to what precise network and training configuration will yield success in each situation. In fact, training an ANN to approximate a given function may be impossible even though the representation exists in theory [45].

Nonetheless, the excitement around ANNs stems from their demonstrated ability to approximate unknown functions. Their apparent failure to guarantee training success, however, makes these results puzzling. Indeed, the extensive training time and hyperparameter tuning required to train an ANN were notable obstacles for several decades, even leading into the twenty-first century. Though backpropagation in its efficient form was already known, attempts to train wider or deeper networks were stalled out due to the exponential increase in gradient operations. The key enabling ANN training to proceed was the corresponding rise in computational power over the last two decades [45].

The dual interpretation of the universal approximation theorem, which involves an ANN of unbounded depth but of bounded width, has since become favoured. Though offering no more clues for construction than the primal version, it gives a theoretical basis for experimentation with numerous hidden layers of fixed width. Along with greater

practicality, these deep learning approaches have also shown encouraging results in recent years, leading to their high popularity. Other network architectures, first developed decades ago, have simultaneously resurfaced to exploit the new gains in computational power [45]. We now mention two main types of ANNs, though several more exist in the literature.

The first type of ANN is the feedforward neural network. Its name is derived from its directed acylic network structure, where all connections point forwards to the next layer. The simplest kind of feedforward neural network is the single-layer perceptron, composed of only an input and an output layer. In practice, the successes found by adding one or more hidden layers have caused multilayer perceptrons (MLPs) to be much more commonly used. Nearly always, the layers of an MLP are fully connected such that every neuron in a layer receives input from every neuron in the previous layer and sends output to every neuron in the next layer, forming a dense network. Because of their straightforward and widespread use in machine learning, MLPs are often referred to as vanilla neural networks. It is perhaps confusing that many ML enthusiasts will also use the term MLP to denote just a dense hidden layer component of a more complex ANN [45].

A more specialized variant of feedforward neural network that has gained popularity in the last decade is the convolutional neural network (CNN). Its distinguishing characteristic is the use of one or more convolutional hidden layers, either alone or together with dense hidden layers. In the former case, it is termed a fully convolutional neural network. The convolution operation at the heart of CNNs causes input features to become abstracted according to the kernel, or convolution matrix. By stacking convolutional layers each with different kernels, it is possible to efficiently learn the patterns hidden in many two- or three-dimensional data types. Another advantage is in the sharing of weights among convolutional neurons, which significantly reduces the number of parameters involved in training convolutional layers compared to dense layers [45]. Though we do not use CNNs in this work, we recognize their existence and contrast them with the next architecture.

The second type of ANN is the recurrent neural network (RNN). Its defining feature is its theoretical ability to learn temporal functions, with connections pointing both forwards to the next layer and, in the one or more recurrent hidden layers, backwards on itself to the next time step. The term fully recurrent neural network describes an RNN where all layers are fully connected, including any optional dense hidden layers [45]. More discussion on RNNs will follow in the next subsection.

Early applications of ANNs to bioinformatics mainly dealt with problems in disease classification and biological marker identification for gene or protein expression. Data would commonly be sourced from protein mass spectrometry and DNA microarray experiments, forming complex high-dimensional datasets [47]. With the rise in computational power, ANNs are being tested against a variety of new problems inspired by biology, including robust modeling of sequence evolution. More detail on these applications will follow in the next two subsections.

1.2.2 Recurrent Neural Networks

In traditional feedforward neural networks, it is commonly assumed that the collection of input examples is independent and identically distributed. However, the DNA sequences used in this work clearly introduce context dependencies both within and among sequences, breaking this assumption. Recurrent neural networks permit the processing of this type of dynamic temporal data. More precisely, each time step corresponds to a character in the input DNA sequence. The recurrent hidden layer of an RNN is composed of one neuron for each time step. Each such neuron sends its hidden state, or output, to the next layer and to the next neuron in the temporal sequence. Weight sharing across time reduces the total number of trainable parameters, simplifying training [45].

See Figure 1.4 for a diagram of a minimal RNN in folded and unfolded form. For convenience, a recurrent layer is usually represented succinctly by a symbolic self-connection to denote the hidden state propagation through time. Training an RNN uses a variant

method called backpropagation through time (BPTT), which requires unfolding the RNN in time. In this process, connections between each time step neuron are laid out explicitly in a directed manner similar to deep feedforward neural networks. The interesting RNN operations are: $h_t = \sigma_h(U \cdot x_t + W \cdot h_{t-1} + b_h)$ and $o_t = \sigma_o(V \cdot h_t + b_o)$. Here, h_t is the hidden state of the neuron at time t, x_t is the input at time t, and o_t is the output at time t. $U, W,$ and V denote respectively the shared weight matrices for the connections between input and hidden layer neurons, between hidden layer neurons, as well as between hidden and output layer neurons. Finally, σ_h and σ_o are any nonlinear activation function, while b_h and b_o are bias terms, for the hidden and output layers, respectively [45]. Note that $\cdot$ denotes matrix multiplication.

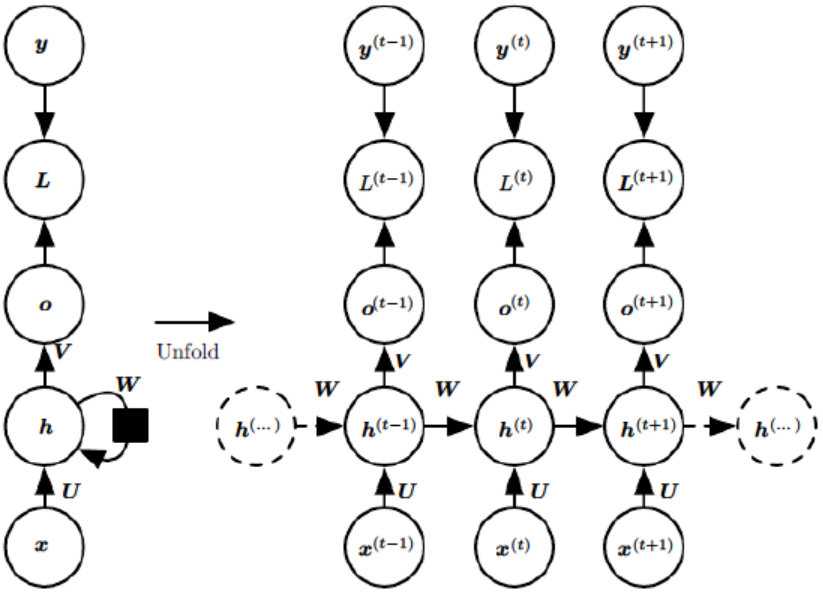

Figure 1.4: Computational graph of an RNN in folded and unfolded form. Time steps are in superscript. The hidden state h is propagated to both the output layer and across time within the hidden layer; x is the input layer; o is the output layer; L is the loss function; y is the true label; U, W, V are the shared weight matrices. Reproduced from [45].

In theory, RNNs should work well to process data with a large temporal dimension. Their implementation, unfortunately, gives rise to two related but opposing computational issues: the vanishing and exploding gradient problems. During BPTT, the gradient is computed with respect to the weights of each time step neuron in the recurrent layer.

The unfolded RNN network topology causes chain rule multiplications to be repeated through each time step. Despite a judicious choice of activation function whose range is not restricted to between 0 and 1, BPTT still often involves successive multiplication of some small values. The dreaded result is a gradient very close to 0, which yields no meaningful update to the network weights. In fact, fundamental limitations in floating point representation cause values very near 0 to end up exactly 0. Increasing the system's memory allocation for a number only postpones the point of underflow. Thus, the vanishing gradient problem ensues in either situation and, as is said colloquially, training is killed. On the other hand, the exploding gradient problem occurs ironically with the naive attempt to avoid vanishing gradients by using an activation function whose range extends to infinity, such as the rectified linear unit ($f(x) = \max(0, x)$). Repeated chain rule multiplications during BPTT of values much greater than 1 lead to very large gradient values, intensely destabilizing training. Similar numerical limitations to the ones previously described can then induce overflow and result in infinity. Worse yet, many operations with infinity are resolved as NaN (Not a Number), which subsequently propagates to all weight matrices and also kills training. Indeed, both gradient problems are greatly exacerbated by sequences with many time steps and networks with multiple hidden layers. To be precise, the gradients vanish or explode exponentially quickly with the length of the input and the depth of the network. Which problem occurs first depends on the network and training configuration used, though the result can still be unpredictable due to the stochastic nature of training. These observations, first made three decades ago, effectively stalled any hope of training RNNs on time series data of useful length [45].

Long Short-Term Memory

The Long Short-Term Memory (LSTM, 1997) network architecture is a specialized RNN variant that proposes a solution to the vanishing gradient problem [48]. Its breakthrough is the addition of gates within each neuron in the recurrent layer, having the effect of creating an internal state whose potential for modification is controlled. For convenience, the term vanilla RNN denotes a traditional recurrent neural network without

such gates. The four specific new components introduced by LSTM networks are: the cell state, the input gate, the forget gate, and the output gate. In vanilla RNNs, neurons in the recurrent layer each only have a hidden state to propagate to either the next layer or to the next time step neuron within the same recurrent layer. On the other hand, neurons in the recurrent LSTM layer are called cells because of the increased complexity in each also maintaining an internal cell state. The addition of cell state permits each cell to have an individual memory. Then, the three gates regulate the flow of information in and out of a cell [45]. The interesting LSTM operations are summarized below.

$$\begin{aligned}
\tilde{c}_t &= \tanh(U_c \cdot x_t + W_c \cdot h_{t-1} + b_c) \\
i_t &= \sigma(U_i \cdot x_t + W_i \cdot h_{t-1} + b_i) \\
f_t &= \sigma(U_f \cdot x_t + W_f \cdot h_{t-1} + b_f) \\
o_t &= \sigma(U_o \cdot x_t + W_o \cdot h_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}$$

See Figure 1.5 for a diagram of an LSTM cell. Here, $\tilde{c}_t$ is the activation for new input to the cell at time t, where input consists of x_t and the hidden state from the previous cell h_{t-1}. Meanwhile, i_t, f_t, and o_t are the input, forget, and output gates for the cell at time t, which respectively control the effect of new input, the level of information transfer from previous input, and the flow of output from the cell. Each of these components has their own weight matrices U and W shared across time, as well as bias terms b. The internal cell state c_t is notable for the sum that takes place between the previous cell state c_{t-1} and the activated new input $\tilde{c}_t$, both modulated by their respective gates. The intuition for how this sum greatly reduces the vanishing gradient problem is by providing a path for processed inputs to not decay exponentially fast, which was the case with vanilla RNNs. Finally, h_t is the hidden state of the cell at time t, propagated to the next layer and to the next cell at time $t + 1$ [45]. Note that $\odot$ denotes element-wise multiplication,

while $\tanh$ and σ signify the nonlinear activation functions of the same name ($\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, $\sigma(x) = \frac{1}{1+e^{-x}}$).

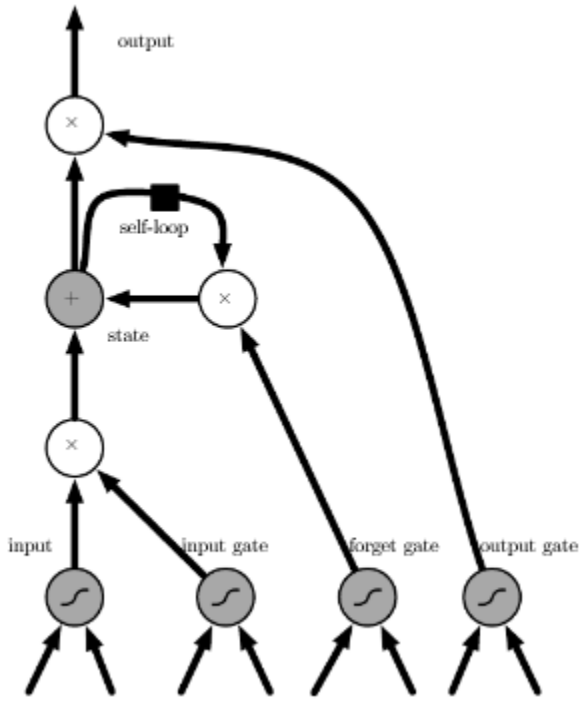

Figure 1.5: Architecture of one LSTM cell. Note there are as many cells in a recurrent LSTM layer as there are time steps. Input corresponds to $\tilde{c}_t$ above. Input, forget, and output gates provide control on information transfer. The left and right arrow input for each bottom node is, respectively, x_t and the hidden state from the previous cell h_{t-1}. State and output correspond to c_t and h_t above, respectively. Reproduced from [45].

Conventional RNNs and LSTM networks only consider temporal data in one time direction, generally in left-to-right sequence from the user's perspective. However, the nature of many data types is that future time steps have a context effect on the current time step. To address this limitation, bidirectional RNNs and LSTM networks are developed. The idea is to use two recurrent hidden layers, one for positive time direction and another for negative time direction. More precisely, both of these layers receive the same input but process it in reverse directions. They also both send their output to the next layer, and not to each other. A slight variation to backpropagation through time is required to correctly update weights in this special network structure. Already, the use of LSTM networks over

vanilla RNNs involves significantly more trainable weight parameters due to the addition of cell state and three gates. The next step of exploiting bidirectional LSTM networks in fact doubles this number. Since training time is an important consideration, bidirectional architectures should therefore not always be favoured. Coupled with the uncertainty of hyperparameter tuning, it can be difficult to find a working network configuration. One mitigating strategy is to find suitable hyperparameters based on a unidirectional architecture, then apply them to an analogous bidirectional design [45].

On the application side, the previously mentioned Lim-Blanchette (2020) model uses a sequence-to-sequence bidirectional LSTM network architecture, within the framework of an encoder-decoder, to simulate context-dependent sequence evolution [20]. In particular, it uses an LSTM-encoder to convert an ancestral sequence into a concatenated vector representation of the forward and reverse sequence context. Using the known ancestral sequence, the LSTM-decoder takes these vectors and outputs a probability distribution for every character of the descendant sequence. Each character is randomly selected according to its distribution, emitted, and sent as output to the next decoder cell in sequence. The set of all possible descendant sequences for a given ancestor can be obtained by repeatedly decoding the encoded ancestral sequence. The model considers the 14 nucleotides flanking each position in the input sequence, giving an unprecedented context size of 15. It has shown promising results when tasked with modeling mammalian and plant DNA sequence evolution, successfully capturing context dependencies for both substitutions and short indels.

1.2.3 Generative Adversarial Networks

In machine learning, there are broadly two classes of models: generative and discriminative. Suppose we have a set of data instances X and a set of associated labels Y. A generative algorithm learns the joint probability distribution $P(X, Y)$, while a discriminative algorithm captures the conditional probability distribution $P(Y|X)$. More intuitively, the former learns the complete data distribution as well as the probability of

a specific data instance, while the latter captures the probability of a label applying to a specific data instance. Typically, generative algorithms are more difficult to train because they need to incorporate more information to successfully generate new data instances. Once trained, however, they can simulate fake data that closely resembles, or is ideally indistinguishable from, the real data distribution [45].

In this work, the objective is to develop a generative approach that automatically learns the correct parameters and weights. In turn, the trained model is expected to output fake instances that cannot be differentiated from the instances found in the real dataset. We exploit a generative adversarial network (GAN, 2014) to achieve this goal [49]. In this framework, a generator network G initially creates plainly fake descendant DNA sequences. A discriminator network D is then fed descendant sequences, which are a mix of the real dataset and the output from G. D must consistently distinguish G's generated sequences from real descendant sequences. G is trained such that the probability of D mislabeling G's generated sequences as real is maximized. On the other hand, D is trained such that the probability of a mislabel is minimized. G and D operate in a feedback loop. As training progresses, G eventually produces descendant sequences that D can no longer distinguish from real sequences. Consequently, D is effectively guessing with a 50% average success rate as to the real or fake nature of its input [45].

Supervised and unsupervised learning are two broad categories of approaches in machine learning. In the former, the goal is to capture a general mapping between data instances and data labels. In the latter, labels are not provided and the target is to learn the underlying structure of data instances. To be precise, a GAN is an unsupervised learning algorithm that is trained using a supervised loss. The objective is for G to model the real data and generate new sequences based on what it has captured. Since all generated sequences are fake, G does not consider labels and in effect only learns the probability distribution of real data $P(X)$. On the other hand, D traditionally uses a binary cross-entropy loss ($-\frac{1}{N}\sum_{i=1}^{N} y_i \log(P(y_i)) + (1 - y_i)\log(1 - P(y_i))$) to determine how well it is discriminating real from fake sequences. To this end, arbitrary labels are assigned to mark

the source of each data instance for D to compute the loss function. Conventional labels of 1 for real and 0 for fake data instances are used in this work.

See Figure 1.6 for a diagram of GAN structure. The interaction between G and D is essentially a two-player game where the overall value is given by the following minimax function: $E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))]$. G minimizes this function, while D maximizes it. Here, E_x is the expected value over all real data instances x, and $D(x)$ is the probability output of D on real instance x. Meanwhile, z is a random input, or latent, vector commonly sampled from a standard normal distribution to act as a trigger for G to generate fake data instances. E_z is the expected value over all random inputs z, while $D(G(z))$ is the probability output of D on fake instance $G(z)$. More formally, G seeks to maximize $D(G(z))$ and thus minimize the second term of the function. G is indirectly trained only by aiming to better fool D, hence the adversarial nature of GANs [49].

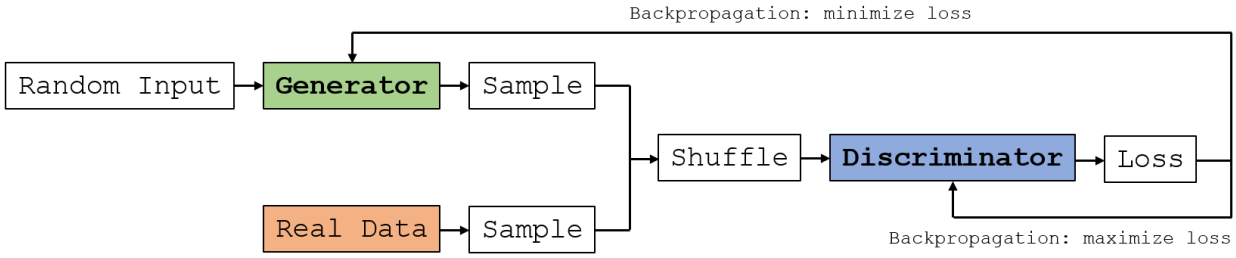

Figure 1.6: Diagram of GAN structure. The discriminator's loss value is used to update the weights of both itself and the generator via backpropagation. The discriminator seeks to maximize its ability to distinguish real from fake samples, while the generator wishes to minimize this ability of the discriminator.

Conditional GANs

A GAN is a generative algorithm that implicitly models the target probability distribution. Unfortunately, the use of a randomly changing latent vector z to generate a variety of data instances from G is wildly unpredictable. Given only a specific data instance from G, there is no way to trace the latent vector that generated it. A straightforward modifi-

cation to restore a degree of control is to package some additional information with z to act as a condition, giving rise to the term conditional GAN.

See Figure 1.7 for a diagram of conditional GAN structure. Since the goal of this work is to simulate DNA sequence evolution and G already generates descendant sequences, it is reasonable to use ancestral sequences as the condition. In practice, a conditional ancestor is merged with z via simple concatenation, or a more complex operation may be used. On D's side, the conditional ancestor also merges with the input descendant sequence in a similar way. True ancestral sequences are provided when D receives real descendant sequences. The intuition for these changes is that D can now more easily assess if its input descendant sequence corresponds plausibly to the associated ancestral sequence [45]. In this way, G eventually learns to evolve its input conditional ancestor into a realistic descendant. The stochastic nature of evolution is preserved by continuing to give randomly changing latent z vectors to generate a diversity of descendants.

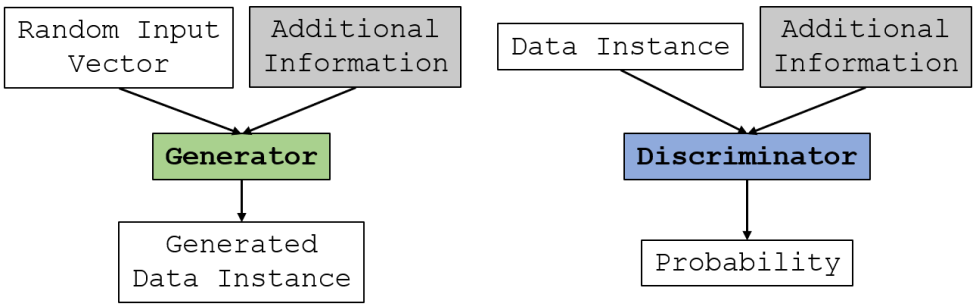

Figure 1.7: Diagram of conditional GAN structure. When the discriminator receives a generated data instance, its additional information is the same as what was given to the generator to produce the data instance. Note that the discriminator outputs a probability, the collection of which over all data instances is then used to compute a loss function.

There are three common problems with GANs using cross-entropy loss and operating under the minimax function described above. First, vanishing gradients can occur where D is too effective at distinguishing real from fake sequences, which is often the case early in training. The result is that G lacks the information to produce better fake sequences, having essentially only received a signal to stop and try again from scratch. Second, G can produce output that completely lacks diversity despite a randomly changing latent

vector z. Termed mode collapse, this phenomenon is nearly always undesirable. It indicates that D's training has stalled, since G succeeds in fooling it with the same sequence consistently. Third, the GAN can fail to converge, or reach the point where D is guessing with an average 50% success rate as to the real or fake nature of its input. Since convergence is hard to detect and D is effectively useless at this stage, GANs can also overshoot the convergence point, which then leads to the deteriorated quality of G's sequences. An alternative indication is constant oscillation between G and D winning with no clear direction of improvement [50]. Overall, GANs require consistent attention throughout training to ensure computational time is not wasted. Promising training configurations often fail inexplicably and cannot be salvaged. Persistent effort in hyperparameter tuning is key to training success.

Wasserstein GANs

To tackle the difficulty of training GANs, a variant called Wasserstein GAN (WGAN, 2017) has been proposed [50]. Instead of using cross-entropy, the WGAN's loss function is based on the Wasserstein, or Earth-Mover's, distance. This new metric is intuitively described as a transport problem to find the minimum cost of moving dirt to convert one dirt pile into a reference dirt pile. A formal treatment of the Wasserstein distance is out of scope here. For convenience, the term vanilla GAN denotes a traditional generative adversarial network using a cross-entropy loss.

Recall that the discriminator output of a vanilla GAN is a probability bounded between 0 and 1. The threshold of 0.5 is most commonly used to make a real or fake prediction. Using the Wasserstein distance, the WGAN's discriminator is unbounded and no longer predicts whether an input descendant sequence is real or fake. Instead, it simply outputs a scalar value, or score, that should be larger for real sequences than for fake ones. This fundamental change in objective gives the WGAN discriminator the new name of critic. More precisely, the critic network C seeks to maximize the Wasserstein distance $\frac{1}{N}\sum_{i=1}^{N} C(x_i) - C(G(z_i))$, which is the average difference between the critic's score on

real sequences x and fake sequences $G(z)$, respectively. On the other hand, G wishes to minimize the Wasserstein distance and does so by updating its weights to maximize the average score assigned to its generated sequences $\frac{1}{N}\sum_{i=1}^{N} C(G(z_i))$ [50].

Empirical evidence indicates that Wasserstein distance correlates with generated sample quality, while there is no such pattern when using cross-entropy loss. In other words, G's performance is maximized as $C(x) - C(G(z))$ approaches 0. However, this result depends on a smoothness guarantee called the 1-Lipschitz constraint, which is a method of limiting the change in the Wasserstein distance at each update such that it is not too large. The WGAN clips the gradients to some range $[-c, c]$ to ensure the gradient Euclidean norm is at most 1 everywhere in the critic space. In practice, finding the optimal c value to both respect this constraint and not overly impair training is difficult [51].

An elegant solution, superseding clipping, is introduced with the concept of a gradient penalty. To be precise, the new term $\lambda(||\nabla_{\tilde{x}} C(\tilde{x})||_2 - 1)^2$ is subtracted from the existing Wasserstein distance described above. Here, C is softly penalized if the gradient Euclidean norm moves away from 1, and this penalty is multiplied by some coefficient λ. Though the gradient norm constraint is ideally enforced at all points in the critic space, this operation is unfortunately intractable. It suffices in reality to check the gradient norm at a uniformly random point $\tilde{x}$ on the line between real and fake sequences [51].

Note that implementations of WGAN with gradient penalty (WGAN-GP) flip the sign of the previously described equations due to how gradients are optimized in practice. More detail on this change will follow in the next subsection. In other words, C minimizes the Wasserstein distance while G maximizes it. The gradient penalty term is then added to the Wasserstein distance to preserve C's correct penalization. The WGAN-GP loss function used therefore becomes $C(G(z)) - C(x) + \lambda(||\nabla_{\tilde{x}} C(\tilde{x})||_2 - 1)^2$.

In recent years, there have been several promising applications of GANs to sequence-based problems in bioinformatics. For instance, the Killoran-et-al. (2017) model proposes a WGAN-GP approach to create convincing fake DNA sequences with certain desired properties, such as the appearance of a particular motif [52]. The generator and critic net-

works are both based on a convolutional architecture. The Ghahramani-Watt-Luscombe (2018) model also uses a WGAN-GP method to simulate realistic single cell RNA sequencing data spanning a range of epidermal, neural, and hematopoietic cell types [53]. Here, the generator and critic networks are each composed of a dense MLP with one hidden layer. The Wang-Dizaji-Huang (2018) model introduces a conditional vanilla GAN approach to infer new gene expression profiles using landmark gene expression profiles as the condition [54]. It adds a custom loss term to the original minimax function to avoid mode collapse and achieve improved results.

The Gupta-Zou (2019) model proposes a novel feedback WGAN-GP method to better generate fake DNA sequences with desired properties [55]. In particular, it uses an external function analyser to assess the generated sequences, the best of which become part of the real dataset fed to the critic. Over time, the entire real dataset is replaced by generated sequences. The Berman-et-al. (2020) model applies an LSTM-based WGAN without gradient penalty to the task of simulating protein sequence evolution [56]. The generator is a sequence-to-sequence bidirectional LSTM network with an encoder-decoder architecture, while the critic is an encoder that shares the weights of the generator's encoder to simplify training. The Yelmen-et-al. (2021) model exploits a vanilla GAN with no modification to create artificial human genome segments that augment genetic datasets limited by individual privacy laws [57]. Expected challenges with mode collapse are discussed.

1.2.4 Training Strategies

In this work, a conditional WGAN-GP is used as the variant of choice. The underlying generator and critic networks are LSTM-based. Even with all of its theoretical improvements over the vanilla GAN, the WGAN-GP still needs some effort to train.

Optimization

Recall that ANNs are usually trained using backpropagation, which computes the gradient of the loss function with respect to the network weights. A second algorithm then applies the gradient in the direction of steepest descent to the loss function. Termed gradient descent, this method requires a differentiable loss function that is configured for minimization. Unfortunately, gradient descent only guarantees finding a local minimum, though the global minimum is generally desired. A greater complication comes from the existence of saddle points, where the gradient appears to be 0 in all directions but in reality the loss minimizes further. These points offer a plateau region of loss function that is difficult for gradient descent to escape and optimize [45].

There are three general approaches to gradient descent. First, the default method of batch gradient descent processes the entire training set as a single batch and applies the average gradient over all the data instances. Since loading data instances in and out of memory is a bottleneck operation, this variant is the slowest, though it is the most reliable at approximating the true gradient. Second, stochastic gradient descent involves applying the gradient computed using batches of a single data instance at a time. It provides the fastest speed of convergence because its computational simplicity enables many more iterations in the same time span. However, it suffers from a propensity for noisy oscillations that can potentially continue indefinitely and avoid the global minimum. Nonetheless, stochastic gradient descent has the best empirical performance in escaping saddle points. Third, mini-batch gradient descent presents a hybrid approach by using batches of a few to at most several hundred data instances to apply an average gradient. A compromise is therefore obtained between fast and steady convergence [45].

Many extensions to these three gradient descent methods have been proposed in the past decade. The main innovations are momentum and adaptive gradients. The former accelerates gradient descent in the same direction, akin to a ball rolling down a hill, and helps reduce oscillations. The latter offers more flexibility in that each parameter has its own dynamic step size for the effect of each gradient update. A popular algorithm

combining these two advancements is Adaptive Moment Estimation (Adam, 2014) [58]. In particular, running averages of the first and second moments of past gradients, or the mean and variance respectively, are used. A formal analysis of the theoretical properties of this optimizer is out of scope here. Adam has seen promising empirical results for several years as a computationally efficient method that is light on memory and succeeds on complex parameter-rich tasks [45].

Hyperparameters

There are numerous hyperparameters to tune on the level of the WGAN-GP framework, its constituent LSTM networks, and the Adam optimizer. First, the WGAN-GP requires a latent vector z of a certain number of dimensions and a gradient penalty coefficient λ. The training ratio, or the number of times the critic updates for each time the generator updates once, is also usually a user-defined parameter. Fortunately, there exists a proof showing it suffices to have a training ratio of 1 if two different time scales, or learning rates, are used for the optimizer of the generator and critic [59]. Furthermore, the number of epochs, or complete passes of the training set, to execute before terminating training of the GAN is a consideration.

On the LSTM network side, the number of hidden LSTM layers to use in the generator and critic needs thought. The principle of deep learning suggests more complex and generalizable information can be integrated with deeper networks, at the cost of speed. While there are as many cells as time steps in a recurrent layer, the number of hidden units of each cell, which denotes the dimensionality of the cell output, is also a factor. In addition, dropout layers, where cells in the preceding hidden LSTM layer are randomly inactivated for a training step, are often used to prevent overfitting to the training set. The probability of dropout, or of such an inactivation, needs to be user-specified.

The Adam optimizer takes a specified batch size of data instances, which lies ambiguously somewhere in the range of a few to several hundred. Each version of Adam

in the generator and critic networks requires its own base learning rate, or gradient descent step size, α from which the adaptive gradients are computed. This hyperparameter can range from $1\text{e-}3$ to $1\text{e-}7$, with larger values converging faster but risking missing the global minimum, and smaller values taking an almost intractably long time to reach optimality. Furthermore, Adam's first and second gradient moments each have their own exponential decay hyperparameter, respectively β_1 and β_2, such that larger values cause these moments to be retained longer [58].

Curriculum Learning

Recall that DNA sequence data shares characteristics of images and text. Longer sequences contain more information, but are also more challenging to simulate due to the increased contextual complexity. A method called curriculum learning, inspired by the way humans are taught, offers a stepwise approach to ease the learning process.

In the beginning, very short sequences are used to train the GAN until the desired simple concepts are integrated. Then, progressively longer sequences are fed as increasingly complex information is incorporated by the generator and critic models. This strategy has been empirically shown to accelerate overall training and provide additional consistency to the learning process [45].

Chapter 2

Materials and Methods

We introduce EvoSeqGAN, an LSTM-based conditional WGAN-GP approach to modeling context-dependent DNA sequence evolution. In it simplest form, the trained model takes as input a random latent vector of length L and an ancestral DNA sequence S of the same length as the condition. It outputs a generated descendant sequence T that may be identical to S, different via one or more substitutions, and shorter or longer via one or more indels. In particular, we design two models: a substitution-only model where T always has length L, as well as a more complex substitution plus indel model where the length of T may vary. For convenience, we refer to them respectively as the substitution and indel models throughout this work.

This chapter presents the datasets and general methodology used to train EvoSeqGAN in Section 2.1 and Section 2.2, respectively. Specific details concerning the data encoding scheme (2.1.1), the network architecture (2.2.1), the training process (2.2.2), the evaluation procedure (2.2.3), and the implementation (2.2.4) are discussed.

2.1 Datasets

EvoSeqGAN is trained from a set of ancestral-descendant human sequence pairs. Sequences belonging to the modern human, constituting the descendant, are taken from chromosome 20 of the human genome hg38 assembly. For the ancestor, a 100-way alignment of whole vertebrate genomes has been processed through the Ancestors 1.0 program, an algorithm for inferring maximum likelihood ancestors on a genome-wide scale using a simple context-independent substitution model [20, 60, 61]. The generated ancestor selected for this work is the catarrhinian, or Old World monkey, ancestor. Advantages of this ancestor choice include a genetic distance close enough to the modern human to reduce the risk of double mutations at the same sequence positions, while far enough to provide a sufficient number of mutations for training [20].

Alignment data is downloaded from the repositories belonging to the UCSC Genome Browser and McGill Centre for Bioinformatics [62]. The first 48 million lines of the file are found to contain sufficient training data and are subsequently stripped of superfluous information. More precisely, only alignment blocks where a catarrhinian ancestral sequence is predicted are isolated for project use. As a result, genetic regions of more recent origin are excluded. The total amount of sequence data corresponding to these blocks is outlined later in this section. See Figure 2.1 for a visual depiction of the described evolutionary relationship.

The sequences have already been aligned within their respective blocks. However, they vary widely in length, from as short as a single character to several hundred characters long. The data cleaning stage seeks to eliminate sequence length as an additional learning parameter. We therefore concatenate all ancestral and descendant sequences into two very long respective strings. These strings are then split into chunks of arbitrary length L. Note that this operation does not break the initial alignment of the sequence pairs.

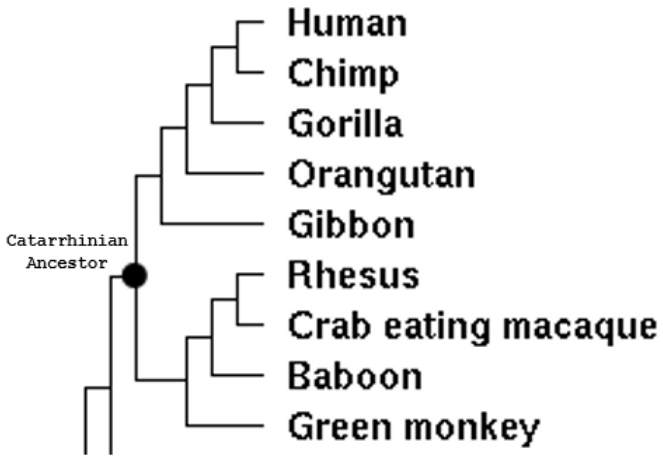

Figure 2.1: A cropped primate phylogenetic tree. The black node corresponds to the catarrhinian ancestor, the most recent common ancestor of humans and the eight other primate species shown. Reproduced from [63].

Gaps are irrelevant for the substitution model, so character pair positions that contain a gap at either or both of the ancestor and the descendant are simply removed. The intuition for this treatment is to preserve in a simple way the maximum amount of original contextual information. While the resulting alignment no longer perfectly matches the true ancestral and descendant sequences, it is acceptable for the purpose of training a substitution model. Recall that the true ancestral sequences are in any case limited in their accuracy given the context-independent model used to infer them. Conversely, gaps are desired for the indel model, but only in sequence pair chunks that contain fewer than 50% gaps after removal of redundant aligned gaps between ancestor and descendant. Here, the idea is that sequences composed of mostly gaps are not as useful for training. The affected chunks are therefore simply excluded. Note again that, despite these processing steps, pairwise alignment between associated sequences is still maintained in either case.

The collection of cleaned and aligned chunks totals approximately 9.4 million character pairs for the gapless case and 9.5 million character pairs for the case with gaps. While these character pairs are not true nucleotide pairs of DNA, we use the term base pair (bp) interchangeably throughout this work for convenience. Multiple versions of the dataset are prepared for chunk sizes of 500 bp, 5,000 bp, and 15,000 bp in the substitution model

versus 50 bp, 500 bp, and 3,000 bp in the indel model. More detail on the difference in series of chunk sizes will follow in Section 2.2.2.

The dataset is then divided into training and test sets according to a 90%-10% split. A validation set, often used in machine learning to give an unbiased indication of a model's performance during hyperparameter tuning, is of limited utility when training GANs. Recall that the goal is to train the GAN's generator to produce descendant sequences that cannot be distinguished from those in the real dataset. Due to the difficulty of training, it is already considered a major success if repeatedly sampling the generator using conditional ancestral sequences from the training set yields varied and plausible output. It therefore suffices to have just a test set that provides an unbiased final evaluation of the generator on unseen conditional ancestral sequences, thus serving as a measure of the model's generalizability.

2.1.1 Data Encoding

The aligned DNA sequence data must be transformed into a form friendly for mathematical operations. While both encoding schemes are based on a one-hot encoding of the ordered alphabet $[A, C, G, T, -]$, the one used for the substitution model is different from the one used for the indel model. In the former case, each character in the aligned gapless sequence pair is one-hot encoded such that the ancestral sequence's alphabet occupies the first five bits and the descendant sequence's alphabet occupies the next five bits, for a total of ten bits to encode each character pair. Overall, the aligned gapless sequence pair is two-hot encoded over ten bits. See Figure 2.2 for a simple example of this gapless encoding. Note that inclusion of the gap character is unnecessary in the encoding of the gapless substitution model, but does not harm training. It is kept for consistency.

In the indel case, the ancestral sequence's alphabet is the same as previously described and likewise occupies the first five bits of each character pair encoding. The difference comes from the encoding of indels. In fact, a deletion event of arbitrary size X from

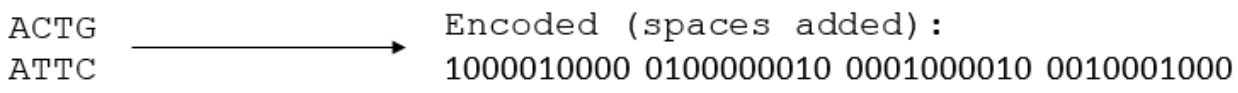

Figure 2.2: A toy example of the gapless encoding. Note that spaces are added for ease of viewing and do not appear in practice.

ancestral sequence to descendant sequence is easily handled as the substitution of the X characters in the descendant into gap characters at the same position. However, it is unclear how best to represent an arbitrary size insertion event in the same direction. The encoding cannot assume that gaps exist in the ancestral sequences, since that would be equivalent to knowing ahead of time where insertions reside.

Our approach is to extend the descendant sequence's alphabet to 90 characters: 4 usual nucleotides, 1 gap, 16 dinucleotides for insertions of size one, 64 trinucleotides for insertions of size two, 4 x 1 special character denoting insertions of size three or more for each possible ancestor nucleotide, and 1 dummy character. To be explicit, each dinucleotide is composed of the descendant's nucleotide at the previous position and the nucleotide to be inserted, and each trinucleotide follows analogously with the two nucleotides to be inserted. The special character is composed of the descendant's nucleotide at the previous position and an arbitrary character 'X' to denote the large insertion. The dummy character's main function is to aid in debugging should the implementation of this encoding prove troublesome. Recall that aligned gaps between ancestor and descendant were removed from the data in earlier processing, so no character is necessary for this case.

While more detail on network architecture will follow in the next section, we clarify that the two EvoSeqGAN models generate descendant sequences that mimic their respective encoding schemes. The recovery of the DNA sequence and the reconstruction of the ancestral-descendant alignment therefore proceeds from a decoding stage. In the substitution model, the decoding is a straightforward reversal of the encoding process with simple character alignment of the generated descendant sequence and the ancestral sequence given as the condition. In the indel model, the encoding process is likewise reversed, but care must be taken to correctly insert gap characters in the ancestral sequence

at the position corresponding to the descendant sequence's instruction: one gap for an encountered dinucleotide and two gaps for an encountered trinucleotide.

The four special characters are decoded deterministically and naively for convenience: they are assumed to denote a respective insertion of exactly AAA, CCC, GGG, and TTT in the descendant. Consequently, three gap characters are inserted in the ancestral sequence. This simplifying decision is justified by the relative rarity in insertions of size three or more, the most likely such insertion being a tandem repeat of this type, as well as the clearly exponential blow-up in extended alphabet size required to encode large insertions precisely. An unattempted alternative is to create insertions of size three or more randomly according to the probability distribution describing the length and composition of such insertions in the real dataset.

2.2 Methodology

EvoSeqGAN is an LSTM-based architecture trained using a conditional WGAN-GP framework. In particular, we examine the methods used for the unidirectional substitution and indel model designs, as well as their bidirectional variants.

2.2.1 Network Architecture

See Figure 2.3 for a diagram of EvoSeqGAN's generator and critic network components. We explain the architecture using one encoded input ancestral DNA sequence of length L for simplicity. However, a batch of sequences is fed at a time to each network in practice. We also highlight the differences between the substitution and indel model designs. Extensive tuning was done to determine the optimal hyperparameter values mentioned in this section.

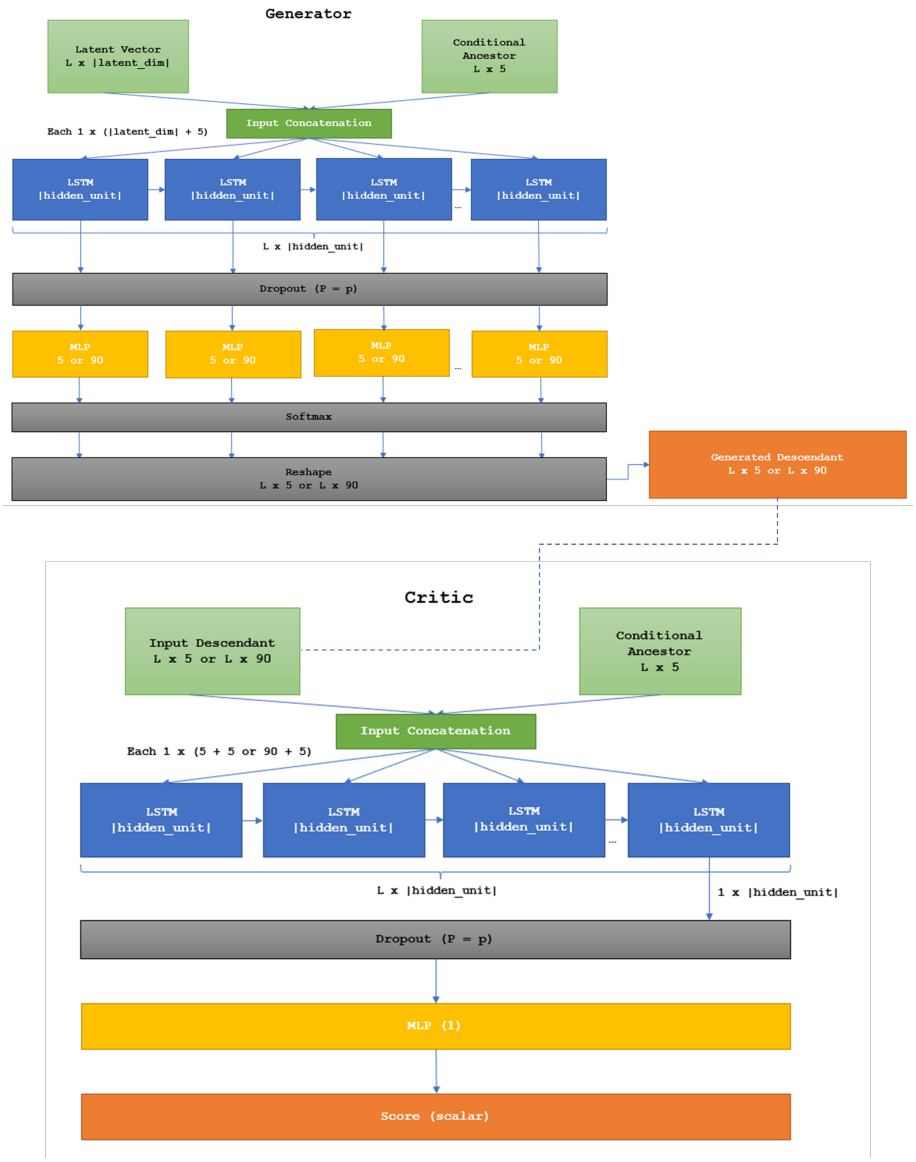

Figure 2.3: A diagram of the unidirectional generator (above) and critic (below) networks. In our implementation, |latent_dim| is 10 and |hidden_unit| is 48, while L depends on the length of the conditional ancestral sequence. Dimensions separated by 'or' denote the difference between the substitution and indel model architectures. In the bidirectional case, a second LSTM layer unconnected to the first is added to both networks whose horizontal arrows run in the reverse direction, but takes the same input as the first LSTM layer and also sends output to the MLP layer.

Beginning with the generator, random latent vectors are sampled from the standard normal distribution. The dimensionality of the latent space is selected to be 10, and there

is one latent vector for each of L characters in the input ancestral sequence, resulting in an L x 10 matrix. The ancestral sequence, used as the condition, has length L and has dimension 5 due to its one-hot encoding, giving an L x 5 matrix. We refer to this sequence as the conditional ancestor throughout this work. These two inputs are then concatenated to form an L x (10 + 5) matrix.

Next, there is an LSTM layer composed of L LSTM cells, where each cell's hidden unit dimensionality is selected to be 48. Each of the L LSTM cells receives a different individual position of the concatenated input, corresponding to a 1 x (10 + 5) matrix. The LSTM layer is configured such that each previous LSTM cell sends its position output, or hidden state, to the next cell in time sequence, as well as to the next layer. Note that we do not explicitly output the internal cell state in any model. A dropout layer follows with an inactivation probability selected to be 0.5, meaning that half of the LSTM cells are randomly set to 0 for a training step, to fight generator overfitting.

An MLP, or dense, layer receives the hidden state output from the LSTM layer post dropout. Each neuron in the MLP is time-distributed, or distinct from the other temporally, when it takes input from its respective LSTM cell. Consequently, each MLP neuron gets input from only one LSTM cell and sends output to no other MLP neuron. The dimensionality of each MLP neuron output is set to 5 or 90 for the substitution and indel models, respectively, which corresponds to the desired dimensionality of each position in the generated descendant sequence.

A softmax layer then applies the softmax function ($y_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e_j^x}$, where y_i is the probability of element i, x_i is the original unnormalized value of element i, and there are k total elements) independently to each MLP neuron output to normalize their values into a probability distribution. In this way, an argmax operation in the decoding stage can provide the most probable encoded character for each position represented by an MLP neuron. Note, however, that the argmax operation is not differentiable and so does not occur in the generator during training. Finally, a reshape layer repackages the individual

temporal slices into an L x 5 or L x 90 matrix depending on the respective substitution or indel model used, giving the generated descendant sequence in encoded form.

On the critic side, an encoded descendant sequence is fed from either the real dataset or from the generator. Termed input descendant, its dimensionality is either L x 5 or L x 90 respectively for the substitution and indel models. The conditional ancestor, meanwhile, is always L x 5 regardless of model design. The same conditional ancestor given as input to the generator to synthesize a fake descendant sequence is again provided to the critic when this fake descendant sequence is the input descendant. On the other hand, true ancestral-descendant sequence pairs from the real dataset are split and fed individually as conditional ancestor and input descendant, respectively. In either case, the input descendant and conditional ancestor are concatenated to form an L x (5 + 5) or L x (90 + 5) matrix, depending on the model used. The intuition with the concatenation is that the critic should certainly learn to consider paired ancestral-descendant information when attempting to discern which input descendants are real or fake.

Next, an LSTM layer composed of L LSTM cells has each cell's hidden unit dimensionality again selected to be 48. Similarly, each of the L LSTM cells receives a different individual position of the concatenated input, corresponding to a 1 x (5 + 5) or 1 x (90 + 5) matrix depending on the model used. While each previous LSTM cell continues to send its hidden state to the next cell in time sequence, only the last cell temporally propagates its hidden state to the next layer. In this way, only one transformed vector representing the product of an entire sequence of LSTM cell processing is sent, and not a sequence of individual vectors. A dropout layer follows where an inactivation probability of 0.5 is again used to combat critic overfitting.

An MLP layer composed of a single neuron receives the 1 x (48) hidden state matrix post dropout from the last cell in the LSTM layer. It outputs a single value regardless of model used. Within the context of WGAN-GP, this scalar denotes the critic's score on the given input descendant. Recall that the goal of the critic is to maximize the difference in scores assigned to real and fake descendant sequences, without restriction to any

$[0, 1]$ range or that scores be strictly positive. Consequently, the critic may well converge on assigning negative scores to sequences perceived as real, depending on the training configuration and occasional chance.

We also experiment with analogous bidirectional LSTM generator and critic models that consider characters of the concatenated input in both the forward and reverse directions. The intention is to capture contextual information from before and after the current position of interest, since that is how biological molecules are known to interact organically as well. In this case, an additional LSTM layer with identical hyperparameters to the first is added that also receives the same concatenated input. The main difference is that hidden state propagation arrows between LSTM cells now point in the reverse direction. Note that the forward and reverse LSTM layers do not interact between each other, but send their respective cells' output to the same neuron in the MLP layer.

2.2.2 Training Procedure

EvoSeqGAN is trained using Wasserstein loss with gradient penalty: $C(G(z)) - C(x) + \lambda(||\nabla_{\tilde{x}} C(\tilde{x})||_2 - 1)^2$. Recall that the critic C seeks to minimize this function, while the generator G maximizes it. One training cycle consists of a critic update stage and a generator update stage. To be clear, parameters are untrainable when it is not a given network's update stage. A batch size of 32 is used throughout training for all network components and model designs, including the substitution and indel models as well as their bidirectional variants.

In the critic update stage, a batch of real input descendants is given as input to the critic together with their associated conditional ancestors. At the same time, a batch of fake input descendants is synthesized on the fly by the generator using the same conditional ancestors and a batch of random latent vectors sampled from the standard normal distribution. These fake descendants are likewise given as input to the critic with the conditional ancestors. Both batches are run through and a gradient update is subsequently

performed only for the critic. Trainable network weights include: the input, forget, output, and cell state weight matrices as well as bias terms for the one or two LSTM layers depending on the variant; weights for the single neuron in the MLP layer.

In the generator update stage, a different batch of conditional ancestors is used with a new batch of random latent vectors sampled from the standard normal distribution to synthesize fake descendant sequences. These fake input descendants together with their conditional ancestors are then run through a frozen version of the critic. A gradient update is subsequently performed only for the generator. Trainable network weights include: the input, forget, output, and cell state weight matrices as well as bias terms for the one or two LSTM layers depending on the variant; weights for the MLP layer.

Training is carried out using the Adam optimizer with a base learning rate of $1\text{e-}4$ for the generator of both models versus $5\text{e-}4$ for the critic in the substitution model and $3\text{e-}4$ for the critic in the indel model. Generally, the critic should update faster than the generator, since it is the component whose feedback is driving gradient updates, but not so fast that the generator does not receive useful signals for improvement. The intuition is to reduce the critic's accelerated learning rate slightly in the case of the more complex indel model. Meanwhile, Adam's first and second moment hyperparameters are set to 0 and 0.9, respectively. With a batch size of 32, the principle of mini-batch gradient descent has been favoured for a compromise between speed and accuracy. A large gradient penalty weight of 100 is also used when computing the total loss of the WGAN-GP to severely discourage gradient update spikes.

Training a WGAN, though easier than a vanilla GAN, remains capricious despite the purported correlation between sample quality and Wasserstein loss approaching 0. In our experience, this correlation is only weakly observed, so regular sampling of the generator is necessary to gauge true training progression. Therefore, no set number of epochs is used as a stopping point. Note that individual network hyperparameters have been outlined in the previous subsection. Drawing full inspiration from the unidirectional architectures, the analogous bidirectional variant of each model reuses the same training

hyperparameters. Intermediate generator and critic models are saved both periodically and checkpoint-style whenever a model pair outputs an improvement on the current best Wasserstein loss.

It is expected that the generator will have an increasingly harder time as input sequence length increases, while the critic will experience the opposite trend. Therefore, a curriculum learning strategy based on sequence length is employed to accelerate the learning process. For all chunk sizes given, the first one in succession refers to the substitution model while the second one is for the indel model. Running time is given in terms of hours; see the next chapter for the precise number of corresponding training epochs. The WGAN-GP is initially trained on batches of short 500 bp and 50 bp sequences. Stage one continues for about five hours until the generator has learned to output descendant sequences that perfectly match the conditional ancestor. Next, batches of 5,000 bp and 500 bp sequences are used in stage two for the next 48 to 72 hours of training until the generator has captured the most common mutation types with the correct associated frequencies by order of magnitude. Finally, batches of 15,000 bp and 3,000 bp sequences are employed for the remainder of training. The immediate goal in stage three is for the generator to learn all mutation types and appropriate frequency orders of magnitude, based on the premise that longer sequences contain more information to discover rare mutations. The hope is that continued training on the longest sequences enables capture of longer-range context dependencies and precisely correct frequencies for all mutations.

Across the board, shorter chunk sizes are used for the indel model than for the substitution model. While this practice may seem counter-intuitive, the reasons are twofold. First, the encoding size of 90 for an encoded indel descendant character is 18 times greater than the size of 5 for an encoded substitution descendant character. In other words, 18 times fewer indel descendant sequences can fit in the same amount of memory as substitution descendant sequences. Second, the greatly added complexity of indels tempers any enthusiasm to jump quickly to long sequences, given that more time is needed to reach the listed milestone of each step.

Unfortunately, memory limitations hinder the use of longer than 15,000 bp and 3,000 bp sequences with a batch size of 32. While a reduction in batch size does enable greater length, recall that there are as many cells in an LSTM layer as there are time steps or positions in the input sequences. The use of even longer sequences thus slows gradient updates from the perspective of increased network width and loss of parallelism related to reduced batch size. The selected sequence length of 15,000 bp for the substitution model offers a good compromise between sufficient length to contain subtle information and no excess length to severely impair training. On the other hand, the 3,000 bp sequence length for the indel model is likely too short to observe rare mutations, including most indels. The ability to handle significantly longer than 3,000 bp sequences requires remarkable memory, computational power, and patience, which are alas unavailable at this time.

2.2.3 Evaluation

An EvoSeqGAN generator model is easily used for evolution simulation, even on ancestral sequence lengths different from what was used for training. It suffices to load the saved model weights into a new generator model and provide one or more random latent vectors of size 10 concatenated with the desired ancestral sequence to evolve in encoded form. The set of all possible descendant sequences for a given ancestor can be obtained by thoroughly sampling the latent space with many latent vectors.

EvoSeqGAN learns to model the evolution that took place between the ancestral and descendant sequences it receives as input. Let the intermediate time span be called one unit of time. Should a longer unit time scale be desired, the generated descendant from one inference step can be fed as the conditional ancestor for the next inference step, repeated integer multiple times. A method showing preliminary success for dealing with non-unit time scales is to decompose the target time scale into its integer and fraction components. The ancestral sequence evolves as previously described integer multiple times. Then, a final evolution occurs where any proposed mutation is accepted with probability equal to the fraction component [20]. Replicating the entire operation from

scratch several times should yield diversity in final descendant sequences, matching the appropriately stochastic nature of sequence evolution.

GANs have traditionally been assessed qualitatively by sampling the generator and noting any perceived improvements in output as training progresses. With images, this approach is relatively straightforward and works adequately as humans are adept at recognizing visual patterns. However, the same cannot be said for natural language strings and biological sequence data. As a result, several objective metrics and measures have since been proposed to render the evaluation of GANs unbiased [59].

EvoSeqGAN uses the Kullback-Leibler divergence $D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in X} P(x) \log(\frac{P(x)}{Q(x)})$ as a measure of the statistical distance between a reference probability distribution Q and a target probability distribution P, each composed of elements x of a set X. In this case, Q is the distribution of mutation probabilities for ancestral to descendant sequences in the real dataset, P is that for ancestral to fake descendant sequences synthesized by the generator, and X is the set of all mutation types. To avoid undefined issues with $Q(x) = 0$ or $P(x) = 0$ for any term in the sum, we smooth the actual probabilities by adding 1 to the observed frequency of each mutation x for both Q and P.

Recall that Wasserstein distance is itself a metric between two probability distributions. However, EvoSeqGAN's version of Wasserstein loss is not considered objective for the purpose of evaluation, since the constituent critic network itself outputs the values that go into computing this distance. Nonetheless, it remains interesting to track.

2.2.4 Implementation

EvoSeqGAN is implemented using Keras 2.3.1 in Tensorflow 2.0 [64, 65]. Code and sample data files are available at `https://github.com/smacra2/EvoSeqGAN`.

Note that CuDNNLSTM is a GPU-accelerated LSTM layer developed by Nvidia. Unfortunately, it currently lacks support for higher order gradients, which occur when back-

propagating loss gradients based on a gradient penalty term. EvoSeqGAN is thus forced to use the CPU-bound version of LSTM layers. While most machine learning frameworks traditionally rely on a massively multi-core GPU to run many linear algebra operations in parallel, EvoSeqGAN directly benefits from a CPU with a faster clock speed to accelerate training.

Several unsuccessful workarounds were attempted to bypass the gradient penalty term and regain GPU functionality. Initial efforts to use the original cross-entropy loss immediately yielded poor results and unstable training. We also experimented with the traditional Wasserstein loss using clipping to ensure smoothness, obtaining mixed results. Finding a good clipping range was a tedious operation that added unnecessary complexity to an already well-populated list of hyperparameters to tune. It was decided that speed could be sacrificed in exchange for a more straightforward training experience with the gradient penalty term.

Chapter 3

Results

Repeated experimentation and extensive hyperparameter tuning were performed to find the most promising model configurations. This chapter presents a performance assessment of three EvoSeqGAN model variants in Section 3.1. An artificial example to test the output diversity of the models is showcased in Section 3.2, followed by a description of training duration in Section 3.3.

3.1 Model Performance

In general throughout this section, we first compare the unidirectional and bidirectional substitution model variants before introducing the unidirectional indel model.

We intended to show results for the bidirectional indel model. However, it suffered from intense training instability that was unsalvageable despite multiple restarts. We observed the consistent but random occurrence of an exploding gradient penalty term that eventually converted to NaN, subsequently killing network training. Despite it remaining unclear what caused the gradient penalty term to explode, a working configuration

could yet exist. We unfortunately lacked the resources for a successful search, and so only show bidirectional results for the substitution model.

3.1.1 Wasserstein Loss Over Time

We tracked the critic's Wasserstein loss output over the training duration for each model. See Figures 3.1, 3.2, and 3.3 for plots of the Wasserstein loss with and without the gradient penalty term versus sampled iterations for each model. We sampled the generator models regularly during each training epoch to gauge progression, and report iteration numbers based on this arbitrary sampling frequency. Unfortunately, there is no consistent conversion between iteration number and number of training epochs. Values were clipped to $[-20, 20]$ for improved readability. The stopping point of training for each model was decided based on results discussed in the next subsection.

Recall that a curriculum learning approach was used to separate training into three broad stages. Intermittent narrow spikes should be ignored as they stem from early volatility related to a reset optimizer state at the start of each training stage. See Section 3.3 for an indication of each stage's running time in terms of epochs, which can be interpreted to be proportional to the iteration number. In addition, running long experiments on a shared server sometimes resulted in out-of-memory issues caused by other users' encroaching programs. Narrow spikes could therefore also be attributable to manual restarts of the experiment to continue training from the last saved checkpoint. In particular, gradient penalty spikes denote network attempts to smooth training and avoid large sudden gradient updates.

For each model, we present the original Wasserstein loss that considers only the difference in critic score on real and fake sequences, as well as the modified Wasserstein loss plus gradient penalty term used to drive training in practice. In all cases, a loss value closer to 0 is thought to be correlated with greater sample quality. Recall that a positive loss value generally indicates that the critic has assigned a positive score to fake sequences

and a negative score to real sequences. Conversely, a negative loss value generally signals that the critic has assigned a negative score to fake sequences and a positive score to real sequences. However, same sign scores are also possible in either situation given the right magnitudes. Note the lack of practical significance of the signs involved beyond hoping for the difference in scores to converge towards 0. Loss values should not be compared between model variants due to the different training configurations.

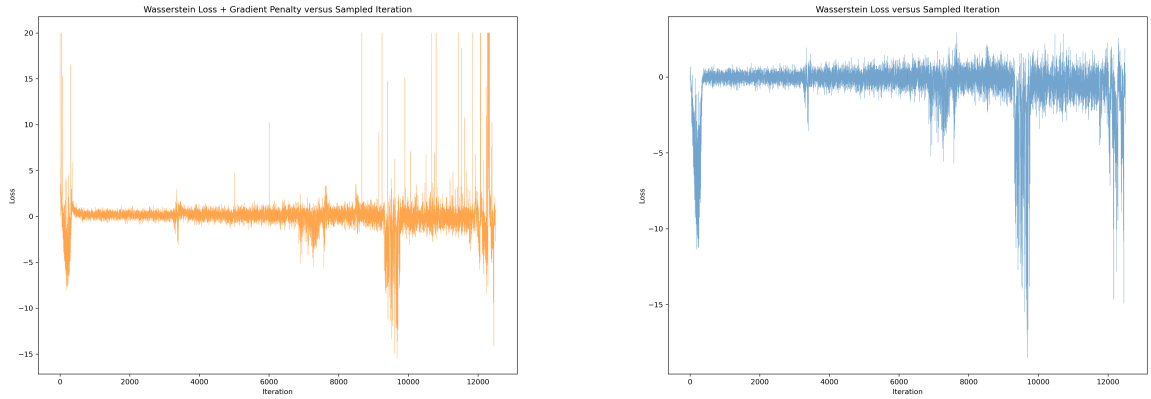

Figure 3.1: Wasserstein loss with (left) and without (right) gradient penalty term versus training time for the unidirectional substitution model.

For the unidirectional substitution model, the Wasserstein loss remained quite stable over time with a noticeable mean around 0. There was increased variance in the latter half of training, suggesting that the critic was improving faster than the generator and thus better able to discern a difference between real and fake sequences. Lending support to this idea is the presence of more frequent gradient penalty spikes, which oppose the generator's response of large gradient updates to keep up with the critic.

For the bidirectional substitution model, the Wasserstein loss became increasingly variable as training progressed. The loss pattern suggests that the addition of a second LSTM layer to the generator and critic was problematic for both networks. While the generator struggled during training, the critic also had difficulty in consistently discriminating between real and fake sequences, evidenced by the inconsistent sign of loss values. The use of analogous hyperparameters to the unidirectional case for training was a good

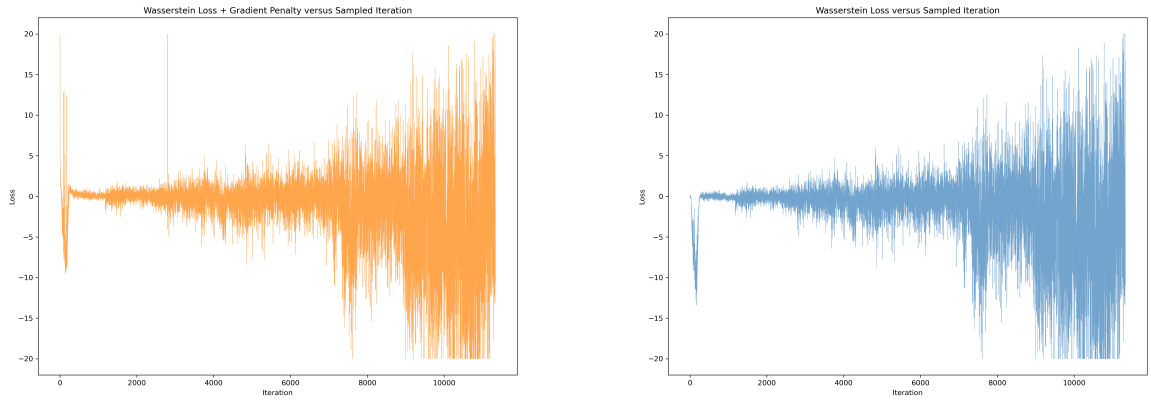

Figure 3.2: Wasserstein loss with (left) and without (right) gradient penalty term versus training time for the bidirectional substitution model.

starting point, but these results hint that a reduction in hidden unit size to reduce network complexity may have been more beneficial, along with other fine-tuning. The lack of gradient penalty spikes does reassure that smooth training is possible with this model.

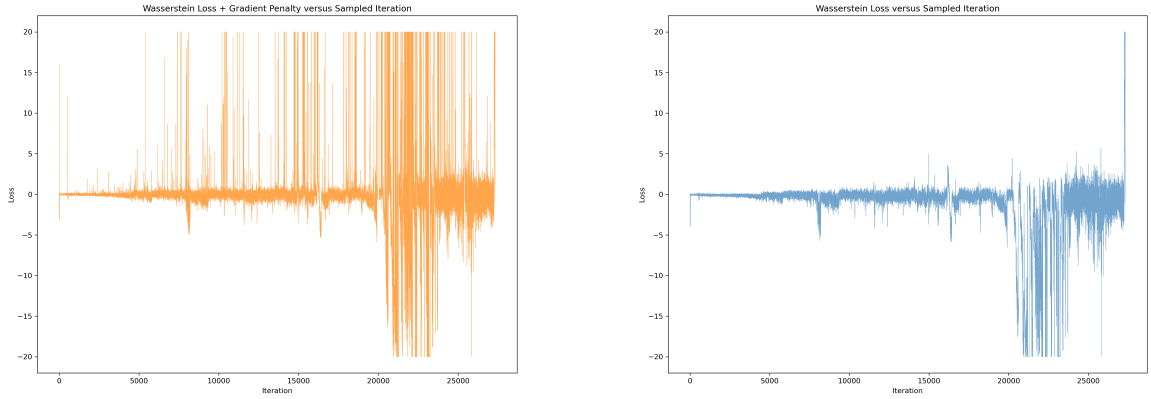

Figure 3.3: Wasserstein loss with (left) and without (right) gradient penalty term versus training time for the unidirectional indel model.

For the unidirectional indel model, the Wasserstein loss remained relatively stable over time with a noticeable mean around 0, until destabilization at the last third of training. At this point, there was much increased variance with the same previously mentioned implication. The presence of more frequent gradient penalty spikes even early

on suggests an abrupt training process, which is perhaps attributable to the large indel encoding size amplifying network complexity.

3.1.2 Kullback-Leibler Divergence Over Time

We present the Kullback-Leibler (KL) divergence results for each model, denoting the statistical distance of the generated probability distribution from the real one. See Figures 3.4, 3.5, and 3.6 for plots of the KL divergence versus sampled iterations for each model divided by stage of training. Divergence values closer to 0 are better. In each plot, we provide divergence results broken down by trinucleotide, dinucleotide, and single nucleotide mutation context probability distributions. We therefore consider a maximum context size of three in the analysis.

More precisely, the substitution model entails 4,096 ($4^3 \times 4^3$) trinucleotide, 256 ($4^2 \times 4^2$) dinucleotide, and 16 (4×4) single nucleotide mutation context probabilities. Meanwhile, the indel model involves 15,625 ($5^3 \times 5^3$) trinucleotide, 625 ($5^2 \times 5^2$) dinucleotide, and 25 (5×5) single nucleotide mutation context probabilities due to the addition of the gap character. For example, the probability of an $AAA \rightarrow AA-$ mutation belongs to the distribution of trinucleotide contexts in the indel model, while the probability of a $CG \rightarrow TG$ mutation belongs to the distribution of dinucleotide contexts in either model. Note that probabilities of conservation events such as $AAA \rightarrow AAA$ and $C \rightarrow C$ are also included in their respective mutation context distributions.

We sampled the generator models regularly throughout training to gauge progression, and report the results based on this arbitrary sampling frequency. Unlike in the Wasserstein loss plots, iteration numbers here do correspond exactly to training epochs. Each saved generator model was fed conditional ancestors from a reserved test set. See the next subsection for more detail on the composition of the test sets. The KL divergence values shown below were then obtained by analyzing the ensuing distribution of fake descendants and comparing with the real descendants. Training was stopped when the

divergence value did not improve overall in any mutation context size for at least three consecutive sampled time points. Each training stage is denoted to start at time 0, though subsequent stages are understood to have been run sequentially from previous ones. In general, divergence values for trinucleotide contexts are greater than for dinucleotide contexts, which are in turn greater than for single nucleotide contexts.

For the unidirectional substitution model (Fig. 3.4), KL divergence values in stage one and two broadly decreased over time across all context sizes. Stage three, however, demonstrated inconsistent fluctuations with divergence values often spiking before obtaining a small but short-lived overall improvement. Analysis of fake descendant output of the generator instances around the time of the biggest peaks in divergence values revealed an incredible excess in the number of substitutions compared to the real descendants.

Recall that descendants outputted by the generator are decoded using an argmax operation for each character. Only the decoded fake sequences are compared with the real descendants in the computation of KL divergence values. We suspect intuitively that adjustments to the generator's weight matrices as directed by the critic's feedback are incremental and ultimately result in modifications to all the softmax-ed values in a character's encoding. Therefore, it is plausible that the training process temporarily changes the argmax character to one that is incorrect according to the real data. Meanwhile, the correct character's value may be approaching the maximum and will overtake in subsequent iterations. Following this explanation, transient increases in divergence values signal that the generator is responding to critic feedback.

For the bidirectional substitution model (Fig. 3.5), KL divergence values in stage one and the first half of stage two were in decline over time across all context sizes. Fluctuations were apparent in the remainder of stage two and throughout stage three, with the same previously mentioned interpretation. The second half of stage three did not net an overall improvement in any context size. We hypothesize again that the addition of a second LSTM layer to the generator and critic was problematic for both networks. Recall

that the bidirectional generator and critic have double the number of trainable weights compared to their unidirectional counterparts. Perhaps the critic's feedback became less useful as training progressed, leading the generator to overshoot the convergence point. However, it is more probable that the generator and critic had yet to converge, and that much more training time may have led to an overall improvement in divergence value. In all cases, a reduction in network complexity through other hyperparameters would have been more effective.

For the unidirectional indel model (Fig. 3.6), KL divergence values in stage one and two decreased over time across all context sizes. Stage three involved a very slowly trending reduction in the first half of training, followed by intense destabilization in the latter half with no net improvement thereafter. Analysis of fake descendant output of the generator instances around the time of the peak in divergence values revealed an incredible excess in the number of substitutions, insertions, and deletions compared to the real descendants. We suppose that the significantly increased number of probabilities for the indel model as well as the large indel encoding size were the biggest hindrances to successful training. Any slight bias for a particular character in the argmax of the generator's output easily spiked the divergence value.

We anticipated that the generator instance with the closest Wasserstein loss to 0 in stage three would also minimize the sum of the KL divergence values over each context size. Unfortunately, we did not observe such a pattern. There was, similarly, no observed correlation between a generator instance experiencing a Wasserstein loss spike and there being a corresponding spike in divergence value for any context size. However, we did not test generator instances at very frequent intervals, so more granularity could reveal a pattern. We also ran a preliminary experiment to assess whether adding a second unidirectional LSTM layer to the unidirectional substitution and indel models would yield better divergence values. Early stage three results were disappointing, and the significantly increased computation time deterred further experimentation.

The process for selecting the best saved generator instance for each model variant was, primarily, based on picking the one minimizing the sum of the stage three KL divergence values over each context size. We also ran some preliminary output tests for each candidate generator instance, and considered if the mean and median number of mutations reported in these tests were broadly in line with the real data for the same sized sequence chunks. Another important component was to privilege instances that had successfully incorporated all mutation types and the appropriate probability orders of magnitude, even if they did not strictly minimize the divergence values. Following this method, the best stage three generator instance occurred at epoch 1,269 for the unidirectional substitution model, epoch 519 for the bidirectional substitution model, and epoch 1,499 for the unidirectional indel model. In the following subsections, we report results based only on the best saved instance of each model variant.

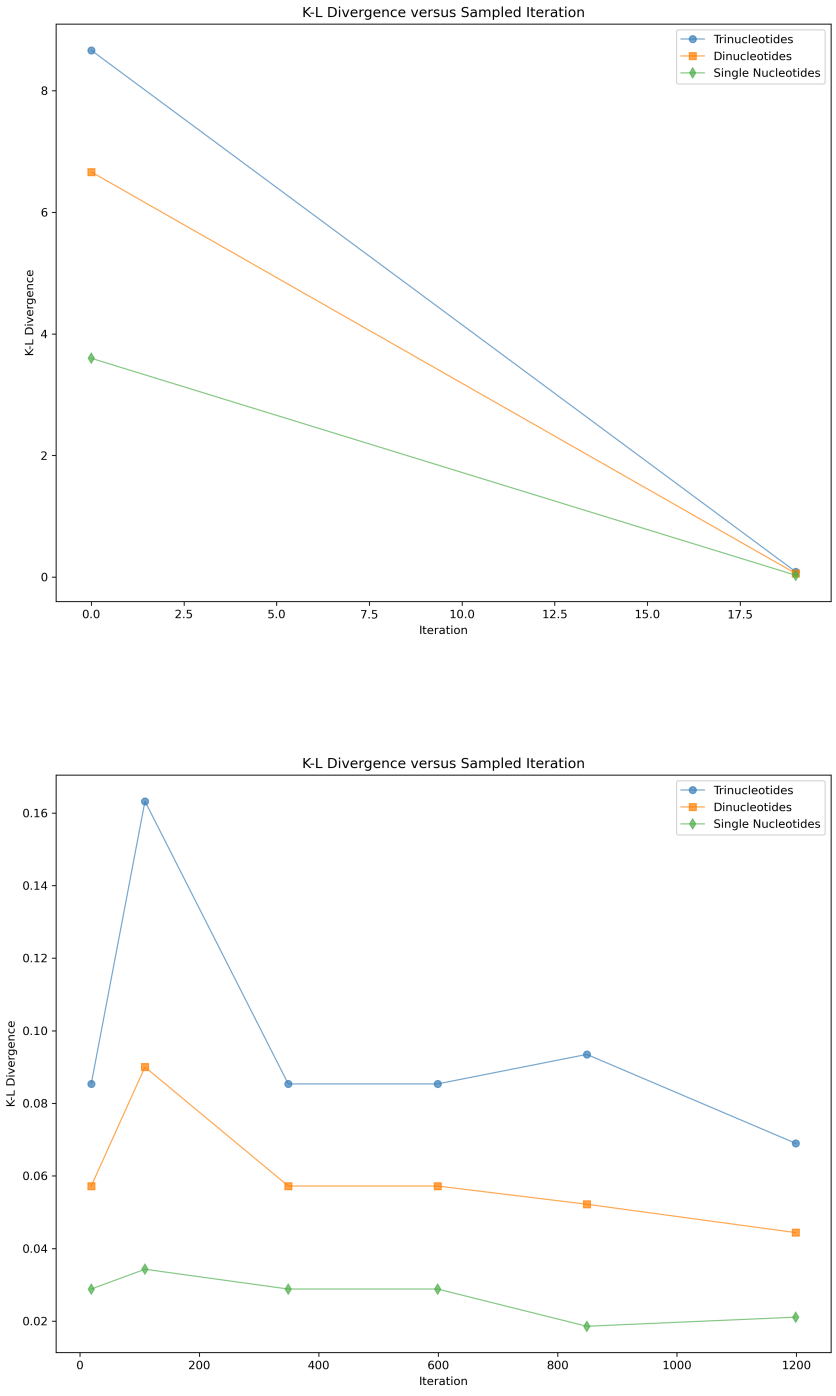

Figure 3.4: KL divergence versus training time for the unidirectional substitution model in stage one (top) and stage two (bottom). Note that data points at time 0 actually occur slightly after time 0.

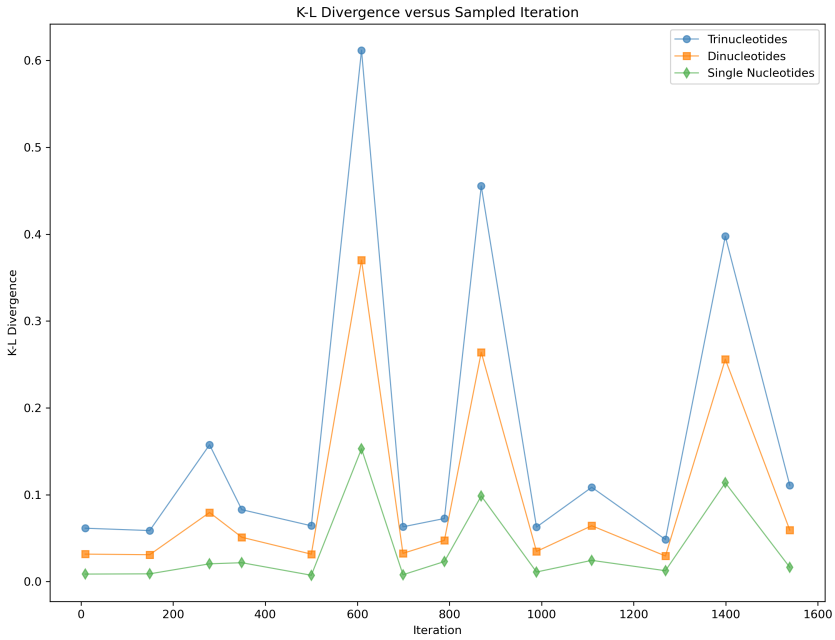

Figure 3.4: (cont.) KL divergence versus training time for the unidirectional substitution model in stage three. Note that data points at time 0 actually occur slightly after time 0.

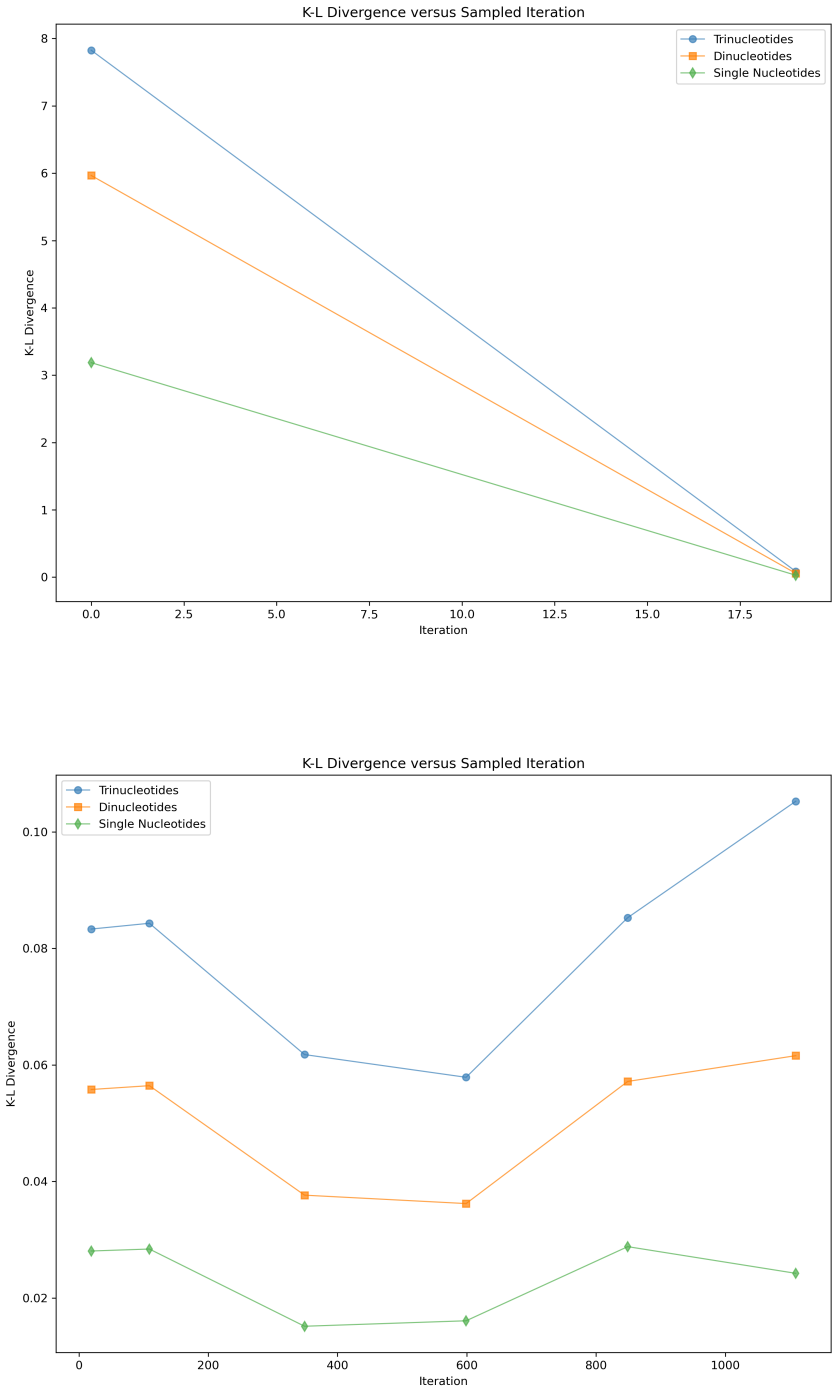

Figure 3.5: KL divergence versus training time for the bidirectional substitution model in stage one (top) and stage two (bottom). Note that data points at time 0 actually occur slightly after time 0.

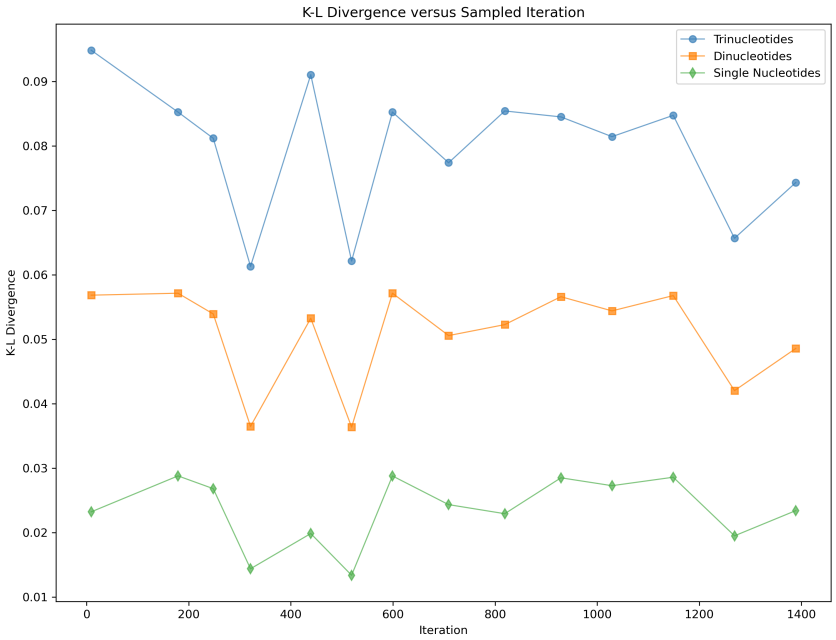

Figure 3.5: (cont.) KL divergence versus training time for the bidirectional substitution model in stage three. Note that data points at time 0 actually occur slightly after time 0.

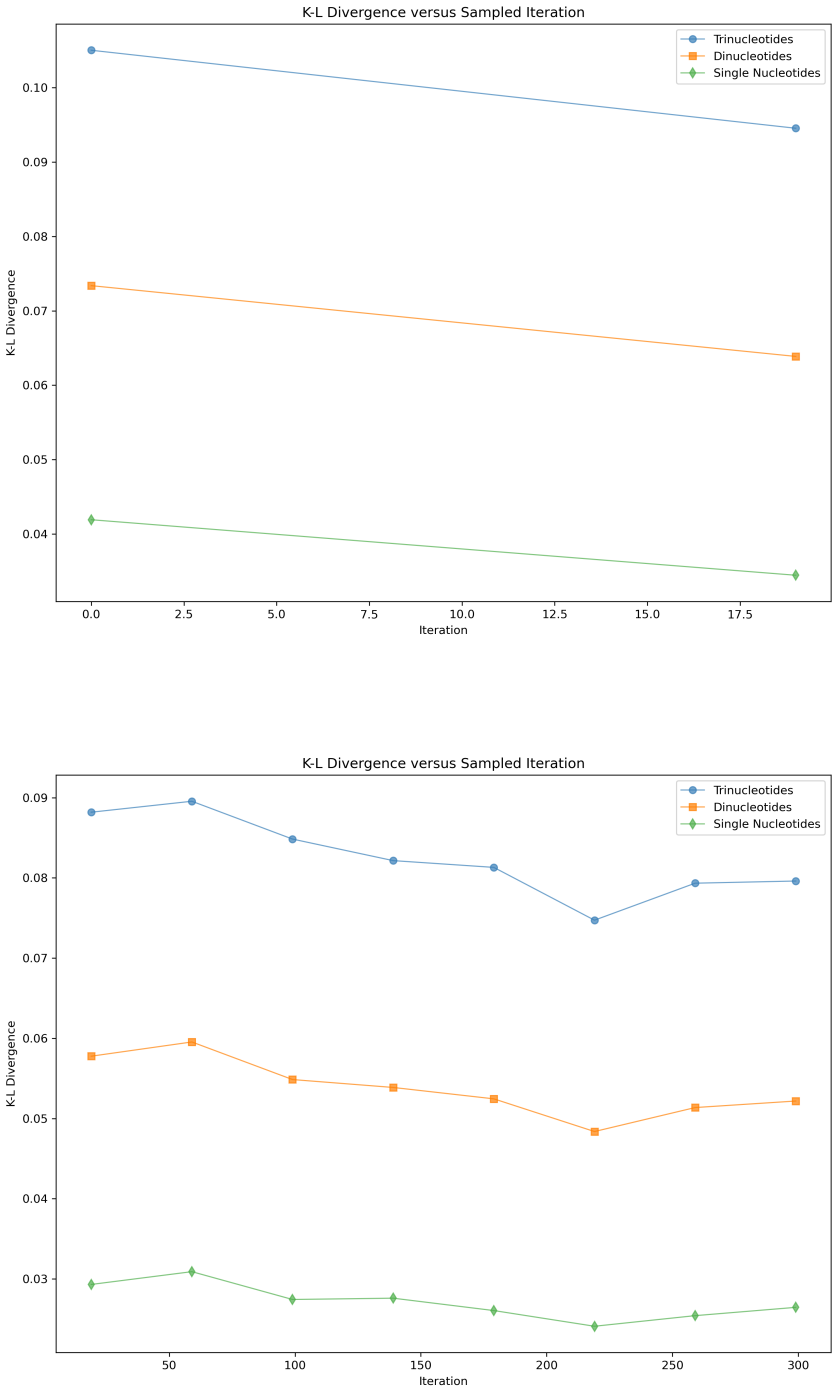

Figure 3.6: KL divergence versus training time for the unidirectional indel model in stage one (top) and stage two (bottom). Note that data points at time 0 actually occur slightly after time 0.

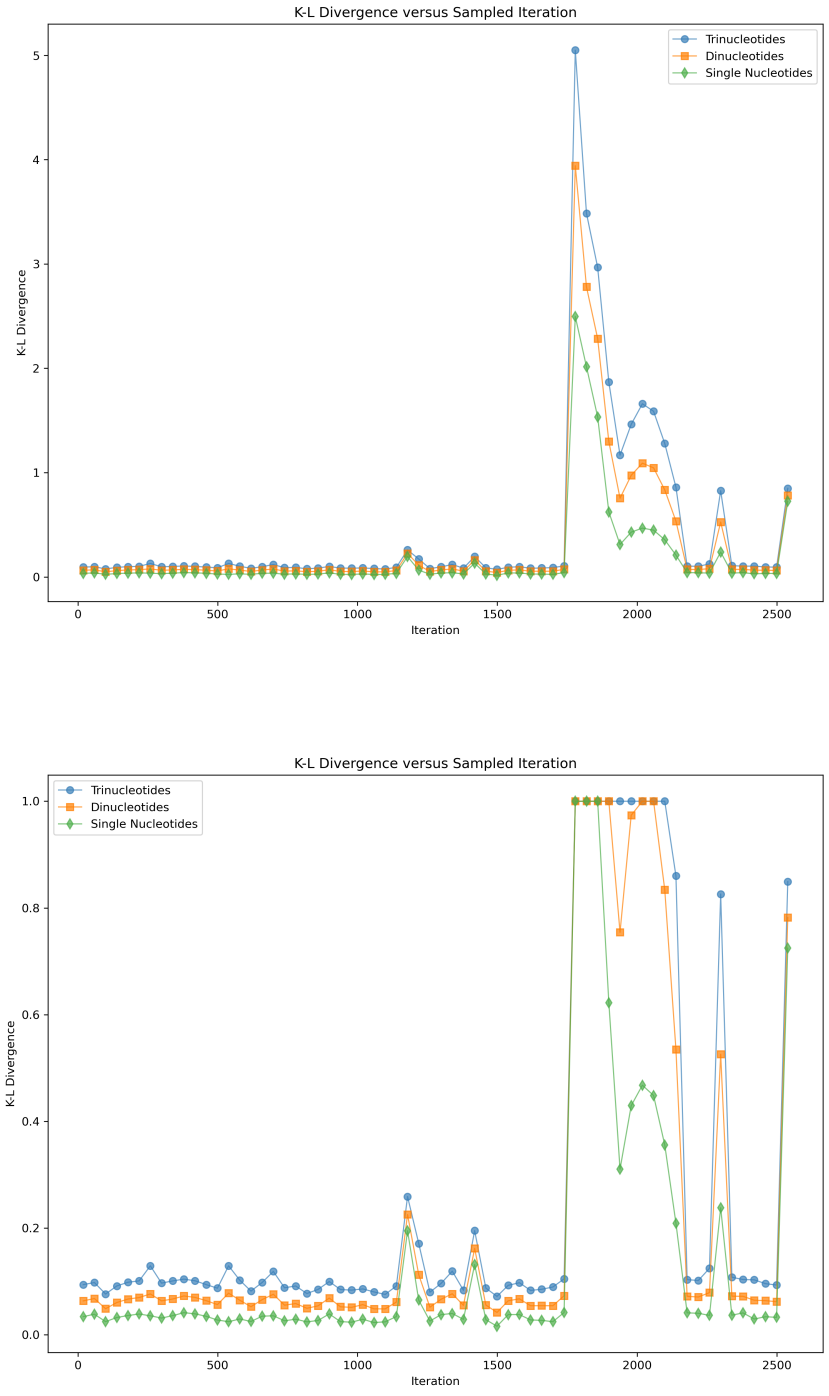

Figure 3.6: (cont.) KL divergence versus training time for the unidirectional indel model in stage three (top: unclipped; bottom: clipped to ≤ 1.0). Note that data points at time 0 actually occur slightly after time 0.

3.1.3 Interesting Probabilities

We report some relevant mutation probabilities following analysis of the real and fake data generated by each of the model variants. See Tables 3.1 and 3.2 for a side-by-side comparison of single nucleotide conservation, transition, transversion, CpG substitution, insertion, deletion, and replication slippage related deletion probabilities between each model where applicable.

The substitution model test set was composed of 705 conditional ancestors of length 15,000 that were each evolved independently three times via repeated random sampling of the generator latent space, giving 2,115 total descendants. Likewise, the indel model test set contained 1,000 conditional ancestors of length 1,500 that were also evolved three times, yielding 3,000 total descendants. Recall that, in each case, the test set comprises a reserved 10% of the initial cleaned data. Test set chunk size was selected to be of the same magnitude as the stage three training chunk size.

For the two substitution models (Tab. 3.1), conservation probabilities were generally in line with the real data. While A-T conservations tended to be overestimated, G-C conservations showed the opposite trend. We suspect that the generator model successfully learned the human genome's unequal base distribution favouring A-T over G-C content, but may have overcompensated and needed to be dialed back. Transition mutation probabilities were mostly of the correct magnitude, though the unidirectional model seemed to struggle with $C \rightarrow T$ transitions, while the bidirectional model had more difficulty with $A \rightarrow G$ transitions. However, transversion mutation probabilities expressed great variability, but were significantly underestimated overall compared to the real data. One single nucleotide mutation context in the unidirectional and three such contexts in the bidirectional model were never observed in the generated test data.

The unidirectional model struggled more with the CpG dinucleotide forward context than with the reverse. We hypothesize that the unidirectional generator architecture requires reading the entire CG dinucleotide forward context before it can propagate the

Probabilities		Real	Unidirectional	Bidirectional
Conservations (4):	$A \to A$	0.278	0.294	0.297
	$C \to C$	0.207	0.200	0.198
	$G \to G$	0.208	0.200	0.200
	$T \to T$	0.280	0.296	0.292
Transitions (4):	$A \to G$	5.57e-3	3.32e-3	7.44e-4
	$G \to A$	4.03e-3	1.86e-3	1.91e-3
	$C \to T$	3.91e-3	3.63e-4	2.89e-3
	$T \to C$	5.59e-3	3.74e-3	6.81e-3
Transversions (8)		8.68e-4 to 1.10e-3	2.08e-6 to 7.42e-5 *	3.15e-8 to 4.89e-5 †
$CG \to TG$ (forward)		1.06e-3	1.53e-5	9.10e-5
$CG \to CA$ (reverse)		1.09e-3	1.16e-4	2.40e-4

Table 3.1: Interesting probabilities for the fake data generated by the unidirectional and bidirectional substitution models, compared to the real gapless data. Dinucleotide probabilities are provided for the forward and reverse complement strands of the unusually mutagenic CpG context. Probabilities are rounded and may not sum to 1. * The $C \to G$ mutation context was not observed in the unidirectional model. † The $A \to T$, $T \to A$, and $C \to G$ mutation contexts were not observed in the bidirectional model.

information to future LSTM cells. However, future LSTM cells do not impact previous cells in this unidirectional model. By the time the generator has read the second nucleotide of the dinucleotide, there is no way to send the context information back one cell such that it outputs TG more frequently. With the reverse complement, meanwhile, the unidirectional generator architecture has an opportunity when it reads CG to mutate the second nucleotide of the dinucleotide, and therefore learns to output CA more often. On the other hand, the bidirectional model considers flanking context of the CG dinucleotide, which explains why the gap between the mutagenicity of the forward and reverse contexts is significantly narrowed. Nonetheless, the reverse complement remains closer to the ground truth even in this case.

For the unidirectional indel model (Tab. 3.2), conservation probabilities approached the real data with the same successful capture of the human genome's bias for A-T over

G-T content. A similar level of overcompensation was noted. Transition mutation probabilities were again mostly of the right magnitude, though the model expressed more trouble with $T \rightarrow C$ transitions. Transversion mutation probabilities were likewise varied and underestimated across the board. Two single nucleotide mutation contexts were never observed in the generated test data.

The same trend concerning the CpG dinucleotide mutation probability was observed in the indel model. The reverse complement context was closer to the ground truth, and we have the same previously described hypothesis to explain it given the unidirectional nature of this model. Insertions and deletions were surprisingly well captured across the board, apart from the $- \rightarrow G$ insertion context. Deletion probabilities in particular were remarkably close to their ground truth. Since deletions are on average about twice as common as insertions, we suppose that the model had more exposure to samples of the former during training to better match its probabilities. However, the relatively small absolute difference in frequency overall between insertions and deletions does counterbalance this idea.

Recall that deletions are more often found in the context of repeated identical nucleotides, as a result of the repair process following cases of DNA polymerase enzyme slippage during DNA replication. Since we limit our analysis to a context size of three, we look at deletions immediately following repeated dinucleotides. In this case, we observe that half of the deletions are of the correct magnitude and that the other half is about one magnitude smaller than expected. Recall that the indel training set had a maximum chunk size of 1,500, meaning that fewer than 1.5% of the sequences could be expected to have an occurrence of this deletion type. From this perspective of relative rarity, the performance of the indel model is not unsatisfactory.

It appears, based on this analysis, that the unidirectional substitution model successfully incorporated the most mutation types and achieved the most reasonable probability orders of magnitude across the board. This result is in line with our expectations given it is the simplest model operating on the easier dataset.

In all cases, we recognize that the stochastic nature of the training process and the significant impact of random restarts lends great variability to the descendants generated by a model variant, even given the same training configuration. It is possible that more rigorous testing of the same training configuration would have revealed closer probabilities, especially with regards to transitions and transversions. Nonetheless, time and resource limitations discourage this level of diligence. More detail on training effort will follow in Section 3.3. The trade-off between exploration and revision of a training configuration is ever so salient in this work.

Probabilities		Real	Unidirectional
Conservations * (4):	$A \to A$	0.274	0.293
	$C \to C$	0.204	0.194
	$G \to G$	0.205	0.194
	$T \to T$	0.276	0.297
Transitions (4):	$A \to G$	5.51e-3	3.49e-3
	$G \to A$	3.98e-3	3.67e-3
	$C \to T$	3.86e-3	1.13e-3
	$T \to C$	5.52e-3	1.85e-4
Transversions (8)		8.56e-4 to 1.09e-3	1.99e-6 to 1.02e-3 †
$CG \to TG$ (forward)		1.05e-3	2.97e-5
$CG \to CA$ (reverse)		1.07e-3	8.87e-5
Insertions (4):	$- \to A$	1.25e-3	2.29e-3
	$- \to C$	8.44e-4	2.57e-4
	$- \to G$	7.08e-4	0.000 †
	$- \to T$	1.26e-3	3.42e-4
Deletions (4):	$A \to -$	2.63e-3	2.88e-3
	$C \to -$	1.59e-3	1.82e-3
	$G \to -$	1.61e-3	1.70e-3
	$T \to -$	2.66e-3	2.83e-3
Deletions given AA, CC, GG, TT (16; replication slippage)		e-5 (14) to e-6 (2) magnitude	e-5 (8) to e-6 (7) to e-7 (1) magnitude

Table 3.2: Interesting probabilities for the fake data generated by the unidirectional indel model, compared to the real data containing gaps. Dinucleotide probabilities are provided for the forward and reverse complement strands of the unusually mutagenic CpG context. The first four of the 16 trinucleotide replication slippage related deletions, for example, are $AAA \to AA-$, $AAC \to AA-$, $AAG \to AA-$, and $AAT \to AA-$; the others follow analogously. For comparison, general trinucleotide deletions have e-7 magnitude probabilities in both real and fake data. Numbers in parentheses in the data columns denote the number of probabilities having the related magnitude. Probabilities are rounded and may not sum to 1. * The $- \to -$ conservation context correctly and necessarily has probability 0, since such aligned gaps were cleaned from the real data; the same follows for the analogous dinucleotide and trinucleotide gap conservation contexts. † The $T \to A$ and $- \to G$ mutation contexts were not observed in this model.

3.2 Constructed Examples

Some token evolution tests using artificial ancestors were performed to determine the stochastic diversity in output of the trained EvoSeqGAN unidirectional substitution model. Examples were created according to the following procedure. The first time point evolved an original ancestor to generate a fake descendant. The second time point took the freshly created fake descendant and evolved it again to produce a second fake descendant. In general, a generated descendant from time point $n-1$ was used as the conditional ancestor for time point n. This evolution sequence was repeated as many times as required to reach the desired final time point.

To showcase the diversity of model output at a certain time point, the entire evolution procedure was replicated independently 1,000 times. We present the results visually using sequence logos, where one or more characters may be found at each sequence position [66]. The height of the characters at each sequence position denotes the relative frequency of that character at that position over the 1,000 repetitions. The most common character is found at the top of the stack and the rest, if any, are shown in decreasing order of frequency. See Figure 3.7 for sequence logos depicting the diversity of nucleotide output at various evolution time points.

In the following reported results, a conditional ancestor of length 100 was used. This ancestor was chosen for its enhanced CpG content within an otherwise regular repeating context of four A, four C, four G, and four T. Three unique CG local contexts and eight different CG extended contexts are provided, where local context is defined as the three bases immediately flanking CG and extended context as beyond these three. The sequence is shown at the top of Figure 3.7.

Recall once more that generated descendants are decoded using an argmax operation for each character of the sequence. Incremental adjustments at each evolution time point thus do not manifest unless they succeed in becoming the maximum value. Globally, it is encouraging to see gradually more mutations appear as the timeline lengthens. Intrigu-

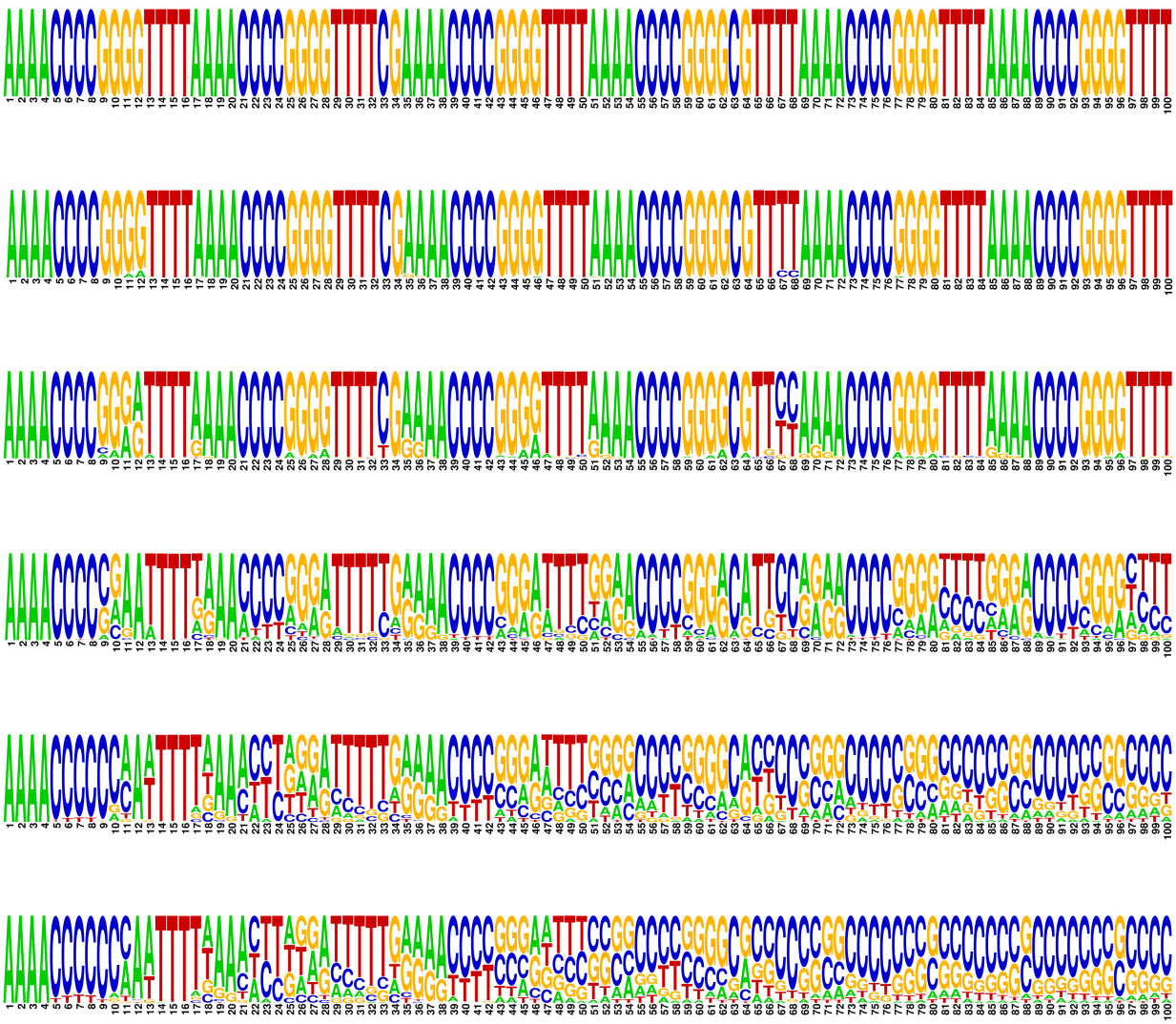

Figure 3.7: Evolution of an artificial ancestral sequence numerous times consecutively, repeated independently 1,000 times per time point, and presented as a sequence logo. From top: original ancestor, generated descendants at time 1, generated descendants at time 10, generated descendants at time 100, generated descendants at time 500, and generated descendants at time 1000. Note that each time point designates a complete evolution using the previous time point's generated descendant as the current ancestor to evolve.

ingly, there seemed to be a relative preference for C in the second half of the sequences during the late stages of evolution, which may indicate a bias in the model. The first string of A at positions 1-4 was consistently immune to mutation. This result is not so surprising given the unidirectional nature of the model that relies solely on previous characters to

provide context for the mutation of future characters. This same explanation supports the observation that mutations become more common overall at later sequence positions.

The first CG context was $CCCGGG$, repeated six times amid slightly different extended contexts. The variety of evolution output at time 1000 suggests that extended context is highly important. For instance, CG at positions 8-9 evolved the grand majority of the time to CC, while CG at positions 24-25 became TA with TT and TC as next most likely outputs. The CG at positions 42-43 and 58-59 seemed to be conserved about half the time or mutated to CC, CG, or TG the other half of the time.

The second CG context was $TTCGAA$, occurring only once at positions 33-34. Here, the most likely evolution output at time 1000 was TG by quite a margin, and CA only a very small fraction of the time in comparison. Meanwhile, the third CG context was $GGCGTT$, present only once at positions 63-64. This time, the most common evolution outcome was about equally split between a conservation and CA. These results support the context-dependent nature of EvoSeqGAN, which provides alternative CG evolution outputs despite the expected domination of $CG \to CA$ mutations that are, statistically for this model, about ten times more common than $CG \to TG$ mutations.

We also note the reversal or redirection of mutations at certain positions, which indicates the model has integrated how genetic drift acts on neutral mutations. For example, the G at position 11 had A as its most common evolution output at time 100, with only a small frequency of conservation otherwise. However by time 500, A had turned to C slightly less than half the time and, by time 1000, C overtook A as the most common evolution outcome. The A at position 70 likewise experienced a redirected output when it first had at time 100 only G as its alternative result, split about equally with a conservation. At time 500, the conservation's share gave way to C and, by time 1000, C had overtaken G as the most common outcome. In general, we note that a mutation never completely dominates the genetic pool of sequences. There always exists at least one other allele, even if rare, at positions that have mutated from the original ancestor, which aligns with the reality of neutral mutations in a population.

Overall, this example based on an entirely synthetic original ancestor is able to demonstrate that EvoSeqGAN incorporates desirable features of an evolution simulator: time-consistency, output diversity, context-dependency, and genetic drift modeling.

3.3 Running Time

EvoSeqGAN was trained on an Intel® Core™ i7-7700K CPU @ 3.70 GHz. Since the entire training process was CPU-bound due to the previously mentioned limitation with GPU-accelerated LSTM layers, the nature of the available GPU was irrelevant.

The multi-stage training pipeline for each model variant consisted of the following. All time indications are approximate averages. For the unidirectional and bidirectional substitution models presented above, 4 hours were spent on stage one training: for both, 500 bp chunks @ 80 minutes per 10 epochs for 20 epochs. Then, 9 days were spent on stage two training: respectively, 5,000 bp chunks @ 120 minutes per 10 epochs for 1,100 epochs, and 5,000 bp chunks @ 130 minutes per 10 epochs for 1,000 epochs. Finally, stage three training lasted 40 days: respectively, 15,000 bp chunks @ 360 minutes per 10 epochs for 1,600 epochs, and 15,000 bp chunks @ 410 minutes per 10 epochs for 1,440 epochs. In total for the substitution models, there were slightly more than 49 days of continuous training.

For the unidirectional indel model presented above, 5 hours were spent on stage one training: 50 bp chunks @ 90 minutes per 10 epochs for 36 epochs. Then, 2 days were spent on stage two training: 500 bp chunks @ 100 minutes per 10 epochs for 300 epochs. Finally, stage three training could only be pushed to last 30 days: 3,000 bp chunks @ 170 minutes per 10 epochs for 2,540 epochs. Beyond this point, gradients began exploding for an unclear reason and soon enough converted to NaN. In total for the indel model, there were slightly more than 32 days of continuous training.

A major training bottleneck was the batch loading of sequence chunks in and out of memory. Concerning the indel model's stage three training, the relatively short training time per epoch suggested the 3,000 bp length could be increased. In practice however, attempts to gradually increase the length resulted consistently in out of memory errors. A decision was made not to reduce the batch size to accommodate only slightly longer sequence lengths, for reasons already mentioned in the previous chapter.

Chapter 4

Discussion and Conclusion

This concluding chapter presents a summary of EvoSeqGAN and of its major implications in Section 4.1. Limitations of our approach and some future directions for improvement are discussed in Section 4.2.

4.1 Summary and Impact

DNA sequence evolution is at the heart of bioinformatics. Accurately modeling this process is a fundamental problem whose earliest solution attempts date back more than fifty years [11]. The known dependence of mutation rates on sequence context has complicated the task. The first models to tackle the new problem variant of context-dependent sequence evolution appeared at the turn of this century [27]. As a simplification, many of these models assume base substitutions to be the only mutation type. However, insertions and deletions are also significantly involved in context-dependent evolution. It is apparent that a successful sequence evolution model needs to account for all of these mutation types.

Traditional probabilistic modeling attempts to compute the mutation probabilities of a given nucleotide given a certain size of its flanking sequence context. Unfortunately, these models have an exponential algorithmic cost. As the context size increases, the sample sizes required to observe the increasingly rare mutation contexts also grow exponentially. EvoSeqGAN deviates from tradition by using a modern machine learning approach to capture context dependencies without explicit computation of conditional probabilities.

A highly accurate DNA mutation model integrating both substitutions and indels has several core applications in bioinformatics. In multiple sequence alignment, current standard approaches involve outputting maximum likelihood alignments given a scoring scheme. Under conventional context-dependent models that inform the provided scoring scheme, this method quickly reaches its useful limit in considered context size due to the exponential blow-up. EvoSeqGAN uses a generative adversarial network to capture the characteristics of a good alignment by simulating descendants that correctly align with an input ancestor. In turn, the generator's learned parameters and weights can be used as a context-dependent mutation model to inform the scoring scheme. The potential benefit of more accurate sequence alignments is the ability to identify homologous DNA sequence regions even in cases of low overall sequence similarity.

The problem of phylogenetic inference relies on accurately aligning DNA sequences of all the species in the given set. A robust mutation model improves the faithfulness of the multiple sequence alignment, which in turn implies an inferred phylogenetic tree closer to the ground truth. In ancestral genome reconstruction, DNA sequences of all the species in the provided set are related by a provided phylogenetic tree. An accurate mutation model ensures the suitability of the tree and minimizes errors in reconstructed ancestors. An important outcome common to all these applications is the ability to handle highly divergent species. More precisely, it permits the production of more complex phylogenetic trees and the prediction of increasingly distant ancestral DNA sequences.

EvoSeqGAN outputs interpretable sequences that can be compared with empirical data. If overall similarity to experimental observations is high for these DNA sequences,

the learned mutation models can be used in research contexts to anticipate the mutation of genomes in non-coding regions. Subsequent analysis may then elucidate existing mechanisms of evolution and reveal new ones. In addition, a user interface can even make EvoSeqGAN accessible as an evolution simulation tool to the layman population for educational use.

4.2 Limitations and Future Work

Overall, the EvoSeqGAN model variants discussed in this work all show promise. Our results indicate that the GAN framework succeeds in integrating short sequence context. In general, the more common a mutation type, the better EvoSeqGAN is able to model it. The main limitations relate to model accuracy, support of longer-range context, and training stability. There are several directions for improvement that maintain the current LSTM-based network architecture.

First, continued experimentation with hyperparameters could reveal a breakthrough. A global annealing schedule could be implemented that periodically reduces the Adam optimizer's base learning rate, permitting finer control to accelerate early training and to slow down in the end stage. This annealing schedule could be different for the critic and generator networks.

Another possibility is the use of smoothed vectors for the encoding of the real descendant data. For example with the substitution model, each character of the DNA sequence is currently one-hot encoded over five bits. Let x be a random value between 0.80 and 0.90. It could be beneficial to change to a fractional allocation that, for instance, assigns x to the on bit and a random share of $1 - x$ to each of the four off bits. The intuition here is that, at present, the critic may have an easy time differentiating between its input of one-hot real vectors and the softmax-ed fake generator output. Since this separation does not reflect the real world, it is not a desirable pattern to learn and does not send a useful training signal to the generator. At the same time, consider that the critic's LSTM

layer already has a sigmoid activation on three of its gates that transforms the one-hot real vectors. Thus, it is unclear if this regularization technique would be strictly helpful.

A slightly more involved alternative to the smoothed vectors is to exploit the Gumbel-Softmax trick to replace the usual softmax layer in the generator [67]. Recall that GANs have trouble outputting to the discrete space since they were initially conceived for continuous data. More specifically, a discrete output step in the generator, because it is not a differentiable function, kills backpropagation for both the critic and itself. The potential issue with the one-hot real descendant data being easily separated from the softmax-ed generated descendants remains. A naive option to eliminate the difference in presentation to the critic would be to simply take the argmax of the generator output. However, the argmax operation is classically not differentiable.

With the Gumbel-Softmax trick, a temperature hyperparameter is added to the conveniently differentiable softmax function. A low temperature can be used to obtain effectively one-hot vectors, while at high temperature the data vector approaches a uniform distribution. The temperature stays between 0 and 1 in practice, but can theoretically extend to infinity. The ideal value can be learned through repeated experimentation, or an annealing schedule can be defined.

If the goal is to maximize single nucleotide mutation accuracy, it could be beneficial to decrease the LSTM hidden unit dimensionality in both the critic and generator. This change would reduce each network's capacity to learn context dependencies, but would encourage a more equal consideration of sequence positions to improve the ability to capture single nucleotide mutations. Furthermore, it could help to add a custom term to the Wasserstein loss equation that discourages generator output values that are too close to 0 or 1. The intuition here is to promote continued neutral mutations, which would be unlikely with extreme sequence vectors that encourage fixation. The risk with entering into custom loss terms is that LSTM cells do not view sequence positions as humans do and neural networks are notoriously difficult to interpret. It is easy to be misled in this venture.

Based on results, the identified problem areas for EvoSeqGAN are threefold: training sample size requirements, network structure, and hardware. Unfortunately, the areas are interconnected such that challenges in one have a ripple effect on the others. On the flip side, improvements also go hand in hand so as to benefit all areas together.

To begin, memory limitations precluded the use of longer sequences in stage three of training for both the substitution and indel models. This restriction was most punishing for the latter model, stuck at only 3,000 bp sequence chunks, due to its large encoding size. The rarity of indels and specific mutations given larger context sizes made it such that the majority of training examples were too short to contain them. Consequently, model accuracy in this regard and how much flanking context could be considered during training was directly impacted.

Suppose then that available memory is not an obstacle. The sequence chunks used can be as large as desired to ensure there are examples of each mutation up to a certain predetermined context size. However, the LSTM-based underlying network structure means that the longer the sequences, the more cells are needed to represent them. The networks then get proportionately wider. Recall that a current software limitation prevents the efficient use of WGAN-GP with a multi-core GPU due to the incompatibility of higher order gradients. So, the networks are CPU-bound and this incredibly large amount of data is being batch-loaded in and out of memory.

The next issue is that the LSTM networks can only process one sequence at a time. Furthermore, future cells rely on propagated hidden state, so they cannot be assigned to different cores and computed before prior cells. To regain the use of a multi-core GPU to at least process multiple sequences in parallel across large batch sizes and to accelerate the linear algebra operations, the gradient penalty term must be avoided. An alternative could be to change from WGAN-GP to a spectral normalized WGAN [68]. The spectral normalization is another method to enforce the 1-Lipschitz constraint that is much lighter on memory than computing the gradient penalty. It can also be used with loss functions

other than Wasserstein loss and still stabilize training. However, the literature is not rich to describe the success of this approach on data other than images.

A more drastic change could be to forego the LSTM-based architecture entirely and focus on more recent self-attention mechanisms together with the stabilizing effect of spectral normalization [69]. Complete parallelism would be restored by removing the recursive requirement of LSTM networks and, in turn, the memory needs would be significantly lightened. In fact, the latest developments in GAN research are to incorporate transformers, a self-attention grid architecture free from other convolutional or recurrent neural network modules that minimizes memory usage [70]. A general pre-trained technique called Bidirectional Encoder Representations from Transformers (BERT) has become popular for natural language tasks [71]. However, very few have attempted to incorporate BERT within a GAN framework [72]. While these self-attention GAN models have achieved state-of-the-art results on images, there are scarce reports of applying these approaches to sequential data. More detail on self-attention may be found in the literature.

In conclusion, there are many partial solutions available to lighten the memory load, permit longer sequences for training, and reinstate the ability to train on a GPU. Piecing several of them together could be the key to unlocking better accuracy, accessing longer-range context, and improving training stability in the late stage. The intersection of evolutionary biology and the latest machine learning advances runs rife with opportunities. Capturing context-dependent mutations represents a significant challenge for the scientific community that, if overcome, has major impacts for sequence analysis, phylogenetics, and potentially human health. EvoSeqGAN is a proof of concept for our objective of developing a framework to learn, in an unsupervised manner, the correct parameters and weights involved in simulating DNA sequence evolution.

Bibliography

[1] E. Pettersson, J. Lundeberg, and A. Ahmadian, "Generations of sequencing technologies," *Genomics*, vol. 93(2), pp. 105–111, February 2009.

[2] M. Kimura, "Evolutionary Rate at the Molecular Level," *Nature*, vol. 217, pp. 624–626, February 1968.

[3] L. Duret, "Neutral Theory: The Null Hypothesis of Molecular Evolution," *Nature Education*, vol. 1(1), p. 218, 2008.

[4] M. Kimura, "The neutral theory of molecular evolution: a review of recent evidence," *Japanese Journal of Genetics*, vol. 66(4), pp. 367–386, August 1991.

[5] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, pp. 443–453, March 1970.

[6] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147(1), pp. 195–197, March 1981.

[7] E. W. Myers and W. Miller, "Optimal alignments in linear space," *Bioinformatics*, vol. 4(1), pp. 11–17, March 1988.

[8] P. Bonizzoni and G. D. Vedova, "The complexity of multiple sequence alignment with SP-score that is a metric," *Theoretical Computer Science*, vol. 259, pp. 63–79, May 2001.

[9] S. Clancy, "Genetic Mutation," *Nature Education*, vol. 1(1), p. 187, 2008.

[10] I. Keller, D. Bensasson, and R. A. Nichols, "Transition-Transversion Bias Is Not Universal: A Counter Example from Grasshopper Pseudogenes," *PLOS Genetics*, vol. 3, pp. 1–7, February 2008.

[11] T. H. Jukes and C. R. Cantor, "Evolution of Protein Molecules," *Mammalian Protein Metabolism*, vol. 3, pp. 21–132, 1969.

[12] M. Kimura, "A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences," *Journal of Molecular Evolution*, vol. 16, pp. 111–120, June 1980.

[13] J. Felsenstein, "Evolutionary trees from DNA sequences: A maximum likelihood approach," *Journal of Molecular Evolution*, vol. 17, pp. 368–376, November 1981.

[14] M. Hasegawa, H. Kishino, and T.-A. Yano, "Dating of the human-ape splitting by a molecular clock of mitochondrial DNA," *Journal of Molecular Evolution*, vol. 22, pp. 160–174, October 1985.

[15] S. Tavaré, "Some probabilistic and statistical problems in the analysis of DNA sequences," *Lectures on Mathematics in the Life Sciences*, vol. 17(2), pp. 57–86, December 1986.

[16] J. L. Thorne and H. Kishino, "An Evolutionary Model for Maximum Likelihood Alignment of DNA Sequences," *Journal of Molecular Evolution*, vol. 33(2), pp. 114–124, August 1991.

[17] M. P. Simmons and H. Ochoterena, "Gaps as Characters in Sequence-Based Phylogenetic Analyses," *Systematic Biology*, vol. 49(2), pp. 369–381, June 2000.

[18] G. McGuire, M. C. Denham, and D. J. Balding, "Models of Sequence Evolution for DNA Sequences Containing Gaps," *Molecular Biology and Evolution*, vol. 18(4), pp. 481–490, April 2001.

[19] T. Nishimaki and K. Sato, "An Extension of the Kimura Two-Parameter Model to the Natural Evolutionary Process," *Journal of Molecular Evolution*, vol. 87, pp. 60–67, January 2019.

[20] D. Lim and M. Blanchette, "EvoLSTM: context-dependent models of sequence evolution using a sequence-to-sequence LSTM," *Bioinformatics*, vol. 36, pp. i353–i361, July 2020.

[21] J. Watson and F. Crick, "Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid," *Nature*, vol. 171, pp. 737–738, April 1953.

[22] D. G. Hwang and P. Green, "Bayesian Markov chain Monte Carlo sequence analysis reveals varying neutral substitution patterns in mammalian evolution," *Proceedings of the National Academy of Sciences*, vol. 101(39), pp. 13994–14001, September 2004.

[23] J. Surrallés, M. J. Ramírez, R. Marcos, A. T. Natarajan, and L. H. F. Mullenders, "Clusters of transcription-coupled repair in the human genome," *Proceedings of the National Academy of Sciences*, vol. 99(16), pp. 10571–10574, August 2002.

[24] Z. Feng, W. Hu, E. Komissarova, A. Pao, M.-C. Hung, G. M. Adair, and M. shong Tang, "Transcription-coupled DNA repair is genomic context-dependent," *Journal of Biological Chemistry*, vol. 277(15), pp. 12777–12783, April 2002.

[25] P. W. Messer and P. F. Arndt, "The Majority of Recent Short DNA Insertions in the Human Genome Are Tandem Duplications," *Molecular Biology and Evolution*, vol. 24(5), pp. 1190–1197, May 2007.

[26] A. Tanay and E. D. Siggia, "Sequence context affects the rate of short insertions and deletions in flies and primates," *Genome Biology*, vol. 9, p. R37, February 2008.

[27] J. L. Jensen and A.-M. K. Pedersen, "Probabilistic Models of DNA Sequence Evolution with Context Dependent Rates of Substitution," *Advances in Applied Probability*, vol. 32(2), pp. 499–517, June 2000.

[28] P. F. Arndt, C. B. Burge, and T. Hwa, "DNA sequence evolution with neighbor-dependent mutation," *Journal of Computational Biology*, vol. 10(3), pp. 313–322, July 2004.

[29] A. Siepel and D. Haussler, "Phylogenetic Estimation of Context-Dependent Substitution Rates by Maximum Likelihood," *Molecular Biology and Evolution*, vol. 21(3), pp. 468–488, March 2004.

[30] V. Aggarwala and B. F. Voight, "An expanded sequence context model broadly explains variability in polymorphism levels across the human genome," *Nature Genetics*, vol. 48, pp. 349–355, April 2016.

[31] Y. Zhu, T. Neeman, V. B. Yap, and G. A. Huttley, "Statistical Methods for Identifying Sequence Motifs Affecting Point Mutations," *Genetics*, vol. 205(2), pp. 843–856, February 2017.

[32] G. Ling, D. Miller, R. Nielsen, and A. Stern, "A Bayesian Framework for Inferring the Influence of Sequence Context on Point Mutations," *Molecular Biology Evolution*, vol. 37(3), pp. 893–903, March 2020.

[33] R. C. Aikens, K. E. Johnson, and B. F. Voight, "Signals of Variation in Human Mutation Rate at Multiple Levels of Sequence Context," *Molecular Biology and Evolution*, vol. 36(5), pp. 955–965, May 2019.

[34] A. Jariani, C. Warth, K. Deforche, P. Libin, A. J. Drummond, A. Rambaut, F. A. M. IV, and K. Theys, "SANTA-SIM: simulating viral sequence evolution dynamics under selection and recombination," *Virus Evolution*, vol. 5(1), January 2019.

[35] L. Appleton and J. Mackie, "Using Digital Organism Evolutionary Software in the Classroom," *The American Biology Teacher*, vol. 81(1), pp. 12–17, January 2019.

[36] J. Stoye, D. Evers, and F. Meyer, "Rose: generating sequence families," *Bioinformatics*, vol. 14(2), pp. 157–163, March 1998.

[37] W. Fletcher and Z. Yang, "INDELible: A Flexible Simulator of Biological Sequence Evolution," *Molecular Biology and Evolution*, vol. 26(8), pp. 1879–1888, August 2009.

[38] R. C. Edgar, G. Asimenos, S. Batzoglou, and A. Sidow, "Evolver: a whole-genome sequence evolution simulator." `http://www.drive5.com/evolver/`. Online; accessed 10-January-2021.

[39] M. Chatzou, C. Magis, J.-M. Chang, C. Kemena, G. Bussotti, I. Erb, and C. Notredame, " Multiple sequence alignment modeling: methods and applications," *Briefings in Bioinformatics*, vol. 17(6), pp. 1009–1023, November 2016.

[40] G. Hickey and M. Blanchette, "A Probabilistic Model for Sequence Alignment with Context-Sensitive Indels," *Journal of Computational Biology*, vol. 18(11), pp. 1449–1464, November 2011.

[41] R. K. Ramakrishnan, J. Singh, and M. Blanchette, "RLALIGN: A Reinforcement Learning Approach for Multiple Sequence Alignment," *2018 IEEE 18th International Conference on Bioinformatics and Bioengineering (BIBE)*, pp. 61–66, 2018.

[42] L. S. Kubatko, "Inference of Phylogenetic Trees," *Tutorials in Mathematical Biosciences IV: Evolution and Ecology*, vol. 1922, pp. 1–38, 2008.

[43] B. Chor and T. Tuller, "Maximum likelihood of evolutionary trees: hardness and approximation," *Bioinformatics*, vol. 21, pp. i97–i106, June 2005.

[44] M. Blanchette, A. B. Diallo, E. D. Green, W. Miller, and D. Haussler, "Computational reconstruction of ancestral DNA sequences," *Methods in Molecular Biology*, vol. 422, pp. 171–184, 2008.

[45] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[46] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5(4), pp. 115–133, December 1943.

[47] L. J. Lancashire, C. Lemetre, and G. R. Ball, "An introduction to artificial neural networks in bioinformatics—application to complex microarray and mass spectrometry datasets in cancer studies," *Briefings in Bioinformatics*, vol. 10(3), pp. 315–329, May 2009.

[48] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9(8), pp. 1735–1780, November 1997.

[49] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[50] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv preprint arXiv:1701.07875*, 2017.

[51] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," *arXiv preprint arXiv:1704.00028*, 2017.

[52] N. Killoran, L. J. Lee, A. Delong, D. Duvenaud, and B. J. Frey, "Generating and designing DNA with deep generative models," *arXiv preprint arXiv:1712.06148*, 2017.

[53] A. Ghahramani, F. M. Watt, and N. M. Luscombe, "Generative adversarial networks simulate gene expression and predict perturbations in single cells," *bioRxiv preprint bioRxiv:262501*, 2018.

[54] X. Wang, K. G. Dizaji, and H. Huang, "Conditional generative adversarial network for gene expression inference," *Bioinformatics*, vol. 34(17), pp. i603–611, September 2018.

[55] A. Gupta and J. Zou, "Feedback GAN for DNA optimizes protein functions," *Nature Machine Intelligence*, vol. 1, pp. 105–111, February 2019.

[56] D. S. Berman, C. Howser, T. Mehoke, and J. D. Evans, "MutaGAN: A Seq2seq GAN Framework to Predict Mutations of Evolving Protein Populations," *arXiv preprint arXiv:2008.11790*, 2020.

[57] B. Yelmen, A. Decelle, L. Ongaro, D. Marnetto, C. Tallec, F. Montinaro, C. Furtlehner, L. Pagani, and F. Jay, "Creating artificial human genomes using generative neural networks," *PLOS Genetics*, vol. 17, pp. 1–22, 02 2021.

[58] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[59] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6629–6640, December 2017.

[60] M. Blanchette, E. D. Green, W. Miller, and D. Haussler, "Reconstructing large regions of an ancestral mammalian genome in silico," *Genome Research*, vol. 14(12), pp. 2412–2423, December 2004.

[61] A. B. Diallo, V. Makarenkov, and M. Blanchette, "Ancestors 1.0: a web server for ancestral sequence reconstruction," *Bioinformatics*, vol. 26(1), pp. 130–131, January 2010.

[62] W. J. Kent and D. Haussler, "Ucsc genome browser." `http://genome.ucsc.edu/index.html`.

[63] W. J. Kent and D. Haussler, "Ucsc genome browser phylogenetic tree." `http://genome.ucsc.edu/images/phylo/hg19_100way.png`.

[64] F. Chollet *et al.*, "Keras." `https://github.com/fchollet/keras`, 2015.

[65] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous systems." `https://www.tensorflow.org/`, 2015.

[66] G. E. Crooks, G. Hon, J.-M. Chaandonia, and S. E. Brenner, "WebLogo: a sequence logo generator ," *Genome Research*, vol. 14(6), pp. 1188–1190, June 2004.

[67] E. Jang, S. Gu, and B. Poole, "Categorical Reparameterization with Gumbel-Softmax," *arXiv preprint arXiv:1611.01144*, 2017.

[68] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral Normalization for Generative Adversarial Networks," *arXiv preprint arXiv:1802.05957*, 2018.

[69] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-Attention Generative Adversarial Networks," *arXiv preprint arXiv:1805.08318*, 2018.

[70] Y. Jiang, S. Chang, and Z. Wang, "TransGAN: Two Pure Transformers Can Make One Strong GAN, and That Can Scale Up," *arXiv preprint arXiv:2102.07074*, 2021.

[71] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[72] D. Croce, G. Castellucci, and R. Basili, "GAN-BERT: Generative Adversarial Learning for Robust Text Classification with a Bunch of Labeled Examples," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2114–2119, July 2020.