SIMULATING REAL EVOLVABILITY
CS 229BR: BIOLOGY AND COMPLEXITY, FINAL PROJECT

AMIR SHANEHSAZZADEH

Abstract. The goal of this project is to simulate and analyze the Wide-Scale Random Noise (WSRN) algorithm described in Paul Valiants's paper "Evolvability of Real Functions" [3]. The paper introduces the real evolvability framework and proves that a WSRN mutator can evolve, with respect to the L_2-squared loss function, polynomials from $\mathbb{R}^n$ to $\mathbb{R}^m$ with coefficients bounded by r to accuracy ϵ in Poly $\left(n, m, \frac{1}{\epsilon}\right)$ number of generations with Poly (n, m, r) children per generation. However, in the paper they do not provide any analysis as to what exactly the number of generations is with respect to the above parameters. Understanding the growth of the number of generations is important to determining the feasibility of this algorithm for evolution in the biological sense. In this paper we present the results of simulations of the algorithm, specifically empirical estimates showing that for fixed n, m and sufficiently small error ϵ, the growth of the number of generations required for evolution is $\Theta\left(\log\left(\frac{1}{\epsilon}\right)\right)$. We show that this relationship holds under slight variation of the domains, ranges, and degrees of the polynomials analyzed as well as variation of the bounds on the coefficients. Furthermore, we alter the number of children per generation to see the effects on the number of generations required for evolution. For any associated code and data see here.

1. Introduction

From an algorithmic perspective, the process of evolution is still quite a mystery. An evolution algorithm must evolve whatever feature is necessary while also being biologically feasible (i.e. tractable in terms of time and memory requirements). Naturally, Darwin's theory of natural selection can be viewed as a sort of optimization problem with individual organisms possessing their own functions that determine their phenotypes and their behaviors. The organisms strive to evolve their functions so as to minimize loss with respect to an optimal *fitness function*, to which they have black box access to.

Early work in this field was done by L. Valiant in 2009, where the Boolean Evolvability Model was introduced [2]. Here the functions are defined with domain $\{0,1\}^n$ and codomain $\{-1,1\}$. Under this model a binary string maps to $\{-1,1\}$ to indicate a negative or a positive response. If we let f be the *fitness function* and h be the function of some organism then the organism's fitness with respect to some distribution D on $\{0,1\}^n$ is defined to be

$$\mathbb{E}_{x\sim D}\left[f(x)h(x)\right].$$

Intuitively, this makes sense as the function of a "fit" organism should agree with f with high probability, whereas that of an "unfit" organism should disagree with f.

The generalization of this model to functions $\mathbb{R}^n \to \mathbb{R}^m$ is a more recent breakthrough. This generalization is a clear superset of the Boolean Evolvability Model and provides a more biologically realistic perspective on the problem of algorithmic evolution since many biological processes rely on continuous probability distributions. One might ask why we cannot view evolution of a polynomial from $\mathbb{R}^n \to \mathbb{R}^m$ as the evolution of m independent polynomials from $\mathbb{R}^n \to \mathbb{R}$. The reason for this, as P. Valiant describes in [3], is because we cannot assume independence. This independence assumption would lead us to think that each of our legs evolved independently from each other instead of at the same time, which is quite an unnatural thought.

2. The Real Evolvability Model

Here we present definitions of the Real Evolvability model adapted from P. Valiant's paper. These definitions will be referenced frequently throughout the remainder of the paper.

Definition 2.1. (P. Valiant [3] Def. 2.1) The (n, m)-dimensional evolvability model is a model of the evolution of a hypothesis $h : \mathbb{R}^n \to \mathbb{R}^m$ towards a target $f : \mathbb{R}^n \to \mathbb{R}^m$. The target and hypothesis are evaluated using samples drawn from some distribution D over the domain $\mathbb{R}^n$ where $D \in \mathcal{D}$ with $\mathcal{D}$ a set of such distributions. We consider the evolution of a set of target functions f, denoted C, which will be called a *concept class.*

Definition 2.2. (P. Valiant [3] Def. 2.2) Under the above model a *loss function* is a function $L : \mathbb{R}^m \times \mathbb{R}^m \to [0, \infty)$ with $L(x, y) = 0$ if and only if $x = y$.

We will consider two types of loss functions, those that are convex and those that are star-convex.

Definition 2.3. A function $f : \mathbb{R}^n \to \mathbb{R}$ is called convex if for all $x_1, x_2 \in \mathbb{R}^n$ and $t \in [0, 1]$

$$f(tx_1 + (1-t)x_2) \le tf(x_1) + (1-t)f(x_2).$$

Date: May 13, 2019.

Definition 2.4. (P.Valiant et al. [1]) A function $g : \mathbb{R}^n \to \mathbb{R}$ is called star-convex if there exists an $x^* \in \mathbb{R}^n$ such that for all $x \in \mathbb{R}^n$ and $t \in [0,1]$

$$g(tx^* + (1-t)x) \leq tg(x^*) + (1-t)g(x).$$

Definition 2.5. The L^p loss function for $p > 0$ is defined as $L^p : \mathbb{R}^n \to \mathbb{R}$ such that

$$L^p(x_1, ..., x_n) = |x_1|^p + \cdots + |x_n|^p.$$

We see that the loss function L^p is convex for $p \geq 1$ and star-convex for $p < 1$.

Definition 2.6. (P. Valiant [3] Def. 2.3) Given a loss function L, hypothesis h, and target f the *performance* of h relative to a distribution D over $\mathbb{R}^n$ is defined to be

$$L\text{Perf}_{f,D}(h) = \mathbb{E}_{x\sim D}[L(f(x), h(x))].$$

For a positive integer s, the s-sample *empirical performance* can be defined by drawing s independent draws from D denoted $x_1, ..., x_s \sim D$:

$$L\text{Perf}^s_{f,D} = \frac{1}{s}\sum_{i=1}^{s} L(f(x_i), h(x_i)).$$

Definition 2.7. (P. Valiant [3] Def. 2.4) Given $\epsilon > 0$ a *mutation algorithm* is a pair (R, M) where

(1) R is a set of hypotheses $\mathbb{R}^n \to \mathbb{R}^m$, where each hypothesis h can be represented by a string of finite length.
(2) M is a randomized $\text{Poly}(n, m, \frac{1}{\epsilon})$ time Turing machine that, given inputs $r \in R$ and $\frac{1}{\epsilon}$ outputs $r_1 \in R$ with probability $\text{Pr}_A(r, r_1)$. The *neighborhood* of r for ϵ is defined as the set of representations that can be output by $M(r, \epsilon)$ and is denoted as $\text{Neigh}_A(r, \epsilon)$.

Definition 2.8. (P. Valiant [3] Def. 2.5) Here we define our *selection rule*. Given a loss function L, tolerance t, candidate pool size p, and sample size s the *selection rule* $\text{SelNB}[L, t, p, s]$ is the algorithm that for any function f, distribution D, mutation algorithm $A = (R, M)$, representation $r \in R$, and accuracy ϵ, $\text{SelNB}[L, t, p, s](f, D, A, r)$ outputs a random variable that takes a value r_1 determined as follows: First, run $M(r, \epsilon)$ p times and let Z be the set of representations obtained. For $r' \in Z$, let $\text{Pr}_Z(r')$ be the relative frequency with which r' was generated among the p observed representations. For each $r' \in Z \cup \{r\}$ compute and empirical value of performance $v(r') \leftarrow L\text{Perf}^s_{f,D}(r')$. Let $\text{Bene}(Z) = \{r' \in Z : v(r') \leq v(r) - t\}$ and $\text{Neut}(Z) = \{r' \in Z : |v(r') - v(r)| < t\}$.

(1) If $\text{Bene}(Z) \neq \emptyset$: return a random $r_1 \in \text{Bene}(Z)$ with distribution according to the probabilities Pr_Z.
(2) If $\text{Bene}(Z) = \emptyset$ and $\text{Neut(Z)} \neq \emptyset$: return a random $r_1 \in \text{Neut(Z)}$ with distribution according to the probabilities Pr_Z.
(3) If $\text{Bene}(Z) = \text{Neut}(Z) = \emptyset$: return r.

Regarding the selection rule, while it may at first seem to be optimal to select the mutation that has highest overall fitness (in our case lowest expected loss), this is an inherently biased approach that leads to, albeit minimized expected loss, but high variance. A more natural approach that is better aligned with Darwin's theory of natural selection is to randomly select from the set of beneficial mutations (assuming they exist) or the set of neutral mutations (again assuming they exist) using a probability distribution based on the likelihood of the mutations occurring.

Definition 2.9. (P. Valiant [3] Def. 2.6) Here we define *evolvability*. A concept class C, set of distributions $\mathcal{D}$, and loss function L are said to be *evolvable* if there exists a mutation algorithm $A = (R, M)$, polynomials $p(n, m, \frac{1}{\epsilon})$, $s(n, m, \frac{1}{\epsilon})$, a tolerance $t(n, m, \frac{1}{\epsilon})$ whose inverse is poly-bounded, and a polynomial number of generations $g(n, m, \frac{1}{\epsilon})$ such that for all n, m, target functions $f \in C$, distributions $D \in \mathcal{D}$, $\epsilon > 0$, and any initial genome $r_0 \in R$, and any number of generations $g' \geq g$, with probability at least $1-\epsilon$, the random sequence defined by $r_i \leftarrow \text{SelNB}[L, t, p, s](f, D, A, r_{i-1})$ will have $L\text{Perf}_{f,D}(r_{g'}) \leq \epsilon$, namely, the mechanism of evolution will yield an almost-optimal genome in a polynomial number of generations, g.

3. Wide-Scale Random Noise (WSRN)

Here we present the definitions of WSRN and the WSRN mutator from P. Valiant's paper [3] as well as their main result regarding this algorithm. The proof of the result is deferred to the paper. We will later analyze what the result means in an empirical sense.

Definition 3.1. (P. Valiant [3] Def. 3.1) Given lower and upper bounds (l, u) and a dimension k the wide-scale random noise selects ρ uniformly randomly from the the interval $[\log_2 l, \log_2 u]$ and returns 2^ρ times a randomly chosen element of the k-dimensional unit ball.

Definition 3.2. (P. Valiant [3] Def. 3.1) For degree d polynomials from $\mathbb{R}^n \to \mathbb{R}^m$, the WSRN mutation algorithm $A = (R, M)$, parameterized by (l, u), is defined so that:

(1) R is a representation of such a polynomial via the vector of its $k = m \cdot \binom{n+k}{n}$ coefficients.
(2) M acts by generating k-dimensional WSRN, parameterized by (l, u), and adding it to the input vector R.

Theorem 3.3. *(P. Valiant [3] Thm. 3.3) Given any positive integer k there exists for any $r \in \mathbb{R}$ bounds (l, u) and an integer $c = Poly(n, m, r)$ such that the WSRN mutator with parameters (l, u) and c children per generation evolves the class of degree $\leq k$ polynomials from $\mathbb{R}^n \to \mathbb{R}^m$ with coefficients at most r over the set of all distributions on the n-dimensional radius r ball, with respect to the L_2-squared (L^2) loss function.*

4. Simulation Methods

Here we explain the methods used to simulate the algorithm.

4.1. Implementation.

The code used to perform the simulations is written in Python 3. A polynomial is represented using nested arrays with individual sub-arrays representing terms in the polynomial. For example, if we consider a polynomial with domain $\mathbb{R}^2$ and input variables x_1, x_2 then the term $[c, [a, b]]$ corresponds to the term $c \cdot x_1^a x_2^b$. The array $[[c_1, [a_1, b_1]], [c_2, [a_2, b_2]], ..., [c_r, [a_r, b_r]]]$ corresponds to the polynomial

$$f(x_1, x_2) = c_1 \cdot x_1^{a_1} x_2^{b_1} + c_2 \cdot x_1^{a_2} x_2^{b_2} + \cdots + c_r \cdot x_1^{a_r} x_2^{b_r}.$$

If we consider polynomials with domain $\mathbb{R}^n$ that have degree at most k then under our framework we must compute all sequences of non-negative integers $d_1, d_2, ..., d_n$ such that $d_1 + d_2 + \cdots d_n \leq k$. We can use a brute-force solution to do this in $O(k^n)$ time, and while this may seem like an impractical solution it is relatively fast for $k \leq 10$ and $n \leq 6$. We actually make the assumption that $n \leq 6$ and $k \leq 10$ as parameters larger than this become biologically in-feasible.

Using the fact that the number of non-negative integer solutions to $d_1 + d_2 + \cdots + d_n = k$ is $\binom{n+k}{k}$, so we can thus store a polynomial from $\mathbb{R}^n \to \mathbb{R}^m$ with degree at most k using an array of

$$m\left(\binom{n}{n} + \binom{n+1}{n} + \cdots + \binom{n+k}{n}\right) = m \cdot \binom{n+k+1}{n+1} = O(m \cdot n^k)$$

sub-arrays of coefficients and polynomial degree arrays.

A problem does arise in actually evaluating the polynomials. The number of terms in a polynomial is $O(m \cdot n^k)$ and so increasing either the domain or degree of the polynomial leads to exponential growth in evaluation time. It is for this reason that we restrict our analysis to degree one polynomials for $n, m \in \{1, 2\}$ and degree two polynomials for $n = m = 1$. Higher-dimensional cases are simply intractable for our time-frame and the randomness of WSRN prevents us from predicting run-time.

4.2. Sampling.

Regarding the sampling of inputs, the program is designed to allow sampling from the following distributions: Normal, Uniform, Exponential, Beta, Gamma, and Weibull. However, we restrict our attention to the Normal distribution, specifically $\mathcal{N}(0, 1)$. Since we will be taking on the order of thousands of samples to compute empirical loss as well as a large number of iterations of the algorithm, we can argue that using other well-behaved random variables (finite 4th moment) as inputs would be asymptotically the same as using the Normal distribution by the central limit theorem.

Now that we have restricted our inputs to $Z \sim \mathcal{N}(0, 1)$ and the degree of our polynomial is at most 2 we set the number of samples used to compute the empirical loss to 1000 as Z and Z^2 are fairly well behaved random variables, so by the general Hoeffding's inequality (with suitable coefficient bounds) we can bound the probability of error to be very small. For more on the general Hoeffding's inequality see chapter 2 of Vershynin's High-Dimensional Probability [4].

To see why 1000 samples is sufficient, let us consider the case where $n = m = k = 1$. For this case we need only use Chebyshev's inequality. Our target function f and our hypothesis function h are both of the form $f(x) = a_0 + a_1 x$ and $h(x) = b_0 + b_1 x$. We want to compute the expected loss

$$L = L\text{Perf}_{f, \mathcal{N}(0,1)}(h) = \mathbb{E}_{x \sim \mathcal{N}(0,1)}\left[|f(x) - h(x)|\right]$$

using the empirical estimator

$$L_s = L\text{Perf}^s_{f, \mathcal{N}(0,1)}(h) = \frac{1}{s}\sum_{i=1}^{s} |(a_0 - b_0) + (a_1 - b_1)Z_i| = \frac{1}{s}\sum_{i=1}^{s} X_i,$$

with $Z_i \sim \mathcal{N}(0, 1)$ and $X_i \sim |Y_i|$ where $Y_i \sim \mathcal{N}(a_0 - b_0, |a_1 - b_1|)$. The X_i are thus half-normals and we have $L = \mathbb{E}(X_i)$ and $\sigma^2 = \text{Var}(X_i) = \text{Var}(Y_i)\left(1 - \frac{2}{\pi}\right) = |a_1 - b_1|\left(1 - \frac{2}{\pi}\right)$. Using the fact that $\text{Var}(L_s) = \frac{\sigma^2}{s}$ Chebyshev's tells us that

$$\Pr(|L_s - L| > \epsilon) \leq \frac{\sigma^2}{s\epsilon^2}.$$

If we let $s = 1000$, $\epsilon = 0.1$, and $|a_1 - b_1| \leq 3$ we have probability of error at most $\frac{3\left(1 - \frac{2}{\pi}\right)}{1000(0.1)^2} \approx 0.1$. We will thus choose bounds appropriately so that the absolute difference between any two coefficients of a respective term in a polynomial (term with identical degrees in all variables) is at most 3.

4.3. WSRN and WSRN-Mutator Parameters.

Recall that the WSRN algorithm requires two bounds, l and u and the WSRN-Mutator requires a threshold t for the selection rule. We modify P. Valiant's argument to derive such values. Given target t and hypothesis h with error $\epsilon < 1$ we let c be the sum of the absolute differences of the coefficients of f and h. Let q be the number of terms in each of f and h and let $r = \frac{\epsilon^2}{36c^2\sqrt{q}}$. Then we will set $l = r\epsilon^4$, $u = r\epsilon^{-4}$, and $t = \frac{\epsilon^2}{18cq}$. Although the use of these values is a bit ad hoc, the reader will see analogous values used in the proof of Theorem 3.3. No particular guidance is given towards picking l and u expect that they should be chosen so that $r \in [l, u]$. Choosing l and u as above does so and allowed for faster evolution in our simulations then say a constant scaling of r.

5. Procedure

Our procedure involves numerous iterations, which consider different combinations of input parameters. Recall that we will consider five cases for the domain, codomain, and degree (n, m, k) of our polynomials: $n, m \in \{1, 2\}$, $k = 2$ and $n = m = 1$, $k = 2$. We will first fix a set of errors (all less than 1), a set of bounds B, and a set of numbers of children per generation C. The errors we will consider are contained in the set $A = \{0.01, 0.0075, 0.005, 0.0025, 0.001, 0.00075, 0.0005, 0.00025\}$. The algorithm will then generate a target f and hypothesis h where $f, h : \mathbb{R}^n \to \mathbb{R}^m$ with degree at most k and with coefficients each selected independently from a $\text{Unif}(-b, b)$ distribution for b the first bound. These polynomials are stored and for each bound are scaled by the ration of the bound to b (i.e. if the bounds are $[1, 2, 3]$ and $f(x) = 1 + x$, $g(x) = 2x$ then the polynomials $1+x$, $2+2x$, and $3+3x$ will all be stored as targets and the polynomials $2x$, $4x$, $6x$ will be stored as hypotheses). We will then evolve t to h and count and store the number of generations of the WSRN-Mutator (with $c \in C$ children per generation) are required to reach each error. We will repeat this for all scaled t and h. Considering the example above, we will evolve $2x$ to $1 + x$, $4x$ to $2 + 2x$, and $6x$ to $3 + 3x$. This scaling is justified since $a \cdot \text{Unif}(-b, b) \sim \text{Unif}(-ab, ab)$ and the reason for doing so is to reduce the randomness in selecting polynomials to get comparable results We will then iterate this process 100 times and average and return the results. Finally, we repeat this process for all numbers of children per generation $c \in C$.

The set of bounds and set of number of children per generation used are

$$B = \begin{cases} \{0.5, 0.75, 1, 1.25, 1.5\} & n = m = k = 1 \\ \{0.5, 1, 1.5\} & \text{otherwise} \end{cases}$$

$$C = \begin{cases} \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} & n = m = k = 1 \\ \{1, 4, 7, 10\} & (n, m) \in \{(1, 2), (2, 1)\}, k = 1 \\ \{1, 5, 10\} & n = m = 2, k = 1 \\ \{10, 20, 30\} & n = m = 1, k = 2 \end{cases}$$

For the case where $n = m = k = 1$ we also use three different loss functions $L^{1/2}$, L^1, and L^2 whereas for all other cases we only consider L^2.

6. Results

Here we present and briefly discuss the results from the simulations in both a quantitative and a qualitative manner. Over 200 different cases were recorded, however, we present only a small illuminating subset of these cases. The other cases act very similarly and the raw data can be found here.

6.1. Asymptotic Growth of Generations.

Let g be the number of generations required to evolve a target polynomial f to a hypothesis h within ϵ error, with respect to the L^p loss function. Our main claim is that

$$g = \Theta\left(\log\left(\frac{1}{\epsilon}\right)\right).$$

Given a domain degree n, codomain degree m, polynomial degree k, coefficient bound r, number of children per generation c, and L^p loss function we use the model

$$g(\epsilon) = a \log\left(\frac{1}{\epsilon}\right) + b \quad \text{for} \quad a \in \mathbb{R}^+, b \in \mathbb{R}$$

for the number of generations required on average to evolve a uniformly generated hypothesis h to a uniformly generated target f within ϵ error.

To justify this we provide graphs of the regressions of this model to the simulation data. These regressions are all computed using Python 3, specifically the numpy module. In the graphs, the variables n, m, k, r, c, p are as defined above and a and b represent the coefficients corresponding to the regression (i.e. $a = 1, b = 1$ corresponds to $g(\epsilon) = \log\left(\frac{1}{\epsilon}\right) + 1$ where by log we mean the base-10 logarithm). First, consider the case where $n = m = k = 1$ in Figure 1. We only provide cases with r and c at the most extreme values, however other cases are very similar.

It is clear from the high R^2 values that this model does, in fact, fit well to this case. Furthermore, the model is independent of the loss function used as wee see similar fits for any $p \in \left\{\frac{1}{2}, 1, 2\right\}$. The model also fits well to the other four cases for n, m, k as evidenced by Figure 2

6.2. Variation in Coefficient Bounds.

It would be expected that increasing the coefficient bound r would result in an increase in g. Our results shown in Figure 3 indicate that this does occur, at least asymptotically. The graphs showcase this behavior for the cases where $n = m = 1$ and $k \in \{1, 2\}$. There are some deviations from this trend, but they can be associated with the innate randomness of the WSRN algorithm.

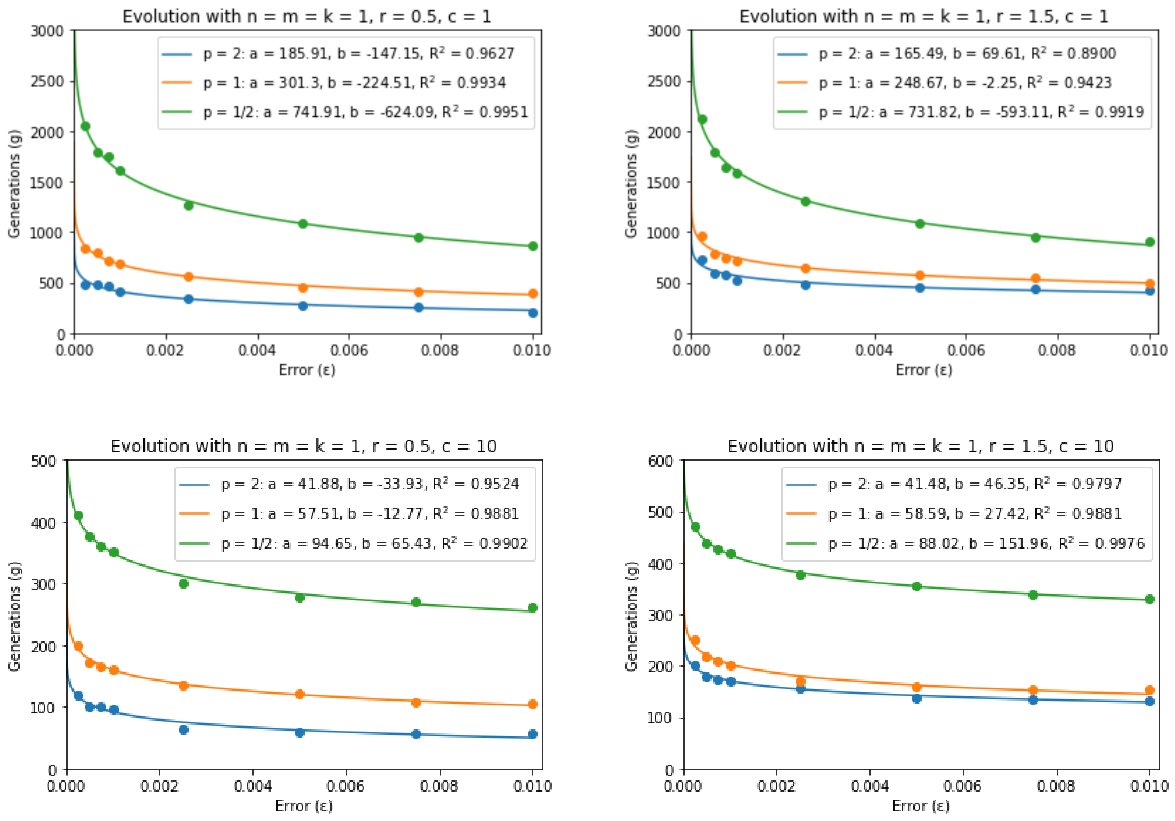

FIGURE 1

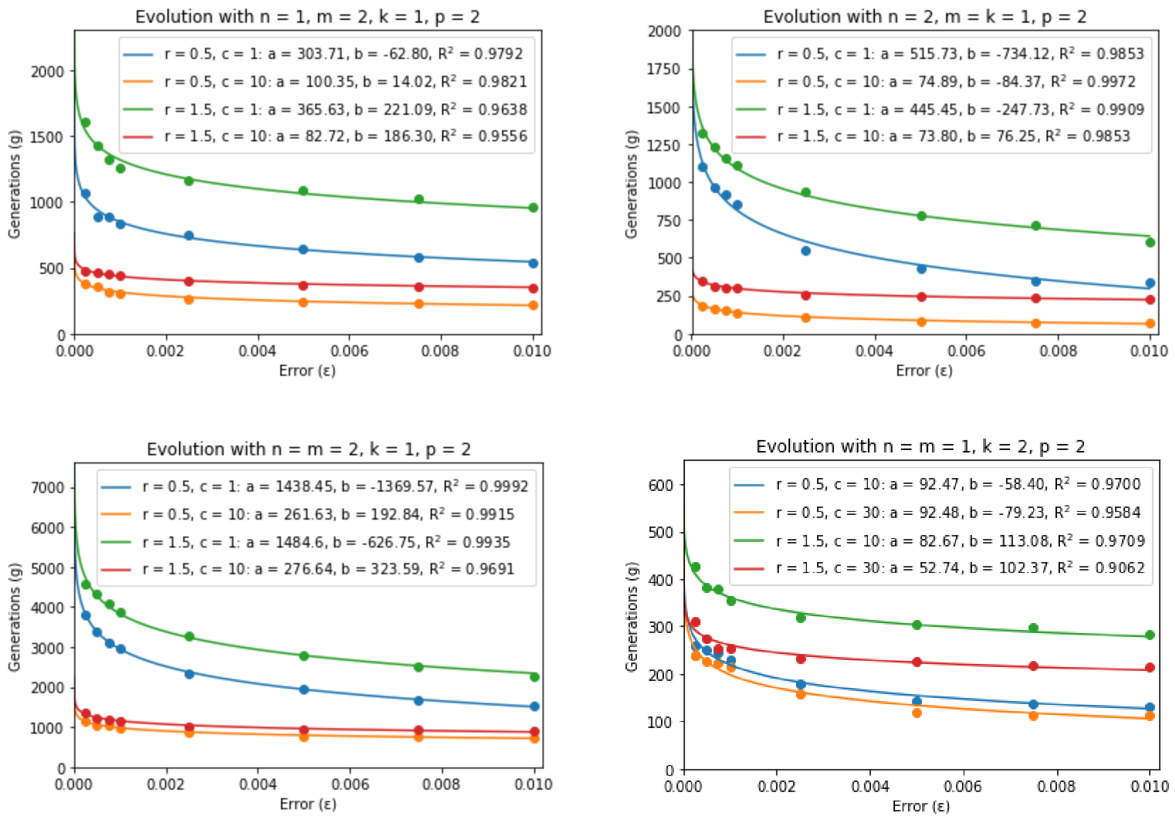

FIGURE 2

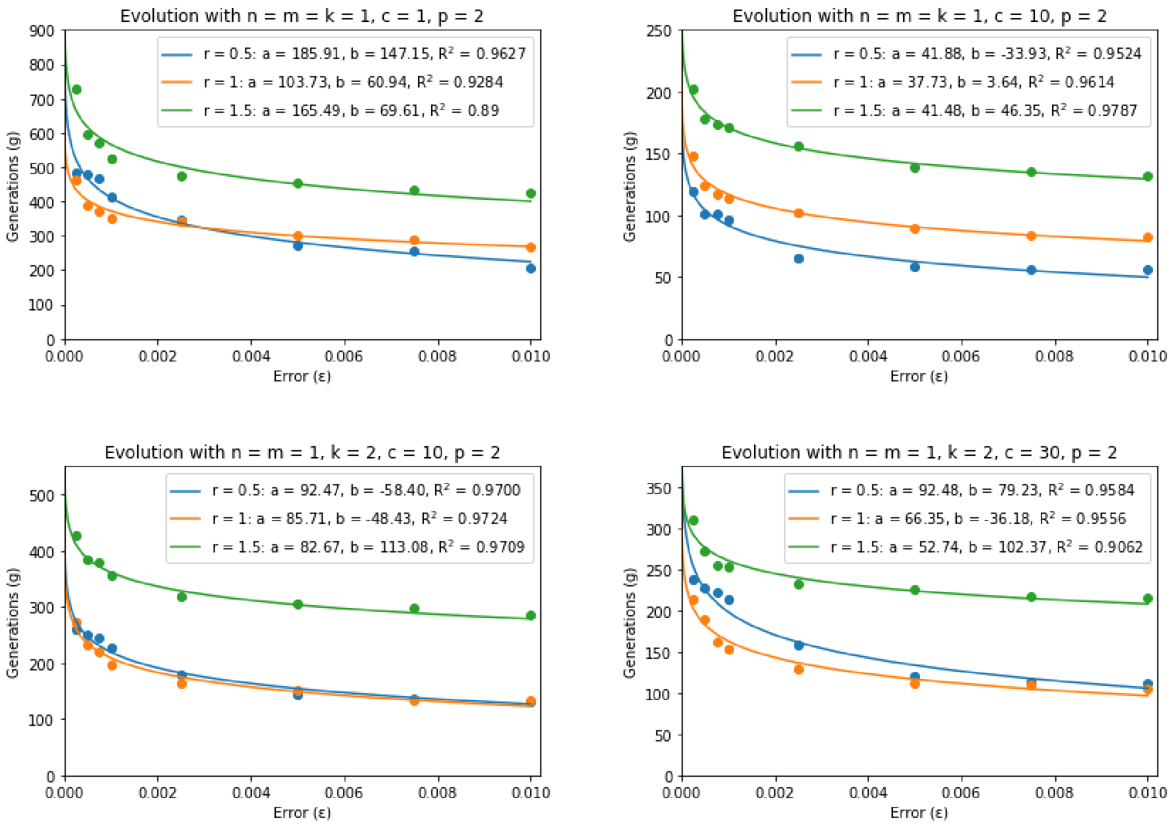

FIGURE 3

6.3. **Variation in Number of Children per Generation.** Increasing the number of children per generation c should result in a decrease in g since the probability of having beneficial mutations each generation goes up. This behavior holds very well asymptotically and is showcased in Figure 4, but there are again some deviations, which, like above, can be chalked up to WSRN's randomness.

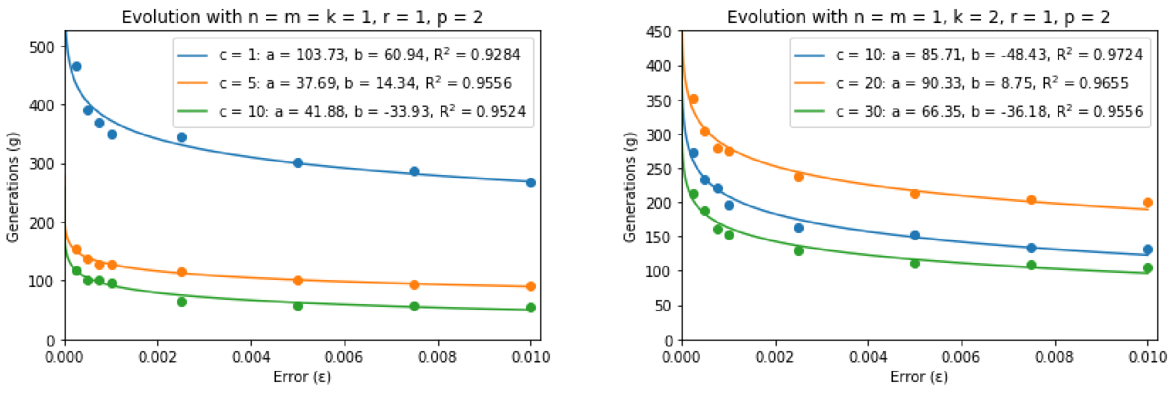

FIGURE 4

7. Conclusion and Future Direction

7.1. **Asymptotic Growth of Generations.** Analysis of the results would indicate that our main claim that $g = \Theta(\log\left(\frac{1}{\epsilon}\right))$ holds quite well in the low-dimensional cases we have considered. The justification for this comes from the quality of the regressions of the corresponding model to our data for all cases (a number of which are presented in this paper). The robustness of the model follows from the high R^2 values. Though R^2 can be scrutinized, our model is incredibly simple including only a single non-constant term, and so over-fitting is unlikely. For further rigor we can consider Adjusted-R^2

defined as

$$\overline{R^2} = 1 - (1 - R^2)\frac{n-1}{n-p-1} = 1 - (1 - R^2)\frac{7}{6} = \frac{7}{6}R^2 - \frac{1}{6},$$

where $n = 8$ is the number of samples and $p = 1$ is the number of non-constant terms in the model. The lowest recorded R^2 value is 0.8900 and so the lowest Adjusted-R^2 is 0.8713 whereas the highest Adjusted-R^2 is 0.9991 corresponding to $R^2 = 0.992$ (note that it is easy to see that $R^2 - \overline{R^2} \leq 0.8900 - 0.8713 = 0.0187$). One important consideration, however, is that this model and regression holds only for sufficiently small error ϵ. This is the greatest degree of over-fitting since the model becomes useless once ϵ increases above a certain point.

Assuming our main claim, it is impossible (besides removing a constant term) to make the model simpler. For the future, however, increasing the number of targets and hypotheses sampled and evolved (currently set at 100) should result in our data behaving according to this model to an even greater extent as a greater number of samples would reduce the inherent variability of the WSRN algorithm.

The importance of this result is quite apparent and its validity would imply the greater feasibility of WSRN as a biological polynomial evolution algorithm, considering the only logarithmic dependence on the inverse of the error. Again, under the assumption of the claim, decreasing the desired error by n orders of magnitude would only require an (for some constant a) more generations for evolution. Thus, a hypothesis function under the WSRN-Mutator converges exponentially quickly (in terms of number of generations) to the target function.

7.2. **Variation in Coefficient Bounds and Number of Children per Generation.** Variation in the coefficient bounds resulted in expected behavior as increasing the bound typically led to a larger number of generations of the WSRN-Mutator until evolution. The intuition behind this is clear as scaling a target and hypothesis by a constant larger than 1 results in an increase in the initial loss and greater deviation between polynomial coefficients.

Variation in the number of children per generation generally resulted in a decrease in the number of generations required for evolution. This behavior is again expected as more children per generation means a higher probability of a beneficial mutation every generation. There were some instances where this behavior did not hold. For example, if we look at the right-most graph in Figure 4 we see that evolution with $c = 10$ required on average less generations than evolution with $c = 20$. Although the behavior for $c = 30$ is as expected, it is interesting as to how and why this deviation could have occurred. One possibility is, as previously mentioned, inherent randomness with WSRN, but this is not sufficient in explaining this degree of deviation from the expectation. Another explanation is the possibility that more children could result in the selection of a mutation that is not the "best" mutation. A key component of our selection rule is that we randomly select from the set of beneficial (or neutral) mutations based on probability of occurrence, which in WSRN is uniform. A set of 10 mutations could contain one particularly "good" mutation that substantially decreases the expected loss along with a few other beneficial mutations. Similarly, a set of 20 mutations could be structured the same way. However, with only 10 mutations it is more likely that this "best" mutation will be selected than if there were 20 mutations. Though it appears that this argument could be extended to the case where $c = 30$, it is likely that by increasing c even more, the probability of having a beneficial improvement increases so much that evolution simply occurs at a faster rate.

7.3. **Future Direction.** As already mentioned, one first step would be to take more samples of target and hypothesis functions to be evolved simply to minimize the inherent variation of WSRN. Another objective should be to perform a more rigorous analysis of the effects of varying the coefficient bounds and the number of children per generation. In this paper, we present some results based primarily on intuition without much rigorous analysis, but a potential approach is to fix a single error ϵ and consider the behavior of g as we alter one of r and c (fixing the other). It would also be valuable to see to what extent our results generalize to higher dimensional examples and to also measure the dependency of g on n, m, and k. The difficulty with this, however, is due to the fact that polynomial evaluation takes $O(m \cdot n^k)$ time and becomes intractable quite quickly. Given more time though, it would be possible to conduct simulations with larger n, m, and k and derive meaningful conclusions.

References

[1] Lee, Jasper C. H., and Paul Valiant. "Optimizing Star-Convex Functions." 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2016. Crossref, doi:10.1109/focs.2016.71.

[2] Valiant, Leslie G. "Evolvability." Journal of the ACM, vol. 56, no. 1, Association for Computing Machinery (ACM), Jan. 2009, pp. 1–21. Crossref, doi:10.1145/1462153.1462156.

[3] Valiant, Paul. "Evolvability of Real Functions." ACM Transactions on Computation Theory, vol. 6, no. 3, Association for Computing Machinery (ACM), July 2014, pp. 1-19. Crossref, doi:10.1145/2633598.

[4] Vershynin, Roman. High-Dimensional Probability: An Introduction with Applications in Data Science. Cambridge, United Kingdom New York, NY: Cambridge University Press, 2018. Print.