Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 89 (2011)

Distribution Free Evolvability of Polynomial Functions over all Convex Loss Functions

Paul Valiant

August 7, 2011

Abstract

We formulate a notion of evolvability for functions with domain and range that are real-valued vectors, a compelling way of expressing many natural biological processes. We show that linear and fixed-degree polynomial functions are evolvable in the following dually robust sense: There is a single evolution algorithm that for all convex loss functions converges for all distributions.

It is possible that such dually robust results can be achieved by simpler and more natural evolution algorithms. In the second part of the paper we introduce a simple and natural algorithm that we call "wide-scale random noise" and prove a corresponding result for the L_2 metric. We conjecture that the algorithm works for more general classes of metrics.

ISSN 1433-8092

1 Introduction

Since the introduction of the evolvability model by L. Valiant in [11], significant work has been done to show both the power and the robustness to modeling variations of this computational framework for investigating how complexity can arise in a fixed environment [2, 3, 4, 7, 10]. In this work we present two complementary constructions, which extend this body of work in a new and very natural direction: while previous papers studied evolvability of Boolean functions (from $\{0,1\}^n \to \{-1,1\}$) we here consider functions from $\mathbb{R}^n \to \mathbb{R}^m$. Many of the functions that evolve in biology, for example, "how should the concentration of protein A in this cell vary in response to the concentrations of proteins B through Z?", might be much more naturally represented as real functions as opposed to Boolean functions. Of course, real functions, when restricted to Boolean domain and range, become Boolean functions, so in some sense this model is more general than the original Boolean model of evolvability.

We first define the notion of evolvability over the reals, and then proceed in two directions. The first direction consists of an adaptation of Feldman's results on evolvability from [4] to our setting, which, because of the different setting, immediately yields results of a somewhat different nature: evolvability can simulate arbitrary polynomial-time optimization algorithms that only require *approximate* access to the function being optimized. In the terminology of Lovasz [9], this is *weak optimization*. We show that his construction of polynomial time weak convex optimization can be leveraged to yield evolution algorithms for linear functions, and further, fixed-degree polynomials, in a distribution-independent sense, for any convex loss function—including *a fortiori* the commonly considered linear and quadratic (L_1 and L_2-squared) loss functions.

For the original Boolean framework of evolvability, Feldman showed that, as long as the underlying distribution is *known*, then the class SQ (statistical queries) defined by Kearns in [8] exactly characterizes the classes of evolvable functions [2]. SQ is both a powerful and natural framework, and seems to capture most of the power of PAC learning. However, the assumption that the evolution algorithm must know the underlying distribution is decidedly unnatural, as one would hope for evolution to function across a broad range of potentially quite intricate and varying distributions of conditions for its creatures. In subsequent work, Feldman showed that if one reinterprets the Boolean model by allowing hypotheses that take real values (even though the hypotheses are then compared, via a loss function, to target values that are Boolean), then if the performance metric is *non-linear*—that is, essentially anything except the correlation metric (the L_1 metric restricted to a small enough region so that it is a linear function, instead of the usual piecewise linear)—one can take advantage of a "kink" in it to, in fact, evolve everything in SQ (Theorem 4.3 of [4]). In a sense, by changing part of the Boolean model to allow for real values, Feldman circumvents an apparent limitation of the purely Boolean evolution setting.

In this current work we consider *all* elements of the evolvability model to be real instead of Boolean. The results we derive are of a rather different form from those in the (partially) Boolean settings previously considered. We show that there exists an algorithm that evolves linear and fixed-degree polynomial functions over the reals, for all distributions and all convex metrics (including both the L_1 and L_2 metrics).

While these results demonstrate the power of the real evolvability framework, they come at the expense of a certain unnaturalness of the underlying evolution algorithm. We balance this out in the second part of this paper by considering perhaps the simplest and most natural algorithm that could be hoped to work, and showing that it in fact can reproduce a significant portion of these results, albeit less efficiently. In a generation of this algorithm, a parent produces a polynomial number of nearby children, each chosen in a uniformly and independently random direction. "Survival of the fittest" turns this into a kind of "steepest descent" strategy, which enables us to prove that, for quadratic loss functions, constant progress is made in each generation, which will rapidly lead to the optimum.

This algorithm, which we call “wide-scale random noise” to emphasize its simple unstructured nature, has in fact been found in simulations to converge rapidly in many cases beyond that of the quadratic loss function, though how it achieves this convergence seems rather different than its provable behavior in the quadratic case. In particular, while for quadratic loss functions, the algorithm provably consistently produces offspring which perform better than the parent, leading to a guaranteed improvement, in contrast for the case of the L_1 loss function the ability to evolve to a descendent whose performance is worse than that of the parent seems crucial for efficient progress. It would seem counterintuitive that such “backtracking” would help in a convex landscape with no spurious local minima. However, this was exactly the effect found in a paper on simulated annealing that also considered a very similar L_1 optimization setting [6]. It would appear that both evolvability and simulated annealing seem effective in unexpected cases, and one might hope that new analysis of one might shed light on the other.[1]

2 Definitions

We adapt much of our notation from [4].

We note a concrete example to motivate the following definitions: consider the task of trying to evolve linear functions. Namely, there is an unknown linear function $f : \mathbb{R}^n \to \mathbb{R}^m$ refered to as the *target* function, and an unknown distribution D over $\mathbb{R}^n$ from which “nature” draws test cases to evaluate the performance of a creature. Creatures are distinguished by the *hypothesis* encoded in their genome, namely a function $h : \mathbb{R}^n \to \mathbb{R}^m$. The creature’s “life” consists of being presented a set of samples from the distribution D; on each sample, it evaluates h and is penalized by nature according to how its answers differ from the target function f. To make this precise, we must introduce the notion of a *loss function*.

Definition 1. *For hypotheses having range $\mathbb{R}^m$, a loss function is a nonnegative function $L : \mathbb{R}^m \times \mathbb{R}^m \to [0, \infty)$ such that for any $x \in \mathbb{R}^m$, $L(x, x) = 0$.*

Definition 2. *Given a loss function L and a target function $f : \mathbb{R}^n \to \mathbb{R}^m$, the* performance *of a hypothesis function $h : \mathbb{R}^n \to \mathbb{R}^m$ relative to a distribution D (over $\mathbb{R}^n$) is defined as $L\mathsf{Perf}_{f,D}(h) = \mathbf{E}_{x \leftarrow D}[L(f(x), h(x))]$. Given a positive integer s, the s-sample* empirical performance *$L\mathsf{Perf}^s_{f,D}(h)$ is defined to be the random variable resulting from drawing s samples $z_1, \ldots, z_s \leftarrow D$ and evaluating $\frac{1}{s}\sum_{i=1}^{s} L(f(z_i), h(z_i))$.*

In a manner which will be made precise shortly, evolution picks a hypothesis h which, empirically, is chosen to have small loss.

In general, instead of considering the class of linear functions, we consider the evolvability of a *concept class C*, consisting of a subset of the functions $f : \mathbb{R}^n \to \mathbb{R}^m$. And, as there may be some pathological distributions that bar progress, we may restrict ourselves to a class $\mathcal{D}$ of distributions over $\mathbb{R}^n$.

In particular, working over the real numbers introduces problems of scale that are not present in the Boolean case. For example, since the “feedback” that nature gives the evolution algorithm in any generation is simply the choice of which, of a bounded (polynomial) number of potential children, survives to the next generation, there is no way to evolve in bounded time a good approximation to an

[1] Feldman has defined “monotone” evolvability to be the restriction where each generation’s performance must be at least that of the previous generation, in part inspired by a desire to consider evolution algorithms that seem more “natural”. Monotone evolvability has been shown in a distribution-independent setting for point functions under the L_1 metric [4], conjunctions under the L_2 metric [3], and very recently, for linear threshold functions with “non-negligible margin”, for L_2 and related metrics [5].

unbounded real number. It is thus important to work with concept classes C that are in some sense bounded. A related issue arises with the distribution class $\mathcal{D}$. Suppose we are working with the L_1 loss function, $L(x,y) = |x-y|$, and suppose D is such that, with probability $1-\tau$, D samples the point $\mathbf{0}$, and with probability τ samples a point more than $\frac{1}{\tau}$-far from the origin. If τ is super-polynomially small, then evolution will likely never see any samples other than $\mathbf{0}$, but meanwhile, the expected loss of hypotheses is unknown and potentially huge. Thus D (and L) must also be reasonably bounded. We make precise the kinds of bounds we use in the particular theorems.

We now define the components of evolvability.

Definition 3 (Definition 3.6 in [4]). *Given parameter $\epsilon > 0$, a* mutation algorithm *A is defined by a pair (R, M) where*

- *R is a representation class of functions $\mathbb{R}^n \to \mathbb{R}^m$.*
- *M is a randomized polynomial (in n, m, and $1/\epsilon$) time Turing machine that, given $r \in R$ and $1/\epsilon$ as inputs outputs a representation $r_1 \in R$ with probability that we denote $\Pr_A(r, r_1)$. The set of representations that can be output by $M(r, \epsilon)$ is referred to as the* neighborhood *of r for ϵ and is denoted by $\mathsf{Neigh}_A(r, \epsilon)$.*

As far as the representation class, recall that the hypothesis functions are ultimately stored as the "genomes" of our creatures, and thus are represented as strings over a finite alphabet. For the results of Section 3 we explicitly represent functions as binary strings, though the class of functions represented by the scheme of Section 3 is somewhat artificial. In Section 4 we consider genomes that can represent the entire class of fixed-degree polynomial functions, and implicitly consider these polynomials as being represented by approximately representing each real-valued coefficient as a short string in the genome—only limited precision is required.

The mutation algorithm is the source of potential genomes for the next generation; which one survives is determined by the *selection rule*, an efficiently-implementable algorithm that we imagine nature running, defined as follows (from Definition 3.7 of [4]).

Definition 4. *For a loss function L, tolerance t, candidate pool size p, and sample size s, the* selection rule *$\mathsf{SelNB}[L, t, p, s]$ is an algorithm such that for any function f, distribution D, mutation algorithm $A = (R, M)$, representation $r \in R$ and accuracy ϵ, $\mathsf{SelNB}[L, t, p, s](f, D, A, r)$ 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 $\Pr_Z(r')$ be the relative frequency with which r' was generated among the p observed representations. For each $r' \in Z \cup \{r\}$, compute an empirical value of performance $v(r') \leftarrow L\mathsf{Perf}^s_{f,D}(r')$. Let $\mathsf{Bene}(Z)$ denote the set of empirically beneficial mutations, $\{r' \in Z \mid v(r') \leq v(r) - t\}$ and $\mathsf{Neut}(Z)$ denote the set of empirically neutral mutations, $\{r' \in Z \mid |v(r') - v(r)| < t\}$. Then*

(i) If $\mathsf{Bene}(Z) \neq \emptyset$ then output a random $r_1 \in \mathsf{Bene}(Z)$ distributed with relative probabilities according to $\Pr_Z$.

(ii) If $\mathsf{Bene}(Z) = \emptyset$ and $\mathsf{Neut}(Z) \neq \emptyset$ then output a random $r_1 \in \mathsf{Neut}(Z)$ distributed with relative probabilities according to $\Pr_Z$.

(iii) If $\mathsf{Neut}(Z) \cup \mathsf{Bene}(Z) = \emptyset$ then output $\perp$.

The situation where all children perform noticeably worse than the parent, in which case the selection rule outputs "$\perp$" is viewed as unnatural, and we view such a case as aborting. Otherwise, if for a concept class (and class of distributions) there exists a mutation algorithm that, under selection rule SelNB, efficiently converges to any target function in the class, then we say that the concept class is evolvable:

Definition 5 (See Definition 3.3 of [4]). *A concept class C, distribution class $\mathcal{D}$, 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 poly-bounded tolerance $t(r, n, m, \frac{1}{\epsilon})$ 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 $r_0 \in R$, with probability at least $1-\epsilon$ the random sequence defined by $r_i \leftarrow \mathsf{SelNB}[L, t, p, s](f, D, A, r_{i-1})$ will have $L\mathsf{Perf}_{f,D}(r_g) \leq \epsilon$.*

(We note that the sign convention most natural for the real case is opposite that used in previous work for the Boolean case, and in particular, a "perfect organism" in our setting has $L\mathsf{Perf} = 0$ while in [4] would have $L\mathsf{Perf} = 1$.)

3 Evolvability as "Weak" Optimization

The idea at the center of this section is that evolvability can reproduce any result efficiently obtainable from *approximate oracle access to* $L\mathsf{Perf}$. In this section we demonstrate this connection, which lets us then leverage the entire field of optimization algorithms towards our goal of evolvability, yielding immediate fruits at the end of this section.

As noted in the introduction, we prove this connection via an adaptation of the analogous result from the Boolean case—which appears as Theorem 5.1 in [4]. The main hurdle in both cases is showing that the selection rule SelNB can efficiently simulate approximate responses to questions of the form: "is $L\mathsf{Perf}_{f,D}(h)$ greater than a threshold θ?" In particular, this will be achieved in a single generation of evolution.

One difference between the real case and the Boolean case—or, more specifically, between how $L\mathsf{Perf}$ is defined here versus in [4]—is that in our case we have no functions whose performance we know *a priori*, while in their case, the Boolean function that returns an independent unbiased coin flip is guaranteed to have performance 0. Without such a reference point, evolvability has no hope of addressing such threshold queries. In lieu of an absolute benchmark like that, we instead adopt a relative benchmark, comparing performance always against $L\mathsf{Perf}(\mathbf{0})$.[2] Namely, our evolution algorithm will function as though it had approximate oracle access to $L\mathsf{Perf}(\cdot) - L\mathsf{Perf}(\mathbf{0})$.

We give an overview of the intuitive idea for the construction to approximately answer, in a single generation, queries of the form "is $L\mathsf{Perf}(h) - L\mathsf{Perf}(\mathbf{0}) > \theta$?". We assume genomes may represent probabilistic functions, and, moreover, assume as a sort of induction hypothesis that the parent's genome defines a function that is the $\mathbf{0}$ function a "large" fraction of the time. Letting $q = q(n, m, \frac{1}{\epsilon})$ be a bound on the total number of threshold queries we would ever need to resolve, we will guarantee going forward that the difference between the parent's probability of expressing the $\mathbf{0}$ function and any child's is at most $\frac{1}{q}$.

Denoting the parent's genome by r, its performance is $L\mathsf{Perf}^s_{f,D}(r)$, and for a given tolerance t, the selection rule SelNB treats children very differently according to whether their observed loss is within t of this (neutral mutations), more than t lower than this (beneficial mutations), or more than t higher than this and doomed to be culled. Since our goal is to make the selection rule have a sharp threshold near where $L\mathsf{Perf}(h) - L\mathsf{Perf}(\mathbf{0}) \approx \theta$, and the selection rule already has these natural sharp thresholds, the natural approach, as in [4], is to make use of these thresholds for our purposes, having r produce two types of children, r_0 that outputs identically to the parent r, and r_1 that outputs the function $\mathbf{0}$ with probability $\frac{t}{\theta}$ less than its parent and h with probability $\frac{t}{|\theta|}$ more than its parent.

[2] We note that here and for the rest of the paper, we use no special properties of the $\mathbf{0}$ function, and indeed any arbitrary function from the hypothesis class could be substituted here and throughout the paper. We use $\mathbf{0}$ simply to avoid introducing further notation. A more meticulous reader might mentally substitute an arbitrarily chosen element of the hypothesis class for $\mathbf{0}$ as it appears in the results below, to handle the odd but perfectly legitimate case that $\mathbf{0}$ is not in the hypothesis class of Theorem 1.

The details of the proof follow the ideas in the appendix of [4] (specifically Theorem A.3) and are given below.

To state the result more cleanly, we introduce "weak" optimization terminology adapted from [9]:

Definition 6. *A* μ-weak evaluation oracle *for a function* $f : \mathbb{R}^k \to \mathbb{R}$ *is an oracle that on input* x *returns a number* a *such that* $|f(x) - a| < \mu$.

Definition 7. *The* ν-weak function minimization problem *for a function* $f : \mathbb{R}^k \to \mathbb{R}$ *is that of finding an* x *such that* $\forall y \in \mathbb{R}^k, f(y) > f(x) - \nu$.

Definition 8. *A class of functions is* weakly optimizable *if there exists a randomized polynomial time oracle algorithm* A *and a polynomial* $\mu = \mu(\nu, \frac{1}{k})$ *such that for every* $\nu > 0$, *and any function* $f : \mathbb{R}^k \to \mathbb{R}$ *in the class,* A *solves the* ν*-weak function minimization problem when given access to a* $\mu(\nu, \frac{1}{k})$*-weak evaluation oracle for* f.

Theorem 1. *If* L *is a loss function,* C *is a concept class, and* $\mathcal{D}$ *is a distribution class such that there is a polynomial* $b(n, m)$ *that bounds* $L(f_1(x), f_2(x))$ *for any* $f_1, f_2 \in C$ *and any* x *in the support of a distribution in* $\mathcal{D}$, *and such that the class of functions* $L\mathsf{Perf}_{f,D}(h) - L\mathsf{Perf}_{f,D}(\mathbf{0})$ *indexed by* $f \in C, D \in \mathcal{D}$ *and evaluated on* $h \in C$ *is weakly optimizable, then* $(C, \mathcal{D}, L)$ *is evolvable.*

We will find it convenient to first prove this result in a restricted model referred to as "evolvability with initialization", where Definition 5 is modified so that instead of assuming evolution starts with an arbitrary genome $r_0 \in R$, we instead assume a fixed starting configuration. (See Theorem A.1 of [4].)

Lemma 1. *Theorem 1 holds under the restricted* evolvability with initialization *model where Definition 5 is changed by replacing the phrase "any initial genome* $r_0 \in R$*" by "initial genome* $r_0 = \star$*".*

Proof. By assumption, there is a randomized polynomial time algorithm and a polynomial $\mu = \mu(\nu, k)$ such that for every $\nu > 0$ and any $f \in C$ and $D \in \mathcal{D}$, the algorithm, when given μ-weak oracle access to $L\mathsf{Perf}_{f,D}(\cdot) - L\mathsf{Perf}_{f,D}(\mathbf{0})$, will return a hypothesis $h \in C$ that is within ν of optimal. Denoting by T a (polynomial) bound on the runtime of this algorithm, we note that we may equivalently reexpress it as a deterministic algorithm that is given as auxiliary input a T-bit uniformly random string. Our goal will be to show that we can simulate the operation of this algorithm in the evolvability framework

As a first step, we will replace the weak evaluation oracle with a simpler oracle, the *weak comparison oracle.*

> The μ-weak comparison oracle for a function g will, on given an input x and a threshold θ, return 1 if $g(x) \geq \theta + \mu$, 0 if $g(x) \leq \theta - \mu$, and either 1 or 0 otherwise.

We note that since by assumption, b bounds the value of the function in question, that is, $L\mathsf{Perf}_{f,D}(\cdot) - L\mathsf{Perf}_{f,D}(\mathbf{0})$, we have that $\log \frac{b}{\mu}$ bounds the number of rounds of binary search we need to μ-weakly approximate the value of the function via weak comparison queries. Denote this bound by β, which since b and μ are polynomial, is hence polynomially bounded itself. We note, as will be important later, that such a binary search can be designed so that none of the thresholds queried ever have magnitude less than μ.

We have thus trivially shown that there is a deterministic algorithm that, when given as an auxiliary input a T-bit uniformly random string, and given weak comparison oracle access to $L\mathsf{Perf}_{f,D}(\cdot) - L\mathsf{Perf}_{f,D}(\mathbf{0})$, will return a ν-weak minimum within $T\beta$ steps. We denote this algorithm A, and for the sake of concreteness, assume that after $T\beta$ steps have passed, it halts and outputs a hypothesis, no matter what.

We now turn to the task of expressing algorithm A in the evolvability framework. Recall that by assumption, the initial genome is uniquely fixed as "$\star$". We thus ask the mutation algorithm to, upon seeing the initial genome, produce children whose genome encodes T bits uniformly generated at random. In each subsequent stage of mutation, these bits will be preserved in the genome; in this manner, future generations will have access to this randomly generated T-bit string, as desired.

All that remains is to describe how to simulate weak comparison queries. We will simulate one query per generation, with the result of the query being stored in the genome for the duration. Thus at time 0 the genome will consist of "$\star$", at time 1, of a T-bit random string, and at time $1+j$ we aim for the genome to consist of the concatenation of this string with a j-bit string that stores the results of the first j weak comparison queries as specified by the algorithm A under simulation. For each such genome, we must specify how the corresponding creature responds to inputs. For the genome "$\star$" and any genome consisting solely of a T-bit string, we return the 0 vector. Otherwise, let R be this T-bit random string, and let z be the remainder of the genome, whose length we denote by j, and whose ith bit we denote z_i. Recall the algorithm A whose results we are trying to reproduce. Iteratively simulate A starting with string R, and let (h_1, θ_1) be the first query sent to the weak comparison oracle; interpreting z_1 as the result of this query, let (h_2, θ_2) be the next query asked by A, and so on. We thus derive (h_i, θ_i) for each $i \in \{0, \ldots, j-1\}$, all computed in polynomial time. We thus define the output behavior of our genome on input $x \in \mathbb{R}^n$: *for each $i \in \{0, \ldots, j-1\}$ such that $z_i = 1$, output $h_i(x)$ with probability $\frac{\mu}{|\theta_i| \cdot T\beta}$ and otherwise output the vector* 0. Since $|\theta_i|$ is guaranteed to be at least μ by construction, the sum of the probabilities over the (up to) $T\beta$ generations involved will never exceed 1.

A complete specification of the scheme requires only that we now specify the mutation probabilities. Namely, given the (random) string R of length T and a string z of length j, where we may determine that (h_{j+1}, θ_{j+1}) is the next query to be simulated, we must choose with what probability the mutation algorithm M should output $Tz0$ as opposed to $Tz1$. Very simply, if $\theta_{j+1} < 0$ then output $Tz0$ with probability $1-\Delta$ and $Tz1$ with probability Δ, otherwise output $Tz1$ with probability $1-\Delta$ and $Tz0$ with probability Δ, where $\Delta = \frac{\epsilon}{3g}$ is chosen so that in g (our target number of generations) rounds of coin flips, a Δ-biased coin will never land heads, except with probability somewhat less than ϵ.

We choose the tolerance parameter t, which specifies the width of the "neutral" zone of performance, to equal $\frac{\mu}{T\beta}$. We choose s, the number of samples taken to evaluate the empirical performance, to be large enough so that with probability $> 1 - \frac{\epsilon}{3}$ the empirical estimates are never off by more than $t\frac{\mu}{2b}$ over the entire course of g generations. We analyze the scheme in two cases, noting that, if we denote the expected performance of genome Tz by ρ, then the expected performance of $Tz0$ equals ρ while the expected performance of $Tz1$ equals $\rho + \frac{\mu}{|\theta_{j+1}| \cdot T\beta}\left[L\mathsf{Perf}_{f,D}(h_{j+1}) - L\mathsf{Perf}_{f,D}(\mathbf{0})\right]$, where as just defined, $\frac{\mu}{T\beta} = t$.

Case 1: $\theta_{j+1} < 0$**.** If the weak comparison query must return negative, that is, if the expected value of $L\mathsf{Perf}_{f,D}(h_{j+1}) - L\mathsf{Perf}_{f,D}(\mathbf{0})$ is at most $\theta_{j+1} - \mu$, then the expected performance of $Tz1$ is at most $\rho + \frac{t}{|\theta_{j+1}|}(\theta_{j+1} - \mu) \le \rho - t - \frac{t\mu}{b}$. Since by assumption, except with probability $< \frac{\epsilon}{3}$, the empirical performance will always approximate the expected performance to within $\frac{t\mu}{2b}$, we have that $Tz1$ will be found to be beneficial, while $Tz0$ will be found to be neutral, and thus the next genome will be $Tz1$, correctly encoding the answer to the weak comparison query. Conversely, if the weak comparison query should return positive, then by analogous argument, the expected performance of $Tz1$ is at least $\rho - t + \frac{t\mu}{b}$, and $Tz1$ is thus either a neutral or negative mutation. Recall that by construction, in this case, an overwhelming majority of the mutations in this generation were constructed to be $Tz0$ instead of $Tz1$, and thus in either case, with very high probability (specifically, at least $\Delta = \frac{\epsilon}{3g}$) $Tz0$ will thus be correctly chosen for the next generation.

Case 2: $\theta_{j+1} > 0$**.** If the weak comparison query must return positive then, from the above argument, the expected loss of $Tz1$ is at least $\rho + t + \frac{t\mu}{b}$, in which case $Tz1$ is a negative mutation, and $Tz0$ will be chosen for the next generation, as desired. Otherwise, the expected loss of $Tz1$ is at most $\rho + t - \frac{t\mu}{b}$, which will be either neutral or beneficial; since the mutation algorithm will construct $Tz1$ instead of $Tz0$ an overwhelming fraction of the time $(1-\Delta)$, with overwhelming probability $Tz1$ will thus by correctly chosen for the next generation.

We conclude by stipulating that once the simulation of A has completed (which will occur with probability at least $1-\epsilon$), the mutation algorithm will compute the result that A would have computed, and thus return a satisfactory hypothesis. □

We now prove Theorem 1, resulting from Lemma 1 and a short argument that initialization is not necessary for the successful evolution of our algorithm. We take a simpler approach than [4] though at the expense of looser bounds.

Proof of Theorem 1. We note that the parameters in the proof of Lemma 1 were chosen so that the probabilistic procedure described will deviate from its expected behavior with probability at most ϵ over g generations, where, significantly, g is a parameter that we are still free to specify. Intuitively, evolution will follow the procedure set up in the proof of Lemma 1, which takes $1 + T\beta$ generations, except that at every generation, there is probability ρ to be defined shortly of *reinitializing*, that is, attempting to start evolution from scratch again. We will exhibit a reinitialization procedure that takes $\frac{2b}{t}$ generations. Thus one round of complete reinitialization and evolution will take $1 + T\beta + \frac{2b}{t}$ generations, while in expectation this will happen only once every $\frac{1}{\rho}$ generations. Let $\rho = \frac{1}{2}\epsilon\big/(1 + T\beta + \frac{2b}{t})$. Since after $\frac{1}{\rho}|1 + \log\epsilon|$ generations, this procedure will have occurred at least once with probability at least $1 - \frac{\epsilon}{2}$, we have that for any moment in time after this, the probability that evolution is at a weak optimum is at least $1-\epsilon$. Thus letting $g = \frac{1}{\rho}|1 + \log\epsilon|$ yields the theorem with probability of success at least $1 - 2\epsilon$. We thus reparameterize $2\epsilon \rightarrow \epsilon$.

We now illustrate the very simple reinitialization procedure, which will take $\frac{2b}{t}$ generations, a number we denote here as c. For each genome representation G in the scheme of Lemma 1, with the exception of "$\star$", we add copies labeled by integers $i \in \{0, \ldots c-1\}$, which we denote as G^i with the interpretation that G^i is "G after i out of c steps towards reinitialization." The mutator described in Lemma 1 we modify so that every time it might output a certain representation G, now with probability ρ it will instead output G^0. The mutation rule for G^i is even simpler: if $i \neq c-1$ then output G^{i+1}, and if $i = c-1$ then output "$\star$", that is, reinitialize.

We now define how elements G^i evaluate an input x: with probability $\frac{c-i}{c}$ output whatever G would output; with probability $\frac{i}{c}$ output the 0 vector. We note that since the performance difference between $\mathbf{0}$ and any other hypothesis is at most b, that the expected change in performance over any generation of reinitialization is thus at most $\frac{b}{c} = \frac{t}{2}$, namely, these are all neutral mutations, and, by the parameter choice of Lemma 1 will be recognized as such, which guarantees that this procedure will operate as claimed. □

While it is fairly immediate that our notion of evolvability itself is indeed a weak optimization procedure, the surprising consequence of this theorem is the converse, that any optimization technique that is "noise-tolerant"—or in our notation "weak"—may be leveraged by evolution.

We may thus immediately leap to what is perhaps the most powerful and robust framework for optimization: the ellipsoid method. The ellipsoid method is famously known to solve any (reasonably bounded) convex optimization problem, and in particular, its weak formulations [9]. (Specifically, both the domain and range of the functions should be bounded.) We thus have that, as long as we can arrange for $L\mathsf{Perf}$ to be convex and bounded, the associated triple $(C, \mathcal{D}, L)$ is evolvable.

As an immediate and important consequence, consider a degree d polynomial $p : \mathbb{R}^n \rightarrow \mathbb{R}^m$, with D a distribution of bounded support. Then for a hypothesis h, performance is evaluated by taking a sample $x \leftarrow D$ and evaluating $L(p(x), h(x))$. We note that if h is a degree d polynomial, considered as a vector of its $m \cdot \binom{n+d}{n}$ coefficients, then $h(x)$ is a linear function of this coefficient vector (though not linear in x!). Thus, if L is a convex function of its arguments, L will be a convex function of the coefficients of h. In short, finding the coefficients of h is a convex optimization problem when L is convex:

Theorem 2. *For any constant positive integer d and positive number r, and an arbitrary convex loss function L bounded on the radius r ball, the class of degree $\leq d$ polynomials from $\mathbb{R}^n \to \mathbb{R}^m$ with coefficients bounded by r is evolvable with respect to all distributions over the radius r ball.*

We note that the case where $m > 1$, though trivial for us to incorporate here, is in fact quite powerful for general choices of loss function L. For example, it might seem natural and sufficient to decompose a function $p : \mathbb{R}^n \to \mathbb{R}^m$ into a vector of m separate functions $\mathbb{R}^n \to \mathbb{R}$ and optimize the performance of each separately, applying the loss function to the vector where each of the other functions is assumed to take some default value, perhaps 0. However, this approach is in some sense analogous to trying to evolve walking by optimizing each leg separately, assuming each other leg were fixed immobile. Evolution seems an inherently high-dimensional problem, in many senses, thus why we emphasize the $m > 1$ case here.

We note that we insist on constant r and d in Theorem 2 because the definition of evolvability (Definition 5) insists that each parameter of performance must be bounded by a (polynomial) function of only n, m, and $\frac{1}{\epsilon}$. However, each of the parameters of evolution in fact depends as mildly as might be expected on r and d, depending polynomially on the number of coefficients needed to describe the hypothesis class of degree-d polynomials, $k = m \cdot \binom{n+d}{n}$, and on r^d, which captures the growth of the output of degree–d polynomials on inputs of magnitude up to r. This same will hold true for the main result of the next section, Theorem 3.

4 A Direct Approach

In this section we construct what is perhaps the simplest conceivable random mutator that could hope to do "evolutionary hill-climbing" (technically, in our case, the less glamorous sounding "valley descent") and show that it is in fact surprisingly adept. In particular, it is capable of efficiently evolving the same class of general multivariate polynomials as we considered at the end of the last section. While we only derive results for the case of a quadratic loss function, that is, $L(x, y) = ||x - y||_2^2$, we conjecture that similar results hold for a much wider range of loss functions, including, perhaps, any loss function $L(x, y) = ||x - y||_c$ for $c > 0$ – including specifically those functions for $c < 1$ which are not convex.

Definition 9. *The k-dimensional* wide-scale random noise *parameterized by a lower and upper bound (ℓ, u) is the result of the following process: choose a uniformly random number ρ from the interval $[\ln \ell, \ln u]$; return e^ρ times a randomly chosen element of the k-dimensional unit ball.*

Our mutation algorithm consists simply of producing several offspring each chosen by adding to the parent an independent sample of wide-scale random noise.[3] Specifically:

Definition 10. *Given a concept class of degree d polynomials from $\mathbb{R}^n \to \mathbb{R}^m$ with bounded coefficients, consider their coefficients as $k = m \cdot \binom{n+d}{n}$-dimensional vectors in $\mathbb{R}^k$. The mutation algorithm $A = (R, M)$ for wide-scale random noise is defined as:*

[3] Vitaly Feldman has pointed out [personal communication] that one can "derandomize" this procedure by instead of choosing random elements of the k-dimensional unit ball, rather taking each of the k standard unit basis vectors, and their negations. It may be, however, that in evolution randomization is the more natural.

- *R is the representation of vectors in $\mathbb{R}^k$, and*
- *M consists of generating wide-scale random noise and adding it to the parent.*

Theorem 3. *Given any positive integer constant d then, for any real number r there exist bounds ℓ, u and an integer $c = poly(n, m, r)$ such that the wide-scale random noise mutator with scale in $[\ell, u]$ and c children per generation evolves the class of degree $\leq d$ polynomials from $\mathbb{R}^n \to \mathbb{R}^m$ with coefficients at most r over the class of distributions on the n-dimensional radius r ball, with respect to the quadratic loss function.*

To prove this theorem, we first will show that progress can always be made if we choose the "right" radius, and then will observe that, because of the exponential way in which the radius is chosen, it is very likely to choose a radius that is almost exactly "right".

For the first part, we note that the expectation (over any distribution) of the quadratic loss function between a polynomial and an arbitrary function, is a positive semidefinite quadratic function of the polynomial's parameters. This is simply because, for any element in the support of the distribution, $x \in \mathbb{R}^n$, the value of the polynomial is a linear function of its coefficients; the value of the other function is fixed; and hence the square of the discrepancy between these two is positive semidefinite. Integrating these positive semidefinite functions over the distribution will thus yield a positive semidefinite function. Consider a rotation and translation of this positive semidefinite function so that it has the form $\sum_{i=1}^k c_i \cdot x_i^2$, for nonnegative c_i, where $\{x_i\}$ are a rotated and translated form of the polynomial's $k = m \cdot \binom{n+d}{n}$ parameters. Viewing the expected loss of a genome evolving in the context of Theorem 3 in this form, we show the following lemma, implying that progress can always be made:

Lemma 2. *Given $\epsilon > 0$ and a vector of non-negative coefficients, $(c_1, \ldots, c_k)$, with $\sigma = \sum_{i=1}^k c_i$, then the quadratic function $p : \mathbb{R}^k \to \mathbb{R}$ defined as $p(x) = \sum_{i=1}^k c_i \cdot x_i^2$ has the property that for any vector x of length at most 1, if $p(x) > \epsilon$ then with probability at least $\frac{1}{4}$, a randomly chosen vector y in the ball of radius $\frac{\epsilon}{6\sigma\sqrt{k}}$ about x will have $p(y) < p(x) - \frac{\epsilon^2}{12\sigma k}$.*

The restriction that x has length at most 1 is for the sake of convenience of the proof; when we apply the lemma in the context of Theorem 3, we will scale the inputs so that the radius r ball becomes a diameter 1 ball.

Proof of Lemma 2. To aid with the proof, we first note the following elementary fact: (see for example Chapter 1 of [1])

> Fact: A k-dimensional ball of unit radius has at least $\frac{1}{4}$ of its volume in the region where its first coordinate exceeds $\frac{1}{3\sqrt{k}}$.

Consider p restricted to the line connecting x to the origin. Since p has value 0 and derivative 0 at the origin, and is quadratic, it must have derivative (along this line) of $\frac{2p(x)}{|x|}$ at x; since by assumption $p(x) \geq \epsilon$ and $|x| \leq 1$, this is at least 2ϵ. Since this is just the derivative in one direction, the gradient at x must have magnitude at least this.

We further note that the second derivative in any direction is at most 2σ, from the definition of σ.

Consider the value of p in the ball of radius $r \triangleq \frac{\epsilon}{6\sigma\sqrt{k}}$ around x, and specifically, in the portion that is at least $\frac{r}{3\sqrt{k}}$ in the direction of the (downward) gradient from x. By the above fact, this portion comprises at least a quarter of the ball.

Considering the second-degree Taylor expansion of p about x—which is exact, since p is quadratic—we note that the linear contribution is a decrease of at least 2ϵ (our lower bound on the magnitude of the gradient) times the distance traveled in the direction of the gradient, namely $\frac{r}{3\sqrt{k}} = \frac{\epsilon}{18\sigma k}$, yielding $\frac{\epsilon^2}{9\sigma k}$. The quadratic contribution is bounded by $\frac{1}{2}$ times the directional second derivative in our direction times the square of the distance, which is bounded by r in our ball, yielding $\sigma r^2 = \frac{\epsilon^2}{36\sigma k}$. Subtracting yields the desired bound. □

We now assemble the pieces into a proof of Theorem 3.

Proof of Theorem 3. Let $k = m \cdot \binom{n+d}{n}$ be the dimension of our degree d polynomials when viewed as a vector space, and let b be a bound on the loss of any pair of hypotheses functions evaluated at any point in the n-dimensional radius r ball. These are both bounded by polynomials for constant d.

For any k-dimensional unit vector v, regarded as a degree d polynomial, and any point x in the n-dimensional radius r ball, the loss of an arbitrary multiple of v, αv, relative to the zero polynomial, evaluated on the point x must be a quadratic function $c\alpha^2$, for $c \leq \frac{b}{r^2}$. Thus for an arbitrary distribution in the n-dimensional radius r ball, and arbitrary target function, the expected loss will be a positive semidefinite quadratic form that can be rotated and translated into the form $\sum_{i=1}^{k} c_i x_i^2$ with each $c_i \geq 0$, and if we further scale the input by $\frac{1}{2r}$ so that the radius r ball in k dimensions maps into a region of diameter 1, then we have $\sigma \triangleq \sum_{i=1}^{k} c_i \leq 4kb$.

We thus consider the application of Lemma 2 to this transformed expected loss function. If there exists a genome in the ball with expected loss greater than ϵ, then Lemma 2 guarantees that there exists this "magic radius" $\mu = \frac{\epsilon}{6\sigma\sqrt{k}}$ such that moving the genome by a vector randomly chosen in the k-dimensional ball of radius μ will, with probability at least $\frac{1}{4}$, improve the expected loss by at least $\frac{\epsilon^2}{12\sigma k}$. Since we have polynomial upper bounds on σ, Lemma 2 thus provides for inverse-polynomial progress, in exactly those cases where we are not already within ϵ of optimal.

We note that we have already bounded $\mu \geq \frac{\epsilon}{24k\sqrt{k}}$; to upper bound μ, we note that $\sum_{i=1}^{k} c_i x_i^2$ is bounded by σ since $||x|| \leq 1$, and hence by assumption, $\sigma > \epsilon$, yielding the bound $\mu < \frac{1}{6\sqrt{k}}$. Recalling that we scaled the coordinates by a factor of $\frac{1}{2r}$ to apply Lemma 2, we have that in the original coordinates and problem setup: either evolution is within ϵ (plus sampling error) of optimal, or there is a "magic radius" between $\frac{r\epsilon}{12k\sqrt{k}}$ and $\frac{r}{3\sqrt{k}}$ such that a randomly chosen mutation in the k-dimensional ball of this radius will yield significant improvement. We thus declare the lower and upper bounds of our wide-scale random noise to be $\frac{r\epsilon}{12k\sqrt{k}}$ and $\frac{r}{3\sqrt{k}}$ respectively.

We note in general that a pair of k-dimensional balls whose radii r, r' have logarithms are within $\frac{1}{k}$ of each other will share a constant fraction of their volume. Thus with at least inverse-polynomial probability, choosing a radius that is e to the power of a number uniformly chosen between $\ln\left(\frac{r\epsilon}{12k\sqrt{k}}\right)$ and $\ln\left(\frac{r}{3\sqrt{k}}\right)$ will yield with constant probability a mutation that improves the expected loss by at least $\frac{\epsilon^2}{12\sigma k}$. The candidate pool size is chosen so that with overwhelming probability, say, at least $1 - \frac{\epsilon}{2}$, such a mutation will be present in each generation.

Thus we may choose s—the number of samples with which to evaluate the empirical performance—high enough that with probability at least $1 - \frac{\epsilon}{2}$, over the entire course of the algorithm all estimates will be accurate to within a third of the minimum improvement, $\frac{\epsilon^2}{36\sigma k}$. Further, we choose t, the threshold for declaring a mutation beneficial, to be equal to $\frac{\epsilon^2}{18\sigma k}$, so that, assuming each empirical estimate is in fact accurate to within $\frac{\epsilon^2}{36\sigma k}$, then each of the beneficial mutations guaranteed by Lemma 2 will be recognized and declared to be beneficial. Thus with probability at least $1 - \epsilon$, the performance of *every* generation will be at least $\frac{\epsilon^2}{36\sigma k}$ better than that of the previous, unless we are already within $\epsilon + \frac{\epsilon^2}{36\sigma k}$ of optimum, yielding the desired result. □

References

[1] K. Ball. An Elementary Introduction to Modern Convex Geometry. *MSRI Publications*, Volume 31, 1997.

[2] V. Feldman. Evolvability from Learning Algorithms. *STOC* 2008.

[3] V. Feldman. A Complete Characterization of Statistical Query Learning with Applications to Evolvability. *FOCS*, 2009.

[4] V. Feldman. Robustness of Evolvability. *COLT*, 2009.

[5] V. Feldman. Distribution-Independent Evolvability of Linear Threshold Functions. Manuscript, February 2011.

[6] A. Kalai and S. Vempala. Simulated Annealing for Convex Optimization. *Mathematics of Operations Research*, 31(2), 2006, pp. 253–266.

[7] V. Kanade, L. Valiant, and J. Vaughan. Evolution with Drifting Targets. *COLT*, 2010.

[8] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 25(6):983–1006, 1998.

[9] L. Lovász. *An Algorithmic Theory of Numbers, Graphs, and Convexity*, Chapter 2. SIAM, 1986.

[10] L. Michael. Evolvability via the Fourier Tranform. *Theoretical Computer Science*, to appear.

[11] L. Valiant. Evolvability. *Journal of the ACM*, 56(1), 2009.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de