Toward Machine Learning Through Genetic Code-like Transformations

Hillol Kargupta (hillol@cs.umbc.edu) and Samiran Ghosh (sghosh1@cs.umbc.edu)
Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
Baltimore, MD 21250

Abstract. The gene expression process in nature involves several representation transformations of the genome. *Translation* is one among them; it constructs the amino acid sequence in proteins from the nucleic acid-based mRNA sequence. Translation is defined by a code book, known as the universal genetic code. This paper explores the role of genetic code and similar representation transformations for enhancing the performance of inductive machine learning algorithms. It considers an abstract model of genetic code-like transformations (GCTs) introduced elsewhere (Kargupta, 2001) and develops the notion of randomized GCTs. It shows that randomized GCTs can construct a representation of the learning problem where the mean-square-error surface is almost convex quadratic and therefore easier to minimize. It considers the functionally complete Fourier representation for Boolean functions to analyze this effect of such representation transformations. It offers experimental results to substantiate this claim. It shows that a linear classifier like the Perceptron (Rosenblatt, 1961) can learn non-linear XOR and DNF functions using a gradient-descent algorithm in a representation constructed by randomized GCTs. The paper also discusses the immediate challenges that must be solved before the proposed technique can be used as a viable approach for representation construction in machine learning.

Keywords: Genetic Code, gene expression, representation construction, machine learning.

1. Introduction

Gene expression involves a series of representation transformations that converts the information coded in the DNA into proteins. These transformations play an important role in constructing the phenotype from the genotype. Representation transformations are frequently used in science, engineering, and business as an efficient problem solving technique. So it is natural to wonder if gene expression plays any role in efficient genetic search and problem solving. This paper explores a particular stage of gene expression, called translation. The translation process in gene expression uses the genetic code to construct an amino-acid-based representation of the genome from the messenger RNA, a sequence of nucleic acids. This code, known as the universal genetic


© 2002 *Kluwer Academic Publishers. Printed in the Netherlands.*

code, is defined by a small redundant table that assigns an amino acid for every three consecutive nucleic acids in the mRNA. This paper explores the computational role of such codes.

This paper investigates the role of genetic code-like transformations (GCTs) (Kargupta, 2001) in learning functions from non-uniformly distributed data. It considers the supervised learning problem where the objective is to learn a function from a given training data set with unknown distribution. It introduces randomized GCTs and shows that they can be effectively used to construct a representation of the learning problem where the error surface is almost convex quadratic and therefore easier to minimize. It also shows that a non-linear function like XOR can be learned using a linear classifier like Perceptron (Rosenblatt, 1961) and the well-known Perceptron learning algorithm that performs greedy error minimization in a representation constructed by randomized GCTs.

Section 2 reviews the biological background and the related work. Section 3 presents an inductive function learning problem from the perspective of representation construction. Section 4 introduces randomized GCTs and presents a Fourier analysis of such transformations. It points out that these transformations construct a representation of the sample data where the higher order Fourier basis vectors become almost orthogonal. This results in a new representation of the learning problem where the mean-square-error surface is a "quasi-quadratic" function that is easier to minimize. Section 5 describes the Perceptron and the XOR problem. It presents the effect of such transformations on learning the XOR problem using a Perceptron. It shows that the gradient descent-based Perceptron learning law can successfully learn the XOR problem in a representation constructed by the randomized GCTs. Section 6 presents similar experimental results with DNF functions. Section 7 discusses the results presented in this paper and identifies future work.

2. Background

This paper considers the problem of representation construction in learning functions from data and explores genetic code-like transformations from this perspective. This section presents the background material. It first offers a brief review of the universal genetic code used in a living organism during the process of translation. Next it summarizes existing work on the role of gene expression in evolutionary search.

2.1. Biological Motivation

The work presented in this paper is motivated by the gene expression process in nature. In order to fully appreciate the contribution we need to understand the biological process to some extent. So let us first review the biology.

The gene expression process in nature involves a series of representation transformations. It starts by first transforming the DNA sequence to the mRNA. DNA is a sequence of nucleotides. It uses four different types of nucleotides namely, *adenine* (A), *thiamin* (T), *guanine* (G), and *cytosine* (C); mRNA is also a sequence of different nucleotides. However, the set of nucleotides for mRNAs is different from that of DNA. In case of mRNA the nucleotides are *adenine* (A), *uracil* (U), *guanine* (G), and *cytosine* (C). This step is called the *Transcription*. Next the mRNA sequence is transformed into protein, a sequence of amino acids. This step is known as *Translation*. It uses a code-book that defines the correspondence between nucleotide triplets (known as *codons*) in the mRNA and the amino acids in proteins. This code-book is known as the *genetic code* (Table I). Each codon is comprised of three adjacent nucleotides in an mRNA chain and it produces an amino acid. The translation process takes an mRNA sequence, replaces every three adjacent nucleic acids (every codon) by the corresponding amino acid (listed in Table I), and produces the amino acid sequence. With a few exceptions, the genetic code for most eukaryotic and prokaryotic organisms is the same.

Proteins, mRNAs, and the DNA are different representations of the genome. Proteins control almost every important activity in a living body. For some reason, a living body chooses the representation of the genome defined by the proteins for all the important tasks. So it is quite natural to wonder about the reason behind choosing this special representation that requires a sequence of representation transformations.

This paper considers the genetic code-based translation process that transforms the mRNA sequence to the amino acid sequence in proteins. It suggests that genetic code-like transformations (GCTs) may have an important role in learning and adaptation. It develops randomized GCT-s that may be used to efficiently learn non-linear functions from data. The following section reviews some of the existing work on representation construction techniques that are motivated by natural gene expression, evolution of genetic codes, and other related machine learning techniques.

Table I. The universal genetic code.

Protein feature	mRNA codons
Alanine	GCA GCC GCG GCU
Cysteine	UGC UGU
Aspartic acid	GAC GAU
Glutamic acid	GAA GAG
Phenylalanine	UUC UUU
Glycine	GGA GGC GGG GGU
Histidine	CAC CAU
Isoleucine	AUA AUC AUU
Lysine	AAA AAG
Leucine	UUA UUG CUA CUC CUG CUU
Methionine	AUG
Asparagine	AAC AAU
Proline	CCA CCC CCG CCU
Glutamine	CAA CAG
Arginine	AGA AGG CGA CGC CGG CGU
Serine	AGC AGU UCA UCC UCG UCU
Threonine	ACA ACC ACG ACU
Valine	GUA GUC GUG GUU
Tryptophan	UGG
Tyrosine	UAC UAU
STOP	UAA UAG UGA

2.2. Previous Work

The importance of gene expression in genetic search was realized in the early days of the field of genetic algorithms. Holland (Holland, 1975) described the dominance operator as a possible way to model the effect of gene expression in diploid chromosomes. He also noted the importance of the process of protein synthesis from DNA in the computational model of genetics. Despite the fact that, traditionally dominance maps are explained from the Mendelian perspective, Holland made an interesting leap by connecting it to the synthesis of protein by gene signals, which today is universally recognized as gene expression. He noted the relation between the dominance operator with the "operon" model of the functioning of the chromosome (Jacob, 1961) in evolution and pointed out the possible computational role of gene signaling in evolution (Holland, 1975).

Several other efforts have been made to model some aspects of gene expression. Diploidy and dominance have also been used elsewhere (Bagley, 1967; Brindle, 1981; Hollstien, 1971; Rosenberg, 1967; Smith, 1988). Most of them took their inspiration from the Mendelian view of genetics. The under-specification and over-specification decoding operator of messy GA has been viewed as a mechanism similar to gene signaling (Goldberg, Korb and Deb, 1989). The structured genetic algorithm (Dasgupta, 1992) also shares motivations from the gene expression; it uses a structured hierarchical representation in which genes are collectively switched on and off. This provides the search algorithm with a richer representation and helps capturing properties of the landscape better. The role of genotype-phenotype-mapping in the context of neutral variation and genetic programming is discussed in (Banzhaf, 1994). An empirical study of genetic programming using artificial genetic code is presented in (Keller and Banzhaf, 1999). Kauffman (Kauffman, 1993) offered an interesting perspective of the natural evolution that realizes the importance for gene expression. However, Kauffman's work does not explain the process in basic computational terms on analytical grounds and does not relate the issue to the complexity of search in quantitative terms. The complex nature of the representation in the DNA itself created interest among the researchers. The eukaryotic DNA typically contains many segments that are not used in the gene expression process for producing proteins. An empirical investigation of the role of such "non-coding" segments (introns) in genetic search can be found in (Wu and Lindsay, 1995). A survey of evolutionary algorithms with intron-based representations is presented in (Wu and Lindsay, 1996). Grammatical evolution (O'Neill and Ryan, 1999; Ryan, Collins, and O'Neill, 1998) for evolving programs is also motivated by the gene expression process. Another evolutionary algorithm, called the gene expression programming, motivated by natural gene expression, is reported elsewhere (Ferreira, 2001).

The "neutral network" theory (Reidys and Fraser, 1996; Schuster, 1997) considers sequence-to-structure mapping from the perspective of random graph construction. This work approaches gene expression from the perspective of random graph construction and points out existence of the fitness invariant neutral networks. The translation process maps multiple mRNA sequences to the same protein sequence. As a result, it creates a set of different genomes with same fitness, termed as neutral networks. This work provides interesting insights into the effect of such neutral networks in genetic search. However, its contribution toward inductive function learning and computationally efficient representation construction of genetic fitness function is not clear. Another related effort to understand the properties of the fitness landscape defined by

the mRNA can be found in (Rockmore, Kostelec, Hordijk and Stadler, 1999). This work presents a Fourier analysis of the landscapes derived from the RNAs using Fast Fourier Transformation (FFT).

There also exists a body of literature that investigates the evolution of the genetic code. An algebraic model of the evolution of the genetic code is presented in (Hornos and Hornos, 1993). This work searches for symmetries in the genetic code and points out the existence of a unique approximate symmetry group compatible with the codon assignments. The main idea behind this work is to view the evolution of the genetic code as an iterative process of representation decomposition. The genetic code is viewed as a 64-dimensional representation decomposed into several sub-representations with respect to different subgroups. The number of amino acids correspond to the number of sub-representations and the number of codons for any amino acid corresponds to the dimension of that sub-representation. An extension of this work using Lie super-algebra is presented in (Bashford, Tsohantjis and Jarvis, 1998). Additional work on the different biological theories on the evolution of the genetic code can be found elsewhere (Beland and Allen, 1994; Freeland, Knight, Landweber and Hurst, 2000; Fukuchi, Okayama and Otsuka , 1994; Knight and Landweber, 2000).

An alternate approach has been developed by Kargupta and his colleagues (Kargupta, 1996; Kargupta, 1997; Kargupta, 1996; Kargupta, 1997; Kargupta, 1997; Kargupta, 1998; Kargupta, 1999; Kargupta, 2001). This approach is mainly motivated by a perspective of the gene expression as a mechanism to make genetic search more efficient. This approach notes that the traditional model of evolutionary computation (based on selection, crossover, and mutation)(Holland, 1975) appears to have some serious scalability problems (Thierens, 1999) for reasonably difficult problems. There is also little theoretical result available that proves guaranteed polynomial time performance of existing evolutionary algorithms for reasonably difficult classes of problems. Since the existing models of evolutionary computation do not address the gene expression issue very well and gene expression changes the genetic representation, it may become a natural candidate for exploring the unknown mechanism that makes the genetic search in nature so efficient and scalable.

The early exploration of gene expression-like mechanisms for efficient inductive detection of function structure resulted in a class of heuristics-based techniques, known as the gene expression messy GA (GEMGA) (Kargupta, 1997). More rigorous approaches using Fourier basis representations were recently suggested. Fourier representations exposes the underlying function structure and it is functionally complete. Therefore, efficient learning of such representations can be very

useful for function induction. A randomized algorithm is presented in (Kargupta, 1999; Kargupta, 2000) that can induce a representation in Fourier basis in polynomial time for problems with bounded variable interaction. An alternate technique for estimating the Fourier representations is proposed elsewhere (Jackson, 1995; Kushilevitz and Mansou, 1991). An extension of this technique for detecting function structure in genetic algorithms is reported in (Thierens, 1999).

The research presented in (Kargupta, 2001) is directly related to the current work. The abstract concept of GCTs is introduced there. That paper also explores the role of GCT-s from the perspective of function learning. It considers a few GCTs and shows that some of them can transform an exponentially long Fourier representation to one that can be accurately approximated by only a polynomial number of low order Fourier coefficients. In this paper we consider randomized GCTs and show that they can reformulate inductive learning problems, making them easier to solve. The following section formulates the problem and presents the analytical motivations behind the work.

3. Learning as Representation Construction

Consider a predictive learning problem where the goal is to induce a function $\hat{f} : \Omega \rightarrow I\!R$ from the training data set $S = \{(\mathbf{x}_{(1)}, y_{(1)}), (\mathbf{x}_{(2)}, y_{(2)}), \cdots (\mathbf{x}_{(m)}, y_{(m)})\}$ generated by underlying function $f : \Omega \rightarrow I\!R$, such that $\hat{f}$ approximates f. Any member of the domain Ω, $\mathbf{x} = x_1, x_2, \cdots x_\ell$ is an ℓ-tuple from a discrete space where x_j can take Λ distinct values. In this paper we shall restrict ourselves to learning discrete functions.

In order to express a function we first need to choose a representation. For example, a linear function can be expressed using a representation that takes the form $f(x) = a_0 + a_1 x_1 + a_2 x_2 + \cdots a_\ell x_\ell$. The same function can be expressed in different forms in different representations. For example, the function $logx + x$ can be expressed using a representation comprised of functions $logx$ and x; on the other hand, it can also be represented in the form of a series.

In this paper, we shall consider functions that can be expressed in the form of $f(\mathbf{x}) = \sum_{\mathbf{j}} w_{\mathbf{j}} \psi_{\mathbf{j}}(\mathbf{x})$; where $\psi_{\mathbf{j}}(\mathbf{x})$-s are different functions of $\mathbf{x}$ that can be used to define the representation of $f(\mathbf{x})$; $w_{\mathbf{j}}$-s are constant coefficients. In case of the linear function $\psi_j(\mathbf{x}) = x_j$. For example, when $f(x) = logx + x$ we can represent it using $\psi_1(\mathbf{x}) = logx$, $\psi_2(\mathbf{x}) = x$, and $w_1 = w_2 = 1$. One can actually choose a set of such functions $\psi_{\mathbf{j}}(\mathbf{x})$-s in an appropriate fashion for representing any arbitrary function. For example, in Fourier representation over Boolean

strings any function $f : \{0,1\}^\ell \rightarrow I\!R$ can be expressed using 2^ℓ unique Fourier basis functions. Although such rich representations are capable of expressing any function, restricted representations are frequently used for faster and efficient learning.

For the time being let us consider representations defined by functions $\psi_{\mathbf{j}} : \Omega \rightarrow I\!R$. The objective of the learning process is to construct a function $\hat{f}(x) = \sum_{\mathbf{j}} \hat{w}_{\mathbf{j}} \psi_{\mathbf{j}}(\mathbf{x})$ from data generated by the target function $f(x) = \sum_{\mathbf{j}} w_{\mathbf{j}} \psi_{\mathbf{j}}(\mathbf{x})$. For any given ordered training data set S, we can compute the column vector $(\Psi_{\mathbf{j}}(\mathbf{x}))_S = [\psi_{\mathbf{j}}(\mathbf{x}_{(1)})\psi_{\mathbf{j}}(\mathbf{x}_{(2)}) \cdots \psi_{\mathbf{j}}(\mathbf{x}_{(m)})]^T$. The inner product between two column vectors is defined by $(\Psi_{\mathbf{j}}(\mathbf{x}), \Psi_{\mathbf{k}}(\mathbf{x}))_S = (\Psi_{\mathbf{j}}(\mathbf{x}))_S^T (\Psi_{\mathbf{k}}(\mathbf{x}))_S$.

The square-error introduced by the learned function over the entire training data set S can be derived as follows:

$$\begin{aligned} f(\mathbf{x}) - \hat{f}(\mathbf{x}) &= \sum_{\mathbf{j}} (w_{\mathbf{j}} - \hat{w}_{\mathbf{j}}) \psi_{\mathbf{j}}(\mathbf{x}) \\ \sum_{\mathbf{x} \in S} (f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2 &= \sum_{\mathbf{j},\mathbf{k}} (w_{\mathbf{j}} - \hat{w}_{\mathbf{j}})(w_{\mathbf{k}} - \hat{w}_{\mathbf{k}}) \times \\ &\quad \sum_{\mathbf{x} \in S} \psi_{\mathbf{j}}(\mathbf{x}) \psi_{\mathbf{k}}(\mathbf{x}) \\ &= \sum_{\mathbf{j},\mathbf{k}} (w_{\mathbf{j}} - \hat{w}_{\mathbf{j}})(w_{\mathbf{k}} - \hat{w}_{\mathbf{k}}) \times \\ &\quad (\Psi_{\mathbf{j}}(\mathbf{x}), \Psi_{\mathbf{k}}(\mathbf{x}))_S \end{aligned} \tag{1}$$

If the column vectors $((\Psi_{\mathbf{j}}(\mathbf{x}))_S$are orthogonal to each other then the mean-square-error surface becomes interesting. In the general case these vectors may not be orthogonal. However, for the time being let us assume that the column vectors are indeed orthogonal in our data set. Later in this paper we shall identify a technique to approximately satisfy this condition. Let us also assume that all these orthogonal vectors have a magnitude equal to 1; in other words, they are orthonormal. This means $(\Psi_{\mathbf{j}}(\mathbf{x}), \Psi_{\mathbf{k}}(\mathbf{x}))_S = 0$ for all $\mathbf{j} \neq \mathbf{k}$ and $(\Psi_{\mathbf{j}}(\mathbf{x}), \Psi_{\mathbf{j}}(\mathbf{x}))_S = |S|$. We can write using Equation 1,

$$\sum_{\mathbf{j}} (w_{\mathbf{j}} - \hat{w}_{\mathbf{j}})^2 = \frac{1}{|S|} \sum_{\mathbf{x} \in S} (f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2 \tag{2}$$

If we take the partial derivative of the left side of the equation with respect to $\hat{w}_{\mathbf{j}}$ and set it equal to zero, we get $\hat{w}_{\mathbf{j}} = w_{\mathbf{j}}$. This is the global error minima. So the mean-square-error (MSE) surface is convex quadratic in $\hat{w}_{\mathbf{j}}$ and minimization in such surfaces is relatively easier. We can use an existing quadratic optimization algorithm to minimize

the MSE to some δ. Now we can write that for any $\mathbf{j}$,

$$\begin{aligned}(w_{\mathbf{j}} - \hat{w}_{\mathbf{j}})^2 &\leq \delta \\ |w_{\mathbf{j}} - \hat{w}_{\mathbf{j}}| &\leq \sqrt{\delta} \end{aligned} \tag{3}$$

This provides us a bound on the error in estimating the different coefficients defining the target function in terms of the MSE.

Minimization of MSE over the training data set S may not necessarily guarantee good performance over the data that are not in S. This is the generalization aspect of a learning algorithm. Generalization is an important property of the learned model. Usually machine learning and statistical data modeling algorithms check the generalization capability of the learned model by testing it on data sets that are different from the training data set (e.g. cross-validation). Next, we present a result on the generalization capability of the model learned by MSE minimization in the orthonormal representation.

The bound on the difference between the exact coefficients and their estimates identified in Equation 3 intuitively tells us that the performance of the learned model on any given testing data set should be good as long as the MSE on the learning data set is low. But we can also quantify this and identify an interesting scenario when the column vectors $(\Psi_{\mathbf{j}}(\mathbf{x}))_\Omega$-s defined over the entire domain Ω are orthonormal to each other. In other words $(\Psi_j(\mathbf{x}), \Psi_k(\mathbf{x}))_\Omega = 0$. Let $S_x = \{\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \cdots \mathbf{x}_{(m)}\}$ such that $S = \{(\mathbf{x}_{(1)}, y_{(1)}), (\mathbf{x}_{(2)}, y_{(2)}), \cdots (\mathbf{x}_{(m)}, y_{(m)})\}$.

So we can write the following.

$$\begin{aligned}(\Psi_{\mathbf{j}}(\mathbf{x}), \Psi_{\mathbf{k}}(\mathbf{x}))_\Omega &= (\Psi_{\mathbf{j}}(\mathbf{x}), \Psi_{\mathbf{k}}(\mathbf{x}))_S + \\ &\quad (\Psi_{\mathbf{j}}(\mathbf{x}), \Psi_{\mathbf{k}}(\mathbf{x}))_{\Omega - S_x} \\ &= 0\end{aligned}$$

for all $\mathbf{j} \neq \mathbf{k}$. Now note that, $(\Psi_{\mathbf{j}}(\mathbf{x}), \Psi_{\mathbf{k}}(\mathbf{x}))_S = 0$ by assumption. Therefore, $(\Psi_{\mathbf{j}}(\mathbf{x}), \Psi_{\mathbf{k}}(\mathbf{x}))_{\Omega - S_x} = 0$. Therefore, following Equation 1 we can write,

$$\begin{aligned}\frac{1}{|\Omega - S_x|} \sum_{\mathbf{x} \in \Omega - S_x} (f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2 &= \sum_{\mathbf{j}} (w_{\mathbf{j}} - \hat{w}_{\mathbf{j}})^2 \\ &\leq \delta \end{aligned} \tag{4}$$

This shows that the MSE over the entire portion of the domain members should be equal to δ, the MSE over the training data. This also implies the following bound on the MSE over any subset (Q) of $\Omega - S_x$.

$$\frac{1}{|Q|} \sum_{\mathbf{x} \in Q \subseteq \Omega - S_x} (f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2 \leq \frac{\delta |\Omega - S_x|}{|Q|} \tag{5}$$

The bound on the average error over $\Omega - S_x$ is an interesting property. It tells us that the learned model will come with a performance bound for the entire data set that is not presented to the algorithm during the learning stage, as long as the representation is orthonormal over the given training data set and the entire domain. This observation is indeed rational. For example, consider a linear model constructed from real valued continuous data in absence of noise. Since the domain is comprised of real-valued continuous features, it contains infinitely many unique data points. However, if the underlying target function generating the data is linear, we need no more than $\ell + 1$ samples to learn the $\ell + 1$ unknown coefficients of the exact model comprised of ℓ linear terms and a constant. Equation 5 makes a very similar claim.

The analysis presented so far made the following two main assumptions

1. $(\Psi_{\mathbf{j}}(\mathbf{x}))_S$-s are orthonormal. This resulted in a quadratic MSE over the training data set S.

2. $(\Psi_{\mathbf{j}}(\mathbf{x}))_\Omega$-s are orthonormal. This produced a bound on the generalization capability.

The analysis also assumed that the set of functions ($\psi_{\mathbf{j}}(\mathbf{x})$-s) is rich enough to represent the target function.

Choosing a representation where $(\Psi_{\mathbf{j}}(\mathbf{x}))_\Omega$-s are orthonormal to each other is relatively easy. There exist several known types of basis functions (e.g. Fourier, Walsh, and Wavelet) (Beauchamp, 1984; Walsh, 1923; Wickerhauser, 1994) that satisfy this result. However, constructing a representation where $(\Psi_{\mathbf{j}}(\mathbf{x}))_S$-s are orthonormal to each other is non-trivial. The process is further complicated by the fact that we need to construct the orthonormal representation in such a way that the learned model can be efficiently used for testing and new prediction or classification.

In rest of this paper we propose a novel approach that applies a randomized transformation $\eta(\mathbf{x})$ on every member $\mathbf{x} \in S$ and generates a new representation $\mathbf{x}' \in S'$. We show that in a functionally complete set of $\psi_{\mathbf{j}}(\mathbf{x})$-s the randomized transformation $\eta(\mathbf{x})$ constructs a representation where most of the $\psi_{\mathbf{j}}(\mathbf{x})$-s are almost orthonormal to each other over the training data set S'. We note that this class of transformations has strong similarities with the universal genetic code used in a living organism for producing the amino acid-based representation of proteins from the mRNA sequence. The following section defines the genetic code-like transformations considered in this paper.

Table II. Code A: A GCT in binary representation. Single bit in the protein space maps to 3-bit codons in the mRNA space.

Protein feature	mRNA codon
1	100, 000, 001, 010
0	111, 101, 110, 011

4. Genetic Code-like Transformations

Table III. Code B: A GCT in binary representation. Single bit in the protein space maps to 3-bit codons in the mRNA space. Note that it is different from the Code A presented in Table II. It assigns unequal number of codons to the protein features.

Protein feature	mRNA codon
0	100, 000, 001, 010, 111, 101, 110
1	011

In nature the universal genetic code (Table I) assigns an amino acid for every consecutive nucleic acids in the mRNA sequence. In the following discussion we shall develop an abstract class of such transformations that share some similarities with the genetic code. The objective of this formal abstraction is to analyze the effect of such transformations on the representation and to eventually show that they can be effectively used to construct the $\eta(\mathbf{x})$ introduced in Section 2. The nucleotides and the amino acids that define the natural genetic code are physical entities comprised of physical and chemical properties. This paper simplifies this situation and creates an abstract world where the nucleotides and the amino acids are numbers. In other words, we use their symbolic representation just like any Biology text would. We just choose to use integer numbers instead of alphabet symbols. We shall also restrict the presentation to Boolean representations. However, the analysis can be easily extended to non-Boolean representations. Our analysis also considers the transformation from a non-traditional direction. It considers proteins as the primary representation of the phenotype as far as this paper is concerned. It considers the mRNA as the secondary

representation constructed from the proteins. Although in nature the transformations are applied in the DNA→mRNA→protein direction the sequence of the development of each of these transformations is not obvious. For example, many biologists believe that the RNA evolved before the DNA. So we orient our exploration by setting the coordinate reference to the phenotype. Since proteins define the representation that is closest (compared to DNA and mRNA) to the phenotype we treat it as the original representation $\mathbf{x}$ of the target function $f(\mathbf{x})$. The genetic code is considered as a transformation that constructs the mRNA-based new representation of the proteins.

Let $\mathbf{r}$ and $\mathbf{p}$ be the ℓ_r-bit mRNA and the corresponding translated ℓ_p-bit protein sequences. Just like the natural *translation* process, our artificial translation maps the mRNA sequence to the corresponding protein sequence using a code-book, which we call a genetic code-like transformation, in short a GCT. This transformation is denoted by $\eta_c^{'}$ where the subscript c denotes the number of mRNA features that define a codon. If three features are used like natural codons, $c = 3$; $\eta_c^{'}$ can be defined as $\eta_c^{'} : R^{\ell_r} \rightarrow P^{\ell_p}$. R^{ℓ_r} and P^{ℓ_p} denote the ℓ_r and ℓ_p dimensional space of all mRNAs and proteins respectively. Note that $\ell_r = c\,\ell_p$. Moreover, in our binary case, $R = P = \{0, 1\}$.

Tables II and III show two examples of GCTs introduced elsewhere (Kargupta, 2001). Note that a GCT may be redundant just like the universal genetic code. In other words, a unique protein feature value may be produced by several mRNA codons. As a result, there exist many equivalent mRNA sequences that produce the same protein sequence. All these mRNA sequences have the same genetic fitness since they all map to the same protein sequence. So we can view the space of mRNAs grouped into different equivalence classes. Following (Kargupta, 2001) we shall call this characteristic *Translation Introduced Equivalence* (TIE) and these groups of equivalent mRNAs will be called the TIE classes. Let R_p be the TIE class for the protein sequence $\mathbf{p}$. In other words, $R_p = \{\mathbf{r}_j | \mathbf{r}_j \stackrel{\eta_c^{'}}{\rightarrow} \mathbf{p}\}$. The cardinality of the set R_p depends on the genetic code and the protein sequence $\mathbf{p}$. Let a_0 and a_1 be the total number of codons that map to a protein feature value of 0 and 1 respectively. Let $\ell_{p,0}$ and $\ell_{p,1}$ be the number of 0-s and 1-s in $\mathbf{p}$ respectively. For Boolean strings, $\ell_{p,0} + \ell_{p,1} = \ell_p$. Then the cardinality of the TIE class of protein $\mathbf{p}$ is $|R_p| = a_0^{\ell_{p,0}} a_1^{\ell_{p,1}}$.

The transformation $\eta_c^{'}$ that converts an mRNA sequence to a protein sequence is deterministic. However, a single protein sequence corresponds to a set of different mRNA sequences. The following section considers probabilistic selection of one mRNA sequence from the set of all mRNA sequences in the TIE class of a protein sequence. It introduces

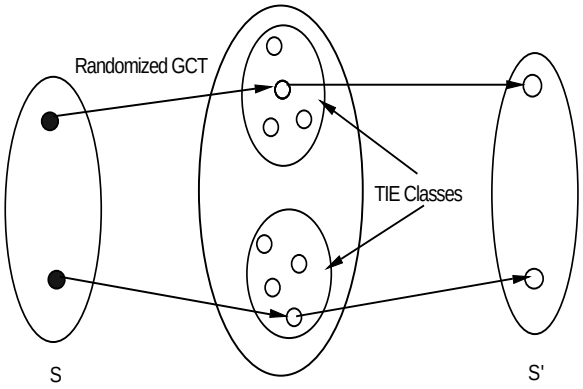

Figure 1. Construction of a new representation of the training data set S by applying a randomized GCT. The entire domain Ω can also be transformed in a similar manner for constructing the corresponding Ω'.

randomized GCTs that transform a string $\mathbf{x}$ to $\mathbf{x}'$ by probabilistically assigning codons from the different possible choices associated with a single value of a protein feature.

4.1. Randomized GCT-s

Redundancy is an important characteristic of GCTs. Redundancy forces us to make a choice from different possible codons while constructing the mRNA sequence from a given amino acid sequence in a protein. We can make this decision probabilistic. For example, an mRNA sequence can be generated from a given protein sequence by assigning codons with a certain probability distribution. Randomized GCTs do exactly that.

A randomized GCT is a GCT where the codons are associated with a certain probability distribution. Given an amino acid sequence we generate the corresponding mRNA sequences by simply following this probability distribution assigned over the codons. In the following discussion we shall consider such randomized GCTs where each codon has uniform probability of occurrence.

Before moving further ahead let us take an example. Consider the protein sequence 10 that needs to be transformed to the mRNA space using code A (Table II). We have four codons associated with an 1 and we choose one among them with uniform probability. Let us say we pick 000. Similarly, we choose another codon for the last bit 0 in the protein sequence 10 and let that codon be 101. The resulting transformed string is therefore 000101. A randomized GCT works in this fashion in order to produce a unique ℓ_r-bit representation of an ℓ_p-bit protein. We can

construct a new representation (S') of the members of the training data set S by applying this transformation. The entire domain (Ω) of the target function can also be converted to $\Omega^{'}$ in a similar manner. Figure 4.1 shows this construction process graphically. The following section explores the representational properties of such transformations using multi-variate Fourier analysis.

4.2. The Fourier Representation and GCT-s

Recall that our earlier discussion on function induction and the convex quadratic formulation of the MSE (in Section 3) made two assumptions regarding the representation: (1) orthonormality of the basis set over the complete domain Ω and (2) orthonormality over the training data set.

As we noted earlier, it is relatively straight forward to come up with a representation where the basis vectors are orthonormal over the complete domain. There are several possibilities. In this paper we use the discrete Fourier representation that satisfies this property. Moreover, it is functionally complete; in other words any discrete function can be represented using Fourier basis.

The Fourier basis set over an ℓ-bit domain contains 2^ℓ orthogonal Fourier functions. Each Fourier basis function is defined as $\psi_{\mathbf{j}}(\mathbf{x}) = (-1)^{(\mathbf{x}\cdot\mathbf{j})}$. Where $\mathbf{j}$ and $\mathbf{x}$ are binary strings of length ℓ. In other words $\mathbf{j} = j_{(1)}, j_{(2)}, \cdots j_{(\ell)}$, $\mathbf{x} = x_{(1)}, x_{(2)}, \cdots x_{(\ell)}$ and $\mathbf{j}, \mathbf{x} \in \{0,1\}^\ell$; $\mathbf{x}\cdot\mathbf{j}$ denotes the inner product of $\mathbf{x}$ and $\mathbf{j}$. $\psi_{\mathbf{j}}(\mathbf{x})$ can either be equal to 1 or -1. The string $\mathbf{j}$ is called a *partition*. The *order* of a partition $\mathbf{j}$ is the number of 1-s in $\mathbf{j}$. A function $f : \mathbf{X}^\ell \rightarrow \{0,1\}$, that maps an ℓ-dimensional space of binary strings to a 0 or 1, can be written using the Fourier basis functions $f(\mathbf{x}) = \sum_{\mathbf{j}} w_{\mathbf{j}} \psi_{\mathbf{j}}(\mathbf{x})$. where $w_{\mathbf{j}}$ is the Fourier Coefficient corresponding to the partition $\mathbf{j}$; $w_{\mathbf{j}} = \frac{1}{2^\ell} \sum_{\mathbf{x}} f(\mathbf{x}) \psi_{\mathbf{j}}(\mathbf{x})$.

We can represent the function $f(\mathbf{x})$ using the Fourier bases in either the mRNA or the protein space. The partitions defined in these two spaces are related. Since one feature in the protein sequence maps to c mRNA features, partitions defined in the mRNA and the protein spaces can be associated with each other. If $\mathbf{j}$ and $\mathbf{j}'$ be partitions in the mRNA and the protein spaces respectively then $\mathbf{j}'$ is the *reflection* of $\mathbf{j}$ in the protein space when $\mathbf{j}^{'}_{(i)} = 1$ if and only if $\mathbf{j}$ takes a value of 1 at the location(s) corresponding to at least one of the mRNA features associated with $\mathbf{j}^{'}_{(i)}$. The order of $\mathbf{j}'$ is called the *absolute order* of partition $\mathbf{j}$.

For example, the reflection of the partition $\mathbf{j} = 101000$ using a genetic code of codon size three is $\mathbf{j}' = 10$. The left three bits of $\mathbf{j}$ are associated with the leftmost bit of $\mathbf{j}'$. Since two of those three bits

are set to 1, $\mathbf{j}'_{(0)} = 1$. However, none of the rightmost three bits in $\mathbf{j}$ takes the value 1. So the corresponding $\mathbf{j}'_{(1)} = 0$. Note that the reflection of 100000 is also 10 since $\mathbf{j}'_{(0)} = 1$ as long as at least one of the leftmost three bits is set to 1. Similarly the reflection of 100110 under a genetic code of codon size three is 11.

4.3. Fourier Analysis of Randomized GCT-s

The mRNA-features corresponding to the positions with 1-s in the partition $\mathbf{j}$ may belong to the (1) same mRNA codon, (2) different codons, and (3) a combination of both. In other words they originate from the (1) the same protein feature (since one feature in the protein sequence maps to c features in the mRNA sequence) or (2) different protein features or (3) a combination of both respectively.

Let $\mathbf{j}$ be a partition in the mRNA space with absolute order q and $\mathbf{j}'$ be its reflection in the protein space. The protein space is defined by an ℓ-bit strings and the codon size is c.

Let $t_{(1)}, t_{(2)}, \cdots t_{(q)}$ be the location of the q fixed bits in the partition $\mathbf{j}'$; τ_i be the Boolean string with all zeros except the c bits associated with the location $t_{(i)}$ which are all set to 1. Now let us define a sub-partition in the mRNA space $\mathbf{j}_i$ as a $c\ell$-bit partition generated by the bit-wise AND operation between $\mathbf{j}$ and τ_i.

So we can represent $\mathbf{j}$, a partition in the mRNA space, using a collection of partitions $\{\mathbf{j}_0, \mathbf{j}_1, \cdots \mathbf{j}_q\}$ where $\mathbf{j}_0$ represents the null partition with all 0-s; every $\mathbf{j}_{i\neq 0}$ represents a sub-partition of the 1-contributing positions of $\mathbf{j}$ that contains only those features that belong to the same protein feature. Note that the reflection of any $\mathbf{j}_{i\neq 0}$ in the protein space has only one 1. The null partition always contribute a value of 1 and it is introduced only for taking care of the case where the partition $\mathbf{j}$ is a sequence of all 0-s. For example, consider a two-bit protein space that is associated with a six-bit mRNA space. The partition 110001 in the mRNA space can be represented in terms of the sub-partitions 000000, 110000, and 000001. Note that $\psi_{110001}(\mathbf{r}) = \psi_{000000}(\mathbf{r})\psi_{110000}(\mathbf{r})\psi_{000001}(\mathbf{r})$. So we can write,

$$\psi_{\mathbf{j}}(\mathbf{r}) = \prod_{\alpha=0,1,\cdots q} \psi_{\mathbf{j}_\alpha}(\mathbf{r}). \tag{6}$$

Now note that the value of $\psi_{\mathbf{j}}(\mathbf{r})$ will be -1 when an odd number of $\psi_{\mathbf{j}_\alpha}(\mathbf{r})$-s take a value of -1. The value of $\psi_{\mathbf{j}_\alpha}(\mathbf{r})$ depends on the codon bits corresponding to the partition $\mathbf{j}_\alpha$. It can either take a value of 1 or -1. In case of a randomized GCT, this is a probabilistic event. In other words, $\psi_{\mathbf{j}_\alpha}(\mathbf{r})$ can either be 1 or -1 with a certain probability

distribution. This distribution can be computed from the code-book. Consider applying a code to any protein sequence $\mathbf{p}$ in a randomized fashion with uniform probability assigned to the codons. Let $P_{\alpha,\mathbf{r}}$ be the probability that $\psi_{\mathbf{j}_\alpha}(\mathbf{r})$ will be -1 after $\mathbf{r}$ is constructed by applying the randomized GCT from some $\mathbf{p}$. Subscript $\alpha, \mathbf{r}$ represents the dependency of this probability on both $\mathbf{j}_\alpha$ and $\mathbf{r}$.

The probability that an odd number of $\mathbf{j}_\alpha$-s take a value of -1 is,

$$\begin{aligned} P_{-1} &= \sum_{k=1}^{q} \binom{q}{k} P_{\alpha,\mathbf{r}}^k (1 - P_{\alpha,\mathbf{r}})^{q-k} (k \bmod 2) \\ &= \frac{1 - (1 - 2P_{\alpha,\mathbf{r}})^q}{2} \end{aligned} \tag{7}$$

Note that $P_{\alpha,\mathbf{r}}$ is a property of the code-book. For example in code A (Table II), $P_{\alpha,\mathbf{r}} = 0.5$ or $P_{\alpha,\mathbf{r}} = 0.75$ for any α and $\mathbf{r}$. Equation 7 is true only when $P_{\alpha,\mathbf{r}}$ is constant across every $\mathbf{j}_\alpha$.

The probability that an even number of $\mathbf{j}_\alpha$-s take a value of -1 is $1 - P_{-1}$. For $0 < P_{\alpha,\mathbf{r}} < 1.0$, we can write $\lim_{q\to\infty} P_{-1} = \lim_{q\to\infty} \frac{1-(1-2P_{\alpha,\mathbf{r}})^q}{2} = 0.5$. In other words, for higher order partitions (with high values of q), $\psi_{\mathbf{j}}(\mathbf{r})$ is either going to be 1 or -1 with probability 0.5. This uniformity of the distribution of 1 and -1 approaches equality at an exponential rate with respect to q.

We can derive the counterpart of Equation 7 when $P_{\alpha,\mathbf{r}}$ takes different values. Let us consider the case where $P_{\alpha,\mathbf{r}}$ can take two different values (as in code A). Let $P_{\alpha,\mathbf{r}} \in \{p_1, p_2\}$. For any arbitrary mRNA string $\mathbf{r}$ let q_1 be the number of sub-partitions ($\mathbf{j}_\alpha$-s) where $P_{\alpha,\mathbf{r}} = p_1$. Therefore, for the rest of the $q - q_1$ sub-partitions $P_{\alpha,\mathbf{r}} = p_2$. Now in order to compute the probability that an odd number of $\mathbf{j}_\alpha$-s take a value of -1 we need to count the number of different ways we can pick some k_1 and $k - k_1$ sub-partitions from q_1 and $q - q_1$ choices respectively. So we can write,

$$\begin{aligned} P_{-1} &= \sum_{k=1}^{q} \sum_{k_1=0}^{k} \binom{q_1}{k_1} \binom{q-q_1}{k-k_1} p_1^{k_1} p_2^{k-k_1} (1 - p_1)^{q_1 - k_1} \times \\ &= (1 - p_2)^{q - q_1 - k + k_1} (k \bmod 2) \end{aligned}$$

The behavior of the above equation is similar to that of Equation 7. P_{-1} approaches 0.5 very quickly. Further analysis of P_{-1} for more general scenarios can be found elsewhere (Kargupta, 2001). Since the above analytical expression is not in closed form we choose to work with the expression for constant $P_{\alpha,\mathbf{r}}$ (Equation 7) in rest of this paper.

Let $(\Psi_{\mathbf{j}}(\mathbf{r}))_{S'}$ be the column vector where the i-th row is the value of $\psi_{\mathbf{j}}(\mathbf{r}_{(i)})$ where $\mathbf{r}_{(i)}$ is the i-th member of S'. According to our analysis

every entry of this column vector will be either 1 or -1 with almost equal probability in general when the order of the reflection of $\mathbf{j}$ is not very close to zero. So if $\mathbf{j}$ and $\mathbf{t}$ are two such partitions the entries of both column matrices $(\Psi_{\mathbf{j}}(\mathbf{r}))_{S'}$ and $(\Psi_{\mathbf{t}}(\mathbf{r}))_{S'}$ are going to be uniformly distributed. Therefore, the expected inner product between any two such basis vectors,

$$E[(\Psi_{\mathbf{j}}(\mathbf{r}), \Psi_{\mathbf{t}}(\mathbf{r}))_{S'}] \approx 0. \tag{8}$$

where E denotes the expectation. Even for partitions with relatively small values of q, the $(\Psi_{\mathbf{j}}(\mathbf{r}), \Psi_{\mathbf{t}}(\mathbf{r}))_{S'}$ is likely to be quite small compared to $|S'|$. Now recall that Fourier basis functions are closed under inner product. In other words, $(\Psi_{\mathbf{j}}(\mathbf{r}), \Psi_{\mathbf{t}}(\mathbf{r}))_{S'} = (\Psi_{\mathbf{k}}(\mathbf{r}))_{S'}$. For Boolean strings $\mathbf{k} = \mathbf{j} \oplus \mathbf{t}$, where $\oplus$ denotes bit-wise XOR operation. Therefore Equation 8 also implies that $E[\sum_{\mathbf{r} \in S'} \psi_{\mathbf{k}}(\mathbf{r})] \longrightarrow 0$ as the absolute order (q) of the partition $\mathbf{r}$ increases. This can also be quantified using Equation 7.

$$\begin{aligned} E[\psi_{\mathbf{k}}(\mathbf{r})] &= (-1)P_{-1} + 1.(1 - P_{-1}) \\ &= (1 - 2P_{\alpha,\mathbf{r}})^q \end{aligned} \tag{9}$$

For $0 < P_{\alpha,\mathbf{r}} < 1.0$, we can write $\lim_{q \to \infty} E[\psi_{\mathbf{k}}(\mathbf{r})] = 0$. The following section shows that this result implies reduction of higher order non-linear dependency among the features.

The results presented in this section shows that we can construct a representation of the given training data set where the higher order Fourier basis vectors approaches orthonormality. This almost satisfies the conditions identified in Section 3 since the Fourier basis vectors are always orthonormal over the entire domain. This essentially means that we can construct a representation using the randomized GCTs where the mean-square-error surface may be accurately approximated by a convex quadratic function where gradient descent leads to the global minima. This is an interesting property which is likely to be useful for non-linear regression, curve fitting, classifier learning, and other predictive modeling applications.

However, randomized GCTs may offer additional interesting properties that are yet to be fully explored. It appears that randomized GCTs generate a representation where the magnitudes of the higher order coefficients are exponentially smaller than the magnitudes of the low order coefficients. This property may be useful for constructing a more efficient representation of the target function. The following section discusses this issue.

4.4. Effect of Randomized GCT-s on the Fourier Coefficients

This section explores some of the properties of the Fourier spectrum of the new representation of the target function constructed by the randomized GCT-s.

Consider the target function $f : \Omega \rightarrow \{0, 1\}$. The domain Ω contains 2^{ℓ_p} different ℓ_p-bit strings. On the other hand, the transformed domain $\Omega^{'}$ contains 2^{ℓ_p} different $c\ell_p$-bit strings. Let us define the corresponding partial function $f_p : \Omega^{'} \rightarrow \{0, 1\}$. In order to understand the properties of this partial function we need to extend the domain to the complete set of $2^{c\ell_p}$ different $c\ell_p$-bit strings. Let us do that in the following manner.

$$\begin{aligned} f'(\mathbf{r}) &= f_p(\mathbf{r}) \quad \text{if} \quad \mathbf{r} \in \Omega^{'} \\ &= 0 \quad otherwise \end{aligned}$$

Following Equation 7 we can write,

$$\begin{aligned} E[w_{\mathbf{j}}^{'}] &= \frac{1}{2^{c\ell_p}} E[\sum_{\mathbf{r}} f'(\mathbf{r})\psi_{\mathbf{j}}(\mathbf{r})] \\ &= \frac{1}{2^{c\ell_p}} \sum_{\mathbf{r}|f'(\mathbf{r})=1, \mathbf{r}\in\Omega'} E[\psi_{\mathbf{j}}(\mathbf{r})] \\ &= \frac{(1-2P_{\alpha,\mathbf{r}})^q |\Omega_{f(\mathbf{r})=1}|}{2^{c\ell_p}} \\ &= \frac{(1-2P_{\alpha,\mathbf{r}})^q w_0}{2^{(c-1)\ell_p}} \end{aligned} \tag{10}$$

Where $|\Omega_{f(\mathbf{x})=1}|$ is the number of members of Ω for which $f(\mathbf{x}) = 1$ and w_0 is the coefficient for the order-zero partition in the original unexpanded protein space. This result shows that the expected magnitude of the Fourier coefficients in the new representation decays exponentially with respect to q, the absolute order of the corresponding partition. It essentially means that the individual higher order coefficients are exponentially less significant compared to the low order ones. However, the new longer representation also introduces more number of partitions and therefore potentially more number of Fourier coefficients. This is because the new representation has $\binom{\ell_p}{q}(2^c - 1)^q$ number of partitions of absolute order q, instead of the $\binom{\ell_p}{q}$ number of partitions of order q in the original representation. So the overall expected contribution of all the coefficients of absolute order q to the energy of the spectrum,

$$E_q = \sum_{\mathbf{j}|o(\mathbf{j}')=q} w_{\mathbf{j}}^2$$

$$
\begin{aligned}
&= \sum_{\mathbf{j}|o(\mathbf{j}')=q} \frac{(1-2P_{\alpha,\mathbf{r}})^{2q} w_0^2}{2^{2(c-1)\ell_p}} \\
&= \frac{(1-2P_{\alpha,\mathbf{r}})^{2q} w_0^2}{2^{2(c-1)\ell_p}} \binom{\ell_p}{q} (2^c-1)^q \\
&\leq \frac{(1-2P_{\alpha,\mathbf{r}})^{2q} w_0^2}{2^{2(c-1)\ell_p}} (2^c \ell_p)^q \qquad (11)
\end{aligned}
$$

Let b be a constant such that $b < 1$. The overall contribution of E_q will decay exponentially with respect to increasing q only if the following condition is true:

$$
\begin{aligned}
(1-2P_{\alpha,\mathbf{r}})^{2q} (2^c \ell_p)^q &= b^q \\
P_{\alpha,\mathbf{r}} = \frac{1}{2} - \frac{1}{2}\sqrt{\frac{b}{2^c \ell_p}} & \qquad (12)
\end{aligned}
$$

So if we can construct a codebook that satisfies the above condition for some $b < 1$, the contribution of the Fourier coefficients with absolute order q to the spectrum energy will decay at an exponential rate. In other words, the new representation will become less "non-linear" since we will be able to neglect the higher order coefficients by exploiting the exponential decay property of E_q. However, it is not clear whether we can construct a codebook that satisfies this property. But this is an interesting possibility. Also note that Equation 12 makes use of the simplified expression of P_{-1}. So the expression is only an approximation. The exact nature of this expression is yet to be explored.

The following sections present a set of experimental results that demonstrate the performance of the gradient descent-based Perceptron learning algorithm for a class of nonlinear classification problems.

5. The Perceptron and the XOR Function

The experiments presented in this section considers the Perceptron (Rosenblatt, 1961) which is widely known to be a linear classifier and the non-linear XOR problem. The Perceptron cannot learn the XOR problem (Minsky and Papert, 1968) since it is a linear classifier. However, we apply the Perceptron on a representation of the XOR constructed by randomized GCT-s and test its performance. The results show that the Perceptron accurately learns the functions in the new representation constructed by GCTs. This results however do not necessarily say anything concrete about the overall properties of randomized GCTs. It is also not clear whether such performance is unique to randomized GCT-s. We need to explore these issues further before

making any concrete claim. The experiments are primarily presented to gain some insights into the behavior of randomized GCT-s. First let us quickly review the Perceptron and its learning mechanism.

5.1. Perceptron A Brief Review

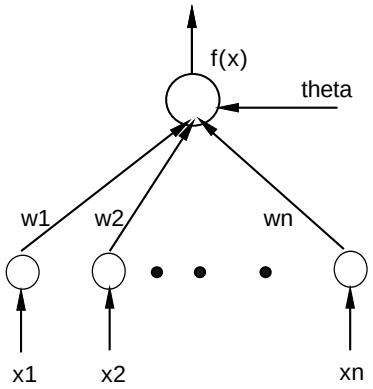

Figure 2. A Perceptron.

A Perceptron (Rosenblatt, 1961) is an artificial model of a neuron that computes a linear function of a set of input variables. In its most common implementations, a Perceptron computes a Boolean output value by filtering the output of the linear function through a threshold logic which fires (a value of 1) only when the output is greater than a certain threshold value. The output of the Perceptron is zero when the threshold logic does not fire. Figure 2 shows the structure of a single node Perceptron. The input feature vector is represented by $\mathbf{x} = [x_1, x_2, \ldots x_{\ell_p}] \in I\!R^{\ell_p}$ and $[w_1, w_2, \ldots w_{\ell_p}] \in I\!R^{\ell_p}$ are the weights associated with the input links connected to the node. The constant input threshold is defined by $\theta \in I\!R$. However, in our current experiments all the input features are Boolean.

The function computed by a Perceptron can be written as follows

$$\phi(x) = \begin{cases} 1 & \text{If } \mathbf{w}.\mathbf{x} \geq \theta \\ 0 & \text{Otherwise.} \end{cases} \tag{13}$$

where $\mathbf{w}.\mathbf{x}$ denotes the inner product defined by $\mathbf{w}.\mathbf{x} = \sum_{i=1}^{\ell_p} w_i x_i$.

Perceptron classifies the inputs into two categories i.e., for those with $\phi(\mathbf{x}) = 1$ and for the others with $\phi(\mathbf{x}) = 0$. The decision boundary of the function defined by the equation $\mathbf{w}.\mathbf{x} - \theta = 0$, is an *affine* subspace of $I\!R^n$.

Perceptron learns a classifier by learning the set of weights $\mathbf{w}$ and θ. Given a training data set it learns the weights by gradient descent in the min-square-error surface. For convex functions such gradient descent is guaranteed to find the error minima. The learning algorithm for Perceptrons is sometimes called Widrow-Hoff or delta learning law (Widrow and Hoff, 1960). The algorithm is described in the following.

1. Initialize the weights ($\mathbf{w}$) and the threshold value (θ) to some arbitrary values.

2. Update $\mathbf{w}$ and θ according to the following rule

$$w_{i,t+1} = w_{i,t} + \alpha(f(\mathbf{x}) - \phi(\mathbf{x}))x_i$$
$$\theta_{t+1} = \theta_t - \alpha(f(\mathbf{x}) - \phi(\mathbf{x}))$$

   where $w_{i,t+1}$ and $w_{i,t}$ are the i-th weights at the $(t+1)$-th and t-th iteration respectively; α is a small constant. It is usually called the learning rate. $f(\mathbf{x})$ is the target function value.

3. Continue until the convergence condition is satisfied. All experiments reported in this paper were stopped when either of the following conditions was satisfied

   a) $w_{i,t+1} \approx w_{i,t}, \quad \forall\ i = 1, 2 \ldots, n$

   b) the total number of iterations is more than a predefined threshold.

Our experiments consider up to fifth decimal places of $w_{i,t+1}$ and $w_{i,t}$ before concluding that $w_{i,t+1} \approx w_{i,t}$. The chosen value for the maximum number of iterations is $10,000$.

It has been shown elsewhere (Rosenblatt, 1961) that this algorithm is guaranteed to converge to the global error minima when the target function is linear.

5.2. The XOR Problem

An XOR is a Boolean function. Its domain is a set of Boolean strings. Table IV shows the truth-table of a two-bit XOR problem. Our experiments also consider a generalized version of the XOR problem for n-bits. An n-bit XOR problem is defined as follows:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ contains odd number of 1-s} \\ 0 & \text{otherwise.} \end{cases}$$

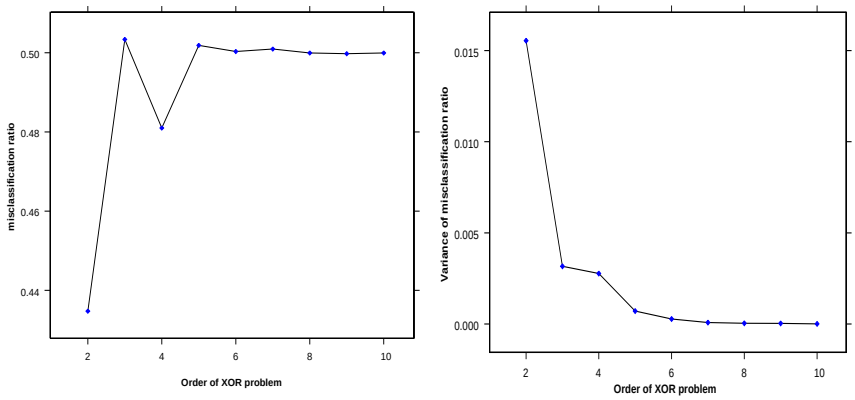

Figure 3. Performance (Error rate vs. problem size) of the Perceptron on XOR problems. Note that the error rate starts from 0.4 for two bit XOR and quickly goes up to approximately 0.5 for larger problems. An error rate of 0.5 is equivalent to random guessing.

A linear classifier like a single Perceptron cannot learn a non-linear function. The XOR function is non-linear and therefore a single Perceptron cannot learn it in its canonical representation where the input features of the Perceptron is nothing but the Boolean variables of the XOR function. Our experiments apply the Perceptron to learn the XOR in a representation constructed by the randomized GCT-s. During the learning stage we present the domain members to the Perceptron. Once the learning is over the trained Perceptron is tested on the domain members. In order to measure the performance of a Perceptron we define mis-classification ratio as the ratio of the number of mis-classification and the number of entries in the testing data set. For all the experiments reported here, the training data set in drawn from the entire domain using uniform distribution. The testing data set is comprised of the entire domain.

Figure 3 presents the mis-classification ratio for the different sizes (number of input variables) of the XOR problem in its canonical representation (i.e. without any randomized GCT-based transformation). The figure at the left shows the average mis-classification ratio for $10,000$ runs. The figure at the right shows the corresponding variance. As the figure shows the absolute variance is quite low. For smaller problems the variance is relatively higher. These figures show that the Perceptron is making about 50% error for most of the problems of different sizes. This implies that the classifier is no better than random guessing since for half of the members of the XOR-truth-table $f(\mathbf{x}) = 1$.

Table IV. The XOR problem.

x_1	x_2	$f(x_1, x_2)$
0	0	0
1	0	1
0	1	1
1	1	0

In other words, according to our expectation the Perceptron fails to correctly learn the XOR function in its given representation.

5.3. Learning XOR with a Perceptron and Randomized GCT-s

This section presents the results of our experiments where the Perceptron is applied to learn the XOR function in a representation constructed by the randomized GCT-s.

5.4. Experimental Setup

The experimental process is comprised of the following steps

1. Given the XOR problem in the canonical representation, construct a new enlarged representation using randomized GCT-s. Table V shows the GCT-s used for the experiments reported here. We consider three different code-books with different codon sizes. Codons are assigned with uniform probability.

2. Learn a single Perceptron in this new representation using the delta learning law.

3. Test the trained Perceptron.

The training set is constructed by uniformly selecting members from the domain $\Omega^{'}$. We used the entire domain ($\Omega^{'}$) for testing.

All the experiments consider code books with equal number of codons for both 1 and 0. Our analysis does not require that. It is just one of the experimental choices that we made. We plan to report results with unequal codon distributions in the future.

Table V. Three code books defined by 2, 3, and 4-bit codons.

0	01, 10
1	00, 11

0	010, 101, 111, 000
1	001, 110, 100, 011

0	0000, 1111, 0001, 1110, 0011, 1100, 0100, 1011
1	0010, 1101, 1000, 0111, 0101, 1010, 1001, 0110

5.5. Experimental Results

Figure 4 presents the average mis-classification error of the Perceptron trained over a representation constructed by the code-book with codons of size two *(top, left)*, three *(top, right)*, and four *(bottom)*. The average error is computed over $10,000$ independent sessions. The leftmost figure at the top shows that a three-bit XOR problem can be learned almost perfectly using the code-book of size two. The error-ratio grows up to 0.5 as we increase the problem size. Note that the increase in error is logistic unlike the fast growth observed in Figure 3. The graph at the top-right corner shows a similar error variation for the code-book of size three; only the boundary of bottom knee of the graph is different. It shows that the code-book with codons of size three is capable of learning up to four-bit XOR problem accurately. Similarly, the Perceptron can learn up to five-bit XOR problem using the code-book with codons of size four (the figure at the bottom of Figure 4). Figure 5 plots the change in the corresponding variances for the three code-books. In absolute terms the variance is quite low. In other words the expected behavior plotted in Figure 4 is reliable. The change in the variances with respect to increasing problem size only reveals a spike around the problem size where the Perceptron starts failing. The analysis presented in this paper did not address the effect of codon size on the performance of the GCT-s. We plan to address this in our future work.

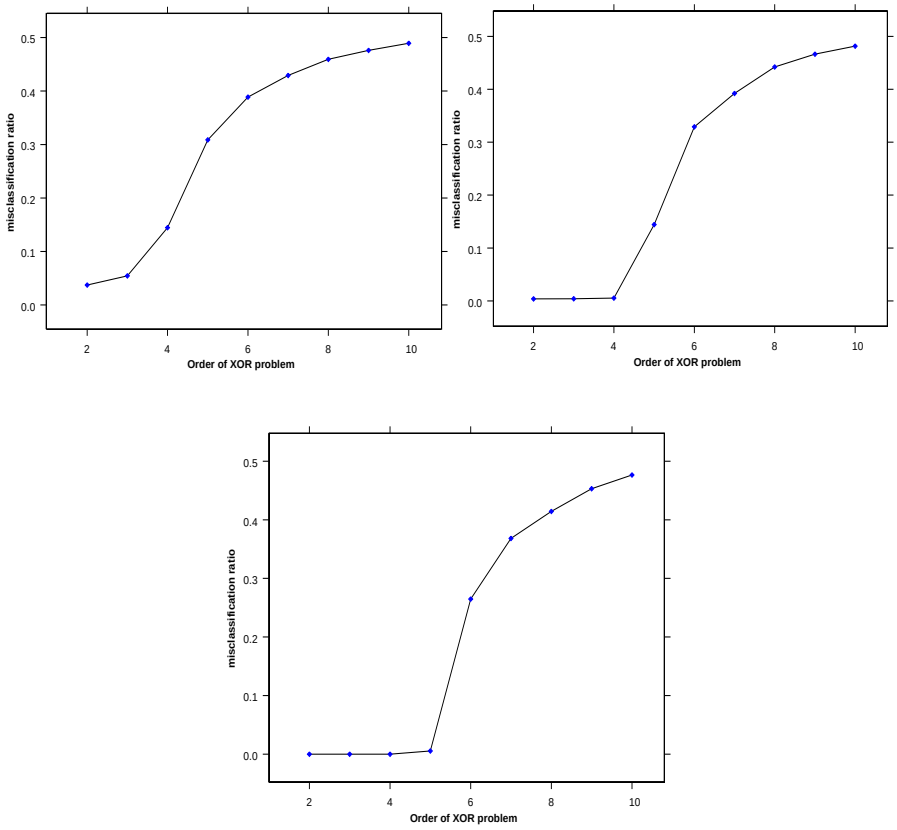

Figure 4. Average mis-classification ratio vs. problem size (number of input variables) for code-books with codon size (top, left) two, (top, right) three, (bottom) four. The reported result is an average of 10,000 independent runs.

6. Experiments with DNF Functions

The DNF (disjunctive normal form) is a Boolean function whose domain is comprised of a set of Boolean strings. Let B be a Boolean formula. A literal is either a variable or the negation of a variable. A clause is defined as conjunction of literals, e.g. $C = x_1 \wedge \neg x_2 \wedge x_3$. The formula is said to be in the disjunctive normal form (DNF) if it is a disjunction of clauses $C_1 \vee C_2 \vee ... \vee C_n$.

This section reports the results of applying the Perceptron to learn a DNF function of the following form:

$$f(\mathbf{x}) = \begin{cases} \bigvee_{i=1}^{\frac{n}{2}} C_i & \text{for an even integer } n > 2 \text{ and } C_i = x_{2i-1} \wedge x_{2i} \\ x_1 \vee x_2 & \text{for n=2.} \end{cases}$$

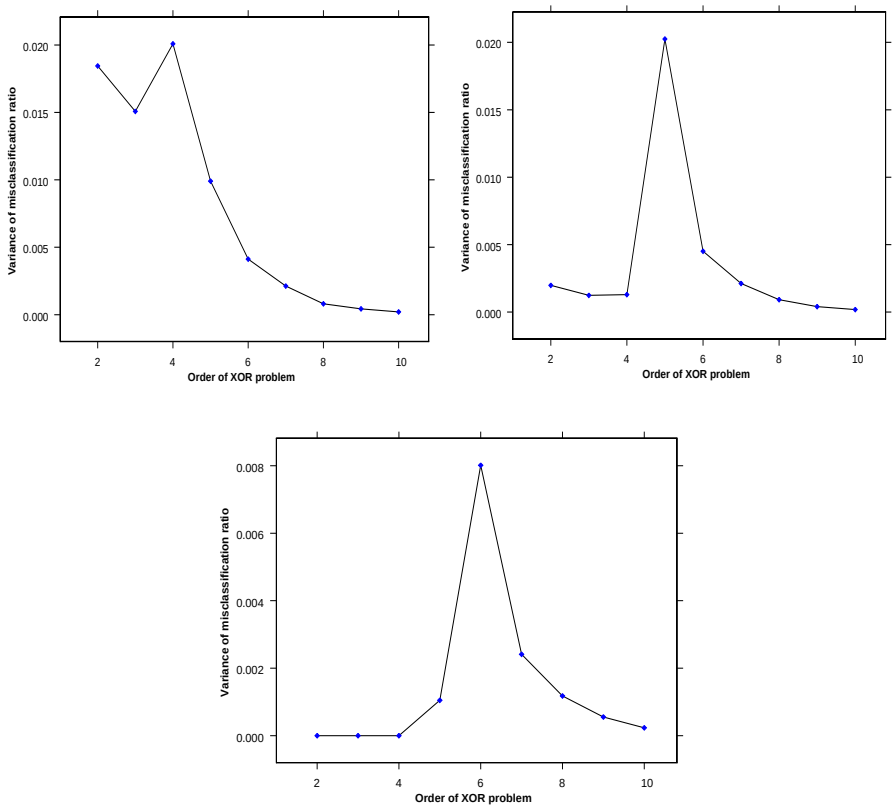

Figure 5. Variance of mis-classification ratio calculated over 10,000 learning and testing sessions for different problem sizes and different codon sizes (top, left) two, (top, right) three, (bottom) four.

Just like the XOR experiment, here we study the relative performance of the Perceptron with and without the the application of randomized GCT-s. During the learning stage we present all the domain members to the Perceptron. Once the learning is over, the trained Perceptron is tested on all the domain members. Performance of the Perceptron is measured in a way similar to that of the XOR experiments.

Figure 6 plots the average (over 1,000 runs) mis-classification ratio for the different sizes (number of input variables) of the DNF problem in its canonical representation (i.e., without GCT). This figure shows that Perceptron performs not so well as the problem size increases.

Figure 7 plots the average mis-classification error of Perceptron trained over a representation constructed by the code book with codons

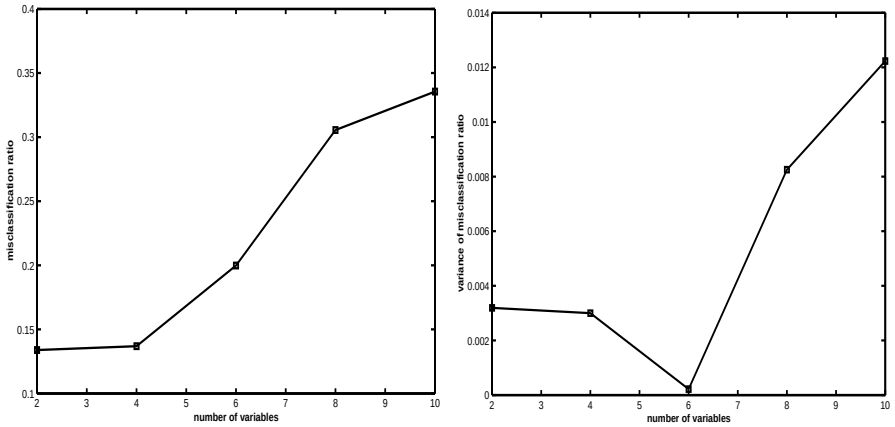

Figure 6. Performance (Error rate vs. problem size) of the Perceptron on DNF problems.

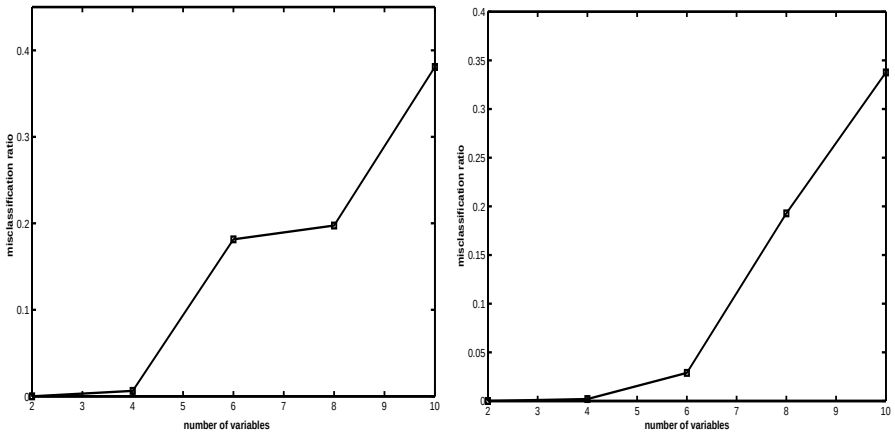

Figure 7. Average mis-classification ratio vs problem size (number of input variables) for code-books with codon size (left) two and (right) three. The reported result is an average over $1,000$ independent runs.

of size two *(left)* and three *(right)*. The average error is computed over 1,000 independent sessions. The left most figure shows that 6 bit DNF can be learned almost perfectly using the code book of size two. The error ratio grows as we increase the problem size. The graph at the right shows similar error variation for code-book of size three. It indicates

that the code book with codon size three is capable of learning up to eight bit DNF.

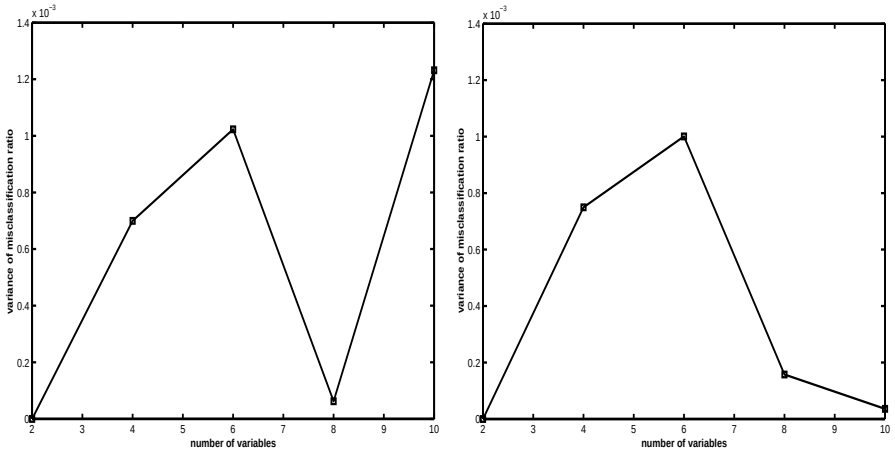

Figure 8. Variance of the mis-classification ratio calculated over 1,000 learning and testing sessions for different problem sizes and different codon sizes (left) three and (right) two.

Figure 8 plots the change in the corresponding variance for both of the code-book. In absolute terms variance is quite low indicating that the results are quite stable. The following section concludes this paper.

7. Discussion, Future Work, and Conclusions

This paper is a part of a series of efforts by the author and his colleagues in understanding the role of gene expression on computational grounds. The observations made in this paper are particularly related to the results presented elsewhere (Kargupta, 2001). That paper showed that the exponentially long representation of some functions using Fourier bases can be transformed to a representation using GCTs, where the low order coefficients are exponentially more significant than the high order ones. This makes the exponentially long representation suitable for approximations using only a polynomial number of low order coefficients. However, that paper considered only the deterministic GCTs.

This paper introduces and explores the randomized GCTs. The primary contribution of the paper is the following. It shows that the randomized GCT-s can be used to construct a representation of the domain where the higher order Fourier basis-vectors defined over the training data set are almost orthonormal to each other. This property

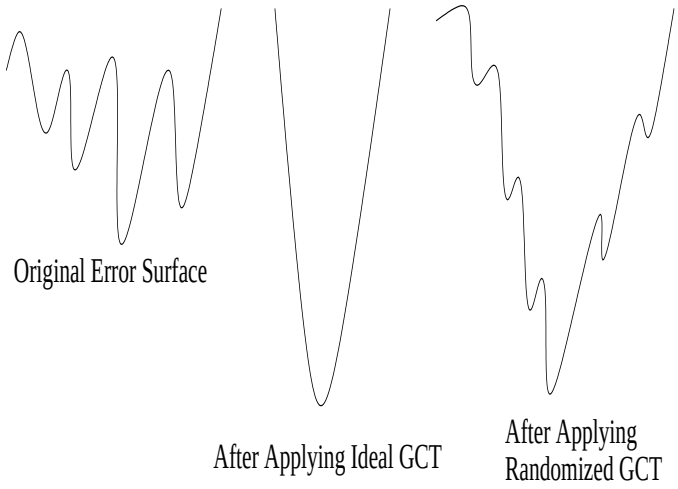

Figure 9. (Left) Original mean-square-error surface. (Middle) The surface produced after the ideal transformation where the bases are perfectly orthonormal over the learning data set S. (Right) The surface obtained after applying the proposed randomized GCTs.

makes the mean-square-error surface approximately convex quadratic, allowing error minimization using gradient descent algorithms.

The analytical observations derived from the Fourier analysis of randomized GCT-s are tested using the well-known linear classifier Perceptron. The Perceptron learns by performing gradient descent in the mean-square-error surface. It fails to learn the widely known non-linear XOR function. The experiments show that Perceptron can indeed learn the XOR function in a representation constructed by randomized GCT-s as long as the codon length (i.e. the size of the expanded representation) is sufficiently long.

Although the approach appears quite promising, several issues should be addressed before it can be offered as a viable technique for representation construction in function induction. Some of them are listed below.

1. Our experiments first constructs $\Omega^{'}$ from Ω. This new set $\Omega^{'}$ is then used for generating the training and the testing data. Therefore, every unique string $\mathbf{x}$ is mapped to a unique string $\mathbf{x}^{'}$ in $\Omega^{'}$. During the testing phase we need to use the same string $\mathbf{x}^{'}$ if we want to find out the class label of the $\mathbf{x}$. This is difficult to do unless we have an efficient way to generate the same $\mathbf{x}^{'}$ from $\mathbf{x}$. Note that the genetic code allows deterministic generation of $\mathbf{x}$ from $\mathbf{x}^{'}$ but not the opposite. So we need to develop a technique to make sure that the trained model is always applied to the members of $\Omega^{'}$.

In other words, we need to make sure that a model learned using samples from Ω' is applied for testing only the members of Ω'. Explicit storage of the new representation of the domain members is a possibility since the length of the representation is increased by a small constant factor.

2. We think eventually the GCT-s may require techniques to introduce bias in the code book in order to construct a more efficient description of the data sets belonging to different classes. Non-uniform distribution of codons to different protein features is an attractive way to introduce bias in the code and we are currently exploring this possibility.

3. The optimality of the code is an important issue. We know that the closer $P_{\alpha,\mathbf{r}}$ is to 0.5 the better it is. If it is very close to 0.5 the higher order coefficients will quickly disappear. We also know that the codon size has a critical role on the performance. We need to quantify that. In general, we need to address the issue of designing the optimal or near optimal code.

The overall contribution of the work can be summarized using the cartoon diagram presented in Figure 9. It shows the original MSE surface, the result of ideal transformation generating perfectly orthonormal bases, and the quasi-quadratic surface generated by the proposed randomized GCTs. A reasonable stochastic minimization algorithm is likely to be able to handle such almost quadratic error surfaces with some relatively small humps. Note that the diagram is provided only for intuitive description of the results presented here. The exact transformation of an arbitrary MSE surface under randomized GCTs and the exact nature of the "quasi-quadratic" surface are yet to be explored.

The possibility of reducing function non-linearity briefly sketched in this paper is worth further explorations. Linearization through construction of longer transformations is exploited in other machine learning techniques. For example, the support vector machines (SVMs) (Cristianini and Shawe-Taylor, 2000; Vapnik, 1995) formulate a linear version of a given non-linear function induction problem by expanding the number of features in the original representation. SVMs construct a longer representation where the target function is linear using user-provided kernel functions. Finally the classifier is learned by minimizing the error using quadratic programming algorithms. Although the proposed technique shares some philosophical similarities with SVMs the technical approaches are different.

Acknowledgments
This work was supported by the United States National Science Foundation (NSF) Grants IIS-9803660 and IIS-0083946. The first author would also like to acknowledge support from the NSF CAREER award IIS-0093353. The authors thank Rajeev Ayyagari for many useful discussions.

References

J. D. Bagley. The behavior of adaptive systems which employ genetic and correlation algorithms. *Dissertation Abstracts International*, 28(12)5106B, 1967. (University Microfilms No. 68-7556).

W. Banzhaf. Genotype-phenotype mapping and neutral variation—A case study in Genetic Programming. In: *Proceedings of the Parallel Problem Solving from Nature III*, Eds. Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner. Lecture notes in Computer Science 866, Springer-Verlag, Berlin, pages 322–332.

J. Bashford, I. Tsohantjis, and P. Jarvis. A super-symmetric model for the evolution of the genetic code. In: *Proceedings of the National Academy of Science USA*, 95987–995, 1998.

P. Beland and T. Allen. The origin and evolution of the genetic code. *Journal of Theoretical Biology*, 170359–365, 1994.

K. G. Beauchamp. *Applications of Walsh and Related Functions.* Academic Press, USA, 1984.

A. Brindle. *Genetic Algorithms for Function Optimization.* Unpublished doctoral dissertation, Department of Computer Science, University of Alberta, Edmonton, Canada, 1981.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines.* Cambridge University Press, 2000.

D. Dasgupta and D. R. McGregor. Designing neural networks using the structured genetic algorithm. *Artificial Neural Networks*, 2263–268, 1992.

C. Ferreira. Gene Expression Programming: a New Adaptive Algorithm for Solving Problems. *Complex Systems*, 2(13):87-129, 2001.

S. J. Freeland, R. D. Knight, L. F. Landweber, and L. D. Hurst. Early fixation of an optimal genetic code. *Molecular Biological Evolution*, 17(4)511–518, 2000.

S. Fukuchi, T. Okayama, and J. Otsuka. Evolution of genetic information flow from the viewpoint of protein sequence similarity. *Journal of Theoretical Biology*, 171179–195, 1994.

D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5)493–530, 1989. (Also TCGA Report 89003).

J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

R. B. Hollstien. Artificial genetic adaptation in computer control systems. *Dissertation Abstracts International*, 32(3)1510B, 1971. (University Microfilms No. 71-23,773).

J. Hornos and Y. Hornos. Algebraic model for the evolution of the genetic code. *Physical Review Letters*, 71(26)4401–4404, 1993.

J. Jackson. *The Harmonic Sieve A Novel Application of Fourier Analysis to Machine Learning Theory and Practice*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1995.

F. Jacob and J. Monod. Genetic regulatory mechanisms in the synthesis of proteins. *Molecular Biology*, 3318–356, 1961.

H. Kargupta. The gene expression messy genetic algorithm. In: *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 814–819. IEEE Press, 1996.

H. Kargupta. Gene Expression The missing link of evolutionary computation. In C. Poloni D. Quagliarella, J. Periaux and G. Winter, editors, *Genetic Algorithms in Engineering and Computer Science.*, page Chapter 4. John Wiley & Sons Ltd., 1997.

H. Kargupta. SEARCH, computational processes in evolution, and preliminary development of the gene expression messy genetic algorithm. *Complex Systems*, 11(4)233–287, 1997.

H. Kargupta. A striking property of genetic code-like transformations. *Complex Systems Journal*, 13(1)1–32, 2001.

H. Kargupta and S. Bandyopadhyay. A perspective on the foundation and evolution of the linkage learning genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2000)269–294, 2000. Special Issue on Genetic Algorithms, Guest Editors Goldberg, D. E. and Deb, K.

H. Kargupta, D. E. Goldberg, and L. W. Wang. Extending the class of order-k delineable problems for the gene expression messy genetic algorithm. In: *Proceedings of the Second Annual Conference on Genetic Programming*, pages 364–369, San Francisco, California, 1997. Morgan Kaufmann Publishers.

H. Kargupta and H. Park. Fast construction of distributed and decomposed evolutionary representation. *Journal of Evolutionary Computation*, 9(1)1–32, 2000.

H. Kargupta and K. Sarkar. Function induction, gene expression, and evolutionary representation construction. In: *Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, USA.*. Eds. Wolfgang Banzhaf, Jason Daida, A. E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela and Robert E. Smith, pages 313–320, San Francisco, California, 1999. Morgan Kaufmann.

H. Kargupta and B. Stafford. From DNA to protein Transformations and their possible role in linkage learning. In: Proceedings of the Seventh International Conference on Genetic Algorithms. Eds. Thomas Back, pages 409–416, Morgan Kaufmann Publishers, San Francisco, USA, 1997.

H. Kargupta, R. Ayyagari, and S. Ghosh. Learning Functions Using Randomized Expansions: Probabilistic Properties and Experimentations. In communication, 2001.

S. Kauffman. *The Origins of Order*. Oxford University Press, New York, 1993.

R. Keller and W. Banzhaf. The evolution of genetic code in genetic programming. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1077–1082. Morgan Kaufmann Publishers, San Francisco, USA, 1999.

R. D. Knight and L. F. Landweber. The early evolution of the genetic code. *Cell*, 101569–572, 2000.

S. Kushilevitz and Y. Mansour. Learning decision trees using Fourier spectrum. In: *Proceedings of 23rd Annual ACM Symp. on Theory of Computing*, pages 455–464, 1991.

M. Minsky and S. Papert. *Perceptrons*. MIT Press, MIT, USA, 1968.

M. O'Neill, and C. Ryan. Genetic code degeneracy: Implications for grammatical evolution and beyond. In: *Proceedings of the Fifth European Conference on Artificial Life*, Lausanne, Switzerland, 1999.

C. Ryan, J. J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. Lecture notes in Computer Science 1391. pages 83–95, Springer-Verlag, 1998.

C. Reidys and S. Fraser. Evolution of random structures. Technical Report 96-11-082, Santa Fe Institute, Santa Fe, 1996.

D. Rockmore, P. Kostelec, W. Hordijk, and P. Stadler. Fast Fourier transform for fitness landscapes. Technical Report 99-10-068, Santa Fe Institute, Santa Fe, 1999.

R. S. Rosenberg. Simulation of genetic populations with biochemical properties. *Dissertation Abstracts International*, 28(7)2732B, 1967. (University Microfilms No. 67-17,836).

F. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, Washington DC., 1961.

P. Schuster. The role of neutral mutations in the evolution of RNA molecules. In S. Suhai, editor, *Theoretical and Computational Methods in Genome Research*, pages 287–302. Plenum Press, New York, 1997.

R. E. Smith. An investigation of diploid genetic algorithms for adaptive search of non-stationary functions. TCGA Report No. 88001, University of Alabama, The Clearinghouse for Genetic Algorithms, Tuscaloosa, 1988.

D. Thierens. Estimating the significant non-linearities in the genome problem-coding. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, Eds. Wolfgang Banzhaf, Jason Daida, A. E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela and Robert E. Smith, pages 643–648. Morgan Kaufmann Publishers, San Francisco, USA, 1999.

D. Thierens. Scalability problems of simple genetic algorithms. *Evolutionary Computation*, 7(4)331–352, 1999.

V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

J. L. Walsh. *A closed set of orthogonal functions. Ann. Journ. Math.*, 55, 1923.

M. V. Wickerhauser. *Adapted Wavelet Analysis from Theory to Software.* A. K. Peters Ltd., 1994.

B. Widrow and M. Hoff. Adaptive switching circuits. In *IRE WESCON Convention Record*, pages 96–104, New York, 1960.

A. Wu and R. Lindsay. Empirical studies of the genetic algorithm with non-coding segments. *Journal of Evolutionary Computation*, 3(2)121–147, 1995.

A. Wu and R. Lindsay. A survey of intron research in genetics. In H. Voigt, W. Ebeling, I. Rechenberg, and H. Schwefel, editors, *Parallel Problem Solving from Nature- PPSN IV*, pages 101–110. Springer-Verlag, Berlin, 1996.