Learning Functions Using Randomized Genetic Code-Like Transformations: Probabilistic Properties and Experimentations

Hillol Kargupta, Rajeev Ayyagari, and Samiran Ghosh
Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
Baltimore, MD 21250
{hillol, arajeev1, sghosh1}@cs.umbc.edu

Abstract

Inductive learning of nonlinear functions plays an important role in constructing predictive models and classifiers from data. This paper explores a novel randomized approach to construct linear representations of nonlinear functions proposed elsewhere [14, 17]. This approach makes use of randomized codebooks, called the Genetic Code-like Transformations (GCTs) for constructing an approximately linear representation of a nonlinear target function. This paper first derives some of the results presented elsewhere [17] in a more general context. Next it investigates different probabilistic and limit properties of GCTs. It also presents several experimental results to demonstrate the potential of this approach.

Key Words: Inductive function learning, genetic code-like transformations, representation construction, randomized transformations.

1 Introduction

Learning nonlinear functions from data is important in many domains like data mining, pattern recognition, statistics, blackbox optimization, and machine learning. Often it is used for constructing predictive models and classifiers. However, as we all know learning nonlinear functions from data is usually more difficult compared to learning a linear function. So, any technique to efficiently construct a good approximation of nonlinear functions in an appropriate linear representation will be extremely useful. There exist techniques like the support vector machines (SVMs) [5, 30] that work on a linear representation of a nonlinear problem and they are gaining increasing popularity. This paper explores an alternate approach.

This paper presents a class of randomized transformations with interesting properties that may be useful for learning nonlinear discrete functions in an "almost" linear representation. It draws motivations from the Universal Genetic Code that transforms the mRNA sequence to the Protein sequence in a living organism. This code, is defined by a small redundant table that assigns an amino acid for every three consecutive nucleic acids in the mRNA. This paper investigates the role of genetic code-like transformations (GCTs) [14, 17] in learning nonlinear functions from data with unknown underlying distribution. This paper revisits the foundation of randomized GCTs and explores the analytical properties of such transformations. It also offers experimental results to demonstrate the potential of this approach.

Section 2 revisits randomized GCTs developed elsewhere [14, 17] and reviews multidimensional discrete Fourier transformations, a tool used throughout this paper to study randomized GCTs. It also offers the biological motivation behind this research. Section 3, defines the GCTs themselves and motivates the main problems tackled in this paper—understanding probabilistic and limit properties of the randomized GCTs. Section 4 explores several properties of these transformations. Some transformations are better than others for a given problem. So finding an appropriate randomized GCT for a given problem is an important problem. Section 5 formulates the codebook optimization problem for finding a suitable transformation. Section 6 explores the distributions of such randomized transformations and it shows that the probability of finding a "good-enough" transformation increases "very quickly" as the number of features of the new representation constructed by the randomized GCTs increases linearly. Section 8 tests the hypotheses of this paper using a set of experiments. Finally, Section 9 concludes the paper.

2 Background

This section presents the necessary background material and the motivation behind this research. It offers a brief review of the universal genetic code used in a living organism during the process of translation. It also revisits an abstraction of this code, called genetic code-like transformations (GCTs) introduced elsewhere [14, 17].

2.1 Function Induction and Representational Issues

Consider the discrete set $X = X_1 \times X_2 \times \cdots \times X_n$, where each X_i is a finite set $\{0, 1, \ldots, k_i - 1\}$. We use the notation $[k_i]$ to denote the set $\{0, 1, \ldots, k_i - 1\}$. We are interested in *inducing* functions of the form $f : X \to \mathbb{R}$. We are given a *sample* from this function: a set of points $S = \{(\mathbf{x}_1, f(\mathbf{x}_1)), (\mathbf{x}_2, f(\mathbf{x}_2)), \ldots, (\mathbf{x}_m, f(\mathbf{x}_m))\}$, where $\mathbf{x}_i \in X \; \forall i = 1, \ldots, m$. The problem is to learn, or *induce*, a *good* approximation $\hat{f} : X \to \mathbb{R}$ to f based on the information in S.

Function induction algorithms must work with a *representation* of the functions f and $\hat{f}$. One example of a representation of a function is a vector in $\mathbb{R}^d$, where d is the number of points in X. Each coordinate of the vector represents the value of the function at a point in the space. This is an explicit representation of the function. It is not difficult to see that there is a decomposition of the function in terms of a particular basis of the set $\mathcal{F}$ of all real-valued functions on X. Various bases of $\mathcal{F}$ can be used for different decompositions of the function; each basis yields a representation of the function as a point in a d-dimensional space. The computational complexity of inducing a function from data depends on the chosen representation.

For most non-trivial learning problems the target function f is nonlinear in the given natural representation. Therefore, if we learn it well in the same representation we are most likely to come up with an $\hat{f}(x)$ which is also nonlinear. In this paper we shall show that this nonlinear learning process can be efficiently reduced to an approximately linear learning problem by applying the randomized genetic code-like transformations, at least for the classes of problems considered here. Randomized GCTs apply a probabilistic transformation $\eta(\mathbf{x})$ to every member of the sample set S and generate a new representation in a different space. These transformations have 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. In order to fully appreciate the proposed approach, we must understand the biology to some extent. The following section presents this biological motivation.

2.2 Biological Motivation

The gene expression process in nature involves a series of representation transformations of the genome. Representation transformations are often used in many fields like Physics, Engineering, Machine Learning, and Mathematics for transforming difficult problems into suitable forms that are easier to solve. Therefore representation transformations in gene expression may have intriguing computational implications in genetic search. The gene expression process starts by first transforming the DNA sequence to the mRNA. The DNA is a sequence of four different nucleotides namely, *adenine* (A), *thiamin* (T), *guanine* (G), and *cytosine* (C). The mRNA is also a sequence of four different nucleotides namely, *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 1). Each codon is comprised of three adjacent nucleotides in an mRNA chain and it produces an amino acid. The genetic code takes an mRNA sequence, replaces every three adjacent nucleic acids (codon) by the corresponding amino acid (listed in Table 1), and produces the amino acid sequence. With a few exceptions, the genetic code for most eukaryotic and prokaryotic organisms is the same. There exists a considerable body of literature exploring the models of evolutionary computation in gene expression [10, 13, 1, 3, 9, 19, 18, 27, 29, 11, 6, 7, 20, 2]. Interested readers may refer to [17] for a detailed literature review.

Proteins control almost every important activity in a living body and they define the phenotype of a living organism. This construction of the phenotype from the genome can be viewed as an evaluation of a genetic "fitness" function. So Proteins, mRNAs, and the DNA can be viewed as different representations of this underlying genomic "fitness' function.

Most living organisms choose protein-based representation of the genome 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. However, this paper does not consider the biological implications directly.

This paper considers the genetic code-based translation process that transforms the mRNA sequence to the amino acid sequence in proteins in the light of efficient function induction. It considers GCT-s, an abstract class of transformations, that draws direct mo-

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

Table 1: The universal genetic code.

tivation from the genetic code. It suggests that these genetic code-like transformations may have an important role in learning and adaptation. It develops randomized GCTs that can be used to efficiently learn nonlinear functions from data.

In order to understand why GCTs work, we use the well-known multidimensional Fourier transform as a representation to study the effect of the GCTs on a function. The next section reviews the MFT of a function.

2.3 Multidimensional Fourier Transformation and Probabilistically Orthonormal Representations

In this section, we first review Multidimensional Fourier Transformation (MFT), (identical to the Walsh transformations in Boolean representations) [8, 4, 12, 21]. MFT is a useful tool for detailed study of the functions we will be considering as well as the GCTs themselves. Next we explore the structure of the mean-square error surface in orthonormal representation of the target and estimated function. We also introduce a probabilistic notion of orthonormality.

2.3.1 A Brief Review of MFT

We use the MFT to study real valued functions defined on X. Let $\mathcal{F}$ be the set of all such functions. $\mathcal{F}$ forms a $(\prod_{i=1}^{n} k_i)$-dimensional vector space over $\mathbb{R}$. The MFT of a function $f \in \mathcal{F}$ is essentially a decomposition of f according to a suitable basis of $\mathcal{F}$, the Fourier basis.

In the binary case, where $k_i = 2$ for $i = 1, \ldots, n$, the Fourier basis (also called Walsh basis) is $\{\psi_{\mathbf{j}}(\mathbf{x}) = (-1)^{\mathbf{j}\cdot\mathbf{x}} : \mathbf{j} \in [2]^n\}$. Here $\mathbf{j}\cdot\mathbf{x}$ is defined as $\sum_{l=1}^{n} \mathbf{j}_l \mathbf{x}_l$, the subscript l denoting the l-th component. As can be seen, the basis consists of 2^n functions whose indexes are themselves the elements of the space $[2]^n$. It can be shown that it is orthonormal with respect to the inner product defined on $\mathcal{F}$ by $\langle f, g\rangle = 1/2^n \sum_{\mathbf{x}\in[2]^n} f(\mathbf{x})g(\mathbf{x})$. (The orthonormality of this basis is a crucial property, as we shall see repeatedly in this paper.) Any $f \in \mathcal{F}$ can be represented *uniquely* as a linear combination of the basis elements: $f(\mathbf{x}) = \sum_{\mathbf{j}\in[2]^n} w_{\mathbf{j}}\psi_{\mathbf{j}}(\mathbf{x})$. The coefficients $\{w_{\mathbf{j}} : \mathbf{j} \in [2]^n\}$ are collectively called the multidimensional Fourier transform of f. Given a function f, the MFT can be computed using the equation: $w_{\mathbf{j}} = \langle \psi_{\mathbf{j}}, f\rangle = \frac{1}{2^n}\sum_{\mathbf{x}\in[2]^n} f(\mathbf{x})\psi_{\mathbf{j}}(\mathbf{x})$. This is a direct consequence of the orthonormality of the basis.

In the case when the k_i's take arbitrary positive integral values, the Fourier basis is similar. With X as defined above, the basis is: $\left\{\psi_{\mathbf{j}}(\mathbf{x}) = \exp\left(2\pi i \sum_{l=1}^{n} \frac{\mathbf{j}_l \mathbf{x}_l}{k_l}\right) : \mathbf{j} \in X\right\}$. When $k_l = 2$ for $l = 1, \ldots, n$, this reduces to basis for Boolean domain defined earlier. The other expressions are almost the same. We summarize them below for completeness.

$$f(\mathbf{x}) = \sum_{\mathbf{j}\in X} w_{\mathbf{j}}\psi_{\mathbf{j}}(\mathbf{x}) \tag{1}$$

$$w_{\mathbf{j}} = \langle f, \psi_{\mathbf{j}}\rangle = \frac{1}{|X|}\sum_{\mathbf{x}\in X} f(\mathbf{x})\overline{\psi_{\mathbf{j}}(\mathbf{x})} \tag{2}$$

It is worthwhile to note that the MFT in the non-binary case has complex numbers as its elements. Note also that an inner product here involves the complex conjugate: $\langle f, g\rangle$ is defined as $\frac{1}{|X|}\sum_{\mathbf{x}\in X} f(\mathbf{x})\overline{g(\mathbf{x})}$.

2.3.2 Mean Square Error and Orthonormal Representation

This section introduces a probabilistic framework for defining orthonormal representations. It also studies the structure of Mean Square Error (MSE) surface in this framework. It notes that the MSE has a nice quadratic shape when the linear basis are orthonormal to each other.

If f and $\hat{f}$ are the target function and its approximation respectively the MSE over the training data set S is defined in the following manner.

$$\begin{aligned}
\text{MSE} &= \frac{1}{S}\sum_{\mathbf{x}\in S}(\hat{f}(\mathbf{x}) - f(\mathbf{x}))^2 = \sum_{\mathbf{x}\in S}(\hat{f}(\mathbf{x}) - f(\mathbf{x}))\overline{(\hat{f}(\mathbf{x}) - f(\mathbf{x}))} \\
&= \frac{1}{S}\sum_{\mathbf{x}\in S}\left(\sum_{\mathbf{j}\in X}(\hat{w}_{\mathbf{j}} - w_{\mathbf{j}})\psi_{\mathbf{j}}(\mathbf{x})\right)\left(\sum_{\mathbf{k}\in X}(\overline{\hat{w}_{\mathbf{k}}} - \overline{w_{\mathbf{k}}})\overline{\psi_{\mathbf{k}}(\mathbf{x})}\right) \\
&= \sum_{\mathbf{j},\mathbf{k}\in X}(\hat{w}_{\mathbf{j}} - w_{\mathbf{j}})(\overline{\hat{w}_{\mathbf{k}}} - \overline{w_{\mathbf{k}}})\langle\psi_{\mathbf{j}}, \psi_{\mathbf{k}}\rangle_S
\end{aligned} \tag{3}$$

where $\langle f, g\rangle_S$ means $\frac{1}{|S|}\sum_{\mathbf{x}\in S} f(x)\overline{g(x)}$ and $\hat{w}_{\mathbf{j}}$ represent Fourier coefficients corresponding to $\hat{f}$. If $\langle \psi_{\mathbf{j}}, \psi_{\mathbf{k}} \rangle_S$ is close to zero for $\mathbf{j} \neq \mathbf{k}$, this further reduces to

$$\text{MSE} = \sum_{\mathbf{j} \in X} (\hat{w}_{\mathbf{j}} - w_{\mathbf{j}})(\overline{\hat{w}_{\mathbf{j}}} - \overline{w_{\mathbf{j}}}) = \sum_{\mathbf{j} \in X} |\hat{w}_{\mathbf{j}} - w_{\mathbf{j}}|^2 \tag{4}$$

since $\langle \psi_{\mathbf{j}}, \psi_{\mathbf{j}} \rangle_S = 1$ for any $\mathbf{j} \in X$. Note that terms in the error are coefficient-wise independent. This means that an error minimization algorithm that estimates the Fourier coefficients by means of an independent gradient descent along each coefficient performs well. In other words, gradient minimization of MSE offers accurate approximations of the true coefficients.

We now give a probabilistic characterization of the orthogonality of the Fourier basis. This gives us a useful alternative way of looking at orthogonality. Consider the property $\langle \psi_{\mathbf{u}}, \psi_{\mathbf{v}} \rangle = \frac{1}{|X|} \sum_{\mathbf{x} \in X} \psi_{\mathbf{u}}(\mathbf{x}) \overline{\psi_{\mathbf{v}}(\mathbf{x})} = 0$, if $\mathbf{u} \neq \mathbf{v}$, $\mathbf{u}, \mathbf{v} \in X$. Now,

$$\psi_{\mathbf{u}}(\mathbf{x}) \overline{\psi_{\mathbf{v}}(\mathbf{x})} = \exp\left(2\pi i \sum_{l=1}^{n} \frac{\mathbf{v}_l \mathbf{x}_l}{k_l}\right) \exp\left(-2\pi i \sum_{l=1}^{n} \frac{\mathbf{u}_l \mathbf{x}_l}{k_l}\right) = \exp\left(2\pi i \sum_{l=1}^{n} \frac{\mathbf{w}_l \mathbf{x}_l}{k_l}\right)$$

where $\mathbf{w}$ is the string defined by $\mathbf{w}_l = (\mathbf{v}_l - \mathbf{u}_l) \bmod k_l$. Note that $\mathbf{w} = 0$ iff $\mathbf{v} = \mathbf{u}$. Thus, we get

$$\langle \psi_{\mathbf{u}}, \psi_{\mathbf{v}} \rangle = \frac{1}{|X|} \sum_{\mathbf{x} \in X} \psi_{\mathbf{w}}(\mathbf{x}) \tag{5}$$

Now, assume that points are drawn from the space X using a uniform distribution on the points of X. Then the expression in Equation 5 is simply the expectation of $\psi_{\mathbf{w}}(\mathbf{x})$, denoted by $E(\psi_{\mathbf{w}}(\mathbf{x}))$. It is easy to see now that *the basis set* $\{\psi_{\mathbf{j}}(\mathbf{x}) : \mathbf{j} \in X\}$ *is orthogonal if and only if* $E(\psi_{\mathbf{w}}(\mathbf{x})) = 0$ *for every nonzero* $\mathbf{w}$. Note that the same reasoning shows that $\langle \psi_{\mathbf{j}}, \psi_{\mathbf{k}} \rangle_S$ equals $E(\psi_{\mathbf{w}}(\mathbf{x}))$ where $\mathbf{w}$ is suitably defined and the expectation is with respect to a uniform distribution over S. We will use this view of orthogonality in the following sections.

MFT exposes the underlying interactions among the features and quantifies their influence on the function value. For example, consider a binary domain with $n = 5$. Suppose that $w_{\mathbf{j}}$ has a large magnitude compared to the other Fourier coefficients for $\mathbf{j} = (00101)$. This tells us that the interaction between the 3rd and 5th features plays an important role in influencing the value of the function f. On the other hand, if $w_{\mathbf{j}}$ is non-zero only for $\mathbf{j} \in \{(00100), (10000)\}$, the function is a first order (linear) function.

The MFT can also be used to quantify the *degree of nonlinearity* of a function in $\mathcal{F}$ (as can representations in terms of other bases). The linearity or nonlinearity of a function defined on discrete spaces is difficult to quantify because there is no concept of (even piece-wise) differentiability. In this paper we quantify the nonlinearity of discrete functions in terms of a property of its Fourier spectrum.

We define the *order of a function* f as the order of the highest nonzero Fourier coefficient of f. We use the notation $\text{ord}(f)$ to denote that. In this paper we choose to quantify the nonlinearity of a function using its order. In other words $\text{ord}(f)$ is the degree of nonlinearity of function f. We say a function f strictly more nonlinear than another function g if and only if $\text{ord}(f) > \text{ord}(g)$.

We also use a relaxed concept of the degree of nonlinearity of a function. Let us define the distance between two functions $f, g \in \mathcal{F}$ over a subset S of X as the squared error:

$D_S(f,g) = \sum_{\mathbf{x}\in S}(f(\mathbf{x}) - g(\mathbf{x}))^2$. Define

$$\mathrm{ord}(\epsilon, D_S, f, S) = \min\{\mathrm{ord}(g) : g \in \mathcal{F}, D_S(f,g) \leq \epsilon\} \tag{6}$$

That is, the $\mathrm{ord}(\epsilon, D_S, f, S)$ is the order of the lowest order function g that approximates f "well". We call a function g that satisfies $D_S(f,g) \leq \epsilon$ an *ϵ-approximator* of f. Of course, this depends on the set S, but we do not include it in the notation. Now, note that

$$\begin{aligned} D_S(f,g) &= \sum_{\mathbf{x}\in S}(f(\mathbf{x}) - g(\mathbf{x}))^2 = \sum_{\mathbf{x}\in S}\left(\sum_{\mathbf{j}\in X}(\alpha_{\mathbf{j}} - \beta_{\mathbf{j}})\psi_{\mathbf{j}}(\mathbf{x})\right)^2 \\ &= \sum_{\mathbf{x}\in S}\left(\sum_{\mathbf{j},\mathbf{k}\in X}(\alpha_{\mathbf{j}} - \beta_{\mathbf{j}})(\alpha_{\mathbf{k}} - \beta_{\mathbf{k}})\psi_{\mathbf{j}}(\mathbf{x})\psi_{\mathbf{k}}(\mathbf{x})\right) \\ &= |S|\sum_{\mathbf{j}\in X}(\alpha_{\mathbf{j}} - \beta_{\mathbf{j}})^2\langle\psi_{\mathbf{j}},\psi_{\mathbf{j}}\rangle_S + |S|\sum_{\mathbf{j}\neq\mathbf{k}\in X}(\alpha_{\mathbf{j}} - \beta_{\mathbf{j}})(\alpha_{\mathbf{k}} - \beta_{\mathbf{k}})\langle\psi_{\mathbf{j}},\psi_{\mathbf{k}}\rangle_S \end{aligned}$$

If $S = X$, that is, the training set is the whole space, this expression reduces to $D_X(f,g) = |X|\sum_{\mathbf{j}\in X}(\alpha_{\mathbf{j}} - \beta_{\mathbf{j}})^2$ because of orthonormal basis. Hence $D_X(f,g) \leq \epsilon \Leftrightarrow \sum_{\mathbf{j}\in X}(\alpha_{\mathbf{j}} - \beta_{\mathbf{j}})^2 \leq \epsilon/|X|$. It is obvious that the best low-order approximation can be constructed by letting $\beta_{\mathbf{j}} = \alpha_{\mathbf{j}}$ for low-order $\mathbf{j}$. We start at the lowest order and assign $\beta_{\mathbf{j}} = \alpha_{\mathbf{j}}$. We keep going until the error between the function g represented by the $\beta_{\mathbf{j}}$'s falls below $\epsilon/|X|$. (This lets us determine the value of $\mathrm{ord}(\epsilon, D_S, f, S)$. We may need to do some additional work selecting the coefficients of the highest order in g in order to find the *best* ϵ-approximator of order $\mathrm{ord}(\epsilon, D_S, f, S)$.) In other words, the best ϵ-approximator can be found by discarding low magnitude, high order coefficients.

The value $\sum_{\mathbf{j}\in X}\alpha_{\mathbf{j}}^2$ is called the *energy of the Fourier spectrum* $\{\alpha_{\mathbf{j}} : \mathbf{j} \in X\}$. It is also called the energy of the function corresponding to the Fourier spectrum. Thus, the value $\sum_{\mathbf{j}\in X}(\alpha_{\mathbf{j}} - \beta_{\mathbf{j}})^2$ of the last paragraph is the energy of the function $f - g$. Note that if $S \neq X$, $D_S(f,g) \neq |S|\sum_{\mathbf{j}\in S}(\alpha_{\mathbf{j}} - \beta_{\mathbf{j}})^2$ unless $\langle\psi_{\mathbf{j}},\psi_{\mathbf{k}}\rangle_S = 0$ if $\mathbf{j} \neq \mathbf{k}$ and 1 if $\mathbf{j} = \mathbf{k}$. Thus, this property is a desirable one.

We are now in a position to formally define codebooks and genetic code-like transformations, and study their properties.

3 Genetic Code-Like Transformations

In this section, we show how a feature space can be transformed using a *codebook* in order to construct a "more" linear representation of a function with respect to its natural representation . Formally, a GCT, introduced in [14, 17] is a randomized transformation from an n-dimensional space X to a kn-dimensional space X'. It works by using a transformation (such as the one in Table 2) to produce k features for each feature in the n-dimensional space. A set of k-dimensional codons corresponds to every possible value of each feature. This correspondence is called a codebook. In Table 2, the codons corresponding to a feature value of 0 are 100, 000, 001 and 010. Thus, the string 0000 in the 4-dimensional space could get mapped to 100000100010 or 100100001001. It cannot get mapped to 111100000010, because the codon 111 does not correspond to a feature value of 0. On the other hand, the string 1000 could be mapped to 111100000010. A codebook refers to the set of codons corresponding to each feature value in the n-dimensional space, together with a probability distribution on each such set of codons. An important characteristic of these transformations is that they are one-to-many transformations. In particular, the transformations are invertible, as there is exactly one string in the n-dimensional space that could give rise to a particular string in the kn-dimensional space. In this paper, we investigate properties of Boolean codebooks on Boolean spaces. That is, we only consider spaces X of the form $[2]^n$. The codebooks considered are ones in which there are only two "nucleotides". That is, the features of the codons are Boolean. Thus, the transformed strings in the kn-dimensional space are also Boolean strings.

A function $f : X \to \mathbb{R}$ together with a codebook induces a function on the kn-dimensional space as follows. Let $\eta : [2]^{kn} \to [2]^n$ be a function that returns the protein space string corresponding to a kn-dimensional string. The function induced on the kn-dimensional space is defined as $g(\mathbf{x}') = f(\eta(\mathbf{x}'))$. The induced function has several interesting properties that we will study using its Fourier transform.

We first note that the induced function g depends on the codebook through η. We are interested in choosing a codebook that reduces the nonlinearity of the function. That is, we want a codebook for which g is relatively more linear than f in the sense introduced in Section 2.3. To see how to achieve this, we consider a Fourier coefficient $w_\mathbf{j}$ of g. Now,

$$w_\mathbf{j} = \frac{1}{2^{kn}} \sum_{\mathbf{x}' \in [2]^{kn}} g(\mathbf{x}')\psi_\mathbf{j}(\mathbf{x}') = \frac{1}{2^{kn}} \sum_{\mathbf{x} \in [2]^n} \sum_{\mathbf{x}' \in \eta^{-1}(\mathbf{x})} g(\mathbf{x}')\psi_\mathbf{j}(\mathbf{x}') = \frac{1}{2^{kn}} \sum_{\mathbf{x} \in [2]^n} f(\mathbf{x}) \sum_{\mathbf{x}' \in \eta^{-1}(\mathbf{x})} \psi_\mathbf{j}(\mathbf{x}')$$

(Here $\eta^{-1}(\mathbf{x})$ is the inverse image of the point $\mathbf{x}$ under η, or the set of all points in $[2]^{kn}$ that η takes to $\mathbf{x}$.) Note that this expression depends on $\mathbf{j}$ only through $\sum_{\mathbf{x}' \in \eta^{-1}(\mathbf{x})} \psi_\mathbf{j}(\mathbf{x}')$. As mentioned in Section 2.3, we would achieve linearity if the energy of all partitions with large order is low — *if* the orthogonality property mentioned at the end of Section 2.3 holds. In Section 6, this paper does show how this problem can be alleviated by intelligently selecting codebooks. Note that $\sum_{\mathbf{x}' \in \eta^{-1}(\mathbf{x})} \psi_\mathbf{j}(\mathbf{x}')$ is just $E(\psi_\mathbf{j}(\mathbf{x}'(\mathbf{x})))$, up to a constant. Here $\mathbf{x}'(\mathbf{x})$ represents the *random* string to which $\mathbf{x}$ is transformed by the GCT. (In other words, $\mathbf{x}'(\mathbf{x})$ is a Boolean-string-valued random vector, whose probability distribution is determined by $\mathbf{x}$.) Thus, we would like to make $E(\psi_\mathbf{j}(\mathbf{x}'(\mathbf{x})))$ as small as possible. The next section (Section 4) studies factors affecting this quantity, and Section 5 identifies the key problems that need to be solved in order to select a codebook that does so. In addition, we prove in the next section that $E(\psi_\mathbf{j}(\mathbf{x}'(\mathbf{x})))$ is a *decreasing function of the order of* $\mathbf{j}$, apparently at an exponential rate. This is a necessary condition for decreased nonlinearity, but not a sufficient one. The energy of partitions of a certain order needs to be studied in greater detail for a theoretical proof of nonlinearity reduction. However, this paper does not address the energy issue. It explores the variation of the magnitudes of the individual coefficients. It derives analytical results to show that randomized GCTs create a representation where the magnitudes decay very fast to zero as we increase the order of the coefficients and thereby satisfy one necessary condition for nonlinearity reduction. It uses experimental results to demonstrate that nonlinearity is indeed reduced dramatically by applying randomized GCTs.

The linearization property of the codebooks is extremely useful, as we shall see throughout

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

Table 2: A binary GCT. Each bit in the protein space maps to 3 bits in the mRNA space.

the paper. Although the classifier obtained is unlikely to be a perfectly linear classifier, any learning algorithm that is biased towards linearity is likely to learn better after the GCT is applied to the original data. The perceptron [28, 23] and decision trees [26] are just two examples of such classifiers. The perceptron is essentially a linear separating hyperplane of the form $\delta_0 + \sum_{i=1}^{n} \delta_i x_i = 0$, for which the δ_i's are learned using a specific iterative learning rule [31] based on gradient descent. Linear separating hyperplane classifiers for which the coefficients are learned using other optimization techniques have also been used for classification. The coefficients of orders 2 and higher in the MFT of any linear classifier of this type are all zero. Decision trees are very well known and have been used with great success for a wide variety of data mining problems. Decision trees with binary classes can be viewed as real-valued functions over their inputs. Thus, it is possible to calculate their MFT [25]. It turns out that the MFT of decision trees built using C4.5 or ID3 has an *exponential decay* property [25, 24, 22]. Thus, decision trees are good approximations for functions whose Fourier coefficients display a fall-off for higher order coefficients. We shall see in the experiments section of this paper that the GCT does induce functions g that are learned well by these algorithms.

We next present a further motivation for the study of $E(\psi_{\mathbf{j}}(\mathbf{x}'(\mathbf{x})))$, with a focus on gradient descent minimization of the squared error of a function. Suppose the sample in the original space is $S \subset X$. The elements of S are transformed to $S' \subset X'$, where $|S| = |S'|$. As mentioned in Section 2.3, it is easier to calculate a good approximation to g if $\langle \psi_{\mathbf{j}}, \psi_{\mathbf{k}} \rangle_S \approx 0$ whenever $\mathbf{j} \neq \mathbf{k}$. We showed in Section 2.3 that $\langle \psi_{\mathbf{j}}, \psi_{\mathbf{k}} \rangle_S = E(\psi_{\mathbf{w}})$ for appropriately chosen $\mathbf{w}$. The expectation is w.r.t. a uniform distribution over S'. Now,

$$E(\psi_{\mathbf{w}}) \;=\; E\left(\frac{1}{|S'|}\sum_{\mathbf{x}' \in S'} \psi_{\mathbf{w}}(\mathbf{x}')\right) = E\left(\frac{1}{|S|}\sum_{\mathbf{x} \in S} \psi_{\mathbf{w}}(\mathbf{x}'(\mathbf{x}))\right) = \frac{1}{|S|}\sum_{\mathbf{x} \in S} E(\psi_{\mathbf{w}}(\mathbf{x}'(\mathbf{x})))$$

Thus, we would like to find a codebook that minimizes the magnitude of the expectation $E(\psi_{\mathbf{w}}(\mathbf{x}'(\mathbf{x})))$. The following two sections attempt to achieve the goal of minimizing this expectation by studying probabilistic properties of the codebooks.

4 Some Probabilistic Properties of Codebooks

We first express the expectation of the last section in terms of a probability. Since we are in the Boolean case, $\psi_{\mathbf{j}}(\mathbf{x}'(\mathbf{x}))$ can take only the values 1 and -1. Thus, $E(\psi_{\mathbf{j}}(\mathbf{x}'(\mathbf{x}))) = 1 - 2P_{-1}$, where P_{-1} is the probability that $\psi_{\mathbf{j}}(\mathbf{x}'(\mathbf{x}))$ equals -1. Since the random string $\mathbf{x}'(\mathbf{x})$ depends on the codebook and the string $\mathbf{x}$ in the original space, $\psi_{\mathbf{j}}(\mathbf{x}'(\mathbf{x}))$ depends on three factors: the codebook, the specific partition $\mathbf{j} \in [2]^{kn}$, and the string $\mathbf{x}$ in the original space that is being transformed. Thus, we are interested in the behavior of the probability

$$P_{-1}(\mathbf{j},\mathbf{x}) = P(\psi_{\mathbf{j}}(\mathbf{x}'(\mathbf{x})) = -1) \tag{7}$$

We have placed emphasis on both $\mathbf{j}$ and $\mathbf{x}$ as factors affecting the value of this probability, but the choice of codebook itself does affect this probability. Having $E(\psi_{\mathbf{j}}(\mathbf{x}'(\mathbf{x}))) = 0$ is equivalent to having $P_{-1}(\mathbf{j},\mathbf{x}) = 1/2$. Thus, we try to understand the conditions under which $P_{-1}(\mathbf{j},\mathbf{x})$ approaches $1/2$ closely.

To understand this probability, we first note that $\psi_{\mathbf{j}}(\mathbf{x}'(\mathbf{x})) = (-1)^{\mathbf{j}\cdot\mathbf{x}'(\mathbf{x})} = \prod_{\alpha=1}^{n}(-1)^{\mathbf{j}_\alpha\cdot\mathbf{x}'_\alpha(\mathbf{x})}$, where $\mathbf{x}'_\alpha(\mathbf{x})$ is the k-bit codon in $\mathbf{x}'(\mathbf{x})$ corresponding to the α-th bit of the lower dimensional point $\mathbf{x}$. Thus it is a random Boolean string. Although we have used the notation $\mathbf{x}'_\alpha(\mathbf{x})$ to show dependence on $\mathbf{x}$, we note that $\mathbf{x}'_\alpha(\mathbf{x})$ depends *only on the α-th feature of* $\mathbf{x}$. $\mathbf{j}_\alpha$ is called a *subpartition*; it is simply the sub-string of $\mathbf{j}$ comprising the features numbered $k\alpha$ through $k\alpha + k - 1$. In other words, it comprises those locations of the string $\mathbf{j}$ that correspond to $\mathbf{x}'_\alpha(\mathbf{x})$. This equation implies that $\psi_{\mathbf{j}}(\mathbf{x}'(\mathbf{x})) = -1$ if and only if $(-1)^{\mathbf{j}_\alpha\cdot\mathbf{x}'_\alpha(\mathbf{x})} = -1$ for an odd number of features of $\mathbf{x}$. Note, however, that in the cases where $\mathbf{j}_\alpha$ is the zero string, we have $(-1)^{\mathbf{j}_\alpha\cdot\mathbf{x}'_\alpha(\mathbf{x})} = 1$. Suppose there are q values of α for which $\mathbf{j}_\alpha$ is not the zero string. This is called the *order of the reflection of* $\mathbf{j}$ [17]. Further, assume without loss of generality that $\mathbf{j}_\alpha$ is not the zero string for $\alpha = 1, 2, \ldots, q$. Then the probability (7) can be written as follows:

$$P(\text{number of } -1\text{s is odd}) = \sum_{\substack{A \subset [q],\\ |A| \text{ odd}}} \prod_{\alpha\in A} P_\alpha \prod_{\alpha\in[q]\setminus A} (1 - P_\alpha) \tag{8}$$

We now show that under certain conditions, this probability converges to $1/2$ as the order of the reflection $q \to \infty$. A subtle point to note here is that we are assuming a fixed partition $\mathbf{j}$ in $[2]^{kn}$ throughout the following analysis. Thus, q cannot actually become arbitrarily large; in particular, $q \le kn$. The rate at which $P_{-1}(\mathbf{j},\mathbf{x}')$ approaches $1/2$ is dependent on the codebook.

Equation 8 can be made more manageable by assuming that the probabilities P_α take r distinct values. Note that since there are $2^k - 1$ different possible nonzero values for sub-partitions $\mathbf{j}_\alpha$, $P((-1)^{\mathbf{j}_\alpha\cdot\mathbf{x}_\alpha} = -1)$ can take $2^k - 1$ distinct values. (Of course, assuming that the lower dimensional string is fixed.) r is the number of distinct values assumed by the probability among those sub-partitions present in $\mathbf{j}$. Thus $1 \le r \le q$. To simplify notation, we use: $BP(\mathbf{p},\mathbf{q},\mathbf{l},r) = \prod_{\beta=1}^{r}\binom{q_\beta}{l_\beta}p_\beta^{l_\beta}(1-p_\beta)^{q_\beta - l_\beta}$. Here $q_1 + \cdots + q_r = q$. q_β is the number of subpartitions for which $P_\alpha = p_\beta$. l_β is the number of -1's actually occurring among those subpartitions for which the probability is p_β. We can rewrite the probability $P_{-1}(\mathbf{j},\mathbf{x})$ as follows.

$$P_{-1}(\mathbf{j},\mathbf{x}) = \sum_{\{l_\beta\}:\sum_1^{r-1} l_\beta \text{odd}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r) \tag{9}$$

We present here a proof that this expression converges to $1/2$ as $q \to \infty$, provided $q_\beta \to \infty$ for *some* β. We note here that there are many possible modes of behavior for the q_βs as $q \to \infty$, ranging from all of them diverging to ∞ to none of them diverging to ∞.

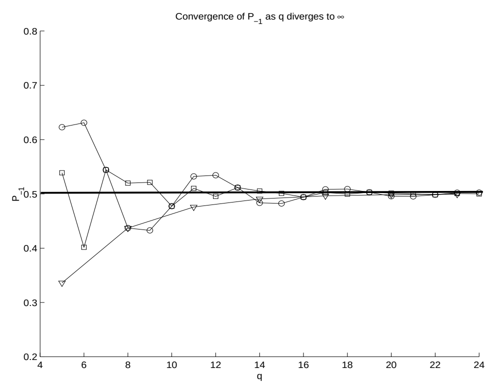


Figure 1: Convergence of P_{-1} to $1/2$ for differing patterns of behavior of q_β. Squares represent convergence when $q_\beta \to \infty \forall \beta$, circles when $q_\beta \to \infty$ for some values of β and triangles when $q_\beta \to \infty$ for no β. $r = 3$ here.

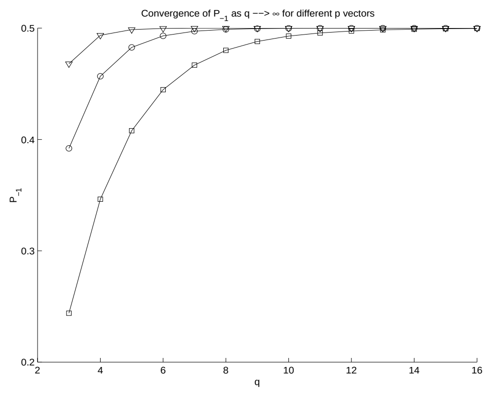


Figure 2: Convergence rates for different probabilities. Convergence is faster when the probability vector has more values close to .5. $r = 3$ in this plot.

Note that for $r = 1$, the above expression reduces to $\frac{1-(1-2p_1)^q}{2}$, which converges to $1/2$ as $q \to \infty$. Assume w.l.g. that l_r is the component of $\mathbf{l}$ that diverges to ∞. We can show that

$$P_{-1}(\mathbf{j}, \mathbf{x}) \longrightarrow \frac{1}{2}. \tag{10}$$

The rationale behind this observation along with the detailed steps of the derivation can be found in Appendix I. The analysis presented in Appendix I does not prove that the convergence holds in the case when none of the q_βs diverge to ∞, our experiments (see for example Figure 1) indicate that this is true as well. We note here that the convergence of $P_{-1}(\mathbf{j}, \mathbf{x})$ to $1/2$ is exponentially fast. For the case $r = 1$, it is exponential: $P_{-1}(\mathbf{j}, \mathbf{x}) = \frac{1-(1-2p_1)^q}{2}$, which shows this. When $r > 2$, the proof by induction shows that it is exponential in q_r. Now, large values of q are not in an exact relationship with large values of the order, or number of nonzero bits in, $\mathbf{j}$. However, high-order strings $\mathbf{j}$ do tend to correspond to large values of q. Thus, the above proof tells us that the reduction in the value of a Fourier coefficient is roughly exponential in the order of the partition $\mathbf{j}$.

As mentioned earlier, this shows that there is a reduction of nonlinearity in the sense of Section 2.3. Note, however, that we are not able to control q — each string $\mathbf{j} \in X'$ has a fixed value of q. Thus, while the above analysis gives us an idea about the behavior of the probability, it does not tell us how to construct a codebook that will help us achieve $P_{-1}(\mathbf{j}, \mathbf{x}) \approx 1/2$. In order to tackle that problem, we note that $P_{-1}(\mathbf{j}, \mathbf{x})$ is a continuous function of the p_βs and converges to $1/2$ as $p_\beta \to 1/2$. This is shown in Figure 2, where the rate of convergence is higher when the probabilities approach $1/2$. Thus, the ideal we should aim for is to have $p_\beta \approx 1/2$ for every subpartition of $\mathbf{j}$. The next section deals with the problem of constructing codebooks that achieve this goal.

5 Codebook Design Factors

Codebooks that minimize $E(\psi_{\mathbf{w}}(\mathbf{x}'(\mathbf{x})))$ are desirable for the current application. We first need to find out the best possible solution that we can ever get. This section explores this question and concludes that there exist no GCT that satisfies the ideal scenario where $p_\beta = 1/2$ for every partition. However, the analysis shows that as we increase the codon-size, the space of all GCTs become increasingly populated by codebooks that are close to the ideal scenario. Therefore, for codebooks with large codon-sizes a randomly generated GCT may have the desirable properties.

In order to study the behavior of codebooks at the subpartition level, we first introduce some notation. We also introduce the *parity matrix*, which allows for an efficient formulation of the problem.

We would like to create a codebook $\{C_0, C_1\}$, such that $C_0 \cup C_1 = [2]^k$, where C_0 is the set of codons corresponding to a bit value of 0 in the protein space and C_1 is the set of codons corresponding to 1. In order to be able to focus on subpartitions, we shift notation a bit. In the remainder of this section, $\mathbf{j}$ denotes a subpartition and $\mathbf{x}$ denotes a codon.

As mentioned at the end of the last section, our goal in the design of a codebook is $\pi(\mathbf{j}, i) = P(\psi_{\mathbf{j}}(\mathbf{x}) = -1 | b = i) = 1/2$, where b denotes a single feature in the protein space, i denotes its value, and $\mathbf{x}$ is a random k-dimensional string (the codon). Let $\mu(\mathbf{j}, i)$ denote the expectation $E[\psi_{\mathbf{j}}(\mathbf{x}) | b = i]$. Note that this is defined for a single subpartition and codon, and is different from the expectation of the last section, which was for the entire partition. The relation $\mu(\mathbf{j}, i) = 1 - 2\pi(\mathbf{j}, i)$ holds. The random bit-string $\mathbf{x}$ is chosen according to the distribution assigned to strings in C_0 and C_1.

There are several possible directions for optimization of the codebook:

1. We can try to achieve $\mu(\mathbf{j}, i) = 0$ for *most* values of $\mathbf{j}$. That is, $\mu(\mathbf{j}, i)$ may be large (close to 1) for a few subpartitions $\mathbf{j}$, but is mostly equal to 0.

2. We could choose a codebook that minimizes $\max_{\mathbf{j}} |\mu(\mathbf{j}, i)|$. That is, we may have many non-zero expectations for individual values of $\mathbf{j}$, but in no case is the deviation severe.

3. We could try to minimize $\sum_{\mathbf{j}} |\mu(\mathbf{j}, i)|$ or $\sum_{\mathbf{j}} (\mu(\mathbf{j}, i))^2$. These give a more balanced codebook. The former gives equal weights to all nonzero expectations, whereas the latter penalizes large nonzero expectations more than small ones.

We next introduce the *parity matrix*, which helps in analysis of the codebooks. In what follows, matrices and vectors are represented in boldface. The parity matrix is a matrix whose elements represent the parity between $\mathbf{x}$ and $\mathbf{j}$. We define the parity matrix $\mathbf{A}_k$ for k-bit codons as $\mathbf{A}_k = ((\psi_{\mathbf{j}}(\mathbf{x})))_{\mathbf{j},\mathbf{x}}$. Thus, each row corresponds to a subpartition and each column corresponds to a codon. We assume that the partitions and codons appear in their natural order. The parity matrices for codons of orders 1 and 2 follow.

$$\mathbf{A}_1 = \left[\begin{array}{c|cc} {}_{j}\diagdown^{x} & 0 & 1 \\ \hline 0 & 1 & 1 \\ 1 & 1 & -1 \end{array}\right] \quad \mathbf{A}_2 = \left[\begin{array}{c|cccc} {}_{j}\diagdown^{x} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 1 & 1 & 1 \\ 01 & 1 & -1 & 1 & -1 \\ 10 & 1 & 1 & -1 & -1 \\ 11 & 1 & -1 & -1 & 1 \end{array}\right]$$

The parity matrix has many interesting properties. It is nonsingular: note that, in general,

$$\mathbf{A}_k = \left[\begin{array}{c|c} \mathbf{A}_{k-1} & \mathbf{A}_{k-1} \\ \hline \mathbf{A}_{k-1} & -\mathbf{A}_{k-1} \end{array}\right]$$

Non-singularity of $\mathbf{A}_k$ can be proved inductively using this identity. Also note that the element-wise product of two rows of $\mathbf{A}_k$ always results in a row of $\mathbf{A}_k$.

We now re-state the codebook design goals in terms of the parity matrix. Note that the first row of $\mathbf{A}_k$, corresponding to the partition 00...0, is not significant. Partition the parity matrix as follows:

$$\mathbf{A}_k = \left[\begin{array}{c} 11\cdots1 \\ \hline \mathbf{B}_k \end{array}\right]$$

$\mathbf{B}_k$ is a $(2^k-1)\times 2^k$ matrix. Each row of $\mathbf{B}_k$ corresponds to a nonzero subpartition. Selection of a codebook is reflected in the matrix by first post-multiplying by a permutation matrix $\mathbf{P}_k$ (thus permuting the columns) and then partitioning the resulting matrix. In other words, with $\mathbf{C}_k = \mathbf{B}_k\mathbf{P}_k$, selecting a codebook leads to a partitioning of $\mathbf{C}_k$ as $\mathbf{C}_k = [\mathbf{C}_k^0 : \mathbf{C}_k^1]$, where $\mathbf{C}_k^0$ has order $(2^k-1)\times p$ and $\mathbf{C}_k^1$ has order $(2^k-1)\times(2^k-p)$. Here p is the size of the set of codons corresponding to a bit value of 0 in the protein space, and the columns of $\mathbf{C}_k^i$ correspond to the codons that a bit-value of i is transformed to for $i=0$ or 1.

Assume that we use the individual probability distributions $(f_1^0, f_2^0, ..., f_p^0)$ and $(f_1^1, f_2^1, ..., f_{2^k-p}^1)$ on the codons corresponding to 0 and 1 respectively. Thus $f_j^i \geq 0$ and $\sum_j f_j^i = 1$ for $i=0,1$. Define $\mathbf{f}_0 = (f_1^0, f_2^0, ..., f_p^0)^T$ and $\mathbf{f}_1 = (f_1^1, f_2^1, ..., f_{2^k-p}^1)^T$. Then the elements of the vectors $\mathbf{C}_k^i\mathbf{f_i}$ equal the lists of conditional expectations $\mu(\mathbf{j}, i)$ for $i=0$, 1. The design goals can be restated as minimizing $||\mathbf{C}_k^i\mathbf{f}_i||$ for a suitable definition of the norm $||\cdot||$. In particular, minimizing $||\mathbf{C}_k^i\mathbf{f}_i||$ corresponds to optimization 1 if $||\cdot||$ is the counting norm, to optimization 2 if $||\cdot||$ is the ℓ_∞ norm, and to optimization 3 if $||\cdot||$ is the ℓ_1 or ℓ_2 norm. (The counting norm of a vector is defined as the number of non-zero components of the vector, and is not strictly a norm since it does not satisfy the linearity property. This does cause problems with the analysis of the first design goal. However, as we show later in this paper, design goal 1 is easier to achieve than the other design goals and does not require the use of the norm.) We observe that in the language of matrices, selecting a codebook translates to choosing the permutation matrix $\mathbf{P}_k$, the codon-set size p, and the distributions $\mathbf{f}_i$.

A natural question is: can we select a codebook that achieves $||\mathbf{C}_k^i\mathbf{f}_i|| = 0$? We now state a lemma which shows that this is not possible.

Lemma 1 (Impossibility of Finding a Perfect Codebook) *For any choice of $p (p \neq 0$ or $2^k)$, $\mathbf{P}_k$ and $\mathbf{f}_i$, we have $\mathbf{C}_k^i\mathbf{f}_i \neq 0$ for one of the partitions (corresponding to 0 or 1).*

Proof: See Appendix II. ■

Thus, it is not possible to construct a perfect codebook. Given this, the next natural step is to optimize the codebook by minimizing the norms described above with respect to choice of $p, \mathbf{P}_k$ and $\mathbf{f}_i$. The next section describes methods of doing this for some of these norms.

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

Table 3: A 3-bit codebook that minimizes the counting norm.

6 Minimizing the Norms to Optimize Codebooks

The methods for minimizing the various norms are different. The easiest to minimize is the "counting" norm, which is described in the next subsection.

6.1 Minimizing the Counting Norm

We know that $\mathbf{C}_k^i \mathbf{f}_i$ cannot be the zero vector. Thus the smallest possible number of nonzero entries is 1. We show here that it is possible to find a codebook that yields a nonzero expected value for just one partition $\mathbf{j}$ by giving a method for constructing such a codebook.

We specify the codebook in terms of its codons as follows. Assume that the codon size is k. We assign the codons for a bit value of b to be the elements of the set $T_b = \{\mathbf{x} \in [2]^k | \mathbf{x}_0 = b\}$. Here $\mathbf{x}_0$ refers to the first bit of the binary string $\mathbf{x}$. Thus, this is a symmetric codebook (one for which the number of codons for each "amino acid" is the same). As an example, the codebook for codons of length 3 is given in Table 3.

6.2 Minimizing the ℓ_2 Norm

As mentioned before, the ℓ_2 norm penalizes large deviations from 0 more and small deviations less. This is required in some situations. Minimizing the ℓ_2 norm is simplified by the fact that the norm is invariant under choice of the minimization parameters (p, permutation matrix $\mathbf{P}_k$ and $\mathbf{f}_i$), as proved in the Lemma below. As a consequence, any codebook is as good as another for minimization of the ℓ_2 norm. We first introduce a slightly different formula for the norm that we are trying to minimize.

As stated before, we would like to minimize the norm $||\mathbf{C}_k^i \mathbf{f}_i||_2$. Notice that this expression is the same as the expression $||\mathbf{C}_k \mathbf{f}||_2$, where f is a $2^k \times 1$ vector with an added restriction (either the first p components or the last $2^k - p$ components are zero, and $\sum_i f_i = 1$, where f_i here represents the i-th component of $\mathbf{f}$).

Lemma 2 (Invariance of the ℓ_2 Norm) *The norm $||\mathbf{C}_k \mathbf{f}||_2$, where $\mathbf{C}_k = \mathbf{B}_k \mathbf{P}_k$, is independent of the choice of p, $\mathbf{P}_k$ and the probability vector $\mathbf{f}$.*

Proof: See Appendix II.

Thus, minimization of the ℓ_2 norm is not an issue. We proceed to a more interesting norm, the ℓ_∞ norm.

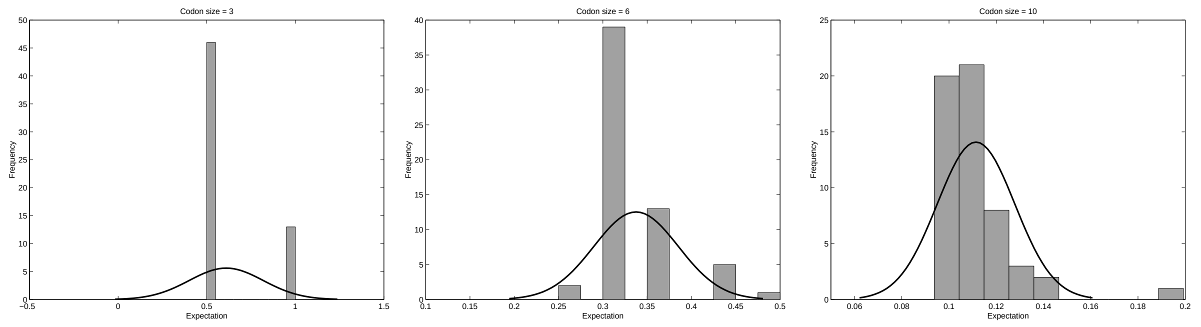


Figure 3: Distribution of the ℓ_∞ norm for codon sizes 3, 6, and 10. These were estimated by randomly generating 60 codebooks of each order and creating histograms of the ℓ_∞ norm for each order.

6.3 Minimizing the ℓ_∞ Norm

As mentioned in the last section, minimizing the ℓ_∞ norm means minimizing the maximum expectation $\mu(\mathbf{j}, i)$. That is, we wish to find the codebook whose ℓ_∞ norm equals

$$\min_{C_0, C_1} \max_{\substack{\mathbf{j} \in [2]^k \\ i \in \{0,1\}}} |\mu(\mathbf{j}, i)|$$

Figure 8 shows how the norm of the optimal codebook behaves as the codon size increases, assuming a uniform distribution on the codebook corresponding to each feature in the original n-dimensional protein space, and $p = 2^{k-1}$. The ℓ_∞ norm decreases rapidly with the size of the codebook. Finding an *optimal* codebook may be difficult in this case because of the combinatorial optimization problem involved, *viz.* finding a number p, a permutation matrix $\mathbf{P}_k$ and a distribution $\mathbf{f}_i$ that optimize the norm. However, Figure 3 indicates that searching for near-optimal codebooks using a randomized search algorithm may yield near-optimal results. In addition, this is not an expensive process because near-optimal codebooks once found can be stored in a table, eliminating the need for expensive re-calculation of these codebooks.

This section presented methods of finding optimal and near-optimal codebooks with respect to three different criteria: the counting norm, the ℓ_2 norm, and the ℓ_∞ norm. Optimizing these norms gives us codebooks that make the vector of codon-wise probabilities approach 1/2 more closely. As pointed out in Figure 2, this leads to faster convergence of P_{-1} (see Section 4) to 1/2. The next section shows simulations and experimental results performed using the codebooks found using the techniques of this paper.

7 The Perceptron and the XOR Function

Section 4 noted that the randomized GCTs construct a representation with reduced nonlinear dependence among the features, and that the reduction grows exponentially with respect to the order of the corresponding partition. In this section we test this hypothesis in the following manner. We consider the Perceptron [28], a linear classifier, and the nonlinear XOR problem. An n-bit XOR problem is defined as follows:

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

The Perceptron cannot learn the XOR problem [23] since it is a linear classifier. However, we apply Perceptron on a representation of the XOR constructed by randomized GCTs and test its performance. If the higher order nonlinearity is sufficiently reduced by the transformation then the Perceptron should be able to learn the problem. In fact that is exactly what we observed.

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-classifications and the number of entries in the testing data set. For all the experiments reported here, the training data set is drawn from the entire domain using uniform distribution. The testing data set is also comprised of the entire domain. Table 4 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.

Although the representation of the Perceptron uses the input Boolean features themselves, not their Fourier transformation themselves, the observations remain valid. This is because $\mathbf{x}.\mathbf{j} = \frac{\psi_{\mathbf{j}}(\mathbf{x})+1}{2}$ for any order-1 partition $\mathbf{j}$. In other words, for a linear representation, the input features are related to their respective Fourier transformation through a linear transformation, comprised of a translation and a scaling.

0	01, 10
1	00, 11

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

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

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

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

Figure 4 (left) shows the average mis-classification error of the Perceptron with no transformation. Figures 4 (middle) and (right) show the performance in a representation that uses GCT-s with codons of size two and four respectively. The average error is computed over 10,000 independent sessions. These figures show 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. The figures also show that the code-book with codons of size four is capable of learning up to five-bit XOR problem accurately. The following section

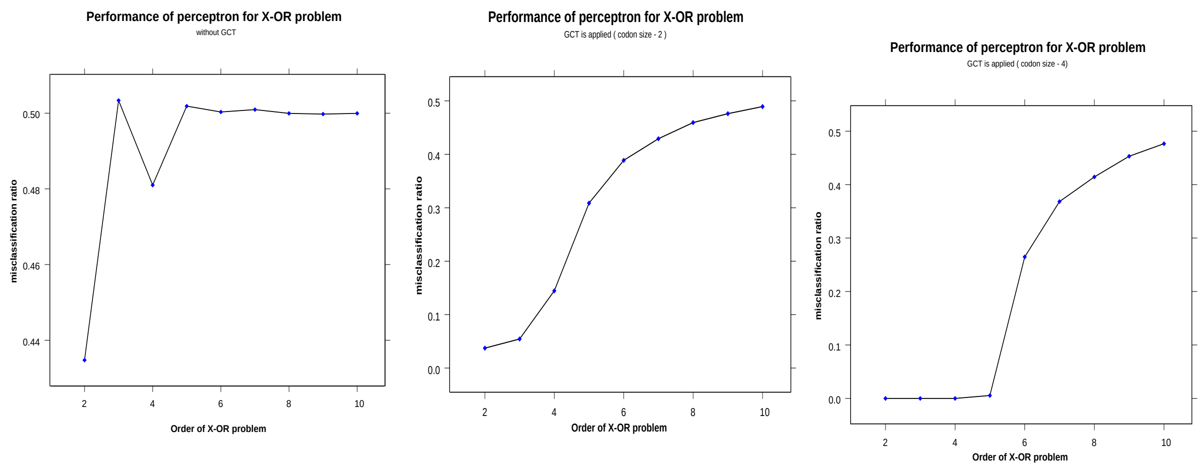


Figure 4: Average mis-classification ratio vs. problem size (number of input variables) for (left) no transformation, (middle) code-books with codon size two, and (right) code-books with codon size four. The reported result is an average of $10,000$ independent runs.

considers an additional set of experiments that demonstrate the performance of the proposed algorithm for larger problems with bounded nonlinearity.

8 Learning Problems with Bounded Nonlinearity

The XOR functions considered in the previous section are fully nonlinear where all the features interact with each other. Learning a function like XOR requires observing all the domain members. However, for function with only a subset of nonlinearly interacting features, all domain members may not be needed for learning. In other words, a fraction of domain members can be used for learning the model. This section presents experiments with such boundedly nonlinear functions.

8.1 Disjunction of Multiple XOR Problems

We have noted earlier that the code-book of fixed codon size is capable of classifying XOR problem of certain order (Figure 4) accurately. However to understand the effect of GCT on nonlinearity, we need to consider problems with lesser nonlinearity (then XOR).

Consider the following disjunctive function constructed using a collection of smaller XOR functions: $\phi(\mathbf{x}) = f(\mathbf{x_1}) \vee f(\mathbf{x_2}) \vee ... \vee f(\mathbf{x_m})$ where each $f(\mathbf{x_i})$ is defined as follows,

$$f(\mathbf{x_i}) = \begin{cases} 1 & \text{If } \mathbf{x_i} \text{ contains odd number of 1-s} \\ 0 & \text{Otherwise.} \end{cases}$$

Our experiments apply the Perceptron to learn a new function, ϕ constructed by the randomized GCT-s. Earlier in the case of XOR, during the learning stage we presented the entire domain member to the Perceptron and once the learning is over the trained Perceptron is tested on the domain members. However as we are considering a less nonlinear function ϕ

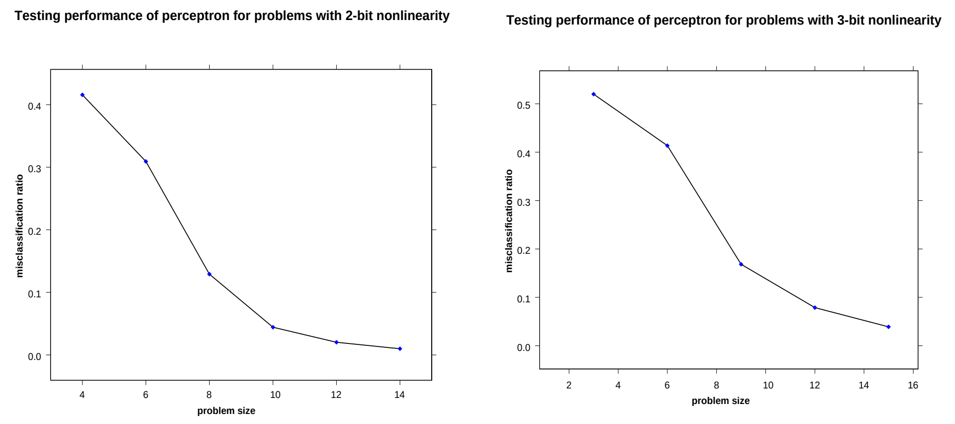


Figure 5: Average testing mis-classification ratio vs. problem size (number of input variables) with two (left) and three (right) bit nonlinearity for code-books with codon size three. The reported result is an average of $1,000$ independent runs.

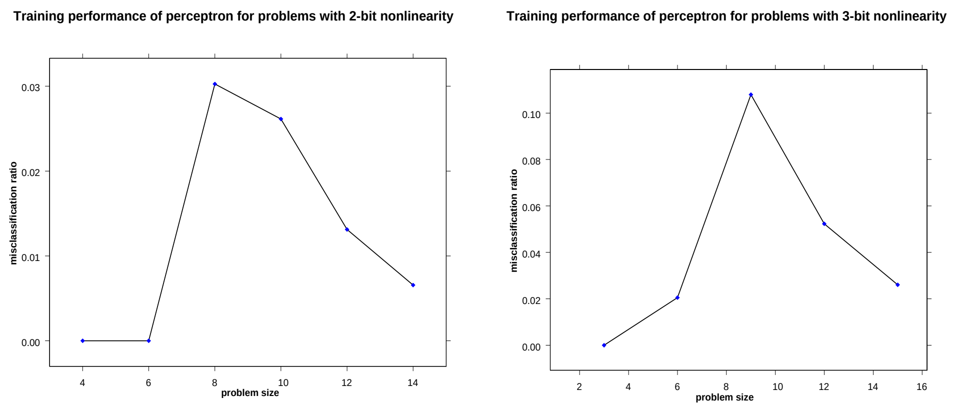
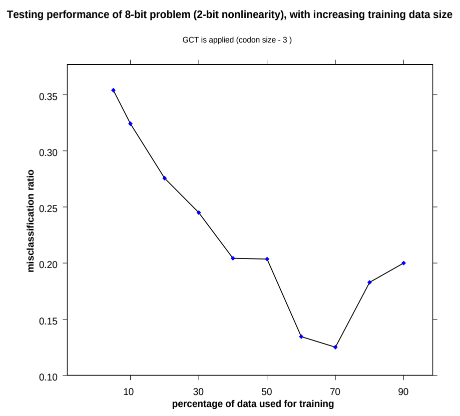


Figure 6: Average training mis-classification ratio vs. problem size (number of input variables) with two (left) and three (right) bit nonlinearity for code-books with codon size three. The reported result is an average of $1,000$ independent runs.

here, instead of presenting the entire domain to the Perceptron we present a random subset of the domain member to the Perceptron and for testing we use rest of the domain members.

8.2 Experimental Results

Figure 5 presents the average testing mis-classification error of the Perceptron trained over a representation constructed by the code-book with codon size three and using only 60% of the domain member. The average error is computed over $1,000$ independent trials. The top left most figure shows that a ten bit ϕ-function with two bit nonlinearity, can be classified almost perfectly using only 60% of the domain member and three bit codon. The graph on the top right shows similar testing error variation for the three bit nonlinearity, except that the size of the ϕ -function that can be properly classified is twelve. The training error is reported in Figure 6. It is low for all cases.



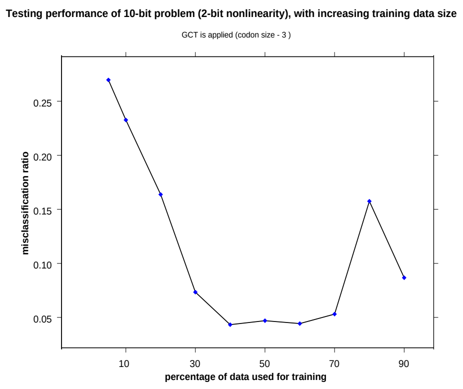


Figure 7: Average testing mis-classification ratio calculated over $1,000$ learning sessions with different percentage of the data in use for different problem sizes of order eight (left) and ten (right) respectively.

Having investigated the effect of GCT on a simplified nonlinear problem, another interesting question is about the percentage of training data required to classify ϕ accurately.

Next we present experimental result for the function ϕ with fixed number of nonlinearly interacting features per sub-function $(f(x_i))$. The variation of accuracy is presented with respect to increasing fraction of the entire domain used to train to the Perceptron. Once the learning is over trained perceptron is tested on rest of the domain members. Figure 7 presents the average mis-classification error of the Perceptron trained over different fraction of the domain member, starting from 5% to 90% of the actual data set, keeping nonlinearity constant (2-bit). The average testing error is computed over $1,000$ independent sessions. The left most figure shows to classify 8-bit problem (ϕ) accurately we need 60% of the domain member. The error ratio comes down to 8% when 60% data is in use. The graph on the right shows a similar error variation for 10-bit problem size, except for the fact that we need at least 30% of the domain member to classify it accurately.

The set of simulations in the next study consists of a set of experiments on an XOR problem of varying degrees of nonlinearity, from 2 bits to 8 bits. The XOR is taken over all the bits. For this set of experiments, we computed the training error of the perceptron. The performance of a perceptron learned on the data, transformed using codons of different sizes, is shown in Figure 9. As can be seen, there is a clear decrease in the training error as the order of the codon is increased, for all the problems. This indicates that it becomes easier to approximate the transformed function with a linear function as codon size increases. Also as expected, XOR problems of higher order are more difficult to approximate using a linear function, but these too show a decreasing trend in the training error.

Figure 11 shows the extent to which a codon of a certain size is useful. It contains two plots: one for an n-bit XOR problem on an n-bit input, and one for an $n/2$-bit XOR problem on an n-bit input. The n-bit XOR of n inputs is computed by XOR-ing all n bits. The $n/2$-bit XOR on n bits is computed by computing the disjunction of the XORs of the first $n/2$ bits and the last $n/2$ bits. This plot is intended to show what codon sizes are required if we are to approximate the function on the higher dimensional space induced by the GCT with

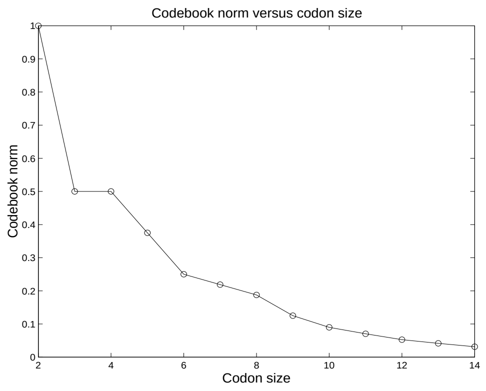


Figure 8: The ℓ_∞ norm of near-optimal codebooks of varying orders.

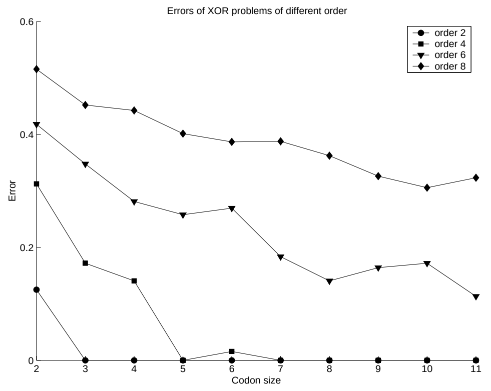


Figure 9: Perceptron errors on transformed XOR problems of varying order.

a linear function. For each codon size, the graph plots that value of n for which the error exceeds 0.3. There is a clear increasing trend for this value as the size of the codon increases, which tells us that we can solve more difficult problems by using a larger codon size.

These results can be explained from the perspective of the concept $\text{ord}(\epsilon, D_S, f, S)$, defined in Section 2.3. Let us agree to call a function *ϵ-linear* if its $\text{ord}(\epsilon, D_S, f, S) = 1$. That is, we call it ϵ-linear if it can be approximated well by a linear function. Then Figure 9 can be interpreted as showing the ϵ for which ϵ-linearity is achieved in the higher dimensional space, for different codon sizes. Thus, for an order 2 XOR problem, 0.2-linearity is achieved in the higher dimensional space with a codon size of 2. For an order 4 XOR problem, 0.2-linearity is achieved with a codon size of 3. For an order 6 XOR, it is achieved with a codon size of 8. Figure 11 shows the largest problem size for which 0.3-linearity is achieved using a codon of a specified size. For an n-bit XOR, a codon size of 3 gives 0.3-linearity up to problems of size 6. A codon size of 4 gives 0.3-linearity up to problems of size 7, and so on.

8.3 Additional Experiments

This section reports additional classification experiments using the GCT-based representation construction technique.

Classifying Tomography Images

This set of experiments considered a dataset describing diagnosing of cardiac Single Proton Emission Computed Tomography images (available from UCI Machine Learning Data Repository). The data consisted of 267 samples of 22 binary features and 1 binary classification each. We compared the performance of perceptrons on the both the untransformed data and the data transformed by codebooks with codons sizes ranging from 2 to 20. 100 trials were done for each size of codebook. Our results show a clear reduction in nonlinearity resulting from the application of the GCT. The Perceptron learned the untransformed data within an error rate of 20.5% or 0.205. Figure 10 shows the performance of the perceptron on the transformed data. This particular experiment reports training and testing using the complete data set. However, we have also performed experiments with different proportions

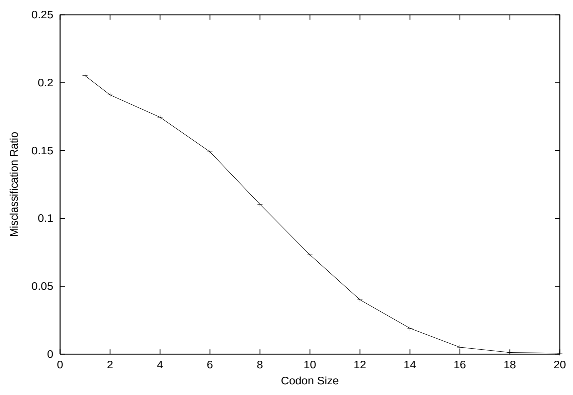


Figure 10: Average testing mis-classification ratio calculated over the complete data set for different codon sizes.

of the data set and noted similar results. Codebooks with codons of size 2 exhibited a MSE of .191. where codebooks with 10 bit codons exhibited a MSE of .073, and codebooks with 20 bit codons showed a MSE of only .0006.

Classifying Stock Market Data using GCTs, Perceptrons, and C4.5

The data considered in this set of experiments is stock market data, parsed from various web-sites publishing stock quotes. The data downloaded consisted of quotes of stock prices for various scripts, along with several continuous and discrete valued features of interest. The stock data was pre-processed into Boolean attributes, with the Boolean classification being equivalent to "up" or "down" — that is, a rise or a fall in the value of the script. Two different experiments were tried on this data. Performance of a simple linear classifier, the perceptron, was compared to C4.5 in the first set of experiments. The data for this set of experiments had 25 features in 97 rows. The classification accuracy of C4.5 on the *untransformed* data was found to be 12.4%, or 0.124. There is a significant reduction in the error, which disappears completely at a codon size of about 4. This indicates that the GCTs enable us to approximate the function with a linear function with almost zero error for this small data set. While this result was interesting, it should be noted that the data was too small.

The second experiment conducted on the stock data used a larger data set, with 6390 rows and 19 features. A perceptron run on the untransformed data produced an error of 37%. C4.5 gave a training error of more than 16% when applied to the untransformed data. The results when C4.5 was used on the transformed data are summarized in Figures 12, 13 and 14. The first point in Figure 12 (i.e. the one corresponding to a codon size of 1) represents the performance of C4.5 on untransformed data. This figure shows the training error of a C4.5 tree learned on transformed data as a function of the size of the codons used to transform the data. The error shows dramatic improvement with increase in codon size. Figure 13 shows how the size of the decision tree (number of nodes) changes. The tree size exhibits an interesting pattern, increasing drastically up to a codon size of 3, and then falling off gradually. The depth of the trees built is shown in Figure 14. As is seen from

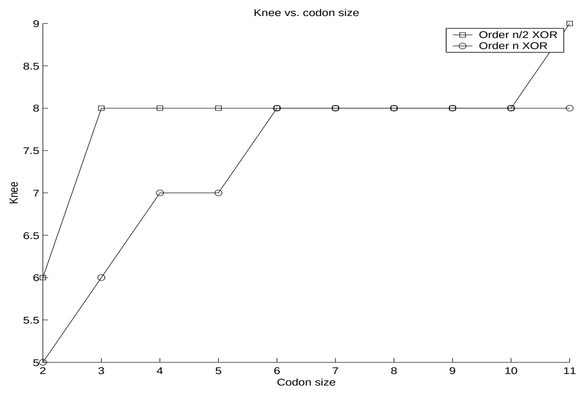


Figure 11: Behavior of the knee with changing codon size for order n and $n/2$ XOR problems.

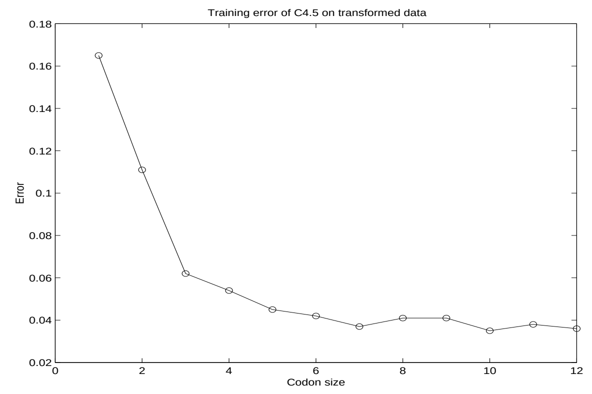


Figure 12: Error of C4.5 decision tree on data transformed with different codon sizes.

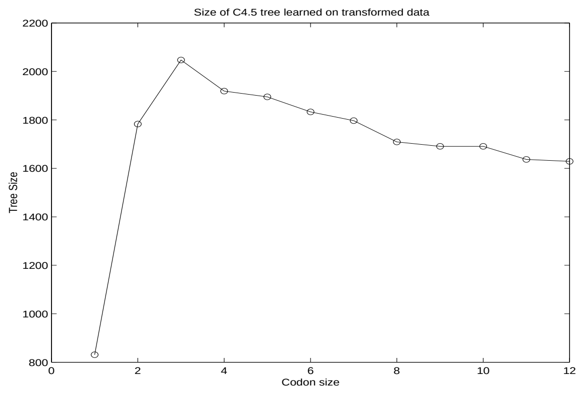


Figure 13: Change is size of C4.5 decision tree with different codon sizes.

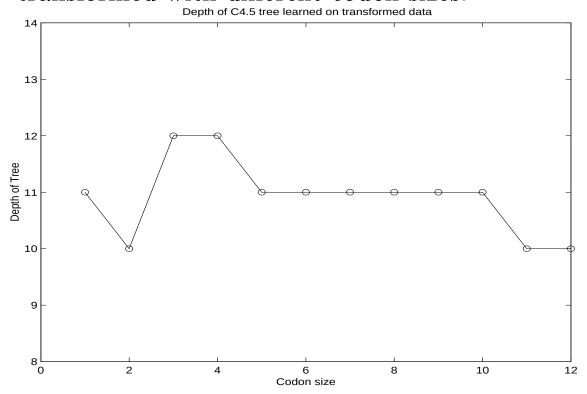


Figure 14: Depth of decision tree learned after transforming with different codon sizes.

Figure 12, the error of all trees learned using transformations with codon sizes greater than 4 is less than 0.05. Tree depth does not change significantly, as seen in Figure 14. As noted elsewhere [25, 24], the depth of a decision tree and the magnitude of its Fourier coefficients are intimately related. In particular, a decision tree has depth d if and only if all its Fourier coefficients of order greater than d are zero. Thus, if the error of a decision tree of depth d is less than ϵ, it means that $\text{ord}(\epsilon, D_S, f, S) \leq d$, where f is the function represented by the decision tree. We conclude here that $\text{ord}(0.05, D_S, f, S)$ is less than 12 when codons of size 4 and above are used. The same two graphs also tell us that $\text{ord}(0.04, D_S, f, S)$ is less than 11 when codons of sizes 7 and above are used. The $\text{ord}(\epsilon, D_S, f, S)$ of the induced function decreases as the size of the codon used increases. In addition, this graph represents an increase in accuracy without a corresponding increase in nonlinearity. This suggests that for a fixed value of the error, one would get a shallower C4.5 tree (demonstrating a decrease in nonlinearity).

9 Summary, Future Work and Conclusions

This paper extends the work on randomized GCTs begun elsewhere [14, 17]. Results in these works indicated that it was possible to obtain a relative linearization of nonlinear functions using randomized transformations resembling the translation process in nature. This paper is an attempt to study the effects of choosing a particular codebook on the linearization of a function under a systematic theoretical framework. The theory is developed using the MFT for representing functions defined on discrete spaces. An examination of the effects of codebooks on orders of nonlinearity is initially conducted at a macroscopic level (see Section 4). It is shown that the overall reductions in nonlinearity depend heavily on the properties of the codebooks at the level of individual codons. Probabilistic properties of codebooks are then studied at this level to come up with a set of criteria for good codebooks. Section 6 offers several techniques for finding codebooks according to the criteria in Section 5. Experimental results corroborating the claims in the paper, using the perceptron and decision trees, are then presented.

While this paper addresses many issues in codebook design, several topics still need to be addressed:

1. Deeper investigations into the properties of the expression in Equation 9 may yield better insights as to what properties of the probability vector $\mathbf{p}$ yield the most uniformly low values for $P_{-1}(\mathbf{j}, \mathbf{x})$.

2. A better and more efficient method for finding codebooks optimized for the ℓ_∞ norm needs to be found. The method presented in this paper essentially exploits the fact that the probability of finding a good codebook is high, and is a simple random search.

3. Note that this paper does not claim that the GCTs construct a more efficient (complexity-wise) representation. GCTs construct a representation that can be solved using linear techniques. An analysis of the energy distribution of the spectrum is essential for identifying the cases where the new representation is more efficient.

4. While the results of this paper do indicate that GCTs linearize functions, techniques for generalizing the model built using the transformed data to the entire lower dimensional domain need to be studied. This may involve some *derandomization* (i.e. finding a function that is relatively deterministic, yet retains crucial properties of the randomized GCTs). An alternative approach is to find a way to use the model learned on the transformed data set to induce an accurate model on the entire lower-dimensional space.

5. Our use of GCTs currently considers only discrete data. It is worthwhile to investigate the applicability of randomized GCT-like techniques to functions defined over continuous feature spaces and mixtures of continuous and discrete feature spaces.

6. The exact effect of GCTs on the Fourier spectrum of a function needs to be studied in greater detail. To illustrate this, we consider the experiments on stock-market data, where GCTs produce almost complete linearization for smaller data sets, but do not produce complete linearization for larger data sets, for most of the codon sizes that we have experimented with. The underlying noise and non-stationarity of the stock-market data may have a role in that. As explained in Section 8, it is clear that some degree of linearization is taking place. An identification of the extent of linearization that takes place for a given set of data would prove useful.

In summary, this paper presents an introductory analytical treatment of a promising new linearization technique. There are many areas that are as yet unexplored, and further steps need to be taken before the randomized GCT technique can be used as a machine learning and data mining tool.

9.1 Acknowledgments

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. IIS-9803660 and IIS-0083946. The first author would also like to acknowledge support from the NSF CAREER award IIS-0093353. The authors would like to thank Sweta Pittie for providing the stock market data used in this paper and Todd Olsen for performing the experiment with Tomography data. The authors would also like to thank Dr. Thomas Mathew for many valuable discussions.

References

[1] 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).

[2] J. Bashford, I. Tsohantjis, and P. Jarvis. A supersymmetric model for the evolution of the genetic code. *Proceedings of the National Academy of Science USA*, 95:987–995, 1998.

[3] P. Beland and T. Allen. The origin and evolution of the genetic code. *Journal of Theoretical Biology*, 170:359–365, 1994.

[4] C. Bridges and D. E. Goldberg. The nonuniform Walsh-schema transform. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 13–22, 1991.

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

[6] 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.

[7] S. Fukuchi, T. Okayama, and J. Otsuka. Evolution of genetic information flow from the viewpoint of protein sequence similarity. *Journal of Theoretical Biology*, 171:179–195, 1994.

[8] D. E. Goldberg. Genetic algorithms and Walsh functions: part I. *Complex Systems*, 3(2):129–152, 1989.

[9] 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).

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

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

[12] 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.

[13] F. Jacob and J. Monod. Genetic regulatory mechanisms in the synthesis of proteins. *Molecular Biology*, 3:318–356, 1961.

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

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

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

[17] H. Kargupta and S. Ghosh. Towards machine learning through genetic code-like transformations. *Genetic Programming and Evolvable Machine Journal*, 3(3):231–258, 2002.

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

[19] 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, 1999.

[20] R. D. Knight and L. F. Landweber. The early evolution of the genetic code. *Cell*, 101:569–572, 2000.

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

[22] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuit, Fourier transform and learnability. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 574–579, 1989.

[23] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, Mass., 1969.

[24] B. Park. *Knowledge Discovery From Heterogeneous Data Streams Using Fourier Spectrum of Decision Trees*. PhD thesis, Washington State University, Pullman, WA, 2001.

[25] B. Park, R. Ayyagari, and H. Kargupta. A Fourier analysis based approach to learning decision trees in a distributed environment. In *Proceedings of the first SIAM international conference on data mining*, 2001.

[26] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

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

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

[29] 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.

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

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

Appendix: I

Detailed derivation of Equation 10:

$$\begin{aligned}
P_{-1}(\mathbf{j},\mathbf{x}) &= \sum_{\mathbf{l}:\sum_1^r l_\beta \text{odd}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r) \\
&= \sum_{\mathbf{l}:\sum_1^{r-1} l_\beta \text{odd}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r-1) \sum_{l_r \text{even}} \binom{q_r}{l_r} p_r^{l_r}(1-p_r)^{q_r-l_r} \\
&+ \sum_{\mathbf{l}:\sum_1^{r-1} l_\beta \text{even}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r-1) \sum_{l_r \text{odd}} \binom{q_r}{l_r} p_r^{l_r}(1-p_r)^{q_r-l_r} \\
&= \sum_{l_r \text{even}} \binom{q_r}{l_r} p_r^{l_r}(1-p_r)^{q_r-l_r} \sum_{\mathbf{l}:\sum_1^{r-1} l_\beta \text{odd}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r-1) \\
&+ \sum_{l_r \text{odd}} \binom{q_r}{l_r} p_r^{l_r}(1-p_r)^{q_r-l_r} \sum_{\mathbf{l}:\sum_1^{r-1} l_\beta \text{even}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r-1) \\
&\longrightarrow 1/2 \cdot \sum_{\mathbf{l}:\sum_1^{r-1} l_\beta \text{odd}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r-1) + 1/2 \cdot \sum_{\mathbf{l}:\sum_1^{r-1} l_\beta \text{even}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r-1) \\
&= 1/2 \cdot \left(\sum_{\mathbf{l}:\sum_1^{r-1} l_\beta \text{odd}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r-1) + \sum_{\mathbf{l}:\sum_1^{r-1} l_\beta \text{even}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r-1) \right) \\
&= \frac{1}{2} \sum_{\mathbf{l}} BP(\mathbf{p},\mathbf{q},\mathbf{l},r-1) = \frac{1}{2}
\end{aligned}$$

The convergence in the above derivation holds because the induction hypothesis implies that each of the two smaller sums in the final expression converges to 1/2 if $q_\beta \to \infty$ for every β.

Appendix: II

Lemma 3 (Impossibility of Finding a Perfect Codebook) *For any choice of $p (p \neq 0$ or $2^k)$, $\mathbf{P}_k$ and $\mathbf{f}_i$, we have $\mathbf{C}_k^i \mathbf{f}_i \neq 0$ for one of the partitions (corresponding to 0 or 1).*

Proof: It is enough to prove that $\mathbf{C}_k^i$ is a full row-rank matrix for some i. In order to do this, we define $\mathbf{D}_k$ by

$$\mathbf{A}_k = \left[\begin{array}{c|c} 1 & 11\cdots1 \\ \hline 1 & \\ 1 & \\ \vdots & \mathbf{D}_k \\ 1 & \end{array}\right]$$

where $\mathbf{D}_k$ is bottom right $(2^k-1)\times(2^k-1)$ sub-matrix of $\mathbf{A}_k$. The matrix $\mathbf{D}_k$ is nonsingular. We can write $\mathbf{D}_k$ recursively as

$$\mathbf{D}_k = \left[\begin{array}{c|c|c} & 1 & \\ & 1 & \\ \mathbf{D}_{k-1} & \vdots & \mathbf{D}_{k-1} \\ & 1 & \\ \hline 11\cdots1 & \text{-1} & -1-1\cdots-1 \\ \hline & \text{-1} & \\ & \text{-1} & \\ \mathbf{D}_{k-1} & \vdots & -\mathbf{D}_{k-1} \\ & \text{-1} & \end{array}\right]$$

Using this recursion, and assuming that $\mathbf{D}_{k-1}$ is a nonsingular matrix, it is easy to show inductively that for $\mathbf{y} \in \mathbb{R}^{2^k-1}$, $\mathbf{D}_k\mathbf{y} = 0$ implies $\mathbf{y} = 0$. Thus $\mathbf{D}_k$ is nonsingular for every k. In particular, this implies that the set of columns of $\mathbf{D}_k$ is linearly independent. Note that the matrix $\mathbf{B}_k$ equals $[\mathbf{1} : \mathbf{D}_k]$, where $\mathbf{1} = [11\cdots1]^T$. Thus, in any partition of $\mathbf{C}_k$, which is just a column-wise permutation of $\mathbf{B}_k$, into $\mathbf{C}_k^0$ and $\mathbf{C}_k^1$, at least one of these, say $\mathbf{C}_k^i$, two matrices will consist of only columns from $\mathbf{D}_k$. Thus $\mathbf{C}_k^i\mathbf{f}_i \neq 0$, since $\mathbf{f}_i \neq 0$. This completes the proof. ■

Lemma 4 (Invariance of the ℓ_2 Norm) *The norm $||\mathbf{C}_k\mathbf{f}||_2$, where $\mathbf{C}_k = \mathbf{B}_k\mathbf{P}_k$, is independent of the choice of p, $\mathbf{P}_k$ and the probability vector $\mathbf{f}$.*

Proof: Minimizing $||\mathbf{C}_k\mathbf{f}||_2$ is equivalent to minimizing $||\mathbf{C}_k\mathbf{f}||_2^2$, which equals $[\mathbf{C}_k\mathbf{f}]^T[\mathbf{C}_k\mathbf{f}] = [\mathbf{B}_k\mathbf{P}_k\mathbf{f}]^T[\mathbf{B}_k\mathbf{P}_k\mathbf{f}] = \mathbf{f}^T\mathbf{P}_k^T\mathbf{B}_k^T\mathbf{B}_k\mathbf{P}_k\mathbf{f}$. Now $\mathbf{A}_k$ is a Hadamard matrix, and hence we have $\mathbf{A}_k^T\mathbf{A}_k = 2^k\mathbf{1}\mathbf{1}^T$, where $\mathbf{1}$ is the $2^k \times 1$ vector of 1s. (It is easy to verify this directly as well.) Now,

$$\mathbf{A}_k^T\mathbf{A}_k = [\mathbf{1} : \mathbf{B}_k^T]\left[\begin{array}{c} \mathbf{1}^T \\ \hline \mathbf{B}_k \end{array}\right] = \mathbf{1}\mathbf{1}^T + \mathbf{B}_k^T\mathbf{B}_k = 2^k\mathbf{1}\mathbf{1}^T$$

and we deduce from this that $\mathbf{B}_k^T\mathbf{B}_k = (2^k-1)\mathbf{1}\mathbf{1}^T$. (This too can be verified directly.) Thus, $||\mathbf{C}_k\mathbf{f}||_2^2 = \mathbf{f}^T\mathbf{P}_k^T\mathbf{B}_k^T\mathbf{B}_k\mathbf{P}_k\mathbf{f} = \mathbf{f}^T\mathbf{P}_k^T(2^k-1)\mathbf{1}\mathbf{1}^T\mathbf{P}_k\mathbf{f} = (2^k-1)\mathbf{f}^T\mathbf{1}\mathbf{1}^T\mathbf{f}$, since $\mathbf{1}^T\mathbf{P}_k = \mathbf{1}^T$ for any permutation matrix $\mathbf{P}_k$. This completes the proof. Note that the expression for $||\mathbf{C}_k\mathbf{f}||_2^2$ finally simplifies to $(2^k-1)\mathbf{f}^T\mathbf{1}\mathbf{1}^T\mathbf{f} = (2^k-1)(\sum_i f_i)^2 = (2^k-1)$. ■