Lecture 4: Reed-Solomon Codes

Anup Rao

October 6, 2019

Now, we turn to the task of constructing efficient codes. A *linear code* is a code that is a subspace of a vector space over a finite field. Namely, we have $C \subseteq \mathbb{F}^n$, where $\mathbb{F}$ is a finite field, and for all $x, y \in C$, $\alpha, \beta \in \mathbb{F}$, $\alpha x + \alpha y \in C$.

Once we have restricted our attention to linear subspaces we get:

Fact 1. *A subspace $C \subseteq \mathbb{F}^n$ is a linear code of distance d if and only if every non-zero element of C has d non-zero coordinates.*

The number of non-zero coordinates is often called the *weight* of the vector.

Proof. Since C is a subspace, it contains the vector $0 \in \mathbb{F}^n$. If the distance is C, then in particular, for all $x \in C$, $\Delta(x, 0) \geq d$, so x must have at least d non-zero coordinates. Conversely, if every non-zero element of C has d non-zero coordinates, then if $x, y \in C$ are distinct codewords, we have $x - y \in C$ is also a codeword, and it is non-zero, so it must have d non-zero coordinates. This proves that $\Delta(x, y) \geq d$. □

Since C is a subspace, you can express every element as a linear combination of k vectors, the basis for the subspace. So, the encoding function can be viewed as a linear map. Formally, a linear code C can be specified by an $n \times k$ matrix over the finite field, $G \in \mathbb{F}^{n \times k}$, called the *generator* matrix. A message $x \in \mathbb{F}^k$ is interpreted as a column vector, and encoded as $Gx \in C$. The columns of G are just a basis for the linear subspace C.

Given a linear subspace C, described using its generator matrix, it is NP-hard to compute distance of the code.

Having access to the generator matrix is a huge advantage! It means that encoding a message is as simple as multiplying a vector by a matrix. Assuming that multiplication in the field takes unit time, computing Gx takes at most $O(n^2)$ time. Indeed, for many codes, the encoding can be done in almost linear time, as we shall discuss soon.

It can be shown that random linear codes have good distance with high probability.

Decoding from Erasures

What about decoding? There is no known generic decoding algorithm that works for every linear code (this is an NP-hard problem). However, there is a simple algorithm to recover from erasures.

Suppose $d - 1$ symbols of Gx are erased. This corresponds to retaining $G'x = y$ for some matrix G' that corresponds to at least $n - d + 1$ rows of G. Since the code has distance d, this matrix must

be full rank, and so it is enough to find the unique value of x such that $G'x = y$. This can be done by Gaussian elimination in time $O(n^3)$.

Reed-Solomon Codes

REED-SOLOMON CODES ARE the most beautiful of all codes. Given a finite field $\mathbb{F}$ of size q, the messages are viewed as univariate polynomials of degree $k-1$. There are exactly q^k such polynomials. Given such a polynomial $f(X)$, the codeword that corresponds to it is just the vector in $\mathbb{F}^q$ that corresponds to the q evaluations of f on all the elements of $\mathbb{F}$.

This a linear code — taking linear combinations of two polynomials of degree $k-1$ gives you another polynomial of degree $k-1$. The distance of the code is $d = q-k+1$ — any non-zero polynomial of degree $k-1$ can have at most $k-1$ roots, and there are polynomials of degree $k-1$ that have $k-1$ roots. Observe that this means that the code matches the Singleton bound: $d+k = q+1$. It is not possible to have a better code with alphabet size q and codeword length q.

Besides this efficiency, the algebraic structure of the code means that both encoding and decoding can be done extremely efficiently. Because the non-zero elements of the finite field can be written as $\gamma, \gamma^2, \ldots, \gamma^{q-1}$, the generator matrix of the code has a particularly nice structure. It is

$$G = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & \gamma & \gamma^2 & \ldots & \gamma^{k-1} \\ 1 & \gamma^2 & \gamma^4 & \ldots & \gamma^{2(k-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \gamma^{q-1} & \gamma^{2(q-1)} & \ldots & \gamma^{(k-1)(q-1)} \end{bmatrix}.$$

The columns of the generator matrix corresponds to the monomials $1, X, X^2, \ldots, X^{k-1}$. Observe that if

$$f(X) = f_0 + f_1 X + \ldots + f_{k-1} X^{k-1},$$

then

$$G \cdot \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{k-1} \end{bmatrix} = \begin{bmatrix} f(0) \\ f(\gamma) \\ \vdots \\ f(\gamma^{q-1}) \end{bmatrix}.$$

This structure of G allows for extremely fast encoding — there is an algorithm to compute $G \cdot f$ in time $O(q \log k)$ time. The nice thing

Try to prove that the orthgonal complement of the Reed-Solomon code is also a Reed-Solomon code.

This is the fast-fourier-transform algorithm

is that we also have fast algorithms to decode Reed-Solomon codes from errors.

To decode the codeword from erasures is particularly easy. As long as k symbols of the codeword survive, one can reconstruct the polynomial by interpolation. Namely if $f(X)$ is a degree $k-1$ polynomial, and $f(\alpha_1) = \beta_1, f(\alpha_2) = \beta_2, \ldots, f(\alpha_k) = \beta_k$ for k distinct elements $\alpha_1, \ldots, \alpha_k$, then

$$f(X) = \sum_{i=1}^{k} \beta_i \cdot \frac{\prod_{j \neq i} X - \alpha_j}{\prod_{j \neq i} \alpha_i - \alpha_j}.$$

The polynomial given above is of degree k and satisfies the given constraints. There cannot be two polynomials that satisfy the constraints, or this would contradict the distance of the code.

Decoding from errors is a little more tricky. Suppose the original message corresponds to the degree $k-1$ polynomial $f(X)$. Suppose $\alpha_1, \ldots, \alpha_q$ are the elements of the field, and the received word is $(\beta_1, \ldots, \beta_q)$. If there is no error in the ith coordinate, we have $f(\alpha_i) = \beta_i$.

The decoding algorithm we present here is a simplification of an algorithm by Welch and Berlekamp. The simplification is due to Gemmell and Sudan.

A key idea to handle the decoding is the concept of an *error-locator* polynomial, which is useful to design the decoding algorithm. This is a polynomial $e(X)$ whose roots correspond to the inputs α_i where the received word has an error — $e(\alpha_i) = 0$ whenever $\beta_i \neq f(\alpha_i)$. We do not know how to compute $e(X)$ directly, since we do not know where the errors are. Nevertheless, consider the polynomial

$$e(X)(Y - f(X)) = e(X) \cdot Y - e(X) \cdot f(X).$$

This is a low degree polynomial that vanishes on *all* inputs (α_i, β_i), whether there is an error in the ith coordinate or not. We shall try to reconstruct this bivariate polynomial, and then use it to compute $f(X)$.

Suppose the number of errors r is less than $(q-k+1)/2$. Let $h(X, Y)$ be a non-zero bivariate polynomial of the form

$$h(X, Y) = e(X) \cdot Y - g(X),$$

with $e(X)$ of degree at most r and $g(X)$ of degree at most $r+k-1$, such that $h(\alpha_i, \beta_i) = 0$ for all i. Such a polynomial can be efficiently

computed. We are trying to find a solution to the linear system:

$$\begin{bmatrix} \beta_1 & \alpha_1 \cdot \beta_1 & \dots & \alpha_1^r \cdot \beta_1 & 1 & \alpha_1 & \dots & \alpha_1^{r+k-1} \\ \beta_2 & \alpha_2 \cdot \beta_2 & \dots & \alpha_2^r \cdot \beta_2 & 1 & \alpha_2 & \dots & \alpha_2^{r+k-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta_q & \alpha_q \cdot \beta_q & \dots & \alpha_q^r \cdot \beta_q & 1 & \alpha_q & \dots & \alpha_q^{r+k-1} \end{bmatrix} \cdot \begin{bmatrix} e_0 \\ e_1 \\ \vdots \\ e_r \\ g_0 \\ g_1 \\ \vdots \\ g_{r+k-1} \end{bmatrix} = 0,$$

where here

$$h(X, Y) = (e_0 + \ldots + e_r \cdot X) \cdot Y + g_0 + \ldots + g_{r+k-1} \cdot X^{r+k-1}.$$

This is a homogenous linear system that has at least one non-zero solution: we could set $e(X)$ to be the correct error-locator polynomial and $h(X, Y) = e(X)Y - e(X)f(X)$ to give a non-zero solution to the above system. So, you can find the coefficients of a non-zero solution for h using Gaussian elimination. Given $h(X, Y)$, we can compute $e(X)$ and $g(X)$. Note that there is no reason to believe that $e(X)$ actually is the error-locator polynomial. Nevertheless, we prove that $g(X)/e(X) = f(X)$:

Here we are using the fact that the set of solutions to $Mx = 0$ is always a subspace, and Gaussian elimination can be used to find a basis for this space. Actually, it is possible to compute the solution in near linear time using polynomial interpolation.

Claim 2. *If $h(X)$ is computed as above, we must have $f(X) \cdot e(X) = g(X)$.*

Proof. Consider the polynomial $u(X) = f(X) \cdot e(X) - g(X)$. Whenever α_i is such that the codeword does not have an error in location i, we have

$$\begin{aligned} u(\alpha_i) &= e(\alpha_i)f(\alpha_i) - g(\alpha_i) \\ &= e(\alpha_i)\beta_i - g(\alpha_i) \\ &= h(\alpha_i, \beta_i) \\ &= 0. \end{aligned}$$

So, $u(X)$ is a polynomial of degree $r + k - 1$ that has $q - r$ roots. Since $q - r > r + k - 1$, $u(X) = 0$. □

Is it possible that the $e(X)$ found by the algorithm is *not* the correct error-locator polynomial?

To review the whole decoding algorithm: first compute $h(X, Y)$ that vanishes on (α_i, β_i) for all i, and then compute $f(X) = g(X)/e(X)$. This gives a fast algorithm for decoding from up to $(q - k + 1)/2$ errors, and it is impossible to uniquely recover the message from more errors.

Code concatenation

THE MAIN DRAWBACK of the Reed-Solomon code is that it uses a large alphabet. What if we really want a code that uses the binary alphabet?

The most naive solution is to simply encode the alphabet symbols in binary. When $q = 2^r$, recall that $\mathbb{F}_q$ consists of polynomials of degree $r-1$ with coefficients from $\mathbb{F}_2$. So, just write down each element using bits that represent its coefficients. This gives a linear code $C \subseteq \{0,1\}^{q \log q}$ of dimension k and distance $q-k+1$. The nice thing is that linearity is preserved by this encoding. However, the rate and relative distance of the code drop by a factor of $\log q$.

A more sophisticated idea is to use another code to encode each alphabet symbol. This idea works, but in the interest of time we leave it to the exercises to explore the details.

That said, in many applications, like CD-ROM's, errors are localized in space. For that situation the large alphabet does not hurt so much, because when part of an alphabet symbol is corrupted, it is quite likely that most of the alphabet symbol will be corrupted anyway.