Class 5 Exercises

CS250/EE387, Winter 2025

In this class, we'll investigate/develop the (starting point[1] for the) *Berlekamp-Massey Algorithm* for decoding Reed-Solomon codes. Some notation:

- We will be working with a RS code $C \subseteq \mathbb{F}_q^n$ with length $n = q-1$ over $\mathbb{F}_q$ and with evaluation points $1, \gamma, \gamma^2, \ldots, \gamma^{n-1}$.
- We will try to decode C from e errors. Suppose that $v \in \mathbb{F}_q^n$ is the received word, so $v = c + p$ for $c \in C$ and $p \in \mathbb{F}_q^n$ is an error vector so that $\mathrm{wt}(p) \leq e$.
- Let $p = (p_0, \ldots, p_{n-1})$ be the error vector, and let $E = \{i : p_i \neq 0\}$. be the locations of the errors.
- Let $d = n - k + 1$ be the distance of the RS code, and assume that $e \leq \left\lfloor \frac{d-1}{2} \right\rfloor$, so that unique decoding is possible.

Now onto the questions.

1. The first step of the Berlekamp-Massey algorithm is to compute the *syndrome* $s = Hv$, where H is the parity-check matrix for C. Write $s = (s_1, \ldots, s_{d-1})$, and let $s(Z) = \sum_{i=1}^{d-1} s_i Z^i$.

   Show that
$$s(Z) = \sum_{i \in E} p_i \cdot \left(\frac{\gamma^i Z - (\gamma^i Z)^d}{1 - \gamma^i Z} \right).$$

   Hint/Outline:

   - *First, show that $s(Z) = \sum_{i \in E} p_i \sum_{\ell=1}^{d-1} (\gamma^i Z)^\ell$. (To do that, write out what $s(Z)$ is, in terms of the γ^i's and p_j's, by using the definition of the parity-check matrix and the fact that $Hv = s$).*
   - *Then, remember that $\sum_{j=1}^{n} x^j = \frac{x^{n+1} - x}{x - 1}$...*

   Solution

   First, $Hv = H(c+p) = Hp$. Next, we know that $H_{j,i} = \gamma^{ij}$ where j is 1-indexed and i is (obnoxiously) zero-indexed. Thus,
$$\begin{aligned} s_j &= j\text{'th row of } Hp \\ &= \sum_{i=0}^{n-1} \gamma^{ij} p_i \end{aligned}$$

[1] In fact, what's usually called the "Berlekamp-Massey Algorithm" picks up where these questions leave off. Today we are going to design an algorithm that runs decently fast. The BM algorithm starts with this outline and makes it much faster.

and so

$$\begin{aligned} s(Z) &= \sum_{j=1}^{d-1} s_j Z^j \\ &= \sum_{j=1}^{d-1}\sum_{i=0}^{n-1} p_j \gamma^{ij} Z^j \\ &= \sum_{i=0}^{n-1} p_i \sum_{j=1}^{d-1} (\gamma^i Z)^j \\ &= \sum_{i\in E} p_i \left(\frac{\gamma^i Z - (\gamma^i Z)^d}{1-\gamma^i Z} \right) \end{aligned}$$

as desired.

2. Let $\sigma(Z) = \prod_{i\in E}(1-\gamma^i Z)$. Explain why we will be in good shape for decoding if we can figure out what σ is.

Solution

We have $i \in E \Leftrightarrow \sigma(\gamma^{-i}) = 0$. So if we find σ, we can factor it and find E. Once we know E, we can, for example, treat the errors as erasures and just solve a linear system to find the original codeword.

3. Consider $\sigma(Z)\cdot s(Z)$. Show that this can be written as

$$\sigma(Z)\cdot s(Z) = w(Z) + Z^d r(Z),$$

where $w(Z)$ and $r(Z)$ are polynomials, and $\deg(w) \le e$. Write down an expression for $w(Z)$.

Solution

We can use our expression earlier for $s(Z)$ to write

$$s(Z)\sigma(Z) = \sum_{i\in E} p_i(\gamma^i Z - (\gamma^i Z)^d) \prod_{j\in E\setminus\{i\}} (1-\gamma^j Z).$$

(Essentially, we are using $\sigma(Z)$ to clear the denominator in our expression for $s(Z)$). By staring, this polynomial has the desired form, and

$$w(Z) = \sum_{i\in E} p_i \gamma^i Z \prod_{j\in E\setminus\{i\}} (1-\gamma^j Z).$$

4. Let's explore this polynomial $w(Z)$ a bit more.

   (a) For all $r \in E$, $w(\gamma^{-r}) = p_r \cdot \prod_{j\in E\setminus\{r\}}(1-\gamma^{j-r})$. Use this to explain why we will be done (that is, we can figure out the error vector p) if we can figure out *both* $\sigma(Z)$ and $w(Z)$.

   (b) **(Optional:)** $w(Z)$ and $\sigma(Z)$ are relatively prime. That is, there is no polynomial $g(Z)$ (other than $g(Z) \equiv 1$) that divides them both.

   Note: if you don't feel like showing this, take it as given and skip this part; you can come back and think about it later if you have time!

Solution

(a) Plugging in γ^{-r}, we have

$$w(\gamma^{-r}) = \sum_{i \in E} p_i \gamma^{i-r} \prod_{j \in E \setminus \{i\}} (1 - \gamma^{j-r}).$$

The product vanishes unless $i = r$, so we are left with only the $i = r$ term, which is

$$w(\gamma^{-r}) = p_r \gamma^{r-r} \left(\prod_{j \in E \setminus \{i\}} (1 - \gamma^{j-r}) \right) = p_r \prod_{j \in E \setminus \{r\}} (1 - \gamma^{j-r})$$

as desired.
This means that if we can figure out *both* $\sigma(Z)$ and $w(Z)$, we'll be done: $\sigma(Z)$ can tell us where the errors occur, as in Question 2. Then, if we know an error occurs at location r, we can plug γ^{-r} into w to get $p_r \cdot$ (stuff we can compute), from which we can recover p_r. Then we'd know all of p, which is what we wanted.

(b) Since $\sigma(Z)$ factors completely, it suffices to show that $(1 - \gamma^r Z)$ does not divide $w(Z)$ for any $r \in E$. But we just saw that for all $r \in E$, $w(\gamma^{-r}) = p_r \neq 0$, so γ^{-r} is not a root of w, so $(1 - \gamma^r Z)$ cannot divide it.

5. The previous part implies that, for all r with $e + 1 \leq r \leq d - 1$, we have

$$\text{coefficient on } Z^r \text{ in } s(Z)\sigma(Z) = 0.$$

What is that coefficient, in terms of the coefficients of s (which we know) and the coefficients of σ (which we don't know)? Write down a system of $d - e - 1$ linear constraints that the coefficients of σ must satisfy. Your constraints should be in terms of the s_i. Explain why there is at least one solution to this system of equations.

Solution

For each r, the coefficient on Z^r in $s(Z)\sigma(Z)$ is given by the convolution

$$\text{coefficient on } Z^r = \sum_{i=0}^{e} \sigma_i s_{r-i},$$

where $\sigma(Z) = \sum_{i=0}^{e} \sigma_i Z^i$. This gives us a system of equations

$$\begin{pmatrix} S_{e+1} & S_e & S_{e-1} & \cdots & S_1 \\ S_{e+2} & S_{e+1} & S_e & \cdots & S_2 \\ S_{e+3} & \ddots & \ddots & & \\ \vdots & & & & \\ S_{d-1} & S_{d-2} & S_{d-2} & \cdots & S_{d-e-1} \end{pmatrix} \cdot \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \vdots \\ \sigma_{n-1} \end{pmatrix} = \vec{0}.$$

There is at least one solution to these equations, because σ is a solution!

6. Suppose we were to solve your system of equations to obtain $(\tilde{\sigma}_0, \ldots, \tilde{\sigma}_{n-1})$ and a corresponding polynomial $\tilde{\sigma}(Z) = \sum_{i=0}^{e} \tilde{\sigma}_i Z^i$. Explain why we can write

$$s(Z)\tilde{\sigma}(Z) = \tilde{w}(Z) + Z^d \tilde{r}(Z)$$

for some polynomials $\tilde{w}(Z), \tilde{r}(Z)$ with $\deg(\tilde{w}) \leq e$.

Hint: Don't overthink it. (Think about what we did to answer the previous question...)

Solution

The system of linear equations that we solved to find $\tilde{\sigma}$ precisely says that all of the coefficients of $s(Z)\tilde{\sigma}(Z)$ between Z^{e+1} and Z^{d-1} (inclusive) are zero. Therefore, $s(Z)\tilde{\sigma}(Z)$ has the desired form. In more detail, you can imagine extending the system of equations above to

$$\begin{pmatrix} S_0 & 0\cdots & & & \\ S_1 & S_0 & 0\cdots & & \\ \vdots & & & & \\ S_e & S_{e-1} & \cdots & & \\ --- & & & & \\ S_{e+1} & S_e & S_{e-1} & \cdots & S_1 \\ S_{e+2} & S_{e+1} & S_e & \cdots & S_2 \\ S_{e+3} & \ddots & \ddots & & \\ \vdots & & & & \\ S_{d-1} & S_{d-2} & S_{d-3} & \cdots & S_{d-e-1} \\ --- & & & & \\ S_d & S_{d-1} & S_{d-2} & \cdots S_{d-e} & \\ S_{d+1} & S_d & \cdots & & \\ \vdots & & & & \end{pmatrix} \cdot \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \vdots \\ \sigma_{n-1} \end{pmatrix} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_e \\ -- \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -- \\ r_0 \\ r_1 \\ \vdots \end{pmatrix}.$$

If you replace σ with $\tilde{\sigma}$, you'll get something of the same form, and that will tell you the coefficients for $\tilde{w}(Z)$ and $\tilde{r}(Z)$.

7. Show that $\tilde{\sigma}(Z)w(Z) = \sigma(Z)\tilde{w}(Z)$.

   Hint: Consider $\sigma(Z)s(Z)\tilde{\sigma}(Z)$. Further hint: $(\sigma(Z)s(Z))\tilde{\sigma}(Z) = \sigma(Z)(s(Z)\tilde{\sigma}(Z))$.

Solution

Following the hint, we have

$$s(Z)\sigma(Z)\tilde{\sigma}(Z) = (s(Z)\sigma(Z))\tilde{\sigma}(Z) = (w(Z) + Z^d r(Z))\tilde{\sigma}(Z) = w(Z)\tilde{\sigma}(Z) + Z^d \cdot [\text{stuff}].$$

Symmetrically,

$$s(Z)\sigma(Z)\tilde{\sigma}(Z) = \tilde{w}(Z)\sigma(Z) + Z^d \cdot [\text{stuff}]'.$$

Thus,

$$\tilde{w}(Z)\sigma(Z) = w(Z)\tilde{\sigma}(Z) + Z^d \cdot [\text{stuff}]'',$$

for some (polynomial) value of $[\text{stuff}]''$. Since $\tilde{w}, w, \tilde{\sigma}, \sigma$ all have degree at most e, $\sigma(Z)\tilde{w}(Z)$ and $\tilde{\sigma}(Z)w(Z)$ both have degree at most $2e$, which is strictly less than d by our assumption on the number of errors. Thus, the $Z^d \cdot [\text{stuff}]''$ term doesn't collide with the lower-order terms, and we conclude that

$$\tilde{w}(Z)\sigma(Z) = w(Z)\tilde{\sigma}(Z)$$

as desired.

8. Explain how, given $\tilde{\sigma}$ and $\tilde{w}$, to find σ and w.

   Hint: By Question 7, $\frac{\tilde{w}(Z)}{\tilde{\sigma}(Z)} = \frac{w(Z)}{\sigma(Z)}$.... Part 4(b) might be useful. Further hint: Suppose that I was thinking of two integers that are relatively prime (like 2 and 5). Suppose I told you that the one divided by the other was the same as 40/100. How could you recover the two numbers?

Solution

The previous part implies that

$$\frac{\tilde{w}(Z)}{\tilde{\sigma}(Z)} = \frac{w(Z)}{\sigma(Z)}.$$

Since $w(Z)$ and $\sigma(Z)$ are relatively prime, they have no common factors. Thus, if we reduce the fraction $\frac{\tilde{w}(Z)}{\tilde{\sigma}(Z)}$ by dividing out any common factors, what we are left with must be the rational function $\frac{w(Z)}{\sigma(Z)}$, and we know that the numerator must be w and the denominator must be σ.

9. Put all the pieces together to write down an efficient (polynomial-time) algorithm to recover the codeword c given v. What is the running time of your algorithm, in terms of the number of operations over $\mathbb{F}_q$?

   If it helps, finding the gcd (or reducing a fraction) of two degree-D polynomials (with, say, Euclid's algorithm) takes $O(D^2)$ operations over $\mathbb{F}_q$. Finding the roots of a degree-D polynomial in $\mathbb{F}_q$ can be done with $O(D^2 \log q)$ operations over $\mathbb{F}_q$. Evaluating a degree D polynomial at a point can be done with about $O(D \log D)$ operations over $\mathbb{F}_q$.

Solution

Here's an algorithm:

- Find the syndrome $s = Hv$.
- Solve the system of linear equations that we came up with above (which only depends on s, which we just found) to find $\tilde{\sigma}(Z)$.
- Let $\tilde{w}(Z) = s(Z)\tilde{\sigma}(Z) \mod Z^d$.
- Compute $g(Z) = gcd(\tilde{w}(Z), \tilde{\sigma}(Z))$.
- Let $\sigma(Z) = \tilde{\sigma}(Z)/g(Z)$ and let $w(Z) = \tilde{w}(Z)/g(Z)$.
- Find the roots of $\sigma(Z) = \prod_{i \in E}(1 - \gamma^i Z)$ to find E.
- For $i \in E$, let

  $$p_i = \frac{w(\gamma^{-i})}{\prod_{j \in E \setminus \{i\}}(1 - \gamma^{j-i})}.$$

  For $i \notin E$, let $p_i = 0$.
- Return $c = v - p$.

The running time, calculated as number of operations over $\mathbb{F}_q$, is (naively):

- $O(nd)$ operations to compute the syndrome.
- $O(d^3)$ operations to solve our $\approx d \times d$ linear system.
- $O(ed) = O(d^2)$ to multiply $s(Z)$ with $\tilde{\sigma}(Z)$ and chop off the end to get $\tilde{w}(Z)$.
- $O(d^2)$ operations to take the GCD and reduce the fraction $\tilde{w}(Z)/\tilde{\sigma}(Z)$.
- $O(d^2 \log q)$ to find the roots of $\sigma(Z)$.
- $O(d^2 \log d)$ operations to evaluate $w(Z)$ on $e = O(d)$ points. (This is $O(d \log d)$ per point, since there are $O(d)$ terms to add up, and each term takes $O(\log d)$ operations to compute by repeated squaring).
- $O(n)$ to finally compute c and return it.

So the total is something like $O(nd + d^3 + d^2 \log n) = n\text{poly}(d)$ operations over $\mathbb{F}_q$. (FWIW, each operation over $\mathbb{F}_q$ can be performed in something like $O(\log^2(n))$ time). If we don't need to compute the syndrome, and $d \ll n$ is small enough, this can actually be sublinear in n!

10. **(Bonus)** Can you think of a way to speed up your algorithm? What steps seem especially lossy?

Solution

There are several ways to speed this up. Most notably, the linear system that we are solving is very structured – the matrix is actually a Toeplitz matrix, and there are fast algorithms for solving such systems. This can reduce the running time to something like $O(nd + d\log^2(d)\log\log(d) + d^2\log n)$ operations over $\mathbb{F}_q$.

In fact, the "Berlekamp-Massey" algorithm usually refers to a particular very fast way to solve this system of linear equations.

11. **(Bonus)** A *linear feedback shift register* (LFSR) is defined as follows. We have a *register*, which initially holds t field symbols $s_0, s_1, \ldots, s_{t-1} \in \mathbb{F}_q$. Fix some vector $\vec{\tau} \in \mathbb{F}_q^t$. We are going to modify this register over time, as follows. At time j, suppose that the register holds $(s_j, s_{j+1}, \ldots, s_{j+t-1})$. To move to time $j+1$, we do the following:

    - Write the contents of the register as $\vec{s} = (s_j, \ldots, s_{j+t-1})$, and compute $s_{j+t} = \vec{s} \cdot \vec{\tau}$.
    - Shift all the contents of the register over by one, dropping the first one, so now it looks like $(s_{j+1}, \ldots, s_{j+t-1}, \text{----})$.
    - Fill in the last black with s_{j+t}, so now we have $(s_{j+1}, \ldots, s_{j+t})$.

    Then we repeat for a while! A question that one might ask about such a thing is: *Given a sequence of "dropped" bits $s_0, s_1, s_2, \ldots, s_N$, can you recover the coefficient vector $\vec{\tau}$?*

    If you look up the *Berlekamp-Massey algorithm* on the internet, it is often described as a very fast way to solve this LFSR problem. But today we said that it was a fast algorithm for decoding RS codes. What does the LFSR problem have to do with decoding Reed-Solomon codes??

Solution

Solving the system of equations we wrote down with s_i and σ_j can be seen as finding the coefficients of a LFSR. To see this, notice that the first equation says that $\sum_{i=0}^{e} \sigma_i s_{e+1-i} = 0$, or

$$s_{e+1} = \frac{-\sigma_1}{\sigma_0} s_e + \frac{-\sigma_2}{\sigma_0} s_{e-1} + \cdots + \frac{-\sigma_e}{\sigma_0} s_1.$$

The second equation can be written as

$$s_{e+2} = \frac{-\sigma_1}{\sigma_0} s_{e+1} + \frac{-\sigma_2}{\sigma_0} s_e + \cdots + \frac{-\sigma_e}{\sigma_0} s_2,$$

and so on. So, this is exactly the same problem as the LFSR problem, where the coefficients are $\tau_i = \frac{-\sigma_i}{\sigma_0}$. (In fact, in our setting, we have $\sigma_0 = 1$ (why?), so actually $\tau_i = -\sigma_i$).