Sumcheck

Notes by Linus Tang.

These notes have not been thoroughly reviewed. Any errors below are my own responsibility.

Sources:

- MIT 6.5610 lecture notes by Yael Kalai
  - https://65610.csail.mit.edu/2025/lec/l17-sumcheck.pdf

Sumcheck problem

Consider a finite field $\mathbb{F}_p$ where p is a large prime. Given a polynomial $f : \mathbb{F}_p^m \to \mathbb{F}_p$ of small degree $\leq d$ in each variable and a small fixed set $H \subseteq \mathbb{F}_p$, compute

$$\beta = \sum_{h_1,\dots,h_m \in H} f(h_1, \dots, h_m).$$

In particular, we are interested in an interactive proof for the sumcheck problem:

A verifier has oracle access to f. Can a prover with unlimited computation efficiently convince a verifier of the value of β?

The simple protocol of having the verifier query for every element of the sum has complexity exponential in m. Using interactive proofs, we can do much better.

Although the problem may seem rather specific, it is an important building block in the GKR protocol and in a proof of $\mathsf{IP} = \mathsf{PSAPCE}$.

Sumcheck protocol

A word on notation: We use default typesetting (e.g. g and β) to denote the correct values that should be output by an honest prover and sans letters (e.g. g and $\boldsymbol{\beta}$) to denote what the prover actually outputs.

We describe the protocol in terms of an honest prover below, so $\mathsf{g} = g$ and $\boldsymbol{\beta} = \beta$, but we choose to use default typesetting for prover operations and sans letters when the verifier is performing computations over untrusted values sent by the prover. This notational distinction is mainly meant to improve the clarity of the soundness analysis later.

The protocol is as follows:

- The prover evaluates the sum of interest and sends the result β to the verifier.
- The prover computes $g_1(x) = \sum_{h_2,\dots,h_m \in H} f(x, h_2, \dots, h_m)$, which is a polynomial of degree $\leq d$, and sends its coefficients.
- The verifier uses the coefficients to check that $\sum_{x \in H} \mathsf{g}_1(x) = \boldsymbol{\beta}$.
- For $i = 2, \dots, m$:
  - The verifier sends uniformly sampled t_{i-1} from $\mathbb{F}_p$.
  - The prover computes $g_i(x) = \sum_{h_{i+1},\dots,h_m \in H} f(t_1, \dots, t_{i-1}, x, h_{i+1}, \dots, h_m)$, and again sends its coefficients as a degree $\leq d$ polynomial in x.
  - The verifier checks that $\sum_{h_i \in H} \mathsf{g}_i(h_i) = \mathsf{g}_{i-1}(t_{i-1})$.
- Finally, the verifier uniformly samples t_m from $\mathbb{F}_p$ and checks that $\mathsf{g}_m(t_m) = f(t_1, \dots, t_m)$ using its oracle access to f. The verifier accepts if and only if all checks have passed.

Soundness analysis

The protocol already describes how an honest prover can pass all checks, so completeness of the protocol follows. We focus on proving soundness.

An important fact is that two distinct polynomials $h_1 \neq h_2$ over $\mathbb{F}_p$ of degree at most d coincide on at most d inputs. Specifically, there exist at most d values $x \in \mathbb{F}_p$ such that $h_1(x) = h_2(x)$, and this follows from the fact that $h_1 - h_2$ is a nonzero polynomial over $\mathbb{F}_p$ of degree at most d.

Consider a dishonest prover who lied about β, i.e. $\boldsymbol{\beta} \neq \beta$.

Since the verifier checks that $\sum_{x \in H} \boldsymbol{g}_1(x) = \boldsymbol{\beta}$, the verifier must lie about g_1 as well, so $\boldsymbol{g}_1 \neq g_1$.

At the end, the verifier checks that $\boldsymbol{g}_m(t_m) = f(t_1, ..., t_m)$ for some uniformly random $t_m \in \mathbb{F}_p$. If $\boldsymbol{g}_m \neq g_m$, then the probability that $\boldsymbol{g}_m(t_m) = f(t_1, ..., t_m) = g_m(t_m)$ is at most $\frac{d}{p}$ because $\boldsymbol{g}_m$ and g_m coincide on at most d inputs. In this case, the prover succeeds with probability $\leq \frac{d}{p}$.

Since the prover must give a wrong value $\boldsymbol{g}_1 \neq g_1$ and the correct value of $\boldsymbol{g}_m = g_m$, the prover must find an opportunity to switch from giving wrong values $\boldsymbol{g}_{i-1} \neq g_{i-1}$ to giving correct values $\boldsymbol{g}_i = g_i$. These opportunities are difficult to come by, as shown here:

The verifier will check that $\sum_{h_i \in H} \boldsymbol{g}_i(h_i) = \boldsymbol{g}_{i-1}(t_{i-1})$.

So in order to give the correct value of g_i while still passing the check, the prover will need to have

$$g_{i-1}(t_{i-1}) = \sum_{h_i \in H} g_i(h_i) = \sum_{h_i \in H} \boldsymbol{g}_i(h_i) = \boldsymbol{g}_{i-1}(t_{i-1}).$$

This is possible only if the verifier's randomly sampled t_{i-1} is one of at most d inputs on which g_{i-1} and $\boldsymbol{g}_{i-1}$ coincide, which happens with probability at most d/p.

To summarize, we've shown that

- A dishonest prover who passes the first check must lie about g_1.
- In each round $2 \leq i \leq m$, a prover who has lied about g_{i-1} must lie about g_i in order to pass a check, except in an event with probability at most d/p.
- A prover who lies about g_m passes the final check with probability at most d/p.

Thus, a dishonest prover gets caught with probability at least $\left(1 - \frac{d}{p}\right)^m \geq 1 - \frac{dm}{p}$.

If dm is not sufficiently small compared to p to achieve the desired soundness/security level, then we run the protocol several times. In the analysis below, we assume that $dm \ll p$ and the protocol is only run once.

Efficiency analysis

We leave to the reader to verify that:

- The communication complexity in bits is $O(dm \log p)$.
- The verifier runtime is $O(m \cdot |H| \cdot d \cdot \text{polylog } p)$.
- The prover runtime is $O\big(m \cdot |H|^m \cdot T_f\big)$, where T_f denotes the time to compute f.