zkCNN

2023.8

zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy

Tianyi Liu
Texas A&M University and Shanghai Key Laboratory of Privacy-Preserving Computation
tianyi@tamu.edu

Xiang Xie
Shanghai Key Laboratory of Privacy-Preserving Computation
xiexiang@matrixelements.com

Yupeng Zhang
Texas A&M University
zhangyp@tamu.edu

Outline

- **Sumcheck for FFT**
  - 2-D conv. using FFT
  - Preliminary about FFT
  - Achieve linear prover
    - Generate sumcheck messages in linear time (with initialization)
    - [This] Initialize $\tilde{F}(u, x)$ for all $x \in \{0,1\}^{\log N}$ in linear time ($u \in \mathbb{F}^{\log M}$ is fixed)
  - Achieve logarithmic verifier
    - [This] Delegate the the computation for $\tilde{F}(u, v)$ to prover
- **Generalization of GKR for CNN**
  - Generalized addition and multiplication gates with fan-in ≥ 2
  - Taking inputs from either the layer above or from the input gate.
  - Reduce the number of IFTTs in the entire convolutional layer

2-D Convolution using FFT

2-D Conv → 1-D Conv

result of a 2-D convolution between two matrices X and W of size $n \times n$ and $w \times w$ as a $(n - w + 1) \times (n - w + 1)$ matrix $U = X * W$ such that

$$U_{j,k} = \sum_{t=0,l=0}^{w-1,w-1} X_{j+t,k+l} \cdot W_{t,l} \quad \text{2-D conv} \quad (1)$$

for $j, k = 0, \ldots, n - w$. [2] In convolutional neural networks, the

tion 2.1, let $\bar{X}, \bar{W} \in \mathbb{F}^{n^2}$ be

$$\bar{X}_{tn+l} = X_{n-1-t,n-1-l}, \quad 0 \le t < n, 0 \le l < n$$

$$\bar{W}_{tn+l} = \begin{cases} W_{t,l}, & 0 \le t, l < w \\ 0, & \text{otherwise} \end{cases}$$

$$\bar{U}_j = \sum_{i=0}^{j} \bar{X}_{j-i} \bar{W}_i \quad \text{1-D conv}$$

$$\begin{aligned} U_{j,k} &= \sum_{t=0,l=0}^{(w-1),(w-1)} X_{j+t,k+l} W_{t,l} \\ &= \sum_{t=0,l=0}^{(w-1),(w-1)} \bar{X}_{(n-1-j-t)\cdot n+(n-1-k-l)} \bar{W}_{t\cdot n+l} \\ &= \sum_{t=0,l=0}^{(n-1),(n-1)} \bar{X}_{(n-1-j-t)\cdot n+(n-1-k-l)} \bar{W}_{t\cdot n+l} \\ &= \sum_{i=0}^{n^2-1-j\cdot n-k} \bar{X}_{n^2-1-j\cdot n-k-i} \bar{W}_i \\ &= \bar{U}_{n^2-1-j\cdot n-k} \end{aligned} \quad (11)$$

Thus U can be computed through 1-D convolution between $\bar{X}, \bar{W}$, vectors defined by the input and the kernel of a convolutional layer.

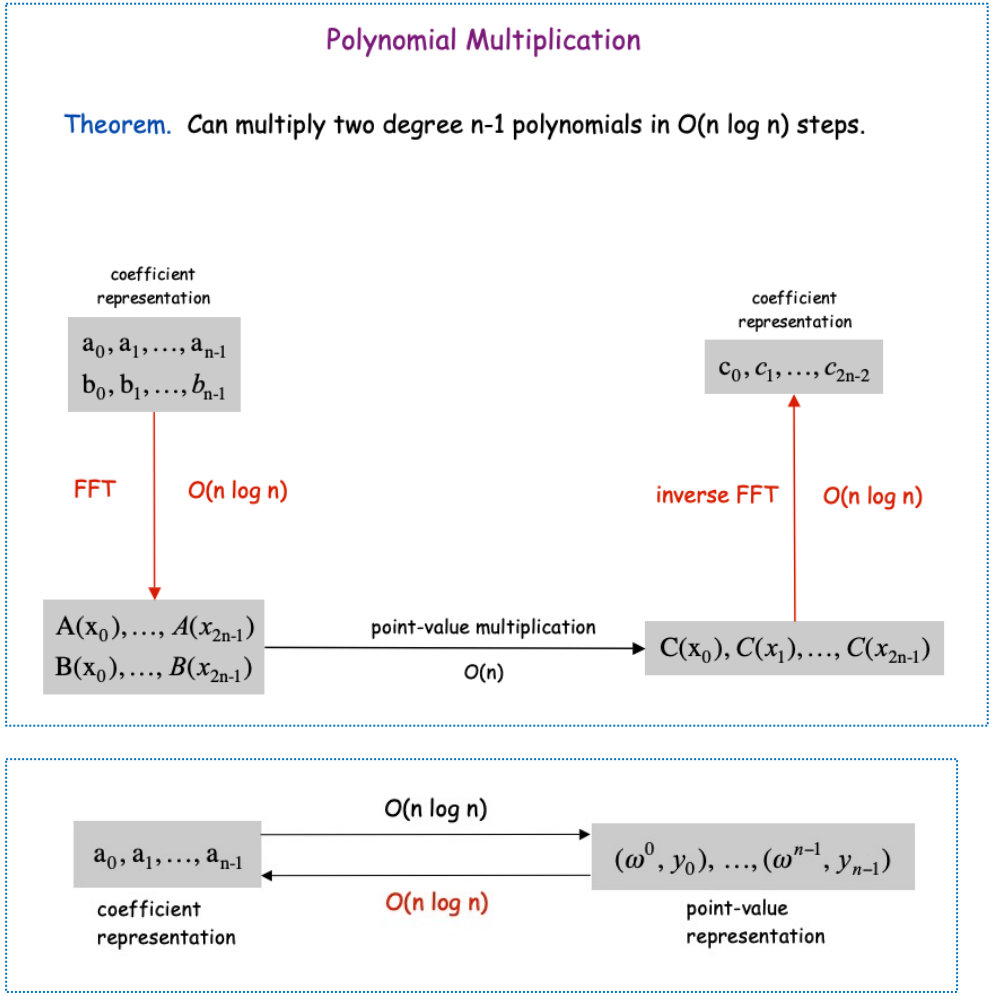
2-D Convolution using FFT

1-D Conv → poly multiplication (3 steps using FFT)

variate polynomials with coefficients $\bar{X}, \bar{W}$ as $\bar{X}(\eta), \bar{W}(\eta)$, then $\bar{U}(\eta) = \bar{X}(\eta)\bar{W}(\eta) \Leftrightarrow \bar{U}_j = \sum_{i=0}^{j} \bar{X}_{j-i}\bar{W}_i$ by taking $\bar{U}$ as the first n^2 coefficients of $\bar{U}(\eta)$.

$$\bar{U} = \bar{X} * \bar{W} = \texttt{IFFT}(\texttt{FFT}(\bar{X}) \odot \texttt{FFT}(\bar{W})) \quad (12)$$

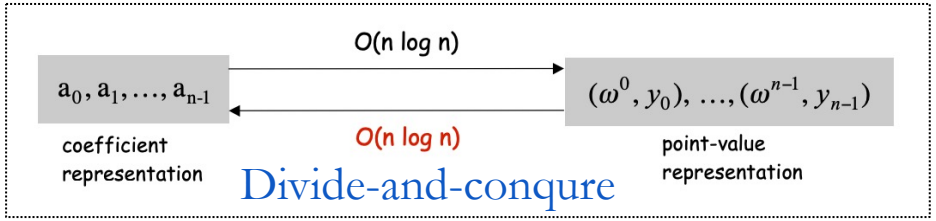
1. FFT: $2n^2$ points on polyX and polyW
2. element-wise product: $2n^2$ points on polyU (point-value rep. of polyU)
3. IFFT: coeff. rep. of polyU (U = first n^2 coeff. of polyU)



https://www.cs.princeton.edu/courses/archive/spr05/cos423/lectures/05fft.pdf

Sumcheck for FFT

FFT & IFFT



FFT

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Discrete Fourier transform — Fourier matrix F_n

IFFT

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Inverse DFT — Fourier matrix inverse $(F_n)^{-1}$

Key idea: choose $x_k = \omega^k$ where ω is principal n^{th} root of unity.

Def. An n^{th} root of unity is a complex number x such that $x^n = 1$.

Fact. The n^{th} roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i / n}$.
Pf. $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$.

$$G_n = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

Consequence. To compute inverse FFT, apply same algorithm but use $\omega^{-1} = e^{-2\pi i / n}$ as principal n^{th} root of unity (and divide by n).

Sumcheck for FFT

Notations

tions at powers of the root of unity. Formally speaking, let $\mathbf{c} = (c_0, c_1, \ldots, c_{N-1})$ be the vector of coefficients of a polynomial, $\mathbf{a} = (a_0, a_1, \ldots, a_{M-1})$ be the vector of evaluations at $(\omega^0, \omega^1, \ldots, \omega^{M-1})$,

By the definition of polynomial evaluations, $a_j = \sum_{i=0}^{N-1} c_i \omega^{ji}$ for $j = 0, 1, \ldots, M-1$, which can also be written as a matrix-vector multiplication $\mathbf{a} = F \cdot \mathbf{c}$, where F is the standard Fourier matrix:

$$F = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega^1 & \omega^2 & \ldots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{M-1} & \omega^{2(M-1)} & \ldots & \omega^{(M-1)(N-1)} \end{pmatrix} \quad (4)$$

$$a = F \cdot c$$

Multilinear Extension

$$\tilde{a}(y) = \sum_{x \in \{0,1\}^{\log N}} \tilde{c}(x) \tilde{F}(y, x), \quad (5)$$

for $y \in \{0,1\}^{\log M}$. Here $\tilde{a}(\cdot)$ and $\tilde{c}(\cdot)$ are multilinear extensions of $\mathbf{a}$ and $\mathbf{c}$, and $\tilde{F}(\cdot,\cdot)$ is the multilinear extension defined by the Fourier matrix F such that $\tilde{F}(y, x)$ is the (y, x)-th entry in F. As x, y are

Definition 2.2 (Identity function). Let $\beta : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be the identity function such that $\beta(x, y) = 1$ if $x = y$, and $\beta(x, y) = 0$ otherwise. Suppose $\tilde{\beta}$ is the multilinear extension of β. Then $\tilde{\beta}$ can be expressed as: $\tilde{\beta}(x, y) = \prod_{i=1}^{\ell}((1 - x_i)(1 - y_i) + x_i y_i)$.

Definition 2.3 (Multilinear Extension [21]). Let $V : \{0,1\}^\ell \to \mathbb{F}$ be a function. The *multilinear extension* of V is the unique polynomial $\tilde{V} : \mathbb{F}^\ell \to \mathbb{F}$ such that $\tilde{V}(x_1, x_2, ..., x_\ell) = V(x_1, x_2, ..., x_\ell)$ for all $x_1, x_2, \ldots, x_\ell \in \{0,1\}$. $\tilde{V}$ can be expressed as:

$$\tilde{V}(x_1, x_2, ..., x_\ell) = \sum_{b \in \{0,1\}^\ell} \tilde{\beta}(x, b) \cdot V(b)$$

$$= \sum_{b \in \{0,1\}^\ell} \prod_{i=1}^{\ell}((1 - x_i)(1 - b_i) + x_i b_i)) \cdot V(b),$$

where b_i is i-th bit of b.

Sumcheck for FFT

Apply sumcheck to $(logN)$-variate poly $g(x) = \tilde{c}(x)\tilde{F}(u,x)$

Claim: $\tilde{a}(u) = \sum_{x\in\{0,1\}^{logN}} \tilde{c}(x)\tilde{F}(u,x)$ for a random fixed point $u \in \mathbb{F}^{\log M}$

Description of Sum-Check Protocol.

- At the start of the protocol, the prover sends a value C_1 claimed to equal the value H defined in Equation (4.1).
- In the first round, $\mathcal{P}$ sends the univariate polynomial $g_1(X_1)$ claimed to equal

$$\sum_{(x_2,\ldots,x_v)\in\{0,1\}^{v-1}} g(X_1,x_2,\ldots,x_v).$$

  $\mathcal{V}$ checks that

$$C_1 = g_1(0)+g_1(1),$$

  and that g_1 is a univariate polynomial of degree at most $\deg_1(g)$, rejecting if not. Here, $\deg_j(g)$ denotes the degree of $g(X_1,\ldots,X_v)$ in variable X_j.
- $\mathcal{V}$ chooses a random element $r_1 \in \mathbb{F}$, and sends r_1 to $\mathcal{P}$.
- In the jth round, for $1 < j < v$, $\mathcal{P}$ sends to $\mathcal{V}$ a univariate polynomial $g_j(X_j)$ claimed to equal

$$\sum_{(x_{j+1},\ldots,x_v)\in\{0,1\}^{v-j}} g(r_1,\ldots,r_{j-1},X_j,x_{j+1},\ldots,x_v).$$

  $\mathcal{V}$ checks that g_j is a univariate polynomial of degree at most $\deg_j(g)$, and that $g_{j-1}(r_{j-1}) = g_j(0)+g_j(1)$, rejecting if not.
- $\mathcal{V}$ chooses a random element $r_j \in \mathbb{F}$, and sends r_j to $\mathcal{P}$.
- In Round v, $\mathcal{P}$ sends to $\mathcal{V}$ a univariate polynomial $g_v(X_v)$ claimed to equal

$$g(r_1,\ldots,r_{v-1},X_v).$$

  $\mathcal{V}$ checks that g_v is a univariate polynomial of degree at most $\deg_v(g)$, rejecting if not, and also checks that $g_{v-1}(r_{v-1}) = g_v(0)+g_v(1)$.
- $\mathcal{V}$ chooses a random element $r_v \in \mathbb{F}$ and evaluates $g(r_1,\ldots,r_v)$ with a single oracle query to g. $\mathcal{V}$ checks that $g_v(r_v) = g(r_1,\ldots,r_v)$, rejecting if not.
- If $\mathcal{V}$ has not yet rejected, $\mathcal{V}$ halts and accepts.

In round k:

- **Prover** sends the degree-2 univariate polynomial $g_k(X)$, which can be specifiied by 3 points $g_k(0), g_k(1), g_k(2)$

$$g_k(X) = \sum_{x_{k+1}\in\{0,1\}} \cdots \sum_{x_{\log N}\in\{0,1\}} \tilde{c}(r_1,\ldots,r_{k-1},X,x_{k+1},\ldots,x_{\log N}) \cdot \tilde{F}(u,r_1,\ldots,r_{k-1},X,x_{k+1},\ldots,x_{\log N})$$

- **Prover** is required to evaluate $\tilde{c}$ and $\tilde{F}$ at all points of the form

$$\tilde{c}(r_1,\ldots,r_{k-1},\{0,1,2\},x_{k+1},\ldots,x_{\log N})$$

$$\tilde{F}(u,r_1,\ldots,r_{k-1},\{0,1,2\},x_{k+1},\ldots,x_{\log N})$$

  where $(x_{k+1},\ldots,x_{\log N}) \in \{0,1\}^{\log N-k}$

In the last round:

- **Verifier** is required to evaluate random points $\tilde{F}(u,v), \tilde{c}(v)$

Sumcheck for FFT

Generate sumcheck messages

Goal: evaluate points $\tilde{c}(r_1, \dots, r_{k-1}, \{0,1,2\}, x_{k+1}, \dots, x_{\log N}), \tilde{F}(u, r_1, \dots, r_{k-1}, \{0,1,2\}, x_{k+1}, \dots, x_{\log N})$ for $k = 1, \dots, \log N$

1. **Initialize** the values of $\tilde{c}(x), \tilde{F}(u, x)$ on all $x \in \{0,1\}^{\log N}$ ($u \in \mathbb{F}^{\log M}$ is fixed)
   - Naive approach requires $O(MN)$ to compute $\tilde{F}(u, x)$
   - This work computes $\tilde{F}(u, x)$ in $O(M + N)$
2. Generate all sumcheck messages in $O(N)$

Outline:

- With initializations, how to generate all sumcheck messages in $O(N)$? / The naive approach to compute $\tilde{F}(u, x)$ in $O(MN$
- How to initialize $\tilde{F}(u, x)$ in $O(M + N)$?

Sumcheck for FFT

Generate sumcheck messages in $O(2^\ell)$ with $O(2^\ell)$-order initialization

Goal:

For simplicity, we denote the ℓ-degree multilinear poly. over $\mathbb{F}^\ell$ by $h(x)$.

1. Initialize: All evaluations of $h(x)$ for $x \in \{0,1\}^\ell$ can be computed in $O(2^\ell)$
2. Compute $\{(r_1, \dots, r_{k-1}, \{0,1,2\}, b_{k+1}, \dots, b_\ell)\}_{k=1,\dots,\ell; b_{k+1},\dots,b_\ell \in \{0,1\}}$ $\#= 3 \cdot 2^\ell$ points

Lemma: Each point can be evaluated in $O(2^\ell)$.

Method 1: compute each point one by one with total runtime $O(2^{2\ell})$.

Method 2: reduce time to $O(2^\ell)$ per round with total runtime $O(\ell 2^\ell)$ over ℓ rounds.

Key Fact:

The points evaluated in round k is **highly structured** that the tailing coordinates are all Boolean.

For any input z of the form $(r_1, \dots, r_{k-1}, \{0,1,2\}, b_{k+1}, \dots, b_\ell)$:

$$\tilde{h}(z) = \sum_{x \in \{0,1\}^\ell} \prod_{i=0}^{\ell} ((1-z_i)(1-x_i) + z_i x_i) \cdot h(x)$$

$= 0$ for all $(x_{k+1}, \dots, x_\ell) \neq (b_{k+1}, \dots, b_\ell)$

Consider all evaluations of $h(x)$ for $x \in \{0,1\}^\ell$ as a list h of size 2^ℓ.

It enables P to evaluate $\tilde{h}(z)$ in round k at all points with a single pass over h.

Sumcheck for FFT

Generate sumcheck messages in $O(2^\ell)$ with $O(2^\ell)$-order initialization

Method 2: reduce time to $O(2^\ell)$ per round with total runtime $O(\ell 2^\ell)$ over ℓ rounds.

Key Fact:
The points evaluated in round k are **highly structured** where the tailing coordinates are all Boolean.
For any input z of the form $(r_1, \dots, r_{k-1}, \{0,1,2\}, b_{k+1}, \dots, b_\ell)$:

$$\tilde{h}(z) = \sum_{x \in \{0,1\}^\ell} \prod_{i=0}^{\ell} ((1-z_i)(1-x_i) + z_i x_i) \cdot h(x)$$

$$= 0 \text{ for all } (x_{k+1}, \dots, x_\ell) \neq (b_{k+1}, \dots, b_\ell)$$

- Consider all evaluations of $h(x)$ for $x \in \{0,1\}^\ell$ as **a table h** of size 2^ℓ.
- Each entry of h **contributes to $\tilde{h}(r_1, \dots, r_{k-1}, \{0,1,2\}, b_{k+1}, \dots, b_\ell)$ for only one tuple** $(b_{k+1}, \dots, b_\ell)$.
- It enables P to evaluate $\tilde{h}(z)$ in round k at all points with a single pass over h.

Method 3: have prover reuse work across rounds reducing time to $O(2^\ell/2^k)$ in round k with $O(2^\ell)$ total runtime.
Informal Fact:
Two entries i, j agree in their last s bits, then h_i, h_j contribute to the same three points in each of the final s rounds.

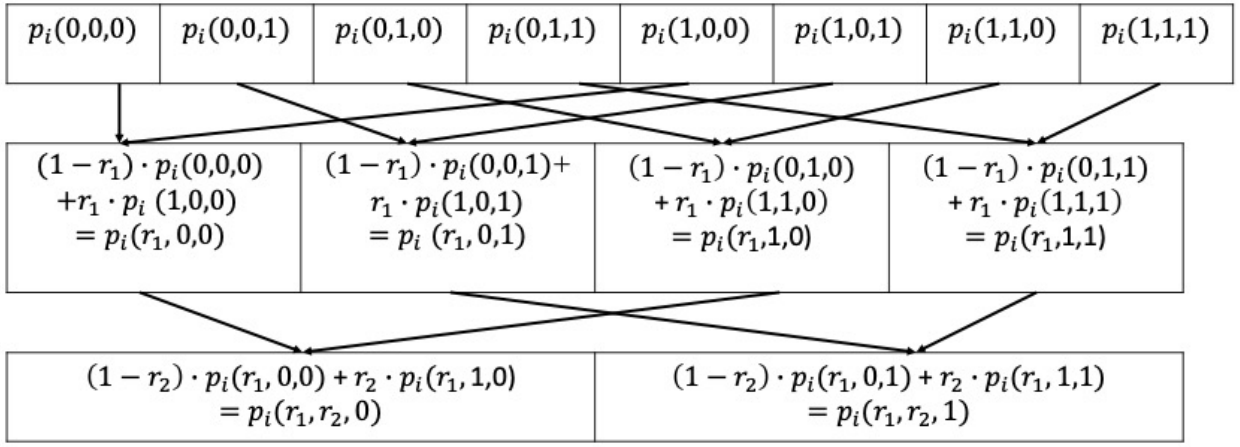
Sumcheck for FFT

Generate sumcheck messages in $O(2^\ell)$ with $O(2^\ell)$-order initialization

Method 3: have prover reuse work across rounds reducing time to $O(2^\ell/2^k)$ in round k with $O(2^\ell)$ total runtime.

Informal Fact:

Two indices i, j agree in their last s bits, then h_i, h_j contribute to the same three points in each of the final s rounds.



Sumcheck for FFT

Generate sumcheck messages in $O(2^\ell)$ with $O(2^\ell)$-order initialization

Method 3: have prover reuse work across rounds reducing time to $O(2^\ell/2^k)$ in round k with $O(2^\ell)$ total runtime.

Informal Fact:

Two indices i, j agree in their last s bits, then h_i, h_j contribute to the same three points in each of the final s rounds.

Lemma 4.3. *Suppose that p is an ℓ-variate multilinear polynomial over field $\mathbb{F}$ and that A is an array of length 2^ℓ such that for each $x \in \{0,1\}^\ell$, $A[x] = p(x)$.*[52] *Then for any $r_1 \in \mathbb{F}$, there is an algorithm running in time $O(2^\ell)$ that, given r_1 and A as input, computes an array B of length $2^{\ell-1}$ such that for each $x' \in \{0,1\}^{\ell-1}$, $B[x'] = p(r_1, x')$.*

Proof:

$$p(x_1, x_2, \ldots, x_\ell) = x_1 \cdot p(1, x_2, \ldots, x_\ell) + (1 - x_1) \cdot p(0, x_2, \ldots, x_\ell).$$

$$B[x'] \leftarrow r_1 \cdot A[1, x'] + (1 - r_1) \cdot A[0, x'].$$

Lemma 4.4. *Let h be any ℓ-variate multilinear polynomial over field $\mathbb{F}$ for which all evaluations of $h(x)$ for $x \in \{0,1\}^\ell$ can be computed in time $O(2^\ell)$. Let $r_1, \ldots, r_\ell \in \mathbb{F}$ be any sequence of ℓ field elements. Then there is an algorithm that runs in time $O(2^\ell)$ and computes the following quantities:*

$$\{h(r_1, \ldots, r_{i-1}, \{0,1,2\}, b_{i+1}, \ldots, b_\ell)\}_{i=1,\ldots,\ell;\, b_{i+1},\ldots,b_\ell \in \{0,1\}} \tag{4.14}$$

Proof:

given:

$$S_i = \{h(r_1, \ldots, r_{i-1}, b_i, b_{i+1}, \ldots, b_\ell)\}_{b_i,\ldots,b_\ell \in \{0,1\}}.$$

$$h(r_1, \ldots, r_{i-1}, 2, b_{i+1}, \ldots, b_\ell) = 2 \cdot h(r_1, \ldots, r_{i-1}, 1, b_{i+1}, \ldots, b_\ell) - h(r_1, \ldots, r_{i-1}, 0, b_{i+1}, \ldots, b_\ell),$$

S_{i+1} can be computed in time $O(2^{\ell-i})$.

Sumcheck for FFT

Back to our goal:

Goal: evaluate points $\tilde{c}(r_1, \ldots, r_{k-1}, \{0,1,2\}, x_{k+1}, \ldots, x_{\log N}), \tilde{F}(u, r_1, \ldots, r_{k-1}, \{0,1,2\}, x_{k+1}, \ldots, x_{\log N})$ for $k = 1, \ldots, \log N$

1. **Initialize** the values of $\tilde{c}(x), \tilde{F}(u,x)$ on all $x \in \{0,1\}^{\log N}$ ($u \in \mathbb{F}^{\log M}$ is fixed)
   - Naive approach requires $O(MN)$ to compute $\tilde{F}(u,x)$
   - This work computes $\tilde{F}(u,x)$ in $O(M+N)$
2. Generate all sumcheck messages in $O(N)$

Algorithm 1 Sumcheck$(\tilde{c}, \mathbf{A}_c, \tilde{F}, \mathbf{A}_F, r_1, \ldots, r_{\log N})$

Input: Arrays $\mathbf{A}_c$ and $\mathbf{A}_F$ storing $\tilde{c}(x)$ and $\tilde{F}(u,x)$ on all $x \in \{0,1\}^{\log N}$, random $r_1, \ldots, r_{\log N}$;

Output: $\log N$ sumcheck messages for $\sum_{x \in \{0,1\}^{\log M}} \tilde{c}(x)\tilde{F}(u,x)$. Each message consists of 3 elements;

1: **for** $i = 1, \ldots, \log N$ **do**
2: **for** $b \in \{0,1\}^{\ell-i}$ **do** *// B is the number represented by b.*
3: **for** $t = 0, 1, 2$ **do**
4: $\tilde{c}(r_1, \ldots, r_{i-1}, t, b) = \mathbf{A}_c[B] \cdot (1-t) + \mathbf{A}_c[B + 2^{\ell-i}] \cdot t$
5: $\tilde{F}(r_1, \ldots, r_{i-1}, t, b) = \mathbf{A}_F[B] \cdot (1-t) + \mathbf{A}_F[B + 2^{\ell-i}] \cdot t$
6: **for** $t \in \{0,1,2\}$ **do** *// Aggregate messages in round i.*
7: Send $\sum_{b \in \{0,1\}^{\ell-i}} \tilde{c}(r_1, \ldots, r_{i-1}, t, b) \cdot \tilde{F}(r_1, \ldots, r_{i-1}, t, b)$
8: **for** $b \in \{0,1\}^{\ell-i}$ **do** *// Update the arrays.*
9: $\mathbf{A}_c[B] = \mathbf{A}_c[B] \cdot (1 - r_i) + \mathbf{A}_c[B + 2^{\ell-i}] \cdot r_i$
10: $\mathbf{A}_F[B] = \mathbf{A}_F[B] \cdot (1 - r_i) + \mathbf{A}_F[B + 2^{\ell-i}] \cdot r_i$

Next task: Initialize $\tilde{F}(u,x)$ for all $x \in \{0,1\}^{\ell}$

Sumcheck for FFT

Initialize $\tilde{F}(u,x)$ for all $x \in \{0,1\}^{\log N}$:

$$\tilde{F}(u,x) = \sum_{z\in\{0,1\}^{\log M}} \tilde{\beta}(u,z)\tilde{F}(z,x)$$
$$= \sum_{z\in\{0,1\}^{\log M}} \tilde{\beta}(u,z)\omega^{\mathcal{X}\mathcal{Z}}$$
$$= \sum_{z\in\{0,1\}^{\log M}} \tilde{\beta}(u,z)\omega^{\mathcal{X}(z_0\cdot 2^{\log M-1}+z_1\cdot 2^{\log M-2}+\cdots+z_{\log M-1})},$$

$$\sum_{z\in\{0,1\}^{\log M}} \prod_{i=0}^{\log M-1} ((1-u_i)(1-z_i)+u_iz_i) \cdot \omega^{\mathcal{X}\cdot\sum_{j=0}^{\log M-1} 2^{\log M-1-j} z_j}$$
$$= \sum_{z\in\{0,1\}^{\log M}} \prod_{i=0}^{\log M-1} ((1-u_i)(1-z_i)+u_iz_i) \cdot \omega^{\sum_{j=0}^{\log M-1} 2^{\log M-1-j}\cdot(\mathcal{X}\cdot z_j)} \quad (7)$$
$$= \sum_{z\in\{0,1\}^{\log M}} \prod_{i=0}^{\log M-1} ((1-u_i)(1-z_i)+u_iz_i) \cdot \prod_{j=0}^{\log M-1} (\omega^{2^{\log M-1-j}})^{\mathcal{X}\cdot z_j}.$$

binary strings, we further denote the values represented by y, x as $\mathcal{Y}, \mathcal{X} \in \mathbb{F}$, and thus $\tilde{F}(y,x) = \omega^{\mathcal{Y}\mathcal{X}}$. The equation basically replaces

Note that $\omega^{2^{\log M-1-j}} = \omega^{\frac{M}{2^{j+1}}}$ above is the 2^{j+1}-th root of unity. We use the same notation as in [19] to denote it as $\omega_{2^{j+1}}$. Then the

$$= \sum_{z\in\{0,1\}^{\log M}} \prod_{i=0}^{\log M-1} ((1-u_i)(1-z_i)+u_iz_i) \cdot \prod_{j=0}^{\log M-1} \omega_{2^{j+1}}^{\mathcal{X}\cdot z_j}$$
$$= \sum_{z\in\{0,1\}^{\log M}} \prod_{i=0}^{\log M-1} ((1-u_i)(1-z_i)+u_iz_i) \cdot \omega_{2^{i+1}}^{\mathcal{X}\cdot z_i}$$
$$= \prod_{i=0}^{\log M-1} \sum_{z_i\in\{0,1\}} ((1-u_i)(1-z_i)+u_iz_i) \cdot \omega_{2^{i+1}}^{\mathcal{X}\cdot z_i}$$
$$= \prod_{i=0}^{\log M-1} \left((1-u_i)+u_i\cdot\omega_{2^{i+1}}^{\mathcal{X}}\right). \quad (8)$$

Let $s_{u_iz_i}$ denote $((1-u_i)(1-z_i)+u_iz_i)\cdot\omega_{2^{i+1}}^{\mathcal{X}\cdot z_i}$

logM=2; z = 00, 01, 10, 11; u=u0u_1

(1) = $s_{u_0z_0}s_{u_1z_0} + s_{u_0z_0}s_{u_1z_1} + s_{u_0z_1}s_{u_1z_0} + s_{u_0z_1}s_{u_1z_1}$

(2) = $(s_{u_0z_0} + s_{u_0z_1})(s_{u_1z_0} + s_{u_1z_1})$

Sumcheck for FFT

Initialize $\tilde{F}(u,x)$ for all $x \in \{0,1\}^{\log N}$:

$$\tilde{F}(u,x) = \sum_{z\in\{0,1\}^{\log M}} \tilde{\beta}(u,z)\tilde{F}(z,x)$$
$$= \prod_{i=0}^{\log M-1} \left((1-u_i) + u_i \cdot \omega_{2^{i+1}}^{X}\right).$$

Note that $\omega^{2^{\log M-1-j}} = \omega^{\frac{M}{2^{j+1}}}$ above is the 2^{j+1}-th root of unity. We use the same notation as in [19] to denote it as $\omega_{2^{j+1}}$. Then the

$\omega_{2^{i+1}}$ is the 2^{i+1}-th root of unity, and $\omega_{2^{i+1}}^{X}$ only has 2^{i+1} distinct values for all $X \in [N]$, which is exactly the property used in the standard FFT algorithm. Therefore, instead of computing $\tilde{F}(u,x)$

Key Idea: have prover reuse work across rounds reducing time to $O(M/2^i)$ in round $\log M - i$.

Compute in $\log M$ rounds with total runtime $O(M+N)$:

1. Prover precomputes all M distinct values of $\omega_{2^{i+1}}^{j}$ for $0 \le i < \log M - 1, 0 \le j < \min(2^{i+1}, N)$ in $O(M)$
2. Prover in round i calculates 2^{i+1} different values of $((1-u_i) + u_i \cdot \omega_{2^{i+1}}^{j})$ and multiplies them to 2^i distinct running products in round $i-1$.
3. In the last round, prover outputs N values of $\tilde{F}(u,x)$ for all $x \in \{0,1\}^{\log N}$. $O(N)$

Sumcheck for FFT

Initialize $\tilde{F}(u,x)$ for all $x \in \{0,1\}^{\log N}$:

$$\tilde{F}(u,x) = \sum_{z\in\{0,1\}^{\log M}} \tilde{\beta}(u,z)\tilde{F}(z,x)$$
$$= \prod_{i=0}^{\log M-1} \left((1-u_i) + u_i \cdot \omega_{2^{i+1}}^{X}\right).$$

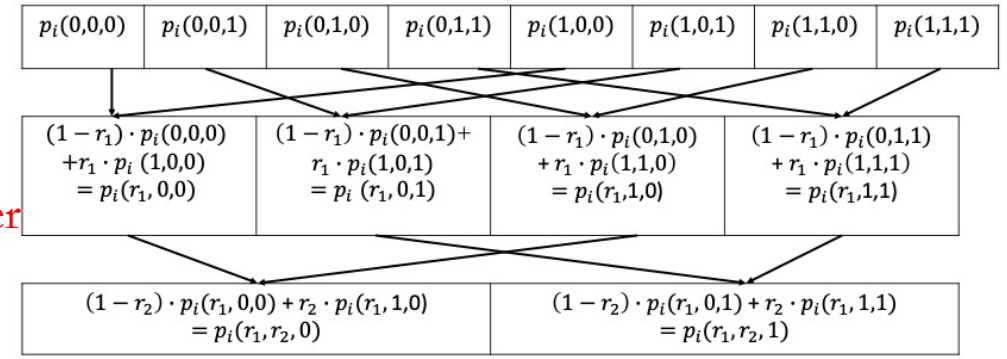
Note that $\omega^{2^{\log M-1-j}} = \omega^{\frac{M}{2^{j+1}}}$ above is the 2^{j+1}-th root of unity. We use the same notation as in [19] to denote it as $\omega_{2^{j+1}}$. Then the

$\omega_{2^{i+1}}$ is the 2^{i+1}-th root of unity, and $\omega_{2^{i+1}}^{X}$ only has 2^{i+1} distinct values for all $X \in [N]$, which is exactly the property used in the standard FFT algorithm. Therefore, instead of computing $\tilde{F}(u,x)$

Algorithm 2 $\mathbf{A}_F \leftarrow$ Initialize(ω, u, N)

Input: M-th root of unity ω, random point $u \in \mathbb{F}^{\log M}$ and the degree N;
Output: $\mathbf{A}_F$ storing $\tilde{F}(u,x)$ for all $x \in \{0,1\}^{\log N}$.

1: $\mathbf{A}_F[0] = 1$;
2: **for** $i = 0, \ldots,$ $\log N - 1$ **do** $\log M - 1$
3: **for** $j =$ $2^{i+1}-1, \ldots, 0$ **do** $\min(2^{i+1}, N) - 1$
4: $\mathbf{A}_F[j] = \mathbf{A}_F[j \mod 2^i] \cdot \left((1-u_i) + u_i \cdot \omega_{2^{i+1}}^{j}\right)$ reverse order
// In round i, $(\omega_{2^{i+1}})^X$ has 2^{i+1} possible values $\forall X \in [N]$, indexed by $j = X \mod 2^{i+1}$.
5: **return** $\mathbf{A}_F$;



Sumcheck for FFT

Apply sumcheck to $(logN)$-variate poly $g(x) = \tilde{c}(x)\tilde{F}(u,x)$

Claim: $\tilde{a}(u) = \sum_{x\in\{0,1\}^{logN}} \tilde{c}(x)\tilde{F}(u,x)$ for a random fixed point $u \in \mathbb{F}^{\log M}$

In round k:

- **Prover** sends the degree-2 univariate polynomial $g_k(X)$, which can be specifiied by 3 points $g_k(0), g_k(1), g_k(2)$

$$g_k(X) = \sum_{x_{k+1}\in\{0,1\}} \cdots \sum_{x_{\log N}\in\{0,1\}} \tilde{c}(r_1, \ldots, r_{k-1}, X, x_{k+1}, \ldots, x_{\log N}) \cdot \tilde{F}(u, r_1, \ldots, r_{k-1}, X, x_{k+1}, \ldots, x_{\log N})$$

- **Prover** is required to evaluate $\tilde{c}$ and $\tilde{F}$ at all points of the form

$$\tilde{c}(r_1, \ldots, r_{k-1}, \{0,1,2\}, x_{k+1}, \ldots, x_{\log N})$$

$$\tilde{F}(u, r_1, \ldots, r_{k-1}, \{0,1,2\}, x_{k+1}, \ldots, x_{\log N})$$

where $(x_{k+1}, \ldots, x_{\log N}) \in \{0,1\}^{\log N - k}$

In the last round:

- **Verifier** is required to evaluate random points $\tilde{F}(u,v), \tilde{c}(v)$

Summary:

- Prover time: $O(M+N)$
- Proof size: $O(\log N)$
- Verifier time: $O(\log N)$ given oracle access of $\tilde{c}(\cdot)\tilde{F}(\cdot)$

on the verifier time. In particular, the oracle accesses of $\tilde{c}(\cdot)$ and $\tilde{a}(\cdot)$ are usually provided by the prover or computed on verifier's input as in existing approaches mentioned above, but our protocol requires an additional evaluation of $\tilde{F}(\cdot)$ at a random point. It takes linear time if the verifier evaluates it on her own using a similar algorithm as the prover in Algorithm 2. We further show

Question: does it require to commit to $\tilde{F}(u,\cdot)$ rather than $\tilde{F}(\cdot)$?

Sumcheck for FFT

Delegate the computation for $\tilde{F}(u, v)$ to prover

Key idea: follow alg2 through a sequence of sumcheck protocols.

$\mathbf{A}_F$, the bookkeeping table, in Algorithm 2. Recall that $\mathbf{A}_F$ stores $\tilde{F}(u, x)\ \forall x \in \{0,1\}^{\log N}$, thus $\tilde{F}(u, v)$ is the multilinear extension of $\mathbf{A}_F$ evaluated at v. Moreover, in Algorithm 2, the values in $\mathbf{A}_F$ in

$A_F^{(i)}(\cdot) : \{0,1\}^{i+1} \to \mathbb{F}$ to denote the array $\mathbf{A}_F$ in the i-th round for $i = 0, \ldots, \log N - 1$, and $\tilde{A}_F^{(i)}(\cdot) : \mathbb{F}^{i+1} \to \mathbb{F}$ to denote its multilinear

extension. Then $\tilde{F}(u, v) = \tilde{A}_F^{(\log N - 1)}(v)$, and we can write $A_F^{(i)}(\cdot)$ as an equation of $A_F^{(i-1)}(\cdot)$:

$$A_F^{(i)}(x, b) = A_F^{(i-1)}(x)((1 - u_i) + u_i \cdot \omega_{i+1}(x, b)), \tag{9}$$

for all $x \in \{0,1\}^i, b \in \{0,1\}$, where $\omega_{i+1}(x, b) = \omega_{2^{i+1}}^j$ for $j = \sum_{k=0}^{i} x_k 2^{k+1} + b$, the number in $\mathbb{F}$ represented by (x, b) in binary.

Algorithm 2 $\mathbf{A}_F \leftarrow$ Initialize(ω, u, N)

Input: M-th root of unity ω, random point $u \in \mathbb{F}^{\log M}$ and the degree N;

Output: $\mathbf{A}_F$ storing $\tilde{F}(u, x)$ for all $x \in \{0,1\}^{\log N}$.

1: $\mathbf{A}_F[0] = 1$;
2: **for** $i = 0, \ldots, \log N - 1$ **do**
3: **for** $j = 2^{i+1} - 1, \ldots, 0$ **do**
4: $\mathbf{A}_F[j] = \mathbf{A}_F[j \mod 2^i] \cdot \left((1 - u_i) + u_i \cdot \omega_{2^{i+1}}^j\right)$
// In round i, $(\omega_{2^{i+1}})^{\mathcal{X}}$ has 2^{i+1} possible values $\forall \mathcal{X} \in [N]$, indexed by $j = \mathcal{X} \mod 2^{i+1}$.
5: **return** $\mathbf{A}_F$;

MLE:

$$\tilde{A}_F^{(i)}(x, b) = \sum_{z \in \{0,1\}^i} \tilde{\beta}(x, z)\tilde{A}_F^{(i-1)}(z)((1-u_i)+u_i \cdot \tilde{\omega}_{i+1}(z, b)), \tag{10}$$

for all $x \in \mathbb{F}^i, b \in \mathbb{F}$, as both sides agree on the Boolean hypercube

Sumcheck for FFT

Delegate the computation for $\tilde{F}(u, v)$ to prover

Key idea: follow alg2 through a sequence of sumcheck protocols.

Algorithm 2 $\mathbf{A}_F \leftarrow \text{Initialize}(\omega, u, N)$

Input: M-th root of unity ω, random point $u \in \mathbb{F}^{\log M}$ and the degree N;

Output: $\mathbf{A}_F$ storing $\tilde{F}(u, x)$ for all $x \in \{0,1\}^{\log N}$.

1: $\mathbf{A}_F[0] = 1$;
2: **for** $i = 0, \ldots, \log N - 1$ **do**
3: **for** $j = 2^{i+1} - 1, \ldots, 0$ **do**
4: $\mathbf{A}_F[j] = \mathbf{A}_F[j \mod 2^i] \cdot \left((1 - u_i) + u_i \cdot \omega_{2^{i+1}}^{j}\right)$

// In round i, $(\omega_{2^{i+1}})^X$ has 2^{i+1} possible values $\forall X \in [N]$, indexed by $j = X \mod 2^{i+1}$.

5: **return** $\mathbf{A}_F$;

MLE:

$$\tilde{A}_F^{(i)}(x, b) = \sum_{z \in \{0,1\}^i} \tilde{\beta}(x, z) \tilde{A}_F^{(i-1)}(z)((1-u_i)+u_i \cdot \tilde{\omega}_{i+1}(z, b)), \quad (10)$$

for all $x \in \mathbb{F}^i, b \in \mathbb{F}$, as both sides agree on the Boolean hypercube

Recursive protocol

- Start: claim $\tilde{F}(u, v) = \tilde{A}_F^{(\log N - 1)}(v)$
- For $i = \log N - 1, \ldots, 0$
  - Reduce its correctness to the evaluation of $\tilde{A}_F^{(i)}(\cdot)$ at a random point through a sumcheck protocol.
  - At the end of each sumcheck, verifier has to evaluate $\tilde{\beta}(\cdot)$ and $((1 - u_i) + u_i \tilde{\omega}_{i+1}(\cdot)$ at a random point.
- In the last round: $\tilde{A}_F^{(0)}(\cdot) = 1$

Round i:

- Prover time: $O(2^i)$ using alg1
- Proof size: $O(i)$
- Verifier time: $O(i)$
  - $\tilde{\beta}(\cdot) = \prod_{i=0}^{i-1} \ldots$ costs $O(i)$
  - $\tilde{\omega}_{i+1}(r) = \sum_{x \in \{0,1\}^{i+1}} \beta(r, x) \omega_{2^{i+1}}^{j}$ for $j = \sum_{k=0}^{i+1} x_k 2^k$ $O(i)$

Sumcheck for FFT

Delegate the computation for $\tilde{F}(u, v)$ to prover

Key idea: follow alg2 through a sequence of sumcheck protocols.

Recursive protocol

- Start: claim $\tilde{F}(u, v) = \tilde{A}_F^{(\log N - 1)}(v)$
- For $i = \log N - 1, \ldots, 0$
  - Reduce its correctness to the evaluation of $\tilde{A}_F^{(i)}(\cdot)$ at a random point through a sumcheck protocol.
  - At the end of each sumcheck, verifier has to evaluate $\tilde{\beta}(\cdot)$ and $((1 - u_i) + u_i \tilde{\omega}_{i+1}(\cdot)$ at a random point.
- In the last round: $\tilde{A}_F^{(0)}(\cdot) = 1$

$$\tilde{F}(u, x) = \sum_{z \in \{0,1\}^{\log M}} \tilde{\beta}(u, z) \tilde{F}(z, x)$$
$$= \prod_{i=0}^{\log M - 1} \left((1 - u_i) + u_i \cdot \omega_{2^{i+1}}^{x} \right).$$

$$\tilde{\omega}_{i+1}(r) = \sum_{x \in \{0,1\}^{i+1}} \beta(r, x) \omega_{2^{i+1}}^{j} \text{ for } j = \sum_{k=0}^{i+1} x_k 2^k$$
$$= \prod_{k=0}^{i+1} \left((1 - r_k) + r_k \omega_{2^{i+1}}^{2^k} \right)$$

Round i:

- Prover time: $O(2^i)$ using alg1
- Proof size: $O(i)$
- Verifier time: $O(i)$
  - $\tilde{\beta}(\cdot) = \prod_{i=0}^{i-1} \ldots$ costs $O(i)$
  - $\tilde{\omega}_{i+1}(r) = \sum_{x \in \{0,1\}^{i+1}} \beta(r, x) \omega_{2^{i+1}}^{j}$ for $j = \sum_{k=0}^{i+1} x_k 2^k$ $O(i)$

Summary: $(\log N)$ **rounds**

- Prover time: $O(N) = O(\sum_{i=1}^{\log N} 2^i)$
- Proof size: $O(\log^2 N) = O(\sum_{i=1}^{\log N} i)$
- Verifier time: $O(\log^2 N) = O(\sum_{i=1}^{\log N} i)$

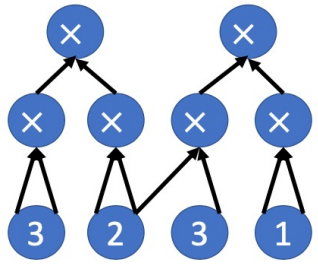
Generalization of GKR for CNN

Generalized addition and multiplication gates

Original GKR:

- Designed for a layered arithmetic circuit of size S, depth d and fan-in two.

$$\tilde{W}_i(z) = \sum_{b,c\in\{0,1\}^{k_{i+1}}} \widetilde{\text{add}}_i(z,b,c)\left(\tilde{W}_{i+1}(b)+\tilde{W}_{i+1}(c)\right) + \widetilde{\text{mult}}_i(z,b,c)\left(\tilde{W}_{i+1}(b)\cdot\tilde{W}_{i+1}(c)\right)$$



$W_0(0) = 36, \quad W_0(1) = 6$

$W_1(0,0) = 9,\ W_1(0,1)=4, W_1(1,0) = 6, W_1(1,1) = 1$

$W_2(0,0) = 3,\ W_2(0,1)=2, W_2(1,0) = 3, W_2(1,1) = 1$

Figure 4.12: Example circuit C and input x, and resulting functions W_i for each layer i of C. Note that C has two output gates.

Issue:

- It takes logn layers to sum n values.
- [Thaler13] partially address it by observing the addition tree can be represented as a single sumcheck.
- [This] consider the a more general case.

Generalized GKR: with fan-in ≥ 2

$$\tilde{add}_i(z,x) = \begin{cases} 1, & \text{if } V_{i+1}(x) \text{ is added to } V_i(z) \\ 0, & \text{otherwise} \end{cases}$$

$$\tilde{mult}_i(z,x,y) = \begin{cases} 1, & \text{if } V_{i+1}(x)\cdot V_{i+1}(y) \text{ is added to } V_i(z) \\ 0, & \text{otherwise} \end{cases}$$

for all $x, y \in \{0,1\}^{s_{i+1}}$ and $z \in \{0,1\}^{s_i}$. With the new definitions,

we can write the multilinear extensions of layer i as:

$$\begin{aligned} \tilde{V}_i(z) &= \sum_{x\in\{0,1\}^{s_{i+1}}} \tilde{add}_i(z,x)\cdot\tilde{V}_{i+1}(x) \\ &+ \sum_{x,y\in\{0,1\}^{s_{i+1}}} \tilde{mult}_i(z,x,y)\cdot\tilde{V}_{i+1}(x)\cdot\tilde{V}_{i+1}(y) \\ &= \sum_{x,y\in\{0,1\}^{s_{i+1}}} \Big(\tilde{\beta}(y,\vec{0})\cdot\tilde{add}_i(z,x)\cdot\tilde{V}_{i+1}(x) \\ &+ \tilde{mult}(z,x,y)\cdot\tilde{V}_{i+1}(x)\cdot\tilde{V}_{i+1}(y)\Big) \end{aligned} \tag{13}$$

Generalization of GKR for CNN

Taking inputs from arbitrary layers

Motivation:

- CNN consists of multiple conv. layers and fully-connected layers but the kernels and weight-matrices of these layers are witness from the prover.

through proof compositions of zero knowledge proofs. Our scheme in this paper only protects the privacy of the parameters while ensuring the integrity of predictions, which is the first step for zero knowledge CNN and the extensions are left as future work.

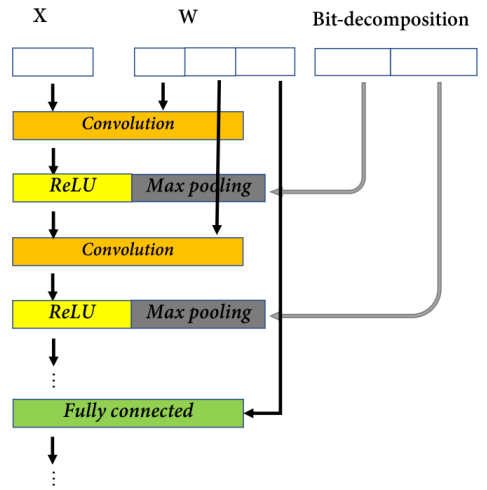
- [Zhang20] proposed a variant GKR protocol where a gate can take input from arbitrary layers instead of only the previous layer.

Notations:

Following the ideas in [51], we denote the subset of values in the input layer connecting to the i-th layer as $V_{i,\text{in}}$ of size $S_{i,\text{in}}$ and $s_{i,\text{in}} = \lceil \log S_{i,\text{in}} \rceil$, and its multilinear extension as $\tilde{V}_{i,\text{in}}(\cdot)$. We also separately define the generalized addition gates between the i-th and the $(i+1)$-th, the i-th and the input layer as $\tilde{Xadd}_{i,i+1}(z,x)$, $\tilde{Xadd}_{i,\text{in}}(z,x)$. Similarly, we define the generalized multiplication

Generalized GKR:

A gate takes input from either the layer above or from the input gate.



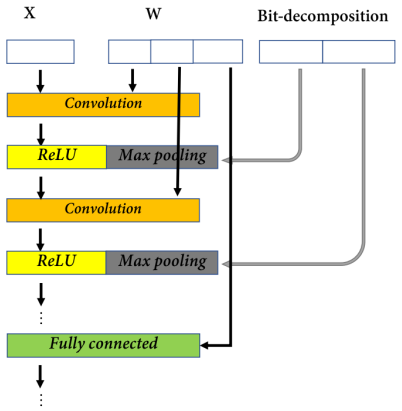
$$\tilde{V}_i(z) = \sum_{x\in\{0,1\}^{s_{i+1}}} \tilde{Xadd}_{i,i+1}(z,x)\cdot\tilde{V}_{i+1}(x) + \sum_{x\in\{0,1\}^{s_{i,\text{in}}}} \tilde{Xadd}_{i,\text{in}}(z,x)\cdot\tilde{V}_{i,\text{in}}(x) + \sum_{x,y\in\{0,1\}^{s_{i+1}}} \tilde{Xmult}_{i,i+1,i+1}(z,x,y)\cdot\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y) + \sum_{x,y\in\{0,1\}^{s_{i,\text{in}}}} \tilde{Xmult}_{i,\text{in},\text{in}}(z,x,y)\cdot\tilde{V}_{i,\text{in}}(x)\tilde{V}_{i,\text{in}}(y) + \sum_{\substack{x\in\{0,1\}^{s_{i+1}},\\ y\in\{0,1\}^{s_{i,\text{in}}}}} \tilde{Xmult}_{i,i+1,\text{in}}(z,x,y)\cdot\tilde{V}_{i+1}(x)\tilde{V}_{i,\text{in}}(y).$$

Generalization of GKR for CNN

Taking inputs from arbitrary layers

Generalized GKR:

A gate takes input from either the layer above or from the input gate.



$$\tilde{V}_i(z) = \sum_{x\in\{0,1\}^{s_{i+1}}} X\tilde{add}_{i,i+1}(z,x)\cdot\tilde{V}_{i+1}(x)$$
$$+\sum_{x\in\{0,1\}^{s_{i,\text{in}}}} X\tilde{add}_{i,\text{in}}(z,x)\cdot\tilde{V}_{i,\text{in}}(x)$$
$$+\sum_{x,y\in\{0,1\}^{s_{i+1}}} X\tilde{mult}_{i,i+1,i+1}(z,x,y)\cdot\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y)$$
$$+\sum_{x,y\in\{0,1\}^{s_{i,\text{in}}}} X\tilde{mult}_{i,\text{in},\text{in}}(z,x,y)\cdot\tilde{V}_{i,\text{in}}(x)\tilde{V}_{i,\text{in}}(y)$$
$$+\sum_{\substack{x\in\{0,1\}^{s_{i+1}},\\ y\in\{0,1\}^{s_{i,\text{in}}}}} X\tilde{mult}_{i,i+1,\text{in}}(z,x,y)\cdot\tilde{V}_{i+1}(x)\tilde{V}_{i,\text{in}}(y).$$

Recursive protocol

- Start: claim $\tilde{V}_0(z)$
- For $i = 0, \dots, d$

  At the end of each sumcheck protocol:
  - **Goal:** Reduce its correctness to evaluation of $\tilde{V}_{i+1}(\cdot)$ at a random point.
  - **But** it is reduced to two evaluations of $\tilde{V}_{i+1}(\cdot)$ and two evaluations of $\tilde{V}_{i,in}(\cdot)$
- In the last round (reaching the input layer): verifier has received two evaluations about the input per layer.

Solution:

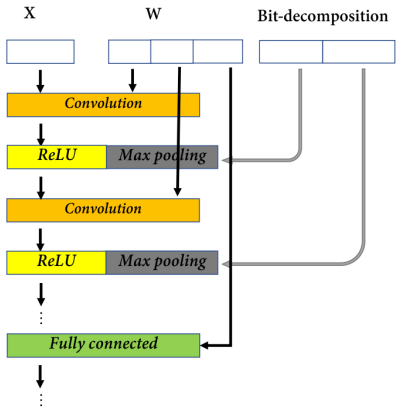
1. Combine all evaluations to a single evaluation of the input $\tilde{V}_{in}(\cdot)$ through **a random linear combination**. [Zhang20]
2. Run the sumcheck protocol, verifier reducing it to a single evaluation of $\tilde{V}_{in}(\cdot)$.

Generalization of GKR for CNN

Taking inputs from arbitrary layers

Generalized GKR:

A gate takes input from either the layer above or from the input gate.



$$\tilde{V}_i(z) = \sum_{x\in\{0,1\}^{s_{i+1}}} \tilde{add}_{i,i+1}(z,x)\cdot\tilde{V}_{i+1}(x) + \sum_{x\in\{0,1\}^{s_{i,\text{in}}}} \tilde{add}_{i,\text{in}}(z,x)\cdot\tilde{V}_{i,\text{in}}(x) + \sum_{x,y\in\{0,1\}^{s_{i+1}}} \tilde{mult}_{i,i+1,i+1}(z,x,y)\cdot\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y) + \sum_{x,y\in\{0,1\}^{s_{i,\text{in}}}} \tilde{mult}_{i,\text{in},\text{in}}(z,x,y)\cdot\tilde{V}_{i,\text{in}}(x)\tilde{V}_{i,\text{in}}(y) + \sum_{\substack{x\in\{0,1\}^{s_{i+1}},\\ y\in\{0,1\}^{s_{i,\text{in}}}}} \tilde{mult}_{i,i+1,\text{in}}(z,x,y)\cdot\tilde{V}_{i+1}(x)\tilde{V}_{i,\text{in}}(y).$$

Solution:

1. Combine all evaluations to a single evaluation of the input $\tilde{V}_{in}(\cdot)$ through **a random linear combination**. [Zhang20]
2. Run the sumcheck protocol, verifier reducing it to a single evaluation of $\tilde{V}_{in}(\cdot)$.

Suppose the evaluations received from layer i are $\tilde{V}_{i,\text{in}}(z_{i,0})$ and $\tilde{V}_{i,\text{in}}(z_{i,1})$, the verifier generates $r_{i,0}, r_{i,1} \in \mathbb{F}$ for layer i and combines all the evaluations through a random linear combination:

$$\sum_i \left(r_{i,0}\tilde{V}_{i,\text{in}}(z_{i,0}) + r_{i,1}\tilde{V}_{i,\text{in}}(z_{i,1})\right)$$

$$= \sum_i \left(r_{i,0} \sum_{z\in\{0,1\}^{s_{\text{in}}}} C_i(z_{i,0},z)\tilde{V}_{\text{in}}(z) + r_{i,1} \sum_{z\in\{0,1\}^{s_{\text{in}}}} C_i(z_{i,1},z)\tilde{V}_{\text{in}}(z)\right)$$

$$= \sum_{z\in\{0,1\}^{s_{\text{in}}}} \tilde{V}_{\text{in}}(z)\left(\sum_i \left(r_{i,0}C_i(z_{i,0},z) + r_{i,1}C_i(z_{i,1},z)\right)\right) \quad (14)$$

where $C_i(z_i, z)$ is defined as:

$$C_i(z_i,z) = \begin{cases} 1, & \text{if the } z_i\text{-th value in } V_{i,\text{in}} \text{ is the } z\text{-th value in } V_{\text{in}} \\ 0, & \text{otherwise} \end{cases}$$

Generalization of GKR for CNN

Convolutional layer

Motivation:

- Have an efficient protocol to verify the result of the 2-D convolution between one input and one kernel.
- In practice, there are multiple channels and kernels in each convolution layer.

Improvement:

- Instead of repeating multiple times, it represents the computation of an entire convolutional layer.
- It utilizes the linearity of FFT.
  - Original: $ch_{in} \cdot ch_{out}$ FFTs and IFFTs with prover time of $O(ch_{in} \cdot ch_{out} \cdot n^2)$
  - Improvements: reduce to ch_{out} IFFTs with prover time of $O(ch_{out} \cdot n^2)$

one input and one kernel

$$\bar{U} = \bar{X} * \bar{W} = \mathtt{IFFT}(\mathtt{FFT}(\bar{X}) \odot \mathtt{FFT}(\bar{W})) \tag{12}$$

multiple channels and kernels

product. Recall that the input data to a convolutional layer is $X \in \mathbb{F}^{\mathsf{ch}_{in} \times n \times n}$ and the kernel is $W \in \mathbb{F}^{\mathsf{ch}_{out} \times \mathsf{ch}_{in} \times w \times w}$. Here with omit the subscript of layer i for the ease of notations. The convolutional layer computes $U \in \mathbb{F}^{\mathsf{ch}_{out} \times (n-w+1) \times (n-w+1)}$ where for each $0 \le \tau < \mathsf{ch}_{out}$, $0 \le j, k < n - w + 1$,

$$U[\tau, j, k] = \sum_{\sigma=0}^{\mathsf{ch}_{in}-1} \sum_{t=0,l=0}^{(w-1),(w-1)} X[\sigma, j, k] \cdot W[\tau, \sigma, t, l]$$

$$= \sum_{\sigma=0}^{\mathsf{ch}_{in}-1} \sum_{i=0}^{n^2-1-jn-k} \bar{X}_\sigma[n^2 - 1 - jn - k - i] \cdot \bar{W}_{\tau,\sigma}[i].$$

algorithm. Let $\bar{U}_\tau$ be the vector defined by the τ-th channel of the output U, as we show in Section 3.2, we have

$$\bar{U}_\tau = \sum_{\sigma=0}^{\mathsf{ch}_{in}-1} \bar{X}_\sigma * \bar{W}_{\tau,\sigma}$$

$$= \sum_{\sigma=0}^{\mathsf{ch}_{in}-1} \mathtt{IFFT}(\mathtt{FFT}(\bar{X}_\sigma) \odot \mathtt{FFT}(\bar{W}_{\tau,\sigma})) \tag{15}$$

$$= \mathtt{IFFT}\left(\sum_{\sigma=0}^{\mathsf{ch}_{in}-1} \mathtt{FFT}(\bar{X}_\sigma) \odot \mathtt{FFT}(\bar{W}_{\tau,\sigma})\right).$$