機械学習を用いた多項式イデアルの学習

石原侑樹

日本大学理工学部数学科

2026年7月18日　可換代数と組合せ論セミナー
@東邦大学 習志野キャンパス

目次

- 機械学習とデータセット生成の概要
- グレブナー基底について
- グレブナー基底の学習
- 数学計算のための機械学習ライブラリ「CALT」の実演
- まとめ

自己紹介

石原侑樹

- **学部(2012-16): 東京理科大学 (理学部数学科)**

  卒研テーマ：有限群の表現論

  3年次にカリフォルニア大学デイビス校に留学

- **大学院(16-21): 立教大学 (数学専攻)**

  博論題目：二重イデアル商と効果的な局所化 （指導教員：横山和弘）

- **助教 (21-24) 東京理科大学 (理学部第一部応用数学科)**

- **助教 (24-): 日本大学（理工学部数学科）**

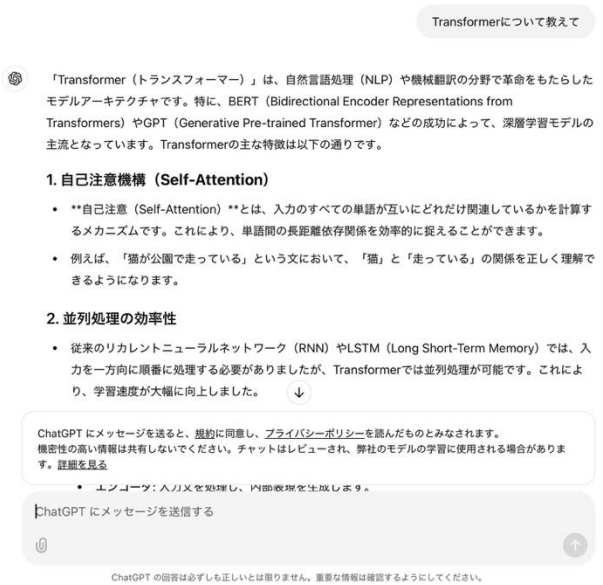
研究分野：計算機代数

キーワード: グレブナー基底、準素イデアル分解

研究背景(AIと数学)

- 近年、ChatGPTを始めとした「生成AI」が普及
- 生成AIの多くは「Transformer」という深層学習モデルがベース
- 最近では、Transformerを数学の計算に応用する研究が活発に

https://chatgpt.com



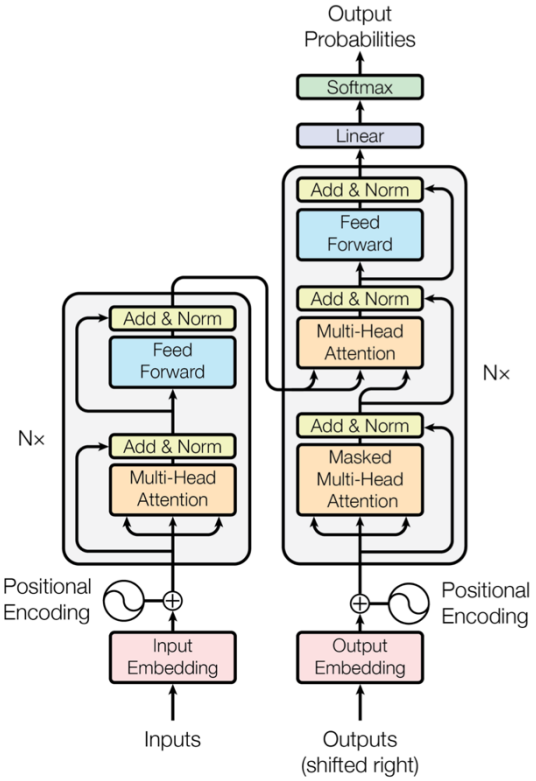


Figure 1: The Transformer - model architecture.

Transformerのモデル図
"Attention is all you need"
(Vaswani et al, 2017)より

計算機代数とTransformer

- Transformer は ChatGPT (Generative Pre-trained Transformer)

  等で使われている深層学習モデル

- Transformerは下記の計算の学習に使われている

  ＋ 最大公約数 (Charton, 2024)

  ＋ 積分 (Lample and Charton, 2020)

  ＋ 行列の計算 (Charton, 2022)

  ＋ グレブナー基底 (Kera et al., 2024)

  ＋ リアプノフ関数 (Alfarano et al., 2024)

➔ Transformerは数式処理の計算を

  高速化する可能性を秘めている

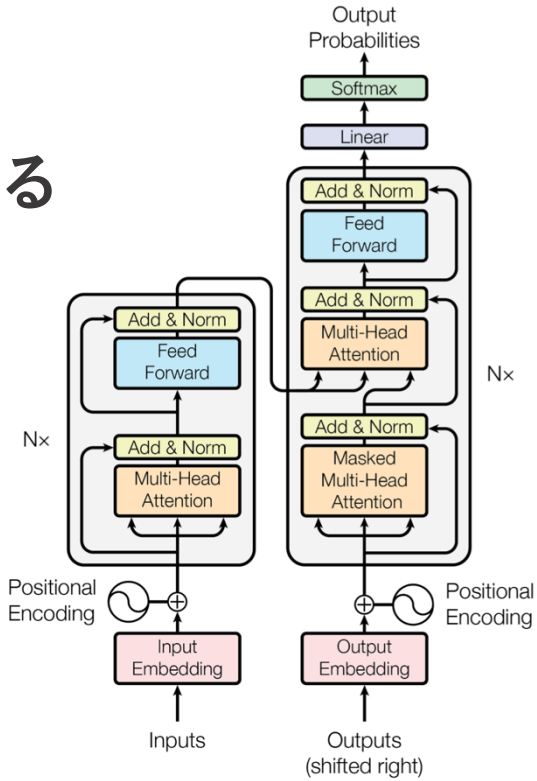


Figure 1: The Transformer - model architecture.

"Attention is all you need"
(Vaswani et al, 2017)

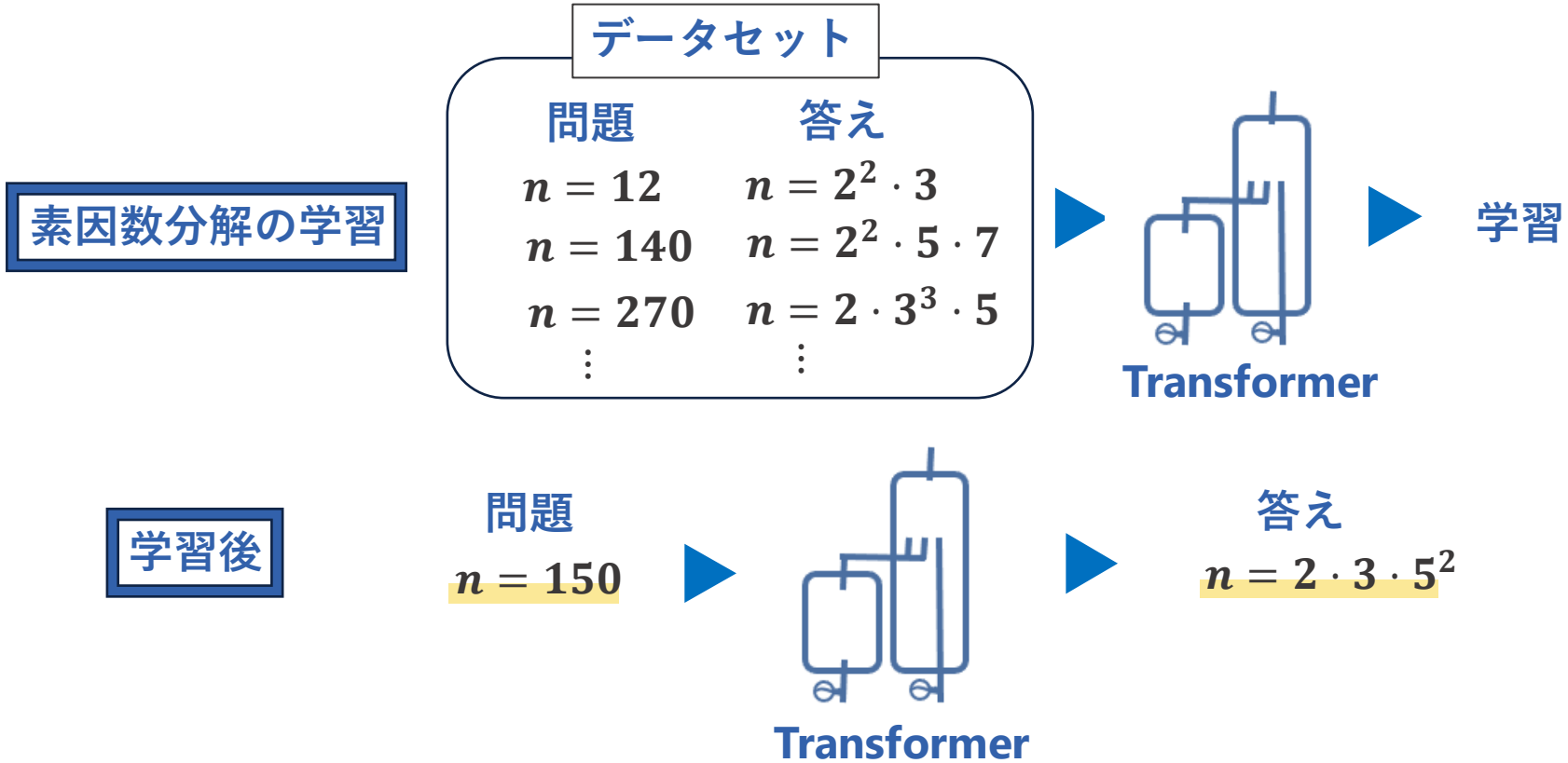
Transformerが難しい積分を解いた例

$f(x)$	$\int f(x)$
$x^2\left(\tan^2(x)+1\right)+2x\tan(x)+1$	$x^2\tan(x)+x$
$1+\frac{2\cos(2x)}{\sqrt{\sin^2(2x)+1}}$	$x+\operatorname{asinh}(\sin(2x))$
$\frac{x\tan(x)+\log(x\cos(x))-1}{\log(x\cos(x))^2}$	$\frac{x}{\log(x\cos(x))}$
$-\frac{2x\cos\left(\operatorname{asin}^2(x)\right)\operatorname{asin}(x)}{\sqrt{1-x^2}\sin^2\left(\operatorname{asin}^2(x)\right)}+\frac{1}{\sin\left(\operatorname{asin}^2(x)\right)}$	$\frac{x}{\sin\left(\operatorname{asin}^2(x)\right)}$

表は Deep Learning For Symbolic Mathematics（**Lample and Charton, 2020**）から

Transformerによる学習

1. 「問題」と「答え」のペアを大量に用意（例：100万個）
2. それらをTransformerに投入し学習
3. 学習結果を元に、新たな問題（入力）に対し答えを出力



数学の計算の学習の問題点

・文章の学習と違い、数学の計算のデータベースが不足しがち

⇨ 新たにデータセットを生成する必要がある

データセット生成のパラドックス

時間がかかる計算を学習で高速化するためには
大量のデータセットが必要であるが、
そのデータセット生成には膨大な時間がかかる

例：計算に１分かかるデータを１００万個作るには
１００万分＝694日かかる。
⇨１秒に高速化できても意味がない

次の積分の答えは？

$$\int (x^2(\tan^2(x)+1)+2x\tan(x)+1)dx$$

実は…

$$(x^2 \tan(x) + x)' = x^2(\tan^2(x) + 1) + 2x \tan(x) + 1$$

から

$$\int (x^2(\tan^2(x) + 1) + 2x \tan(x) + 1)dx = x^2 \tan(x) + x + C$$

⇨積分を求めるより、微分を計算した方がデータが準備しやすい

Transformerが難しい積分を解いた例

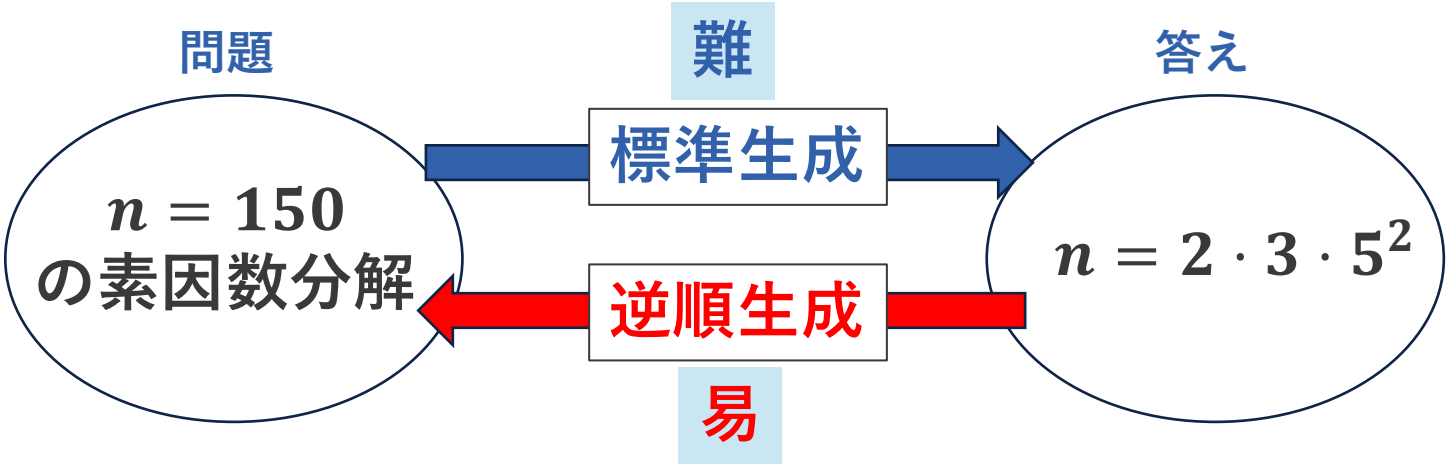
$f(x)$	$\int f(x)$
$x^2\left(\tan^2(x)+1\right)+2x\tan(x)+1$	$x^2\tan(x)+x$
$1+\frac{2\cos(2x)}{\sqrt{\sin^2(2x)+1}}$	$x+\operatorname{asinh}(\sin(2x))$
$\frac{x\tan(x)+\log(x\cos(x))-1}{\log(x\cos(x))^2}$	$\frac{x}{\log(x\cos(x))}$
$-\frac{2x\cos\left(\operatorname{asin}^2(x)\right)\operatorname{asin}(x)}{\sqrt{1-x^2}\sin^2\left(\operatorname{asin}^2(x)\right)}+\frac{1}{\sin\left(\operatorname{asin}^2(x)\right)}$	$\frac{x}{\sin\left(\operatorname{asin}^2(x)\right)}$

表は Deep Learning For Symbolic Mathematics（**Lample and Charton, 2020**）から

問題点の解決方法

「問題」から「答え」を作る（標準生成）のではなく

「答え」から「問題」を作る（逆順生成）



発表者らの結果

グレブナー基底に対し「逆順生成」を適用し

データセット生成の高速化および学習に成功

目次

- 機械学習とデータセット生成の概要
- グレブナー基底について
- グレブナー基底の学習
- 数学計算のための機械学習ライブラリ「CALT」の実演
- まとめ

連立代数方程式の求解

連立代数方程式とは

- 多項式からなる連立方程式
- 代数幾何学や暗号理論など幅広い分野に登場

$$\begin{cases} x^2+y^2=1 \\ 2x+3y=2 \end{cases}$$

（非線形）連立代数方程式

分解

$$\begin{cases} x=1 \\ y=0 \end{cases}$$

$$\begin{cases} x=-\frac{5}{13} \\ y=\frac{12}{13} \end{cases}$$

解を求める=連立方程式を分解する

重要な未解決問題

連立方程式求解問題 （MP問題） ※ MP=Multivariate Polynomial

= n変数m連立方程式の解$(x_1, \ldots, x_n)$を求める問題

- 線形の場合と異なり、非線形の求解アルゴリズムは非常に複雑
- 既存のアルゴリズムの最悪計算量は（二重）**指数的**

未解決問題 = 汎用的かつ高速な求解アルゴリズムの発見

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ f_2(x_1, \ldots, x_n) = 0 \\ \vdots \\ f_m(x_1, \ldots, x_n) = 0 \end{cases} \longrightarrow \begin{aligned} &(x_1, \ldots, x_n) \\ = \ &(a_1, \ldots, a_n), \\ &(b_1, \ldots, b_n), \\ &\ldots \end{aligned}$$

Bernd Sturmfels, “Solving Systems of Polynomial Equations”, CBMS 97 (2002)

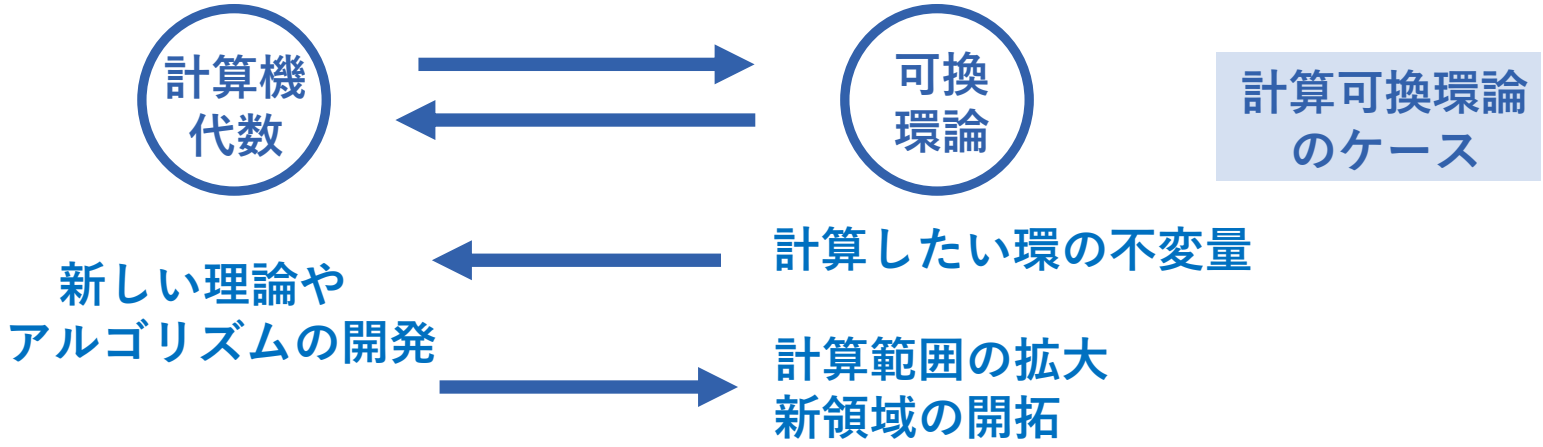
研究の意義

MP問題の解決 → 様々な未解決問題の解決

- 最適化問題の解決、効果的な制御システムの設計
- 耐量子計算機暗号の解読
- 効果的な統計モデルの開発、微分方程式の簡単化

モデルケース：CREST事業「現代の産業社会とグレブナー基底の調和」など

新しい数学の理論の構築



抽象代数によるアプローチ

イデアル = 連立方程式から作れる多項式全体の集合

$f_1 \ldots, f_m$から生成されるイデアル

$$\langle f_1, \ldots, f_m \rangle = \{h_1 f_1 + \cdots + h_m f_m \mid h_1, \ldots, h_m : \text{polynomials}\}$$

- イデアルは整数のように、足し算、掛け算、分解が可能
- 連立方程式よりイデアルの方が扱いやすい

準素分解 = イデアルの分解 = 連立方程式の分解

- 素因数分解のように、イデアルも分解可能（**準素分解**）
- 準素分解をすることで、連立方程式の解がすべて求まる

$\langle f_1 \ldots, f_m \rangle$の準素分解

$$\langle f_1, \ldots, f_m \rangle = Q_1 \cap \cdots \cap Q_r$$

M.F.Atiyah, I.G.MacDonald, “Introduction to Commutative Algebra”, (1969)

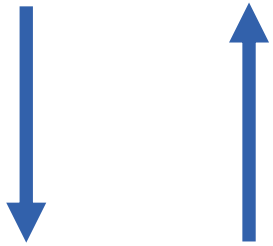
代数幾何学における準素分解

$$V(I) = \{(a_1, \ldots, a_n) \in K^n \mid f(a_1, \ldots, a_n) = 0, \forall f \in I\}$$

イデアル の代数的集合（代数多様体）

$$\boldsymbol{I = \langle f_1, \ldots, f_m \rangle \Rightarrow V(I) : f_1 = \cdots = f_m = 0}$$ **の解集合**

$$I = Q_1 \cap \cdots \cap Q_r$$ **Iの準素分解**



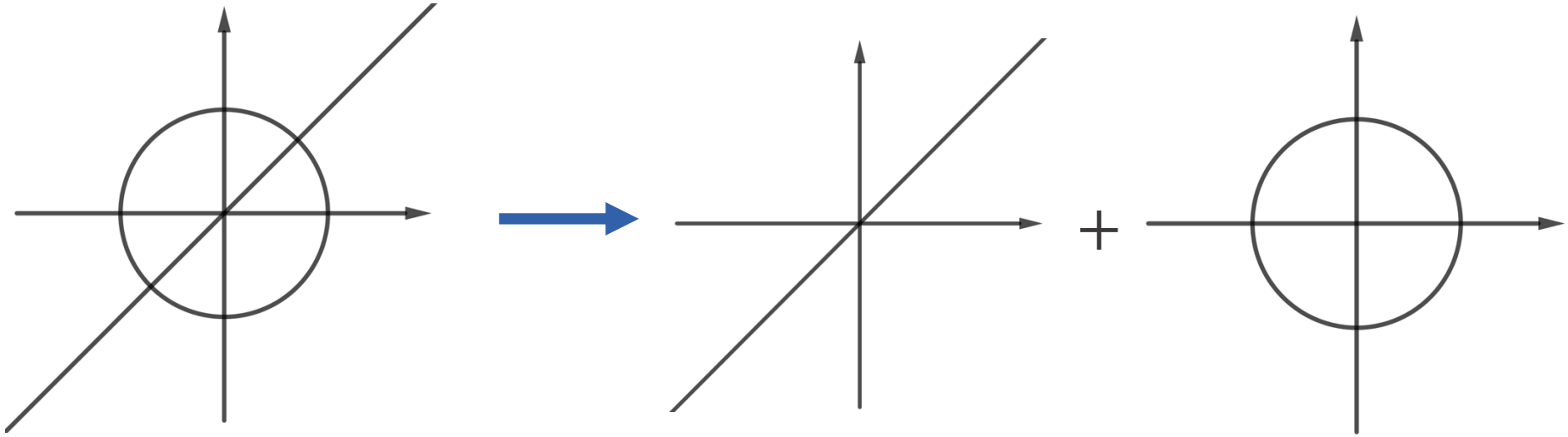
対応

$$V(I) = V(Q_1) \cup \cdots \cup V(Q_r)$$ **$V(I)$の既約分解**

幾何的構造を代数的に分析

準素分解の幾何的対応の例

$$I = \langle (x-y)(x^2+y^2-1) \rangle \quad = \langle x-y \rangle \cap \langle x^2+y^2-1 \rangle$$



$$V(I) \qquad = V(\langle x-y \rangle) \cup V(\langle x^2+y^2-1 \rangle)$$

準素分解のための計算ツール

グレブナー基底 = イデアルの本質的な情報を持った生成系

LT(f): 多項式fの先頭項　　例： $\mathbf{LT}(x^2+2y+3)=x^2$

イデアルIの生成系Gに対し、

GがIの**グレブナー基底** $:\Leftrightarrow \langle \mathrm{LT}(I)\rangle = \langle \mathrm{LT}(G)\rangle$

例： $I=\langle x^2+y^2-1, 2x+3y-2\rangle$ に対し、
$G=\{2x+3y-2, 13y^2-12y\}$ はIのグレブナー基底

$$\begin{cases} x^2+y^2=1 \\ 2x+3y=2 \end{cases} \longrightarrow \begin{cases} 2x+3y=2 \\ 13y^2-12y=0 \end{cases}$$

非グレブナー基底（に対応）　　**グレブナー基底**（に対応）

目次

- 機械学習とデータセット生成の概要
- グレブナー基底について
- グレブナー基底の学習
- 数学計算のための機械学習ライブラリ「CALT」の実演
- まとめ

Transformerで期待されること

① 計算能力の向上（例：機械学習を計算に利用）

Transformer = ChatGPT等に使用される深層学習モデル

▶ 代数計算を学習し、高速に処理できるように目指す

$$\begin{cases} x^2 + y^2 = 1 \\ 2x + 3y = 2 \end{cases}$$

入力

▶

▶ $(x, y) = (0, 1), (-\frac{5}{13}, \frac{12}{13})$

出力

② 応用範囲の拡大（例：耐量子暗号の解読に応用）

耐量子暗号= 量子コンピュータでも解読が難しい暗号

▶ 連立方程式の求解困難性を利用した暗号

多変数多項式暗号（MPKC）

Transformerとグレブナー基底

データセット

学習

Transformer

入力

$$\begin{cases} x^2+y^2=1 \\ 2x+3y=2 \end{cases}$$

出力

$$(x,y)=(0,1),(-\frac{5}{13},\frac{12}{13})$$

$$\begin{cases} x^2+y^2=2 \\ xy=1 \end{cases} \quad (x,y)=(1,1),(-1,-1)$$

学習結果に基づき未知の連立方程式に対しても解を出力できるように汎化する

暗号の概略

代表的な公開鍵暗号の例	安全性の根拠
▶ **RSA暗号**	素因数分解の困難性
▶ **楕円曲線暗号**	（楕円曲線）離散対数問題

RSA暗号、楕円曲線暗号は量子アルゴリズムで多項式時間で解読可能

耐量子計算機暗号の例	安全性の根拠
▶ **格子暗号**	最短ベクトル探索問題
▶ **同種写像暗号**	同種写像計算問題
▶ **多変数多項式暗号**	**多変数二次連立方程式求解問題（MQ問題）**

多変数多項式暗号分析への応用

データセット = 多変数多項式暗号の連立方程式とその解

↓

Transformerで学習・汎化

↓

実用レベルの暗号を解読

期待されるインパクト

- ▶ AIによる暗号解読として、AI分野・暗号分野の双方に大きな反響
- ▶ 他の暗号方式に対しても、Transformerを適応できる可能性
- ▶ AIに強い新しい暗号方式を創り出すことができる可能性

グレブナー基底の学習の問題点

・文章の学習と違い、グレブナー基底のデータベースは不足

⇨ 新たにデータセットを生成する必要がある

データセット生成のパラドックス

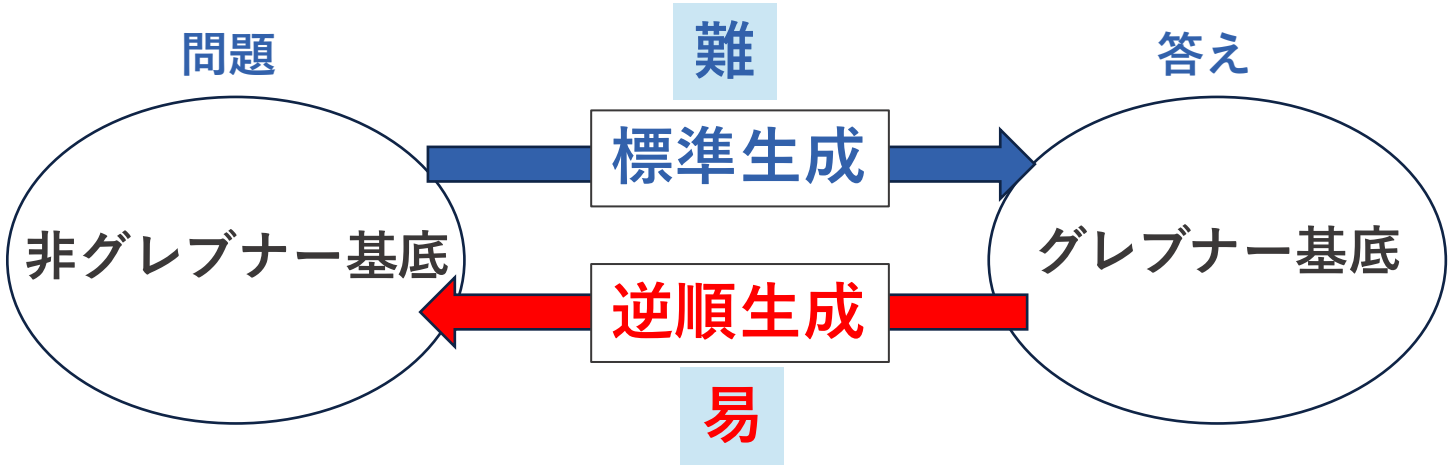
時間がかかる計算を学習で高速化するためには
大量のデータセットが必要であるが、
そのデータセット生成には膨大な時間がかかる

例：1つのグレブナー基底の計算に１分かかる場合、
１００万個作るには１００万分＝694日かかる。
⇨１秒に高速化できても意味がない

グレブナー基底データセットの逆順生成

「非グレブナー基底」から「グレブナー基底」ではなく

「グレブナー基底」から「非グレブナー基底」を生成



$\{3x + 4y - 3, 9xy + 13y^2\}$

$\{3x + 4y - 3, y^2 + 9y\}$

AG ← G

逆順生成の方法

定理（Theorem 4.6. in [Kera et al. 2024]）

I: $k[x_1, \dots, x_n]$における0次元イデアル.

$G = (g_1, \dots, g_t)^T$: Iのグレブナー基底.

$F = (f_1, \dots, f_s)^T = AG$、ここで$A$は$k[x_1, \dots, x_n]$上の$s \times t$行列

1. $\langle F \rangle = \langle G \rangle \implies s \geq n$
2. Aの左逆行列が存在 $\implies \langle F \rangle = \langle G \rangle$
3. $\langle F \rangle = \langle G \rangle \iff$ ある$\mathrm{B} \in k[x_1, \dots, x_n]^{t \times s}$が存在して、

$BA - E_t$の各行がGのsyzygy

グレブナー基底 G に左逆行列を持つ行列をかけることで
非グレブナー基底$F = AG$を大量に生成可能

実験結果①「データセットの生成時間」

Total runtime [sec] for 1000 samples.

Method	$n=2$	$n=3$	$n=4$	$n=5$
F. (STD)	**4.20**	216.3	740.1†	1411.1‡
F. (SLIMGB)	4.29	183.4	697.5†	1322.7‡
F. (STDFGLM)	7.22	8.29	21.0	164.3
B. (ours)	5.23	**5.46**	**7.05**	**7.91**

Exponential growth

Quadratic growth

† (13%+) and ‡ (25%+) timeouts included (with 5-sec budget for each sample)

- 逆順生成は通常生成より、高速にデータセットを生成可能
- 通常生成の計算時間は指数的だが、逆順生成は２次関数的

実験結果②「正答率」

	$n=2$	$n=3$	$n=4$	$n=5$
$\mathbb{Q}$	93.7	88.7	90.8	86.5
$\mathbb{F}_7$	72.3	78.1	71.3	84.3
$\mathbb{F}_{31}$	46.8	50.2	51.1	28.6

- $\mathbb{Q}[x,y]$の場合、**9割越え**の正答率
- 係数体によって学習の正答率は大きく変わり、

  有限体よりも有理数体の方が性能が良い

成功例

F	G
$f_1 = 5x_0^2x_1^5 + 15x_0^2x_1^3 + x_0^2x_1^2 - 2/5x_0x_1^6 + 2/5x_0x_1^5 - x_0x_1^3$ $f_2 = -25x_0^2x_1^6 - 75x_0^2x_1^4 - 5x_0^2x_1^3 + 2x_0x_1^7 - 2x_0x_1^6 + 5x_0x_1^4 + x_0 - 2/5x_1^4 + 2/5x_1^3 - x_1$ $f_3 = -5x_0^4x_1^5 - 15x_0^4x_1^3 - x_0^4x_1^2 + 2/5x_0^3x_1^6 - 2/5x_0^3x_1^5 + x_0^3x_1^3 - 2/3x_0^3x_1 + 4/15x_0^2x_1^5 - 4/15x_0^2x_1^4 + 2/3x_0^2x_1^2 - 5x_0x_1^2 + 2x_1^6 - x_1^5 + 8x_1^3$	$g_1 = x_0 - 2/5x_1^4 + 2/5x_1^3 - x_1$ $g_2 = x_1^5 + 3x_1^3$
$f_1 = -1/5x_0^2x_1 + 1/10x_0x_1^5 + 1/5x_0x_1^4 + 1/2x_0x_1^3 - 1/25x_0x_1^2 + 1/10x_0x_1 - 2/5x_0 + x_1^5 - 2/5x_1^4 + 7/5x_1^3 + x_1^2 - 27/25x_1 + 1$ $f_2 = -1/2x_0^2x_1^3 - 1/4x_0^2x_1^2 + 1/4x_0x_1^7 + 5/8x_0x_1^6 + 3/2x_0x_1^5 + 21/40x_0x_1^4 + 1/5x_0x_1^3 - 7/8x_0x_1^2 - 1/2x_0x_1 + x_0 + 5/2x_1^7 + 1/4x_1^6 + 3x_1^5 + 15/4x_1^4 - 49/20x_1^3 - 27/20x_1^2 + 29/20x_1 - 1/2$	$g_1 = x_0 - 1/2x_1^4 - x_1^3 - 5/2x_1^2 + 1/5x_1 - 1/2$ $g_2 = x_1^5 - 3/5x_1^4 + x_1^3 - x_1 + 4/5$

成功例

F	G
$f_1 = 5x_0^2x_1^5 + 15x_0^2x_1^3 + x_0^2x_1^2 - 2/5x_0x_1^6 + 2/5x_0x_1^5 - x_0x_1^3$ $f_2 = -25x_0^2x_1^6 - 75x_0^2x_1^4 - 5x_0^2x_1^3 + 2x_0x_1^7 - 2x_0x_1^6 + 5x_0x_1^4 + x_0 - 2/5x_1^4 + 2/5x_1^3 - x_1$ $f_3 = -5x_0^4x_1^5 - 15x_0^4x_1^3 - x_0^4x_1^2 + 2/5x_0^3x_1^6 - 2/5x_0^3x_1^5 + x_0^3x_1^3 - 2/3x_0^3x_1 + 4/15x_0^2x_1^5 - 4/15x_0^2x_1^4 + 2/3x_0^2x_1^2 - 5x_0x_1^2 + 2x_1^6 - x_1^5 + 8x_1^3$	$g_1 = x_0 - 2/5x_1^4 + 2/5x_1^3 - x_1$ $g_2 = x_1^5 + 3x_1^3$
$f_1 = -1/5x_0^2x_1 + 1/10x_0x_1^5 + 1/5x_0x_1^4 + 1/2x_0x_1^3 - 1/25x_0x_1^2 + 1/10x_0x_1 - 2/5x_0 + x_1^5 - 2/5x_1^4 + 7/5x_1^3 + x_1^2 - 27/25x_1 + 1$ $f_2 = -1/2x_0^2x_1^3 - 1/4x_0^2x_1^2 + 1/4x_0x_1^7 + 5/8x_0x_1^6 + 3/2x_0x_1^5 + 21/40x_0x_1^4 + 1/5x_0x_1^3 - 7/8x_0x_1^2 - 1/2x_0x_1 + x_0 + 5/2x_1^7 + 1/4x_1^6 + 3x_1^5 + 15/4x_1^4 - 49/20x_1^3 - 27/20x_1^2 + 29/20x_1 - 1/2$	$g_1 = x_0 - 1/2x_1^4 - x_1^3 - 5/2x_1^2 + 1/5x_1 - 1/2$ $g_2 = x_1^5 - 3/5x_1^4 + x_1^3 - x_1 + 4/5$

F	G
$f_1 = x_3^5 + 3x_3^3$	$g_1 = x_0 - 2/5x_3^4 + 2/5x_3^3 - x_3$
$f_2 = -2x_0^2x_1x_3 + 4/5x_0x_1x_3^5 - 4/5x_0x_1x_3^4 + 2x_0x_1x_3^2 + 1/2x_0x_2^2x_3^5 + 3/2x_0x_2^2x_3^3$	$g_2 = x_1 + 1/5x_3^4 + 1$
$f_3 = -x_0^2x_3^2 + 7/5x_0x_3^6 - 2/5x_0x_3^5 + 3x_0x_3^4 + x_0x_3^3 + x_1 + 1/5x_3^4 + 1$	$g_3 = x_2 - 5x_3^4 - 2/5x_3^2 + x_3$
$f_4 = x_0 - 2/5x_3^4 + 2/5x_3^3 - x_3$	$g_4 = x_3^5 + 3x_3^3$
$f_5 = -6x_0^3x_1x_2x_3 + 12/5x_0^2x_1x_2x_3^5 - 12/5x_0^2x_1x_2x_3^4 + 6x_0^2x_1x_2x_3^2 + 2/5x_0x_1^2x_3 + 2/25x_0x_1x_3^5 + 2/5x_0x_1x_3 + x_2 - 5x_3^4 - 2/5x_3^2 + x_3$	
$f_1 = x_1x_3^6 + 4/5x_1x_3^5 - 5/4x_1x_3^3$	$g_1 = x_0 - 5/2x_3^4 + 2/3x_3^3 + x_3^2 + 1/2$
$f_2 = -2x_0^3x_2 + 5x_0^2x_2x_3^4 - 4/3x_0^2x_2x_3^3 - 2x_0^2x_2x_3^2 - 4x_0^2x_2 + 15/2x_0x_2x_3^4 - 2x_0x_2x_3^3 - 3x_0x_2x_3^2 - 3/2x_0x_2$	$g_2 = x_1 - 3/2x_3^4 - 1/2x_3^2 + 2/3x_3 - 4/3$
$f_3 = 8x_0^5x_2x_3 - 20x_0^4x_2x_3^5 + 16/3x_0^4x_2x_3^4 + 8x_0^4x_2x_3^3 + 16x_0^4x_2x_3 - 30x_0^3x_2x_3^5 + 8x_0^3x_2x_3^4 + 12x_0^3x_2x_3^3 + 6x_0^3x_2x_3 - 1/4x_0^2x_1^2 + 3/8x_0^2x_1x_3^4 + 1/8x_0^2x_1x_3^2 - 1/6x_0^2x_1x_3 + 1/3x_0^2x_1 + 5/2x_0x_1x_3 - 15/4x_0x_3^5 - 5/4x_0x_3^3 + 5/3x_0x_3^2 - 10/3x_0x_3 + x_2 + 2/3x_3^4 + 5/2x_3^2 + 5x_3$	$g_3 = x_2 + 2/3x_3^4 + 5/2x_3^2 + 5x_3$
$f_4 = 2/3x_0^2x_2 + 5/2x_0x_1^3 - 5/3x_0x_2x_3^4 + 4/9x_0x_2x_3^3 - 10/3x_0x_2x_3^2 + 1/3x_0x_2 - 8/3x_0x_3^6 - 10x_0x_3^4 - 20x_0x_3^3 - 25/4x_1^3x_3^4 + 5/3x_1^3x_3^3 + 5/2x_1^3x_3^2 + 5/4x_1^3 + 2/3x_2^3 + 4/9x_2^2x_3^4 + 5/3x_2^2x_3^2 + 10/3x_2^2x_3 + x_3^5 + 4/5x_3^4 - 5/4x_3^2$	$g_4 = x_3^5 + 4/5x_3^4 - 5/4x_3^2$
$f_5 = x_0 - 5/2x_3^4 + 2/3x_3^3 + x_3^2 + 1/2$	
$f_6 = 1/20x_0^3x_1^4 - 3/40x_0^3x_1^3x_3^4 - 1/40x_0^3x_1^3x_3^2 + 1/30x_0^3x_1^3x_3 - 1/15x_0^3x_1^3 - 1/2x_0^2x_1^3x_3 + 3/8x_0^2x_1^2x_2^3 + 3/4x_0^2x_1^2x_3^5 + 1/4x_0^2x_1^2x_3^3 - 1/3x_0^2x_1^2x_3^2 + 2/3x_0^2x_1^2x_3 - 9/16x_0^2x_1x_2^3x_3^4 - 3/16x_0^2x_1x_2^3x_3^2 + 1/4x_0^2x_1x_2^3x_3 - 1/2x_0^2x_1x_2^3 - 1/5x_0x_1^2x_2 - 2/15x_0x_1^2x_3^4 - 1/2x_0x_1^2x_3^2 - x_0x_1^2x_3 - 15/4x_0x_1x_2^3x_3 + 45/8x_0x_2^3x_3^5 + 15/8x_0x_2^3x_3^3 - 5/2x_0x_2^3x_3^2 + 5x_0x_2^3x_3 + x_1 - 3/2x_2^4 - x_2^3x_3^4 - 15/4x_2^3x_3^2 - 15/2x_2^3x_3 - 3/2x_3^4 - 1/2x_3^2 + 2/3x_3 - 4/3$	

失敗例

G (Ground Truth)	G' (Transformer)
$g_1 = x_0 - 1/2x_3^4 - x_3^3 - 5/2x_3^2 + 1/5x_3 - 1/2$	$g'_1 = x_0 - 1/2x_3^4 - x_3^3 - 5/2x_3^2 + 1/5x_3 - 1/2$
$g_2 = x_1 - 1/2x_3^4 + 5/3x_3^3 - 2/3x_3^2 + 3/5x_3 - 4/3$	$g'_2 = x_1 - 1/2x_3^4 + 5/3x_3^3 - 2/3x_3^2 + 3/5x_3 - 4/3$
$g_3 = x_2 + 5/4x_3^4 - 2x_3^3 - 1/4x_3^2 - 3x_3 - 3$	$g'_3 = x_2 + 5/4x_3^4 - 2x_3^3 - 1/4x_3^2 - 3x_3 - 3$
$g_4 = x_3^5 - 3/5x_3^4 + x_3^3 - x_3 + 4/5$	$g'_4 = x_3^5 - 3/5x_3^4 + 3/2x_3^3 - x_3 + 4/5$
$g_1 = x_0 + 2/3x_3^4 + 2/5x_3^3 + 3x_3^2 + 5/3x_3$	$g'_1 = x_0 + 2/3x_3^4 + 2/5x_3^3 + 2/3x_3^2 + 3/2x_3$
$g_2 = x_1 + 3/4x_3^4 + x_3^2 - 2x_3 + 2/5$	$g'_2 = x_1 + 3/4x_3^4 + x_3^2 - 2x_3 + 2/5$
$g_3 = x_2 - x_3^4 + 1/5x_3^3 + 2/3x_3^2 + 1$	$g'_3 = x_2 - x_3^4 + 1/5x_3^3 + 2/3x_3^2 + 1$
$g_4 = x_3^5 - 4/3x_3^4 + 3x_3^3 - 1/5$	$g'_4 = x_3^5 - 4/3x_3^4 + 3x_3^3 - 1/5$
$g_1 = x_0 - 5/2x_3^4 + 5/3x_3^3 - 4/5x_3^2 - 2x_3 - 1/3$	$g'_1 = x_0 - 5/2x_3^4 + 5/3x_3^3 - 4/5x_3^2 - 2x_3 - 1/3$
$g_2 = x_1 + 1/3x_3^4 - 1/5x_3^3 - 5/2x_3^2 + 4/3$	$g'_2 = x_1 + 1/3x_3^4 - 1/5x_3^3 - 5/2x_3^2 + 4/3$
$g_3 = x_2 - 4x_3^4 - 4/5x_3^3 - 1/2x_3^2 + 1/5x_3 - 5/4$	$g'_3 = x_2 - 4x_3^4 - 2/5x_3^3 - 1/2x_3^2 + 1/5x_3 - 5/4$
$g_4 = x_3^5 - 1/2x_3^3 - 4/3x_3^2 + 4x_3 - 1$	$g'_4 = x_3^5 - 1/2x_3^3 - 4/3x_3^2 + 4x_3 - 1$

グレブナー基底学習の参考論文

先ほどの結果は、2024年12月10-15日にカナダで開催された国際会議「NeurIPS 2024（旧:NIPS）」で発表した論文

“Learning to compute Gröbner basis”

（Kera-Ishihara-Kambe-Vaccon-Yokoyama）

から引用している

NeurIPSとは

- 世界最大規模のAI国際会議
- 昨年の論文投稿数は15,671 件

  採択率は25.8%

NEURAL INFORMATION PROCESSING SYSTEMS

データセット生成理論の拡張

2024年の結果では生成される非グレブナー基底は

元のグレブナー基底よりサイズが大きくないといけない

⇨ 2025年の論文で、理論を拡張

定理（Kera et al, NeurIPS 2025）

<G>が0次元根基イデアルの時、ランダムな行列Aに対し、

Aの行のサイズがGのサイズより大きければ、ほぼ確率1でAGはイデアル<G>の基底になる

可換環論的には…

$k[x_1, \ldots, x_n]$の極大イデアル$\mathcal{M}$からランダムに$n+1$の元をとると

ほぼ確率1でそれは$\mathcal{M}$の基底になる

目次

- 機械学習とデータセット生成の概要
- グレブナー基底について
- グレブナー基底の学習
- 数学計算のための機械学習ライブラリ「CALT」の実演
- まとめ

CALT

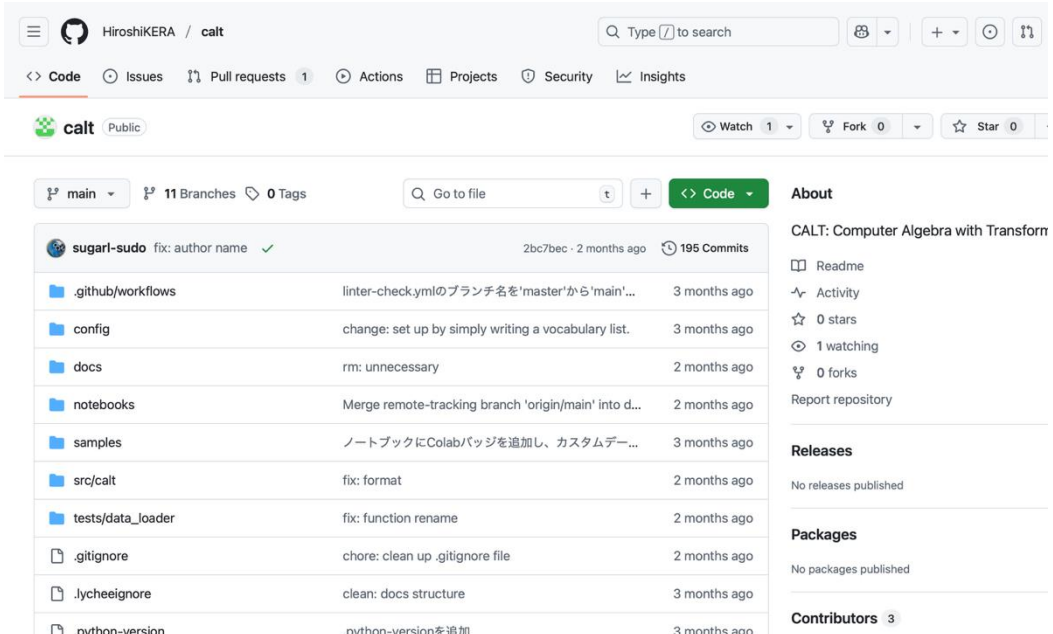
- CALT = 数式処理のための機械学習ライブラリ

  千葉大学の計良-荒川-佐藤によって開発された。

  https://arxiv.org/abs/2506.08600

- 下記のリンクからCALTを利用可能

  https://github.com/HiroshiKERA/calt



CALTで何ができるか?

- CALT はTransformerモデルを用いたPythonライブラリで、数学や記号計算を学習することができる
- 深層学習の専門家ではない人（例えば数学者）でも、データセットがあれば、気軽に学習を試すことができる
- ユーザーに必要なことは、データセット生成器を定義するだけ

（または自前でデータセットを用意するだけ。他の数式処理ソフトでデータを生成してもOK）

- CALTは自動でデータセット生成器を呼び出し、並列に計算することで、膨大なデータセットを生成する。そして、 それらをTransformerで学習する。詳細は下記のリンクから

https://github.com/HiroshiKERA/calt

開発者（計良先生）の推しポイント

- Linux環境ならインストール・セットアップが簡単

  （3行のコマンド・本質は一行だけ）

- データセット生成と学習を分離している

  （数学の人はデータセット生成だけ書けばいい）

- calt-codebaseで実験テンプレートを与えているので,

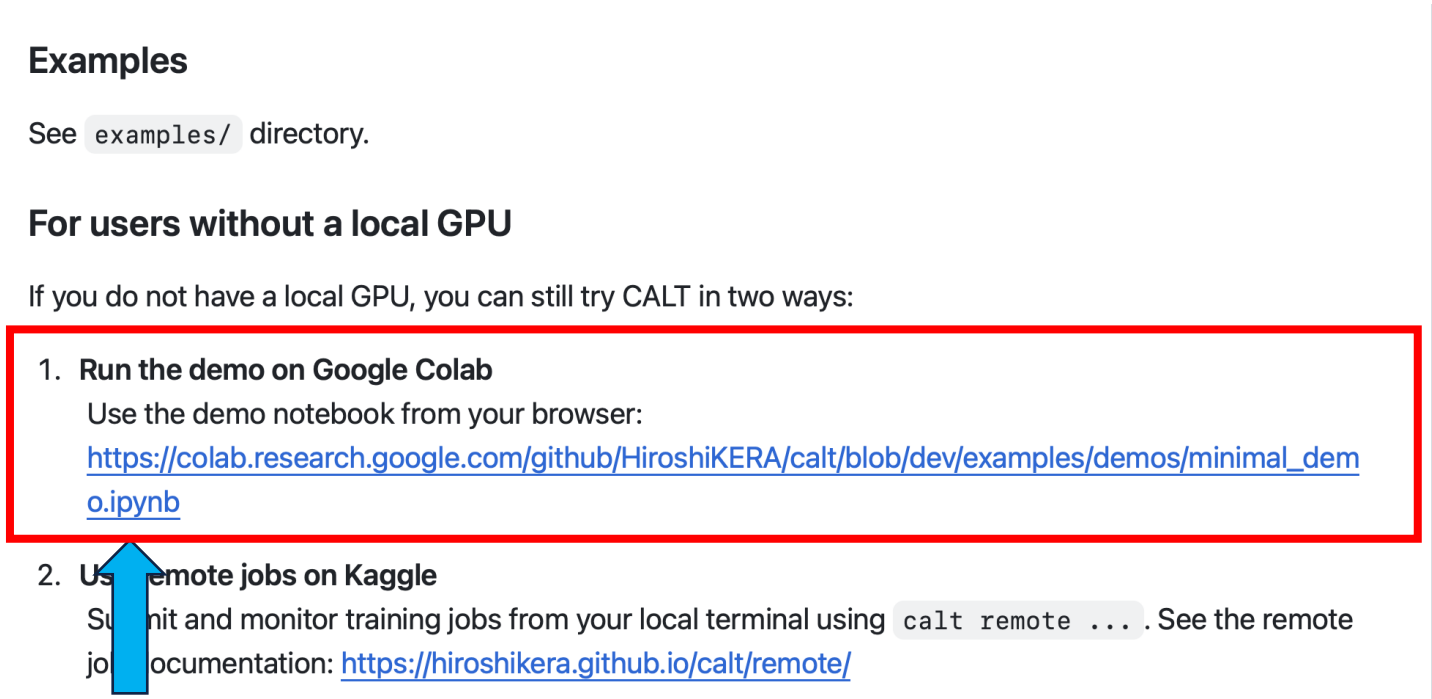
  部分的な書き換えで即実験可能（GPUがあれば）

CALTのドキュメント

https://hiroshikera.github.io/calt-codebase/quickstart/

デモノートブック

- デモノートブックをクリックすると簡単にスタートできる



Examples

See `examples/` directory.

For users without a local GPU

If you do not have a local GPU, you can still try CALT in two ways:

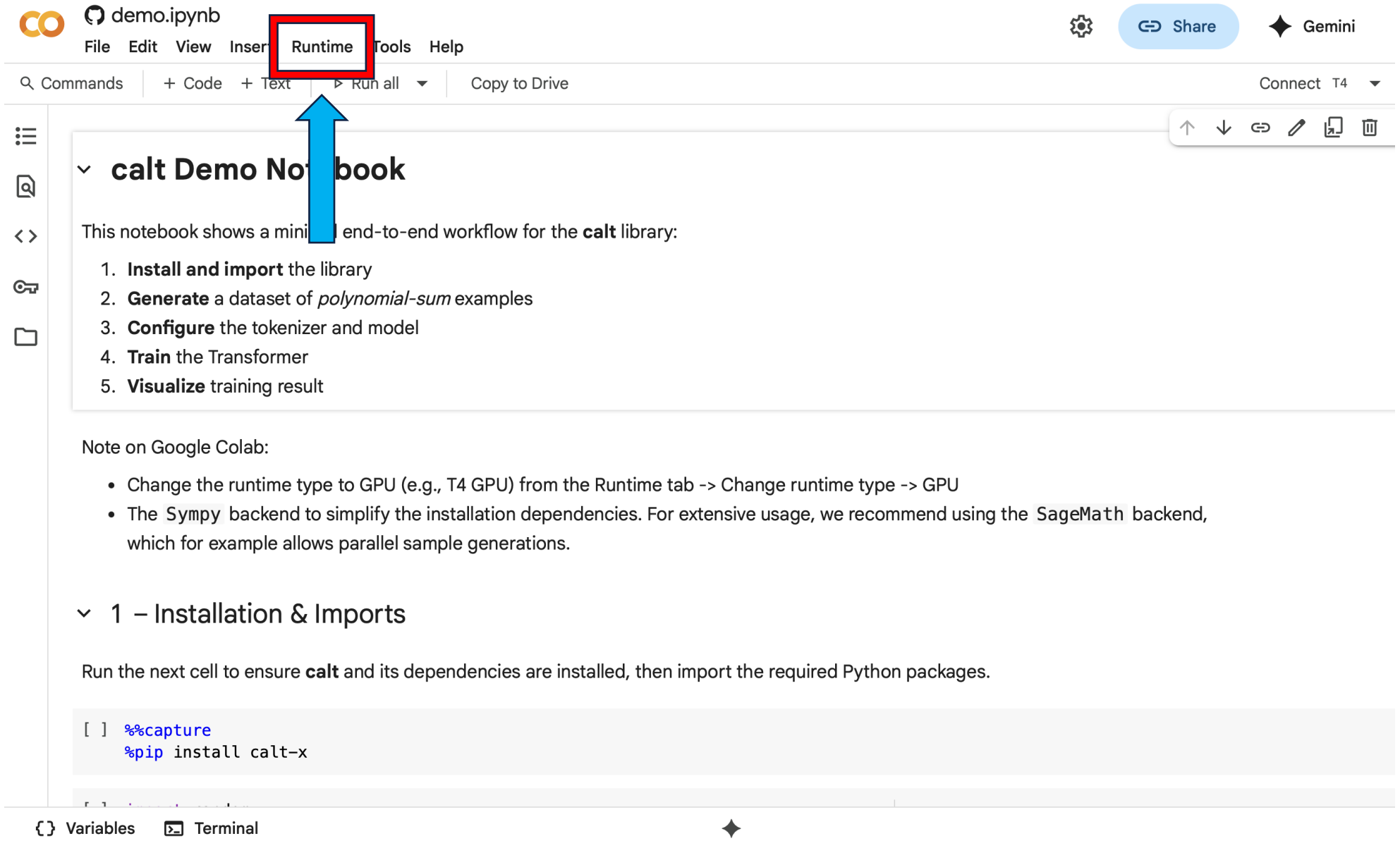
1. **Run the demo on Google Colab**
   Use the demo notebook from your browser:
   https://colab.research.google.com/github/HiroshiKERA/calt/blob/dev/examples/demos/minimal_demo.ipynb

2. **Use remote jobs on Kaggle**
   Submit and monitor training jobs from your local terminal using `calt remote ...`. See the remote job documentation: https://hiroshikera.github.io/calt/remote/

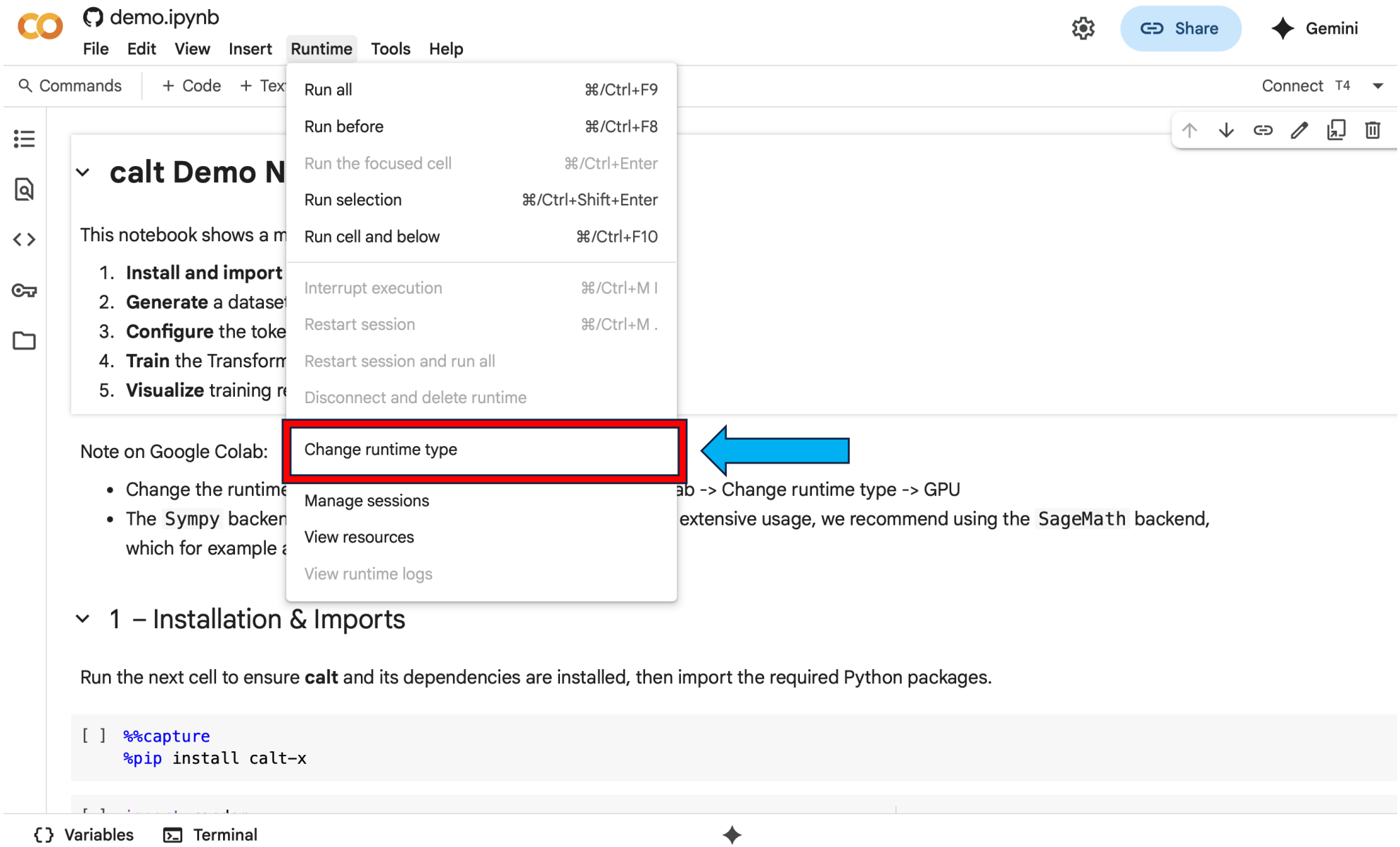
https://colab.research.google.com/github/HiroshiKERA/calt/blob/dev/examples/demos/minimal_demo.ipynb

* 利用にはGoogleアカウントが必要

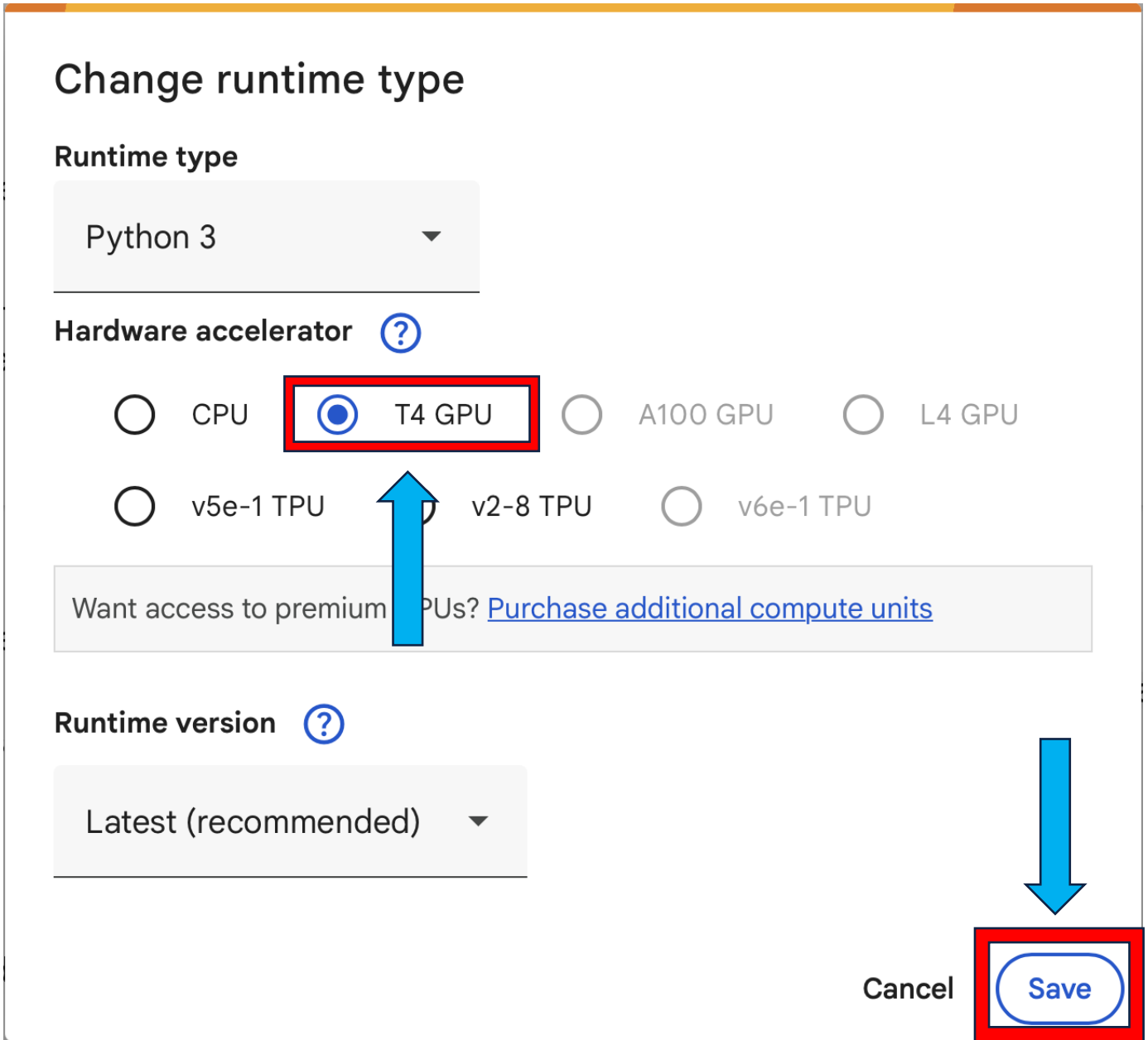
Runtime Type を GPU に変更



Runtime Type を GPU に変更



Runtime Type を GPU に変更



デモをスタート

calt Demo Notebook

This notebook shows a minimal end-to-end workflow for the **calt** library:

1. **Install and import** the library
2. **Generate** a dataset of *polynomial-sum* examples
3. **Configure** the tokenizer and model
4. **Train** the Transformer
5. **Visualize** training result

Step 2のコードを書き換えれば、好きな数学の計算を学習できる

（デモは多項式の和を学習している）

デモをスタート

CALTの大まかな手順

1. ライブラリのインストール
2. データセットの生成
3. モデルの設定
4. Transformerの学習
5. 結果の表示

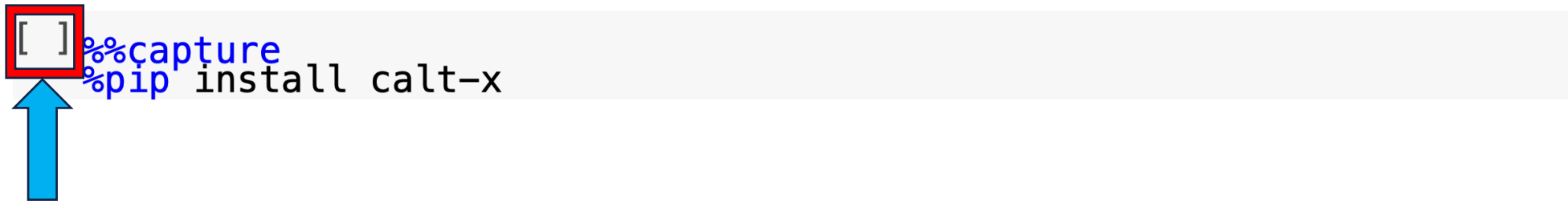
Step 2のコードを書き換えれば、好きな数学の計算を学習できる

（デモは多項式の和を学習している）

ライブラリのインストール

1 – Installation & Imports

Run the next cell to ensure **calt** and its dependencies are installed, then import the required Python packages.

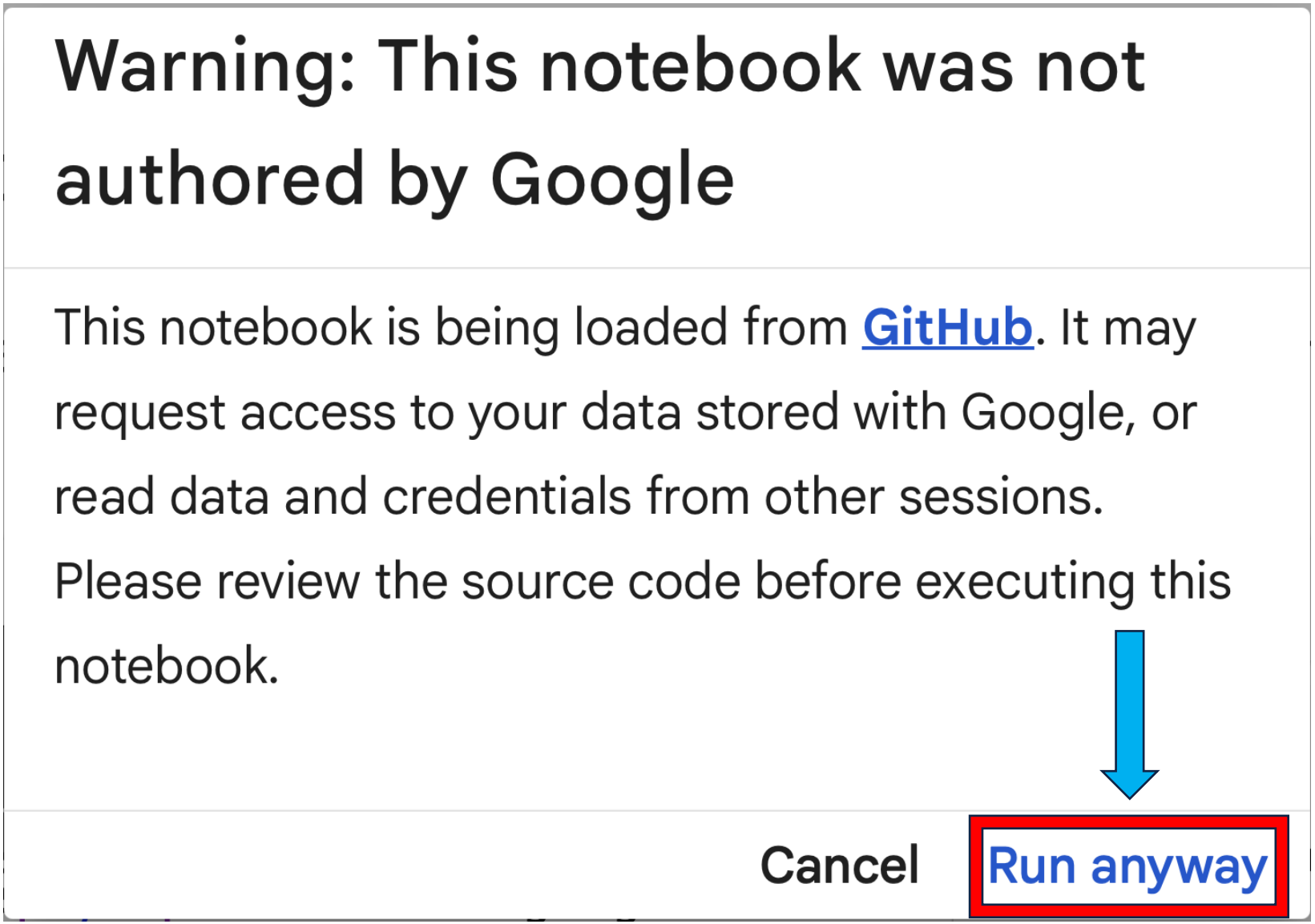


ここをクリックして、calt-x をインストール

もし“Warning: This notebook was not authored by Google”と出た場合には, “Run anyway” をクリック

上から順に、[ ] をクリックしていくだけでOK

ライブラリのインストール



Python パッケージのインポート

```
[ ]  %%capture
     %pip install calt-x
```

```
import random
from sympy.polys.orderings import grevlex
from sympy.polys.rings import PolyElement
from transformers import BartConfig, BartForConditionalGeneration as Transformer
from transformers import TrainingArguments
from calt import (
    Trainer,
    load_data,
)
from calt.dataset_generator.sympy import (
    PolynomialSampler,
    DatasetGenerator,
)
from calt.data_loader.utils import (
    load_eval_results,
    parse_poly,
    display_with_diff
)
```

データセット生成器の定義

2 – Dataset Generation *(Polynomial Addition)*

This cell uses `calt.generator` utilities to create a synthetic dataset of polynomial-addition.

```python
def sum_problem_generator(
    seed: int,
) -> tuple[list[PolyElement], list[PolyElement]]:
    """
    Generate a partial sum problem involving polynomials.

    This function creates problems in which the problem is a list of polynomials F
    and the solution is a list of polynomials G = [g_1, g_2, ..., g_n], where g_i

    Args:
        seed: Seed for random number generator

    Returns:
        Tuple containing (F, G) where F is the problem and G is the solution
    """
    # Set random seed
```

多項式の和の設定（今回のケース）

```
# Initialize polynomial sampler
sampler = PolynomialSampler(
    symbols="x0, x1", # "x, y, z, ... " or "x0, x1, x2, ... "
    field_str="GF(7)", # "QQ", "RR", "ZZ", "GF(p)", "GFp", where p is a prime number
    order="grevlex", # "lex", "grevlex", "grlex", "ilex", "igrevlex", "igrlex"
    max_num_terms=2,
    max_degree=2,
    min_degree=1,
)

# Generate problem polynomials using sampler
F = sampler.sample(num_samples=2)

# Generate solution polynomial g (sum of F)
g = sum(F)

return F, g
```

データセットの生成

```
save_dir = "."

# Initialize dataset generator
dataset_generator = DatasetGenerator(
    backend="multiprocessing",
    n_jobs=-1,
    verbose=False,
    root_seed=100,
)
# Generate training set with batch processing
dataset_generator.run(
    dataset_sizes={"train": 10000, "test": 1000},
    problem_generator=sum_problem_generator,
    save_dir=save_dir,
)
```

データセットの指定

3 – Model Configuration

Here we instantiate the tokenizer, define the Transformer architecture, and prepare the training pipeline.

```
# Point to any dataset you like; here we assume the toy Sum dataset from the data-gene
TRAIN_PATH = "train_raw.txt"
TEST_PATH = "test_raw.txt"
dataset, tokenizer, data_collator = load_data(
    train_dataset_path=TRAIN_PATH,
    test_dataset_path=TEST_PATH,
    field="GF7",
    num_variables=2,
    max_degree=10,  # Should cover the range of generated samples
    max_coeff=7,   # Should cover the range of generated samples
    max_length=256,
)
train_dataset = dataset["train"]
test_dataset = dataset["test"]
```

データセットを別で作ってアップロードすることも可能

モデルの設定

```
[ ]
# Minimal architecture.
model_cfg = BartConfig(
    d_model=256,        # 'width' of the model
    vocab_size=len(tokenizer.vocab),
    encoder_layers=2,  # 'depth' of encoder network
    decoder_layers=2,  # 'depth' of decoder network
    max_position_embeddings=256,  # max length of input/output
    pad_token_id=tokenizer.pad_token_id,
    bos_token_id=tokenizer.bos_token_id,
    eos_token_id=tokenizer.eos_token_id,
    decoder_start_token_id=tokenizer.bos_token_id,
    max_length=256,  # max length of input/output
)
model = Transformer(config=model_cfg)
```

ハイパーパラメータの設定

4 – Training Hyper-parameters

Learning-rate schedule, batch size, number of epochs, and other trainer options are declared in this cell.

```
args = TrainingArguments(
    output_dir="results/",
    num_train_epochs=20,
    logging_steps=50,
    per_device_train_batch_size=int(128),
    per_device_eval_batch_size=int(128),
    save_strategy="no",  # skip checkpoints for the quick demo
    seed=SEED,
    remove_unused_columns=False,
    label_names=["labels"],
    report_to="none",
)
```

学習の開始

5 – Model Training

Launch the training loop. Progress is typically logged to the console (and optionally to Weights & Biases).

```
trainer = Trainer(
    args=args,
    model=model,
    processing_class=tokenizer,
    data_collator=data_collator,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)

# train
results = trainer.train()
trainer.save_model()
metrics = results.metrics

# eval
```

成功率の表示

You can see the success rate of the Transformer learning

1400	0.189100
1450	0.183100
1500	0.183900
1550	0.184300

```
/usr/local/lib/python3.12/dist-packages/transformers/modeling_utils.py:3922: UserWc
  warnings.warn(
```


```
success rate on test set: 62.3 %
```

この場合, Transformerは62.3%の確率で多項式の和に正答

成功例と失敗例の表示

6 – Visualizing Training Results

Finally, we visualize the differences between the mispredicted samples and their correct counterparts.

```
[ ] gen_texts, ref_texts = load_eval_results("results/eval_results.json")

    success_cases = [(i, gen, ref) for i, (gen, ref) in enumerate(zip(gen_texts, ref_text
    failure_cases = [(i, gen, ref) for i, (gen, ref) in enumerate(zip(gen_texts, ref_text

    num_show = 5

    print('------------------------')
    print(''' Success cases ''')
    print('------------------------')
    for (i, gen, ref) in success_cases[:num_show]:
        gen_expr = test_dataset.preprocessor.decode(gen)
        ref_expr = test_dataset.preprocessor.decode(ref)

        print(f"===== sample id: {i+1} =====")
```

成功例と失敗例の表示

成功例

$(4x_0^2 + 5x_1) + (4x_1^2 + 1)$ -> $4x_0^2 + 4x_1^2 + 5x_1 + 1$

失敗例

$(4x_0x_1 + 3x_0) + 2x_0$ -> $4x_0x_1 + \mathbf{3}x_0$
(Ground Truth: $4x_0x_1 + \mathbf{5}x_0$)

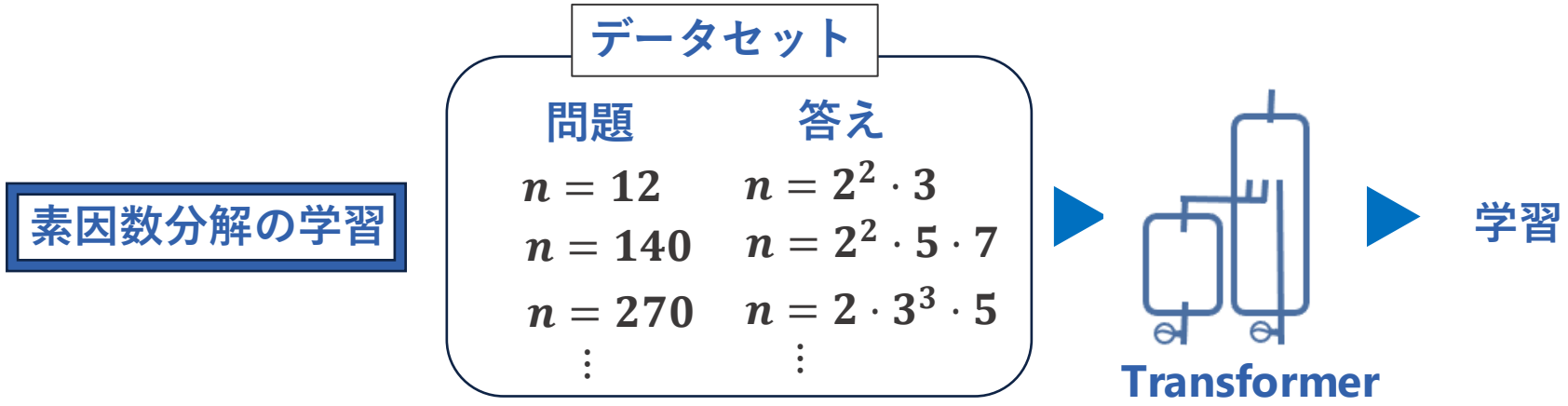
失敗した場合にも間違え方に特徴がある

他の数学の計算の学習

- 整数や有理数の和、積
- 素因数分解
- 多項式の積、因数分解、微分、積分

など…

データセットさえあれば、誰でも気軽に学習することが可能！
（ただし正答率は高いとは限らない）



目次

- 機械学習とデータセット生成の概要
- グレブナー基底について
- グレブナー基底の学習
- 数学計算のための機械学習ライブラリ「CALT」の実演
- まとめ

まとめ

- **Transformer**は近年、数式処理の計算に応用されている
- 学習にはデータセット生成が重要
- 数学的な理論を活用することで、

  データセットを効率良く生成することが可能
- **CALT**を使えば、データセットを作るだけで誰でも気軽に

  Transformerに学習させることが可能