テンソルロジック：AIの言語

ペドロ・ドミンゴス

ワシントン大学 ポール・G・アレン コンピュータ科学・工学部
シアトル、WA 98195-2350、米国

要旨

AIの進歩は、必要な機能をすべて備えたプログラミング言語の不足によって阻害されています。PyTorchやTensorFlowなどのライブラリは、自動微分と効率的なGPU実装を提供していますが、元々AIのために設計されていなかったPythonの追加機能に過ぎません。自動推論と知識獲得のサポートの欠如により、長く費用のかかる無理やりな追加の試みが続いています。一方、LISPやPrologなどのAI言語は、スケーラビリティと学習のサポートが不足しています。本論文は、テンソルロジックを提案しており、これは論理ルールとアインシュタイン和が基本的に同じ操作であること、およびその他すべてがこれらに還元できるという観察に基づいて、ニューラルAIと記号的AIを根本的なレベルで統一することにより、これらの問題を解決します。

テンソルロジックにおける唯一の構成要素はテンソル方程式です。テンソルロジックでトランスフォーマー、形式的推論、カーネル機械、グラフィカルモデルなど、ニューラル、記号的、統計的AIの主要な形式を簡潔に実装する方法を示します。最も重要なことに、テンソルロジックは埋め込み空間での健全な推論のような新しい方向を可能にします。これはニューラルネットワークのスケーラビリティと学習可能性と、記号的推論の信頼性と透明性を組み合わせ、AIのより広範な採用の基礎となる可能性があります。

キーワード： 深層学習、自動推論、知識表現、論理プログラミング、アインシュタイン和、埋め込み、カーネル機械、確率グラフィカルモデル

---

1. 序論

分野は言語を見つけるとき発展します。物理学はニュートンが微積分を発明したときに発展し、それなしには成り立ちませんでした。マクスウェルの方程式はヘビサイドのベクトル計算記号法なしには使用不可能でした。数学者や物理学者が言うように、良い記号法は戦いの半分です。電気工学の多くは複素数なしでは不可能であり、デジタル回路はブール論理なしでは不可能です。現代のチップ設計はハードウェア記述言語により可能になり、データベースは関係代数により、インターネットはインターネットプロトコルにより、ウェブはHTMLにより可能になりました。より一般的には、コンピュータ科学は高水準プログラミング言語なしでは遠くには進まなかったでしょう。定性的分野もその用語に大きく依存しています。芸術家でさえ、彼らのジャンルのイディオムと文体的習慣に頼っています。

分野の言語はその実践者の時間を節約し、彼らの注意を焦点を当て、彼らの思考方法を変えま

す。それは分野をいくつかの共通の方向の周りに統一し、エントロピーを減少させます。重要なことを明白にし、ゼロから何度も解を無理やり組み立てることを避けます。

AIは言語を見つけたでしょうか？LISPは、最初の高水準プログラミング言語の1つであり、記号的AIを可能にしました。1980年代にはPrologも人気になりました。しかし、両方ともスケーラビリティが悪く、学習サポートが不足していました。最終的には、JavaやC++のような汎用言語によってAI内でさえ置き換えられました。グラフィカルモデルは確率的AIの共通言語を提供していますが、推論のコストによって適用可能性が制限されています。マルコフロジックのような形式は記号的AIと確率的AIをシームレスに組み合わせていますが、推論のコストによっても阻害されています。

Pythonは現在AIの事実上の言語ですが、それのために設計されたことはなく、それが明らかです。PyTorchやTensorFlowのようなライブラリは、自動微分やGPU実装のような重要な機能を提供していますが、自動推論や知識獲得のような主要な仕事ではまったく役に立ちません。ニューロシンボリックAIはこれを改善しようとしており、深層学習モジュールを記号的AIモジュールと組み合わせることによって改善しようとしていますが、しばしば両方の欠点を持ってしまいます。要するに、AIは明らかにまだ言語を見つけていません。

そのような言語には明確な要件があります。Pythonと異なり、AIでないすべてを隠し、AIプログラマーが重要なことに焦点を当てることを可能にする必要があります。それはAIシステムに事前知識を組み込み、自動的にそれについて推論することを促進する必要があります。また、自動的に学習を促進する必要があり、結果的なモデルは透明で信頼できるべきです。それは簡単にスケールする必要があります。記号的AIはこれらの特性のいくつかを持ち、深層学習は他のものを持っていますが、どちらもすべてを持っていません。したがって、それらをマージする必要があります。

テンソルロジックはそれらの数学的基礎を統一することによってこれを行います。基本的に本質的にすべてのニューラルネットワークはテンソル代数を使用して構築でき、すべての記号的AIはロジックプログラミングを使用して構築でき、2つは基本的に同等であり、使用される原子データ型によってのみ異なるという観察に基づいています。

本論文はロジックプログラミングとテンソル代数の簡潔なレビューで始まります。論文の中核は、テンソルロジックを定義し、その推論エンジンと学習エンジンを説明しています。その後、テンソルロジックでニューラルネットワーク、記号的AI、カーネル機械、グラフィカルモデルを簡潔に実装する方法を示します。テンソルロジックが埋め込み空間での信頼できる透明な推論を可能にすることを示します。それをスケールアップするための2つのアプローチを提案します。論文は、テンソルロジックの他の潜在的な用途、広範な採用の見通し、その方向への次のステップの議論で結論付けられます。

---

2. 背景

2.1 ロジックプログラミング

記号的AIで最も広く使用されている形式はロジックプログラミングです（Lloyd, 1987）。私たちの目的に十分なロジックプログラミング言語の最も単純なものはDatalogです（Greco and Molinaro, 2016）。Datalogプログラムはルールとファクトのセットです。ファクトは$r(o_1, ..., o_n)$の形式の文です。ここで、rは関係名であり、oは物体名です。たとえば、Parent(Bob, Charlie)はBobがCharlieの親であることを述べ、Ancestor(Alice, Bob)はAliceがBobの祖先であることを述べています。ルールは$A_0 \leftarrow A_1, ..., A_m$の形式の文です。ここで、矢印は「もし」を意味し、コンマは論理積を表し、各Aは$r(x_1, ..., x_n)$の形式を持ち、rは関係名で、xは変数または物体名です。たとえば、ルール

Ancestor(x, y) ← Parent(x, y)

は親が祖先であることを述べ、ルール

Ancestor(x, z) ← Ancestor(x, y), Parent(y, z)

はxがyの祖先であり、yがzの親である場合、xがzの祖先であることを述べています。非形式的には、ルールは、ボディの右側のすべての関係を真にする既知のファクトがある場合、その左側またはヘッドが真であることを述べています。たとえば、上記のルールとファクトはAncestor(Alice, Charlie)が真であることを暗示しています。

データベースの用語では、Datalogルールは一連のジョイン、その後の射影です。2つの関係RとSの（自然）ジョインは、RとSのタプルから形成できるすべてのタプルのセットであり、同じ引数に対して同じ値を持っています。2つの関係が共通の引数を持たない場合、それらのジョインはそれらのデカルト積に減ります。関係Rから引数の部分集合Gへの射影は、Rのタプルから、Gに含まれていないすべての引数を削除することにより得られる関係です。たとえば、ルール

Ancestor(x, z) ← Ancestor(x, y), Parent(y, z)

はAncestor(x, y)とParent(y, z)の関係をyで結合し、結果を{x, z}に射影します。タプルAncestor(Alice, Bob)とParent(Bob, Charlie)はタプルAncestor(Alice, Charlie)を生成します。

ロジックプログラミングで2つの一般的な推論アルゴリズムは前向きチェーンと逆向きチェーンです。前向きチェーンでは、ルールはそれ以上の新しいファクトが導出できなくなるまで、既知のファクトに繰り返し適用されます。結果は演繹的クロージャまたはプログラムのフィックスポイントと呼ばれ、関心のあるすべての質問は単にそれを調べることにより応答できます。たとえば、上記のルールとファクトが与えられたクエリ Ancestor(Alice, x)（「AliceはxのAncestorですか？」）への応答は{Bob, Charlie}です。

逆向きチェーンは、それを一致させるファクトまたはそれをヘッドとして持つルールと本体を一致させるファクトを見つけることにより質問に応答しようとします。以下同様に再帰的に。たとえば、クエリ Ancestor(Alice, Charlie)はどのファクトも一致していませんが、ルール

Ancestor(x, z) ← Ancestor(x, y), Parent(y, z)

と一致し、このルールのボディはファクトAncestor(Alice, Bob)とParent(Bob, Charlie)と一致し、し

たがって答えはTrueです。

Datalogでの前向きと逆向きチェーンは音声推論手順です。つまり、与えられる応答はプログラムのルールとファクトから論理的に従うことが保証されています。ロジックプログラムは宣言的意味論と手続的意味論の両方を持ちます。つまり、ルールは世界についての文と、与えられた引数で本体の手続きを呼び出し、結果を組み合わせることにより、そのヘッドを計算するための手続きの両方として解釈できます。

帰納的ロジックプログラミング（ILP）の分野はデータからロジックプログラムを学習することに関係しています（Lavrač and Dzeroski, 1994）。たとえば、ILPシステムは親と祖先の関係の小さなデータベースから上記のルールを導出するかもしれません。一度導出されると、これらのルールは任意の長さと関係者を含む祖先チェーンについての質問に応答できます。一部のILPシステムはまた述語発明も行うことができます。つまり、データに明示的に表示されない関係を発見することができます。これはニューラルネットワークの隠れた変数に類似しています。

2.2 テンソル代数

テンソルは2つのプロパティにより定義されます：その型（実数、整数、ブール値など）とその形状（Rabanser et al., 2017）。テンソルの形状はそのランク（インデックスの数）とその大きさ（各インデックスに沿った要素の数）で構成されています。たとえば、ビデオはshape(t, x, y, c)の整数テンソルで表現できます。ここで、tはフレームの数で、xとyはフレームのピクセル単位の幅と高さで、cはカラーチャネル数（通常3）です。行列はランク2のテンソルで、ベクトルはランク1のテンソルで、スカラーはランク0のテンソルです。ランクrで、i番目の次元でサイズn_iのテンソルは、合計で$\Pi_{i=1}^{r} n_i$個の要素を含みます。位置i_1に沿った次元1、位置i_aに沿った次元d等のテンソルAの要素は、$A_{i1,\ldots,ia,\ldots,ir}$で表されます。テンソルのこの一般的な要素は、しばしばテンソル自体を表すために使用されます。

同じ形状の2つのテンソルAとBの合計は、テンソルCです。ここで

$$C_{i1,\ldots,ia,\ldots,ir} = A_{i1,\ldots,ia,\ldots,ir} + B_{i1,\ldots,ia,\ldots,ir}$$

2つのテンソルAとBのテンソル積は、それぞれランクrとr'を持つテンソルCで、ランクr + r'であり、

$$C_{i1,\ldots,ia,\ldots,ir,j1,\ldots,ja',\ldots,jr'} = A_{i1,\ldots,ia,\ldots,ir} B_{j1,\ldots,ja',\ldots,jr'}$$

アインシュタイン記法は、すべての合計記号を省略し、繰り返されたインデックスについて暗黙的に合計することにより、テンソル方程式を簡略化します。たとえば、$A_{ij}B_{jk}$はmatrix AとBの積を表し、jについて合計し、インデックスiとkを持つ行列になります：

$$C_{ik} = A_{ij}B_{jk} = \Sigma_j A_{ij}B_{jk}$$

より一般的には、共通インデックスβを持つ2つのテンソルAとBのアインシュタイン和（またはeinsumの短版）はテンソルCです。ここで

$$C_{a\gamma} = \Sigma_\beta A_{a\beta}B_{\beta\gamma}$$

ここで、α、β、γはインデックスのセットで、αはBに表示されないAのインデックスの部分集合で、αとβの要素は任意の順序で交錯する可能性があり、同様にBとγについてです。本質的に、ニューラルネットワークのすべての線形および多重線形操作はeinsumとして簡潔に表現できます（Rocktäschel, 2018; Rogozhnikov, 2022）。

行列のように、テンソルはより小さいテンソルの積に分解できます。特に、Tucker分解はテンソルをランクr同じの、より小さいコアテンソルとk個の因子行列に分解し、各々がコアテンソルのインデックスを元のインデックスに展開します。たとえば、AがランクR-3テンソルの場合、アインシュタイン記法では、その Tucker分解は

$$A_{i\,j\,k} = M_{ip}M'_{\,j\,e}M''_{kr}C_{per}$$

ここで、Cはコアテンソルで、Mは因子行列です。

---

3. テンソルロジック

3.1 表現

テンソルロジックは2つの主要な質問への応答に基づいています：テンソルと関係の関係は何ですか？そしてDatalogルールとeinsumsの関係は何ですか？

最初の質問への応答は、関係は疎なブール値テンソルのコンパクト表現であるということです。たとえば、ソーシャルネットワークはneighborhoodマトリックス$M_{i\,j}$で表現でき、iとjは個人の範囲であり、$M_{i\,j} = 1$（iとjが隣人である場合）、0（そうでない場合）。しかし、大規模なネットワークの場合、これはほぼすべての要素が0になるため、非効率的な表現です。ネットワークはより小さくコンパクトに表現できます。つまり、各隣人ペアに対して1つのタプルを持つ関係として表現できます。隣人ペアのペアは関係に含まれていないと見なされ、隣人ではありません。より一般的には、ランクnの疎なブール値テンソルは、各ゼロ以外の要素に対して1つのタプルを持つn項関係でコンパクトに表現でき、効率向上はnが増加するにつれてエクスポーネンシャルに増加します。

2番目の質問への応答は、Datalogルールはブール値テンソルに対するeinsumで、結果に段階的に要素ごとに適用されたものです。（特にはHeavisideステップ関数、$H(x) = 1$（$x > 0$の場合）、0（それ以外の場合））。たとえば、ルール

$$\text{Aunt}(x, z) \leftarrow \text{Sister}(x, y), \text{Parent}(y, z)$$

を検討してください。関係Aunt(x, z)、Sister(x, y)、Parent(y, z)をそれぞれブール値マトリックスA_{xk}、$S_{x\gamma}$、$P_{\gamma k}$として見ると、

$$A_{xk} = H(S_{x\gamma}P_{\gamma k}) = H(\Sigma_{\gamma}\, S_{x\gamma}P_{\gamma k})$$

は、$S_{x\gamma}$と$P_{\gamma k}$がいくつかのyに対して両方とも1である場合にのみ1になります。言い換えれば、einsum$S_{x\gamma}P_{\gamma k}$はSister(x, y)とParent(y, z)のジョインを実装します。xがzの叔母である場合、yはxの兄

弟姉妹であり、同時にzの親です。ステップ関数は必要です。なぜなら、一般に与えられた(x, z)ペアに対して、複数のyが$S_{x\gamma} = P_{\gamma k} = 1$を満たす可能性があり、結果が1より大きくなり、ステップ関数がこれを1に減らすためです。

UとVを任意のテンソルとし、α、β、γをインデックスのセットとします。その場合、$T_{a\gamma} = H(U_{a\beta}V_{\beta\gamma})$はブール値テンソルです。その要素は、いくつかのβに対して$U_{a\beta}V_{\beta\gamma} = 1$である場合、インデックスαγ付きで1です。言い換えると、TはUおよびVに対応する関係のジョインを表します。

テンソルと関係の間に直接対応があり、einsumsとDatalogルールの間に直接対応があるため、データベースのジョインと射影に直接対応するテンソル操作も必要があります。したがって、テンソル射影とテンソルジョインを次のように定義します。

テンソルTのインデックスのサブセットαへの射影は

$\pi_a(T) = \Sigma_\beta T_{a\beta}$

ここで、βはαに含まれていないTのインデックスのセットです。（βの要素はαの要素と交錯する可能性があります。）言い換えると、Tのαへの射影は、αの各値に対するTのすべての要素の合計です。たとえば、ベクトルはスカラーに射影できます。スカラーはすべての要素を合計することにより、行列を列ベクトルに射影でき、各行を要素に集計できます。立方体テンソルはその面のいずれかに射影でき、その後その面をそのエッジのいずれかに射影でき、その後コーナーに射影できます等。テンソルがブール値である場合、射影がステップ関数に続く場合、テンソル射影はデータベース射影に減ります。

2つのテンソルUとVの共通インデックスセットβに沿ったジョインは

$(U \rtimes\ltimes V)_{a\beta\gamma} = U_{a\beta}V_{\beta\gamma}$

ここで、αはVに含まれないUの次元のサブセットで、同様にγはVのサブセットです。（繰り返しになりますが、α、β、γは任意の順序で交錯する可能性があります。）言い換えると、共通インデックスセットβに沿った2つのテンソルのジョインは、βの同じ値を持つ要素のペアごとに1つの要素を持ち、その要素はそれらの積です。Uのランクがr、Vのランクがr'、$|\beta| = q$の場合、$U \rtimes\ltimes V$のランクは$r + r' - q$です。2つのテンソルが共通のインデックスを持たない場合、それらのジョインはそれらのテンソル積（行列のクロネッカー積）に減ります。すべての次元が共通の場合、それは要素ごとの積（行列のアダマール積）に減ります。テンソルがブール値の場合、テンソルジョインはデータベースジョインに減ります。

テンソルロジックプログラムはテンソル方程式のセットです。テンソル方程式の左側（LHS）は計算されているテンソルです。右側（RHS）は一連のテンソルジョインの後に、テンソル射影であり、オプションで結果に要素ごとに適用された単変量の非線形性があります。テンソルはテンソル名の後に、コンマ区切りのインデックスリストで表記され、角括弧で囲まれています。ジョインサインは左側に暗黙のままであり、射影はLHSのインデックスに行われます。たとえば、単一層のパーセプトロンはテンソル方程式で実装されています

Y = step(W[i] X[i])

ここで、iのジョインと射影は、WとXの内積を実装しています。テンソルは、W = [0.2, 1.9, -0.7, 3]およびX = [0, 1, 1, 0]のように、要素を列挙することで指定することもできます。その後、Y?を入力すると、Yが評価されます。

NumPy、PyTorch等の einsum実装と同様に、テンソル方程式は元のアインシュタイン記法よりも一般的です：合計されたインデックスはLHSに表示されないもので、したがって繰り返されたインデックスは合計される場合と合計されない場合があります。たとえば、インデックスiは

Y[i] = step(W[i] X[i])

は合計されていません。多層パーセプトロンの実装は以下を利用します。

テンソル要素はデフォルトで0であり、同じLHSを持つ方程式は暗黙的に合計されます。これはロジックプログラミングとの対応を保ち、テンソルロジックプログラムをより短くします。テンソル型は宣言またはinferredされる可能性があります。テンソルをファイルに設定すると、ファイルはテンソルに読み込まれます。テキストファイルを読み込むと、ブール値マトリックスが生成されます。このマトリックスの ijth要素は、テキスト内のi番目の位置に語彙のj番目の単語が含まれている場合は1です。（マトリックスはもちろんこの非効率的なフォームに保存されていません。これについては後で詳しく説明します。）たとえば、ファイルが文字列「Alice loves Bob」である場合、それがマトリックスMに読み込まれた場合、結果はM[0, Alice] = M[1, loves] = M[2, Bob] = 1で、他のすべてのi、jについてM[i, j] = 0です。（整数だけでなく、任意の定数をインデックスとして使用できることに注意してください。）逆に、テンソルをファイルに等しく設定すると、テンソルはファイルに書き込まれます。

これがテンソルロジックの全体的な定義です。キーワードなどはありません。ただし、表現の充実度を増やさない一方で、通常のプログラムを書くことをより便利にする構文糖を許可することは便利です。たとえば、次を許可できます：1つの方程式に複数の用語（例えば、Y = step(W[i] X[i] + C)）、インデックス関数（例えば、X[i, t+1] = W[i, j] X[j, t]）、正規化（例えば、Y[i] = softmax(X[i])）、他のテンソル関数（例えば、Y[k] = concat(X[i, j])）、代替射影演算子（例えば、max=またはavg=の代わりに+=、デフォルトはto=）、スライス（例えば、X[4:8]）、手続き型アタッチメント（定義済みまたは外部定義関数）。テンソルロジックはDatalog構文を受け入れます。括弧ではなく角括弧を含むテンソルで表記することは、それがブール値であることを暗示しています。特に、疎なブール値テンソルは、より小さくコンパクトに、ファクトのセットとして書き込まれる可能性があります。たとえば、ベクトルX = [0, 1, 1, 0]はX(1)、X(2)と書き込むこともでき、X(0)とX(3)は暗黙的に0です。同様に、文字列「Alice loves Bob」をマトリックスMに読み込むと、ファクトM(0, Alice)、M(1, loves)、M(2, Bob)が生成されます。

別の簡単な例として、多層パーセプトロンは方程式で実装できます

X[i, j] = sig(W[i, j, k] X[i-1, k])

ここで、iはレイヤー上の範囲で、jとkはユニット上の範囲です。 sig()はシグモイド関数です。異なるレイヤーは異なるサイズである可能性があります（対応する重みマトリックスは全テンソル

を構成するために暗黙的にゼロでパッドされます）。あるいは、各レイヤーに異なる方程式を使用する可能性があります。

基本的なリカレントニューラルネットワーク（RNN）は、

X[i, *t+1] = sig(W[i, j] X[j, *t] + V[i, j] U[j, t])

で実装でき、ここで、Xは状態、Uは入力、iとjはユニット上の範囲、tは時間ステップ上の範囲です。*t表記は、tが仮想インデックスであることを示します。メモリはそれに割り当てられず、連続するX[i]ベクトルは同じ場所に書き込まれます。RNNはTuring-complete（Siegelmann and Sontag, 1995）であるため、上記の実装はテンソルロジックも同様であることを示唆しています。

3.2 推論

テンソルロジックの推論は、前向きと逆向きチェーンのテンソル一般化を使用して実行されます。

前向きチェーンでは、テンソルロジックプログラムは線形コードとして扱われます。テンソル方程式は順番に実行され、各方程式は必要な入力が利用可能な要素を計算します。これは新しい要素がもう計算できなくなるまで、または停止基準が満たされるまで繰り返されます。

逆向きチェーンでは、各テンソル方程式は関数として扱われます。クエリはトップレベルコールであり、各方程式はすべての関連要素が数値またはデータで利用可能になるまで、RHSのテンソルの方程式を呼び出します。または、サブクエリ用の方程式がない場合、サブクエリ要素はデフォルトで0に割り当てられます。

前向きまたは逆向きチェーンを使用するかどうかの選択はアプリケーションに依存します。

3.3 学習

テンソルロジックには1つのタイプのステートメント（テンソル方程式）のみがあるため、テンソルロジックプログラムを自動的に微分することは特に簡単です。単変量の非線形性を除いて、テンソル方程式のLHSの導関数に関するRHS上のテンソルに関しては、RHS上の他のテンソルの積です。より正確に、

$Y[...] = T[...] X_1[...] ... X_n[...]$

の場合、

$\partial Y[...]/\partial T[...] = X_1[...] ... X_n[...]$

このケースの特例としては、Y = AXの場合∂Y/∂X = A。Y = W[i] X[i]の場合∂Y/∂W[i] = X[i]。Y[i, j] = M[i, k] X[k, j]の場合∂Y[i, j]/∂M[i, k] = X[k, j]。

その結果、テンソルロジックプログラムの勾配もテンソルロジックプログラムであり、方程式ごとに1つの方程式でRHS上のテンソルです。簡潔さのため、インデックスを省略すると、損失Lのテンソルに関する導関数Tは

$\partial L/\partial T = \Sigma_e (dL/dY)(dY/dU)\Pi_{uu}H_u T\ X$

ここで、EはそのRHSSがTに表示される方程式です。Yは方程式のLHSです。Uはその非線形性の引数です。Xはユニット内のテンソルです。

テンソルロジックプログラムを学習するには、1つ以上のテンソル方程式を使用して、損失関数と適用されるテンソルを指定する必要があります。たとえば、最後のレイヤーの出力に対する二乗損失を最小化することにより、MLPを学習するには、方程式を使用できます

$Loss = (Y[e] - X[*e, N, j])^2$

ここで、eは訓練例上の範囲、jはユニット上の範囲、Yはターゲット値を含み、Xは上で定義されたMLPであり、例用の仮想インデックスで拡張され、Nはレイヤーの数です。デフォルトでは、訓練データとして提供されないすべてのテンソルが学習されますが、ユーザーは定数（例えば、ハイパーパラメータ）のままであるべきであるかを指定できます。オプティマイザー自体はテンソルロジックでエンコードできますが、通常は事前に供給されたものが使用されます。

伝統的なニューラルネットワークのバックプロパゲーションは、すべての訓練例に対して同じアーキテクチャに適用されますが、テンソルロジックでは、異なる訓練例に適用される異なる方程式の構造が異なる可能性があり、例のすべての可能な導出の和を介してバックプロパゲーションすることは無駄です。幸いなことに、このの問題には、すでに構造を通じてのバックプロパゲーションの形式で利用可能なソリューションがあります。これは、各例について、例の導出に表示されるたびに各方程式のパラメータを1回更新します（Goller and Kücler, 1996）。これをRNNに適用すると、時間を通じてのバックプロパゲーション（Werbos, 1990）の特殊な場合が得られます。

固定セットの方程式で構成されるテンソルロジックプログラムを学習することは非常に柔軟であり、方程式が同じジョイン構造を持つすべてのルールのセットを表現できるためです。（例えば、MLPはサブセットルールのいかなるセットも表現できます。）さらに、テンソルロジックでのテンソル分解は述語発明の一般化です。たとえば、学習されるプログラムが方程式

$A[i, j, k] = M[i, p]\ M'[j, q]\ M''[k, r]\ C[p, q, r]$

である場合、Aが唯一のデータテンソルの場合、学習されたM、M'、M''、Cはネットワークのランクのセットの因子行列となっています。それらをブール値にするとしきい値を設定すると、それらは発明された述語に変わります。

---

4. AIパラダイムの実装

以下の実装は、特に指定されない限り前向きチェーンを使用します。

4.1 ニューラルネットワーク

畳み込みニューラルネットワークは、畳み込みおよびプーリングレイヤーを持つMLP（LeCun et

al., 1998）です。畳み込みレイヤーは画像内のすべての場所にフィルタを適用し、フォームのテンソル方程式で実装できます

Features[x, y] = relu(Filter[dx, dy, ch] Image[x+dx, y+dy, ch])

ここで、xとyはピクセル座標であり、dxとdyはフィルタ座標で、chはRGBチャネルです。プーリングレイヤーは、近くのフィルタのブロックをまとめて、

Pooled[x/S, y/S] = Features[x, y]

で実装でき、ここで/は整数除算で、Sはストライドです。これは、フィルタ出力をS連続位置が各次元にまとめられています。これは合計プーリングを実装しており、最大プーリングは=を max=などに置き換えます。畳み込みおよびプーリングレイヤーは、方程式 Pooled[x/S, y/S] = relu(...)で1つに結合できます。

グラフニューラルネットワーク（GNN）は、グラフ構造化データ（ソーシャルネットワーク、分子、代謝ネットワーク、Web）に深層学習を適用します（Liu and Zhou, 2022）。Table 1は、簡潔なGNNの実装を示します。ネットワークのグラフ構造はNeig(x, y)関係で定義され、各隣接(x, y)ペアに対して1つのファクトがあります。または、ブール値テンソル Neig[x, y] = 1（xとxが隣接し、0 その他の場合）で同等に定義されます。主なテンソルはEmb[n, l, d]で、各層lの各ノードnのd次元埋め込みを含みます。初期化は各ノードの0階層の埋め込みをその特性に設定します（外部定義または学習）。ネットワークは、レイヤーごとに1つ、L回のメッセージパッシング反復を実行します。各反復は1つ以上のパーセプトロンレイヤーを各ノードに適用することで開始します。（Table 1は1つを表示します。順列不変性を保つため、重みWPはノードに依存しません。テンソルロジックにはサブ/スーパースクリプトがありませんが、簡潔さのためここで使用します。）GNNは、テンソル Neig(n, n')とZ[n', l, d]のジョインにより、各ノードの隣接データのZを集計します。各ノードについて、これはすべての非隣接のコントリビューションをゼロアウトし、結果は隣接の特性の合計です。（内部的には、ノードの隣接に対して反復することによって、または他の方法によって効率的に行うことができます。セクション6を参照してください。）集計された特性は、その後、別のMLP（表示されていない）を通じて渡される可能性があり、その後、重みWAggおよびWSelfを使用してノード機能と組み合わせ、次のレイヤーの埋め込みを生成します。

GNNの最も一般的な適用は、ノード分類、エッジ予測、グラフ分類です。2クラス問題の場合、各ノードは最終埋め込みの内積を重みベクトルで行い、結果をシグモイドを通して、クラス確率を得ることで分類されます。マルチクラス問題（表示されていない）の場合、各ノードの最終埋め込みは各クラスcの重みベクトルWOut[c, d]で内積を実行し、ロジットベクトルを生成し、その後softmaxを通じてクラス確率Y[n, c]を生成します。エッジ予測は、各ノードペアの間にエッジがあるかどうかを予測し、埋め込みを内積してシグモイドを通します。グラフ分類は、グラフ全体のクラス予測を生成し、ノード分類と同等で、結果がベクトルY[n]ではなくスカラーYです。

大規模言語モデルの基礎である注目は、テンソルロジックでも実装する方法が直接的です（Vaswani et al., 2017）。埋め込みマトリックスX[p, d]が与えられ、pはテキスト内のアイテム（位置など）に及び、dは埋め込み次元を超えます。クエリ、キー、値マトリックスはXに対応する重みマトリックスを掛けることで得られます：

みマトリックスを掛けることで得られます：

Query[p, dk] = WQ[dk, d] X[p, d]
Key[p, dk] = WK[dk, d] X[p, d]
Val[p, dv] = WV[dv, d] X[p, d]

注目は2つのステップで計算でき、最初のステップは各位置のクエリを各キーと比較します：

Comp[p, p'.] = softmax(Query[p, dk]) Key[p', dk] / sqrt(Dk))

ここで、sqrt(Dk)は内積をキーの次元の平方根でスケーリングします。表記法p'. は、pを正規化するインデックスを示します（各pについて、softmaxはp'でインデックス付けされたベクトルに適用されます）。注目ヘッドはその後、対応する比較により重み付けされた値ベクトルの合計を返します：

Attn[p, dv] = Comp[p, p'] Val[p', dv]

テンソルロジックで実装した完全なトランスフォーマーを作成できます（Table 2）。Subsection 3.1で見たように、テキストはX(p, t)の関係で表現できます。これはテキストのp番目の位置がt番目のトークンを含むことを述べています。（トークン化ルールはDatalogで簡単に表現でき、表示されていません。）テキストの埋め込みEmbX[p, d]は、その後、X(p, t)に埋め込みマトリックスEmb[t, d]を掛けることで得られます。次の方程式は元の論文のように位置エンコーディングを実装しており（Vaswani et al., 2017）、他のオプションは可能です。（ちなみに、この方程式はまた、テンソルロジックで条件とケースステートメントがどのように実装できるかも示します：各式を対応する条件に結合することによって。）残差ストリームは、その後、テキストの埋め込みと位置エンコーディングの合計で初期化されます。

注目は上で説明したように実装され、各テンソルに対する2つの追加インデックスがあります：注目ブロック用のb、および注目ヘッド用のh。注目ヘッドの出力は、その後連結され、残差ストリームに追加され、レイヤー正規化されます。MLP層は前述のように実装されており、ブロック用の追加インデックスがあり、位置は、その出力も正規化されます。（表示されていない場合でも、ストリームに追加されます。）最後に、出力（トークン確率）は、各トークンのストリームを出力重みベクトルで内積しており、softmaxを通じて取得されます。

4.2 記号的AI

Datalogプログラムは有効なテンソルロジックプログラムです。したがって、Datalogで行うことができるすべてはテンソルロジックで行うことができます。これは記号的なシステムを実装するのに十分です。関数空きドメインのレーズニングおよび計画を含みます。関数の適応（Prologのように）には、テンソルロジックで統一を実装する必要があります（Lloyd, 1987）。

4.3 カーネル機械

カーネル機械は方程式で実装できます

Y[Q] = f(A[i] Y[i] K[Q, i] + B)

ここで、Qはクエリの例であり、iはサポートベクトルに及び、f()は出力非線形性です（例えば、

シグモイド）（Schölkopf and Smola, 2002）。カーネルKはその後、それ自体の方程式によって実装されます。たとえば、多項式カーネルは

$K[i, i'] = (X[i, j] X[i', j])^n$

で、ここでiとi'は例に及び、jは特性に及び、nは多項式の次数です。ガウスカーネルは

$K[i, i'] = \exp(-(X[i, j] - X[i', j])^2 / Var)$

です。

（より正確に、KはExamplesに関するカーネルのグラム行列です。）構造化予測、出力が複数の相互関連要素で構成される場合（Bakr et al., 2007）、出力ベクトルY[Q, k]により実装でき、出力と入力の間の相互作用を述べる方程式があります。

4.4 確率グラフィカルモデル

グラフィカルモデルは、ランダム変数のセットの同時確率分布を、因子の正規化された積として表します。

$P(X = x) = (1/Z) \Pi_k \phi_k(x\{k\})$

ここで、各因子ϕ_kは変数xのサブセット{k}の非負関数であり、$Z = \Sigma_x \Pi_k \phi_k(x\{k\})$です（Koller and Friedman, 2009）。各因子が変数の親（いくつかの部分的な順序付けの先行）が与えられた場合の条件付き確率である場合、モデルはベイジアンネットワークであり、Z = 1です。

Table 3は、離散グラフィカルモデルの構成と操作がテンソルロジックのものに直接どのようにマッピングされるかを示しています。因子は非負の実数値のテンソルであり、変数ごとに1つのインデックスと、変数の各値に対して1つの値があります。状態xの正規化されていない確率は、xに対応する各テンソル内の要素の積です。ベイジアンネットワークは、したがって、変数ごとに1つの方程式を使用してテンソルロジックでエンコードでき、条件付き確率テーブル（CPT）と親の分布の観点から変数の分布を述べています：

$PX[x] = CPTX[x, par1, ..., parn] P1[par1] ... Pn[parn]$

グラフィカルモデルの推論は、周辺および条件付き確率の計算であり、2つの操作の組み合わせで構成されます。周辺化とポイントワイズ積。因子ϕでY変数のサブセットの周辺化は、それらを合計し、残りの変数Xに対する因子を残します：

$\phi'(X) = \Sigma Y \phi(X, Y)$

周辺化はテンソル射影です。2つのポテンシャル間のポイントワイズ積はXおよびYの変数のサブセット上の上のXおよびYの上のポテンシャルで、それらを$X \cup Y$の上のポテンシャルに組み合わせ、対応するテンソルのジョインです。

すべてのグラフィカルモデルは結合ツリーで表現できます。結合ツリーは、各因子が元のモデル

の因子のジョインである因子のツリーです。すべての周辺および条件付きクエリは、因子を順序

立てて周辺化し、親の因子でそれらをポイント方向に掛けることにより、ツリーのサイズの線形時間で応答できます。結合ツリーはツリーのようなテンソルロジックプログラムです。つまり、テンソルが複数のRHSに表示されないテンソルロジックプログラムです。その結果、線形時間推論は、このプログラムに対して逆向きチェーンで実施できます。具体的には、分割関数Zは、方程式Z = T[...]をプログラムに追加することにより計算でき。ここで、T[...]はルート因子の方程式のLHS、クエリZであり、テキストの周辺確率P(E)はEをプログラムに一連のファクトとして追加し、クエリZ、元のZを除算し、クエリが与えられた条件付き確率はP(E) = P(Q, E) / P(E)として計算できます。

しかし、結合ツリーは元のモデルより指数関数的に大きく、近似推論の必要性が生じます。2つの最も人気のある方法はループの信念伝播とモンテカルロサンプリングです。ループの信念伝播は、モデルを表すテンソルロジックプログラムの前向きチェーンです。サンプリングは、逆向きチェーンで選択的射影で実装できます（つまり、射影をその用語のランダムサブセットに置き換えます）。

---

5. 埋め込み空間での推論

テンソルロジックの最も興味深い機能は、それが示す新しいモデルです。このセクションでは、埋め込み空間で知識表現と推論を実行する方法を示し、このアプローチの信頼性と透明性を指摘します。

最初に、オブジェクトの埋め込みがランダムユニットベクトルである場合を検討してください。埋め込みはマトリックスEmb[x, d]に保存でき、xはオブジェクトに及び、dは埋め込み次元を超えます。Emb[x, d]にワンホットベクトルV[x]を掛けると、対応するオブジェクトの埋め込みが取得されます。V[x]がセットを表すマルチホットベクトルである場合、

S[d] = V[x] Emb[x, d]

はセット内のオブジェクトの埋め込みの重ね合わせです。内積

D[A] = S[d] Emb[A, d]

は、オブジェクトA用で、Aがセット内にある場合、およそ1であり、そうでない場合は0です（標準偏差√(N/D)で。ここでNはセットのカーディナリティで、Dは埋め込み次元です）。これを1/2でしきい値に設定すると、AがセットのESでエラー確率が埋め込み次元とともに減少して、セット内であるかどうかを教えてくれます。これはブルームフィルター（Bloom, 1970）に似ています。

同じスキームは関係の埋め込みに拡張できます。簡潔さのため、バイナリ関係R(x, y)を検討してください。その場合、

EmbR[i, j] = R(x, y) Emb[x, i] Emb[y, j]

は関係内のタプルの埋め込みの重ね合わせです。ここで、タプルの埋め込みはその引数の埋め込みのテンソル積です。これはテンソル積表現のタイプです（Smolensky, 1990）。|R|で線形時間で計算できます。対応するテンソル積を結果に追加することにより、タプルを反復処理することによってです。方程式

D[A, B] = EmbR[i, j] Emb[A, i] Emb[B, j]

はR(A, B)を取得します。つまり、D[A, B]は、タプル(A, B)が関係にある場合、およそ1であり、そうでない場合は0です。理由は

D[A, B] = EmbR[i, j] Emb[A, i] Emb[B, j]
= (R(x, y) Emb[x, i] Emb[y, j]) Emb[A, i] Emb[B, j]
= R(x, y) (Emb[x, i] Emb[A, i]) (Emb[y, j] Emb[B, j])
≈ R(A, B)

最後の1つ前のステップは、einsumが可換で結合的であるという理由で有効です。（特に、結果はテンソルが表示される順序に依存しません。ただしインデックス構造についてのみです。）最後のステップは、2つのランダムユニットベクトルの内積がおよそ0であるという理由で有効です。

同じ理由で、方程式

D[A, y] = EmbR[i, j] Emb[A, i] Emb[y, j]

はAが関係Rにある対象の埋め込みの重ね合わせを返し、

D[x, y] = EmbR[i, j] Emb[x, i] Emb[y, j]

は関係全体R(x, y)を返します。EmbR[i, j]、Emb[x, i]、Emb[y, j]はデータテンソルD[x, y]のTucker分解を形成し、EmbR[i, j]はコアテンソルで、Emb[x, i]とEmb[y, j]は因子行列です。

関係シンボル自体が埋め込まれる可能性があります。（例えば、R、A、B内のR(A, B)が埋め込まれます。）これはランク3テンソルになります。任意のアリティ関係は（関係、引数、値）トリプルのセットに削減できます。したがって、データベース全体は単一のランク3テンソルとして埋め込まれます。

次のステップはルールの埋め込みです。Datalogルールは、その先行物と成立物をそれらの埋め込みに置き換えることで埋め込むことができます。ルールが

Cons(...) ← Ant1(...), ..., Antn(...)

の場合、その埋め込みは

EmbCons[...] = EmbAnt1[...] ... EmbAntn[...]

です。ここで

EmbAnt1[...] = Ant1(...) Emb[...] ... Emb[...]

等。埋め込み空間での推論は、埋め込みルールについて前向きまたは逆向きチェーンを実行することで実施できます。クエリへの回答は、上記のすべての関係に対して示されたのと同じ方法で、そのテンソルをその引数の埋め込みと結合することで抽出できます。各推定テンソルは埋め

込み関係の結合の射影の合計として表現でき、各引数のために製品Emb[x, i] Emb[x', i]がおよそアイデンティティ行列であるため、これは近似的に正しい結果を与えます。エラー確率は以前のように埋め込み次元とともに減少します。これをさらに減らすために、推定テンソルを定期的に（限界では、各ルール適用の後）抽出、しきい値、再埋め込みできます。

最も興味深い場合は、しかし、オブジェクトの埋め込みが学習される場合です。埋め込み行列とその転置の積

Sim[x, x'] = Emb[x, d] Emb[x', d]

は、今その埋め込みの内積により各オブジェクトペアの相似性を計算するグラム行列です。類似なオブジェクトは、「互いに推論を借り」ており、その相似性に比例する重み。これは、相似性と構成性を明示的に深いアーキテクチャで結合する推論の強力な形式につながります。

各方程式にシグモイドを適用すると、

σ(x, T) = 1 / (1 + e^(-x/T))

その温度パラメータTを0に設定すると、グラム行列がアイデンティティ行列に効果的に削減され、プログラムの推論は純粋に演繹的です。これはLLMと対照的で、T = 0でさえもハルシネーション出力の可能性があります。これはまた、検索拡張生成（Jiang et al., 2025）より指数関数的に強力です。なぜなら、実質的にファクトの演繹的クロージャを取得するルールの下ではなく、単にファクトそのものを検索するだけです。

温度を上げると、推論がますます類推的になり、サンプルはますます類似性を借りる推論から。最適なTはアプリケーションに依存し、異なるルール用に異なる可能性があります（例えば、いくつかのルールは数学的な真実かもしれず、T = 0を持ち、他はエビデンスを蓄積するため、高いT）。

推定されるテンソルは推論中の任意の時点で抽出できます。これはLLMベースの推論モデルとは対照的に、推論を非常に透明にします。また非常に信頼できり、LLMベースのモデルと対照的に、十分に低い温度での幻覚に免疫があります。同時に、埋め込み空間での推論の一般化と推論的能力を持っています。これにより、広範なアプリケーション用に理想的に作られる可能性があります。

---

6. スケールアップ

大規模な学習と推論のため、密なテンソルを含む方程式は、GPU上で直接実装できます。疎なテンソルと混合テンソルの操作は、（少なくとも）2つのアプローチのいずれかを使用して実装できます。

最初は関心の分離です：密な（サブ）テンソルの操作はGPU上で実装され、疎な（サブ）テンソルの操作はデータベースクエリエンジンを使用して実装されます。（サブ）テンソルは関係として扱われます。クエリ最適化の全体的なパノラマを、その後、これらの疎な（サブ）テンソルを

組み合わせるために適用できます。密な（サブ）テンソル全体が、密な（サブ）テンソルを指すポインタを持つ単一のタプルとして、データベースエンジンにより扱われます。密な（サブ）テンソルはその後、GPUを使用してジョインされ、射影されます。

2番目で、より興味深いアプローチは、Tucker分解を介して疎なテンソルを密なテンソルに最初変換するようにGPU上で、すべての操作を実施することです。これは疎なテンソルに直接操作するよりも指数関数的により効率的であり、前のセクションで見たように、ランダムな分解でさえ十分です。コストは小さい確率のエラーがありますが、これは埋め込み次元を適切に設定し、ステップ関数を通じてそれを渡すことにより結果を消音することを制御できます。Tucker分解を通じてスケール上げることは、前のセクション説明した学習と推論アルゴリズムとシームレスに組み合わせる有意な利点があります。

---

7. 議論

テンソルロジックはAI以降に有用である可能性があります。科学計算は本質的に方程式をコードに変換することで構成されており、テンソルロジックで、この変換は以前の言語よりも一層直接的であり、しばしば紙上の記号とコード上の記号の間に一対一の対応があります。科学計算では、関連する方程式は、その実行を制御する論理的ステートメント内にラップされます。テンソルロジックは対応するブール値テンソルを数値のものにリラックスさせることで、この制御構造を自動的に学習可能にし、オプションで結果をロジックにしきい値を戻します。同じアプローチは、原則的には任意のプログラムを学習可能にするために適用できます。

新しいプログラミング言語は、広範な採用のための急な登り坂に直面しています。テンソルロジックの成功の可能性は何ですか？AIプログラミングはもはやニッチではありません。テンソルロジックはJavaがインターネット波に乗ったのと同じ方法でAI波に乗って広範な採用にライドできます。Pythonとの後方互換性はキーであり、テンソルロジックはそれにうまく貸ります。最初はeinsumの、より優雅な実装と推論タスクへのPythonの拡張として使用される可能性があり、開発されるにつれて、NumPy、PyTorch等の，ますます多くの特性を吸収でき、最終的にそれらに取って代わります。

何よりも、新しい言語の採用は、大きな苦しみを治す、そしてサポートするキラーアプリによる推進されます。テンソルロジックはこれらを持っています：例えば、LLMの幻覚と不透明性を治す可能性、および推論、数学的、コーディングモデルに対する理想的な言語。

テンソルロジックの周りのオープンソースコミュニティの育成は、最前線の中央に立ちます。テンソルロジックは密に設計、データレングリング、モデリング、評価を統合するIDE向けに自分自身を貸し、それが起こった場合、ベンダーはそれを支援するために競争します。また、AIの教育と学習に理想的に適切です。そしてこれはそれが広がるこができる別のベクトルです。

次のステップは、CUDAにテンソルロジックを直接実装し、広範な適用で使用し、ライブラリと拡張を開発し、それが可能にする新しい研究の方向を追求することです。

詳細情報は tensor-logic.org を参照してください。

謝辞

この研究は部分的にONR助成金N00014-18-1-2826によって資金提供されました。

参考文献

G. Bakr, T. Hofmann, B. Schölkopf, A. Smola, B. Taskar, and S. Vishwanathan, editors. Predicting Structured Data. MIT Press, Cambridge, MA, 2007.

B. Bloom. Space/time tradeoffs in hash coding with allowable errors. Comm. ACM, 13: 422–426, 1970.

C. Goller and A. Kücler. Learning task-dependent distributed representations by backpropagation through structure. In Proc. Int. Conf. Neural Networks, pp. 347–352, 1996.

S. Greco and C. Molinaro. Datalog and Logic Databases. Morgan & Claypool, San Rafael, CA, 2016.

P. Jiang, S. Ouyang, Y. Jiao, M. Zhong, R. Tian, and J. Han. Retrieval and structuring augmented generation with large language models. In Proc. Int. Conf. Knowl. Disc. & Data Mining, pp. 6032–6042, 2025.

D. Koller and N. Friedman. Probabilistic Graphical Models: Principles and Techniques. MIT Press, Cambridge, MA, 2009.

N. Lavrač and S. Dzeroski. Inductive Logic Programming: Techniques and Applications. Ellis Horwood, Chichester, UK, 1994.

Y. LeCun, L. Bottou, Y. Bengio, and Y. Haffner. Gradient-based learning applied to document recognition. Proc. IEEE, 86:2278–2324, 1998.

Z. Liu and J. Zhou. Introduction to Graph Neural Networks. Morgan & Claypool, San Rafael, CA, 2022.

J. W. Lloyd. Foundations of Logic Programming (2nd ed.). Springer, Berlin, Germany, 1987.

S. Rabanser, O. Schur, and G. Gunnemann. Introduction to tensor decompositions and their applications in machine learning. arXiv:1711.1078, 2017.

T. Rocktäschel. Einsum is all you need – Einstein summation in deep learning. https://rockt.ai/2018/04/30/einsum, 2018.

A. Rogozhnikov. Einops: Clear and reliable tensor manipulations with Einstein-like notation. In Proc. Int. Conf. Learn. Repr., 2022.

B. Schölkopf and A. Smola. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge, MA, 2002.

H. Siegelmann and E. Sontag. On the computational power of neural nets. J. Comp. & Sys. Sci., 50:132–

150, 1995.

P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. Artif. Intel., 46:159–216, 1990.

A. Vaswani, N. Shazeer, N. Parmar, N. Uszkoreit, J. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. Adv. Neural Inf. Proc. Sys., 30:5998–6008, 2017.

P. Werbos. Backpropagation through time: What it does and how to do it. Proc. IEEE, 78:1550–1560, 1990.