型安全でクロスプラットフォームな次世代 ML ライブラリの開発
― 環境依存とランタイムエラーの低減 ―

1. 背景

近年、ニューラルネットワークの持つ可能性が注目され、次々と新たな技術が開発されている。特に、モデルの高度化に伴って、GPU や専用ハードウェアといった計算アクセラレータの使用や学習プロセスの複雑化、モデル自体の複雑化が顕著である。しかし、今日のニューラルネットワークの訓練や推論に使用されるライブラリやそのライブラリを中心とするエコシステムは旧来の技術を前提として設計されており、ユーザ側の実装が比較的簡潔な数値計算には向いているが、近年の複雑なモデルを取り扱うために必要な工学的堅牢さを持たない。そのために、ニューラルネットワークの研究や開発において、本質的に関わらない作業に割く時間が増加している。

このような課題は、Python や GPU の利用規格といった既存のライブラリやエコシステムの根幹をなす要素に起因しているため、既存ライブラリの機能追加や改修によって解決することは難しい。したがって、現代のニューラルネットワーク開発に求められる要件を満たすように設計した、新しいライブラリやエコシステムをゼロから構築し直す必要がある。

2. 目的

本プロジェクトでは、そのような新たなライブラリやエコシステムを作成しうる構成があることを示し、これからの時代の人工知能開発の基盤となるエコシステムの原型を作成し、方向性を示すことを目的としている。長期的には、人工知能開発の効率化の実現を通して次の世代の人工知能開発を支え、もって人間の役に立つ高性能な人工知能の誕生を下支えすることを目指す。

3. 開発の内容

上記の目的のために本プロジェクトで開発した機械学習ライブラリ「Rusty Lantern」の構成概要を図 1 に示す。Rusty Lantern 本体は Rust 言語で実装されており、外部に数値計算ライブラリ等の依存を持たない。

計算バックエンドには CPU バックエンドおよび、WebGPU 規格を使用した GPU バックエンドが用意されている。これらのバックエンド実装は全て四則演算からフルスクラッチされている。WebGPU バックエンドは、FireFox ブラウザの内部実装である Wgpu を使用し、ブラウザの介在なしでネイティブコードとして動作する。それらのバックエンドごとの実装を取りまとめ抽象化するものとして LanternCore があり、汎用的な CPU および GPU での行列計算、行列操作の機能を提供する。さらに LanternCore を使用する形で、LanternDataset と LanternNN が実装されている。LanternDataset は AI モデルの学習に必要な訓練データおよび推論時の入力データをストレージから読み出し、ユーザが指定した方法で前処理を行って、行列型にまとめる機能を持つ。LanternNN はニューラルネットワークの学習アルゴリズムであるバックプロパゲーションを実行するために必要な計算グラフの構築および自動微分の機能を備えている。LanternBoard はこれらの 3 つのシステムに対して GUI によるデバッグを行うことができるブラウザ上で動くアプリケーションであり、計算プロセス側に立て

られた API を介して内部状態の確認や簡単な指示を行うことができる。これらの機能は図 1 の右側に示す 5 つの OS 上で実行することができる。

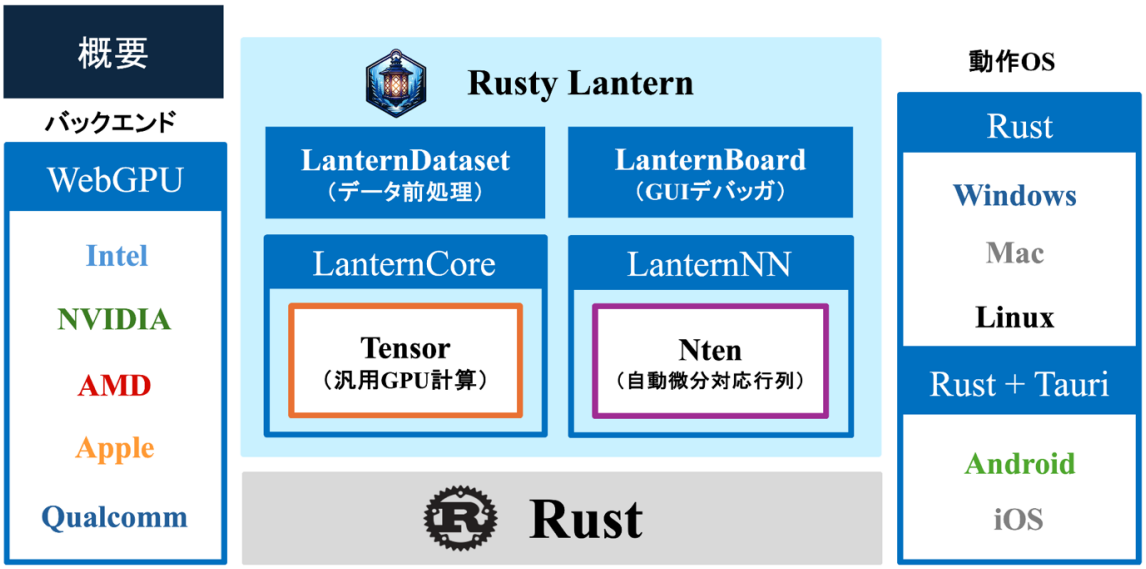

図 1 Rusty Lantern の概要

Rusty Lantern はニューラルネットワークを実行する際に図 2 に示す動作をする。ユーザが実装するものはモデル定義、データ入力、計算グラフ構築、パラメータ更新処理であり、実際の計算実行の管理はシステム側で行う。これによって、システム側が使用する演算の統合、実装の透過的な置き換えや並列化、計算実行の非同期化を行うことを可能にしている。

データ形式と演算の抽象化階層は図 3 のように 5 段階に分けられている。低レイヤ側の機能は LanternCore に含まれており、自動微分レイヤ以降は LanternNN に含まれている。このうち、実体演算レイヤと自動微分レイヤはユーザがライブラリに備えられている一般的な演算のみを使用する場合には意識する必要はない。独自の演算が必要な場合にはカスタム演算として実装することができる。

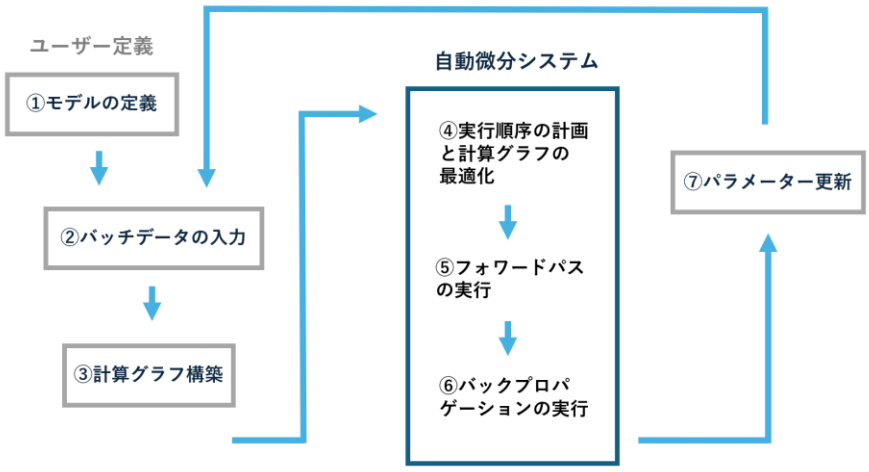

図 2 Rusty Lantern のニューラルネットワーク実行順序

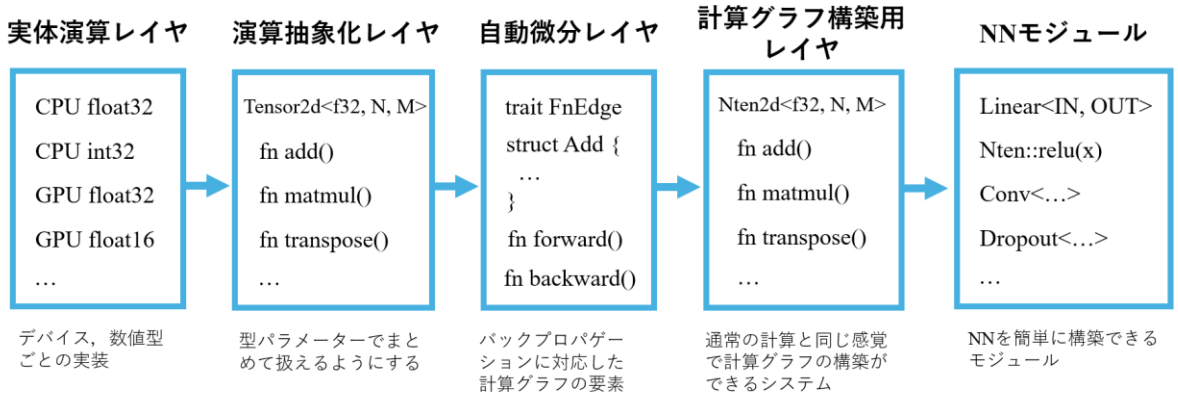

図 3 Rusty Lantern の内部階層

演算における型制約と型推論は図 4 に示すサンプルコードのような使用感を持つ。行列の演算ではスカラーの場合と異なり引数の行列の形状について計算可能な組み合わせと未定義の組み合わせがある。例えば、要素ごとの演算では引数となる 2 つの行列は同形状である必要があるし、行列積の場合は図 4 で N と指定した箇所の大きさが一致している必要がある。したがって、これらの誤りをコンパイル時に検出することはランタイムエラーを削減し、デバッグの手間を減らす効果がある。

計算可能な組み合わせは演算関数ごとに定義されており、その戻り値の形状は型推論によって自動的に決定される。図 4 では matmul 関数の戻り値「c」の後に続く「:」以降の灰色の部分が型推論によって示された形状である。これによって、操作している値の形状変化を追うことができ、可読性が向上する。

```
#[test]
▶ Run Test | Debug
fn matmul_gpu_ones() {
    const N: usize = 8;
    let a: Tensor2d<Gf32, R: 3, N> = Tensor2d::new_ones();
    let b: Tensor2d<Gf32, N, C: 5> = Tensor2d::new_ones();
    let c: Tensor2d<Gf32, 3, 5> = a.matmul(&b);
    println!("{}", c);
}
```

図 4 LanternCore を使用した行列積計算のサンプルコード

4. 従来の技術（または機能）との相違

現在のデファクトスタンダードである PyTroch や TensorFlow といった Python 系ライブラリは、Python ライブラリや GPU バックエンドに起因する環境依存が多く、さらに Python がインタプリタ言語であることに起因して、ランタイムエラーの抑制が困難である。これらは、複雑化する現代のニューラルネットワークを扱う上で生産性の向上を阻害している。

Rusty Lantern では、広範な環境向けにコンパイルできる Rust 言語と、高い可搬性を持つ WebGPU を GPU の抽象化として使用し環境の差異を吸収することによって、どこでも同じコードを実行可能である。

また、Rust の型システムを使用してニューラルネットワーク向けに最適な型設計を行い、コンパイル時にほとんどの計算互換性を検証することによって、ランタイムエラーが大幅に低減された。さらに、GUI デバッガによって内部状態を可視化する機能により、実行時の問題も迅速に特定できる。

5. 期待される効果

Rusty Lanter を使用すれば各種 OS, GPU に対しても単一ソースで実装を記述でき、クラウドからモバイル端末まで広範なデバイスでコードの変更なしで実行可能である。可搬性を持たせることによって、開発マシンの制約がなくなるとともに、開発環境構築の手間を劇的に低減し、さらに環境に依存しないためコードの長期的な再現性が保たれる。また、開発環境と同じコードが使用できるためユーザアプリケーションへの展開も高速になる。

また、専用の型設計と GUI デバッガにより、複雑な AI モデルに対しても高い開発効率を備える。

これによって、開発者の時間や思考力といったリソースを本質に集中させることでさらに高度な AI の研究を加速させる。また、応用アプリケーションの開発や保守も容易にする。

6. 普及（または活用）の見通し

本プロジェクトの成果をカンファレンスでの発表や SNS での発信を通してニューラルネットワークの研究者や応用アプリケーションの開発者に周知し、今後の人工知能開発に貢献する。また、ドキュメントの整備やサンプルコードの充実化、ユーザからのフィードバックの反映、コントリビュータの獲得などを進め、OSS プロジェクトとして発展させる。

7. クリエータ名（所属）

道本将弘（埼玉大学工学部電気電子物理工学科）

（参考）関連 URL

- Rusty Lantern ライブラリの公式広報 X アカウント
  https://x.com/RustyLantern_rs