深層学習の原理探究へ向けたRust製フレームワークの構築

～最適化および機能的拡張へ～

逗子開成高等学校　高校2年　臼井千裕　鈴木翔天

GitHub リポジトリ

SCIENCE

SINCE 1903

研究概要

Rustは近年注目されている言語である一方、機械学習の分野においては開発途上であり pythonやC系言語に比べて文献が少ない。

深層学習の分野ではフレームワークのコミュニティやドキュメントの多くが英語によって説明されているため学習や問題解決までの障壁が高い。

実用的でフルRust実装のフレームワーク

日本語によってコードの説明がなされていてユーザー自身の手で実装してもらう

深層学習の原理探究に向けたRust製フレームワークを構築
我々は開発したフレームワークを*StuCrs*と名付ける

研究にあたって

本研究は下の著書『**ゼロから作るDeep Learning③　フレームワーク編**』をもとにして実装しています。著者である**斎藤康毅**氏に著書の考えや表現の使用を許可していただいたことに感謝を申し上げるとともに、この著書オリジナルのフレームワーク**DeZero**も研究の参考として利用させていただいています。

DeZero

Rustとは

Rustの特徴は**実行速度の速さ**、**安全性の担保**、**モダンな言語**が挙げられます。

実行速度の速さ
- 直接コンパイル
- ガベージコレクションがない
- ゼロコスト抽象化

最も実行速度が速いといわれるC,C++に匹敵する

安全性の担保
- メモリー
- 監視
- コンパイラー

未然にメモリ安全やスレッド安全ではない操作を防ぐ

モダンな言語
- 強力な型推論
- 不変、可変トレイト
- 代数的データ型

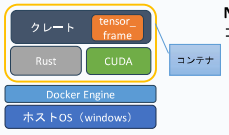
様々プログラミング言語の機能が入っている

環境開発

Dockerを用いて**Rust**の環境をそろえたコンテナを**Visual Studio Code**で立ち上げる。

NVIDIA製のGPUを用いて学習できるようNVIDIAの**CUDA、tensor_frame**をコンテナ内にインストールする。

<環境イメージ>



<スペック>
- CPU: Intel Core i5 12400,
- GPU: GeForce RTX 4060,8GB
- メモリ: 48GB

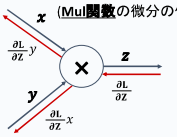
- **Docker**...コンテナ型の仮想環境を構成、管理するソフトウェア。
- **CUDA**... NVIDIAが提供するGPU向けの統合開発環境。
- **ndarray**...多次元配列を効率的に扱うためのクレート。
- **Tensor_frame**... Rustからの GPU上で行列計算を扱うためのクレート。
  ➡ ndarrayのように高度な行列計算を行える。

※詳しいクレートのバージョンや依存関係、スペックはGitHubリポジトリをご参照ください。

実装への主な流れ

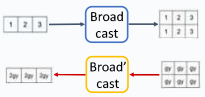
実数での自動微分
1. ***RcVariable***, ***Function*** **トレイト**を実装
2. 一変数関数の微分 (sin, log, tanhなど)
3. 複雑な関数の微分を自動化

(**Mul関数**の微分の例)



演算子のオーバーロード
1. 演算子の関数を実装(Add,Mul,Subなど)
2. 演算子を***RcVariable***の演算子にオーバーロード
   add(&a,&b) → **&a + &b** と書くことができる。

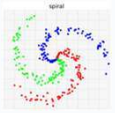
(**Broadcast関数**の逆伝播の例)



行列への対応
1. f32型を**ndarray型**に変更
2. 行列関数を実装(reshape, sumなど)
3. **ブロードキャスト**への対応

ニューラルネットワークの構築
1. ***Layer***, ***Model***, ***Optimizer***を実装
2. **損失関数**を実装(**交差エントロピー誤差**など)
3. ***Dataset***, ***DataLoader***を実装
4. 自作の**スパイラルデータ**で多値分類を試す

自作したスパイラルデータを画にプロット



MNISTの学習
1. **MNIST**のデータセットをダウンロード、実装
2. 学習、および推論
3. 他の**フレームワーク**と比較

TensorFlow
DeZero

公開の準備
1. パッケージの作成
2. リポジトリの公開
3. ドキュメントの作成

※詳しい実装までの説明はGitHubリポジトリのドキュメントをご参照ください。

MNIST での性能比較

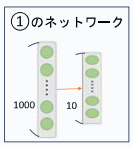
※詳しい学習のデータはリポジトリに載せてあります。

下の表は3種類のフレームワーク(**StuCrs**,**Dezero**,**Tensorflow**)で**MNIST**のデータをCPUで学習した際の処理にかかった時間を表している。
今回は二種類のニューラルネットワークは構築し、比較した。

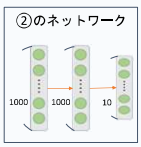
学習の設定
- epoch... 5
- batch... 100

①... 出力が(1000,10)、活性化関数がReLUのDenseレイヤ
②... 出力が(1000,1000,10)、活性化関数がReLUのDenseレイヤ

①のネットワーク



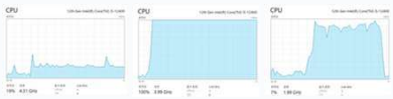
②のネットワーク



また②における学習でそれぞれのCPU使用率を計測した。
右下のグラフが使用率を表しており、これを見ると、
StuCrsが他二つに比べて**CPU使用率が低く抑えられている。**

フレームワーク	StuCrs	DeZero	TensorFlow
①のネットワーク	20.99382s	35.70813s	17.60617s
②のネットワーク	48.97416s	76.68909s	35.39685s

<StuCrs>　<DeZero>　<TensorFlow>



CUDA対応

ここではRustの**tensor_frame**というクレートを用いる。
このクレートは**ndarray**と同じように、**ブロードキャスト**と行列計算に対応し、かつGPU上で計算することができる。しかし必要な関数が存在しない場合があったため、**拡張し独自の関数を実装した。**

独自の関数を実装
- **CUDAカーネル**を書き、追加。
- **Cudarc**で**GPUのメモリ**を管理し、**カーネル**を実行するよう設計。
- CPU上から処理させるよう設計。

↓

ndarrayから**tensor_frame**の行列構造体に変換できるよう設定

↓

各関数を**tensor_frame**の処理に合わせて計算できるよう変更

《GPUで同じくMNISTを学習させる》

CPUと同様に、同じ**MNIST**のデータを**GPU**に対応させたフレームワークで学習させる。

(タスクマネージャーの一部)
CUDAの使用率を表示

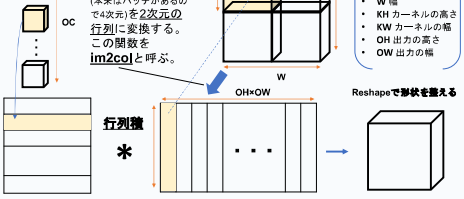
CUDAを用いてGPU上で計算していることがわかる。

※詳しい学習のデータはリポジトリに載せてあります。

CNNの実装

ここでは画像認識の分野でよく使われる**ＣＮＮ**と呼ばれるニューラルネットワークを実装した。
本研究では**畳み込み層**は画像とカーネルを**二層テンソル**に変換しその**行列積**を求め、それを４階テンソルに直し、**プーリング層**はチャンネルごとに**最大値**をとりそれを４階テンソルに直す形式で実装した。

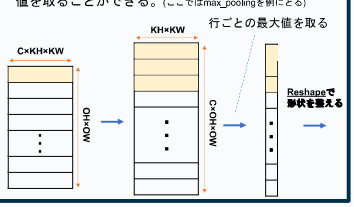
《Conv2d関数の流れ》



《Pooling関数の流れ》

※バッチ、バイアスといった細かい処理のイメージは省略した。

Im2colで２次元にしたものをチャンネル数で行を平たくする。これにより、**チャンネルごと**に最大値を取ることができる。(ここではmax_poolingを例にとる)



GitHubとの連携

本研究は**GitHubリポジトリ**として公開している。
また、リポジトリでは**自ら作成したドキュメント**でプログラミングの概要を説明している。

・ドキュメント

日本語でプログラミングを詳細に説明することによって、ユーザーが**直観的に理解しやすい**ような設計になっている。

深層学習の枠組みがわかりやすく説明されているため、ユーザー自身の手で実装してもらうことができ、**深層学習の原理探求につながる。**

・リポジトリの公開

リポジトリを公開することで様々な方からアドバイスや改善点を教えてもらうことができ、より良い品質のコードが作れるようになる。

考察

今回構築したフレームワークにおいて**「MNISTでの性能比較」**、**「GitHubとの連携」**から次のようなことがいえる。

性能面
CPUでの学習においては層が浅いとTensorflowにせまる性能を持ち、DeZeroと比べて**約2倍近くの性能**があるため、実用的であると言える。

機能面
Functionトレイト、**Layer**トレイト、**関数**で互いに組み込みやすいような設計に改良することで、ユーザーが応用しやすく、柔軟な構造になった。

改善点
行列データが小さいほど、StuCrsはDezeroと比べてより高速に処理するため、行列計算を担う**ndarrayクレート**の処理が**numpy**などよりも遅いことがわかる。さらに高速な行列計算クレートが開発されれば、さらなる**フレームワークの高速化が期待できる。**また、RustとCudaとの連携に関する開発も未発展であり、pythonと同レベルでCudaの性能を十分発揮できるようなシステムを構築できなかった。

反省 ,今後の展望

今回の研究ではRustを用いて実際に一からフレームワークを構築することで**深層学習の仕組みを理解することができた**。しかしTensorFlowなどの既存のフレームワークと比べるとプログラムの最適化をさらに改良できる点も多く、実行速度としてはまだ劣っている。また、機能面においても**実装できていない関数が多い**。

バグの対処やコメントによる説明を増やし、**コミュニティを拡大する**ことでユーザーの理解を深める。

OptimizerやLSTM用のレイヤなど、実装する関数の種類をさらに増やす。

フレームワークの自動的な構造組み立ての恩恵を受けられない設計になっている関数もあり、構造としてのまとまりがない状態なので、ユーザーが理解しやすいようにさらに整理、体系化を進める。

参考文献
- 『ゼロから作るDeep Learning③　フレームワーク編』　斎藤康毅　(2020) オライリー・ジャパン
- 『実践Rust入門[言語仕様から開発手法まで]』Ieen.、河野達也、小松礼人　(2019) 技術評論社
- 『pythonで動かして学ぶ！あたらしい深層学習の教科書　　機械学習の基本から深層学習まで』石川聡彦 (2018) 翔泳社
- 『pythonの年生 ディープラーニングの仕組み体験してわかる会話でまなべる！』巧尚　(2023) 翔泳社
- 『実践RUSTプログラミング入門』　初田直哉、山口聖弘、吉川智史、笠田修貴、松永龍太郎、須田已、中本諒弘　(2020) 株式会社秀和システム
- 『やさしいRust入門』日向俊二　(2021) カットシステム
- 『AlphaZero 深層学習・強化学習・探索 人工知能プログラミング実践入門』布留川英一　(2019) ボーンデジタル

使用した主なクレート
- Tensor_frame <https://docs.rs/tensor_frame/latest/tensor_frame/index.html>
- ndarray <https://docs.rs/ndarray/0.16.0/ndarray/index.html>
- plotters <https://docs.rs/plotters/0.3.1/plotters/index.html>
- mnist <https://docs.rs/mnist/latest/mnist/index.html>
- image <https://docs.rs/image/latest/image/index.html>

使用したロゴ
- Tensorflow <https://www.tensorflow.org/?hl=ja>
- PyTorch <https://pytorch.org/>
- NumPy <https://numpy.org/ja/>
- python <https://www.python.org/>
- GitHub <https://github.com/>
- Rust <https://www.rust-lang.org/ja>

※詳細な参考文献リストは、GitHubのリポジトリをご参照ください。