1E5-1in1

Julia 言語による深層学習ライブラリの実装と評価

Implementation and Evaluation of Deep Learning Library in Julia Language

進藤 裕之 Hiroyuki Shindo　澤井 裕一郎 Yuichiro Sawai　大内 啓樹 Hiroki Ouchi　松本 裕治 Yuji Matsumoto

奈良先端科学技術大学院大学
Nara Institute of Science and Technology

1. はじめに

深層学習とは，多層のニューラルネットワークに基づく機械学習手法であり，画像認識，音声認識や自然言語処理などの分野において，様々なタスクの高精度化に大きく寄与してきた．それに伴い，深層学習のモデル定義やパラメータ最適化を簡潔に記述できるライブラリが公開され，深層学習の専門家でなくても，最先端の深層学習技術を利用したアプリケーションを開発することが容易になりつつある．

一般的な深層学習ライブラリでは，よく用いられる関数群（活性化関数や畳み込み関数）があらかじめ用意されており，ユーザーはそれらを自由に組み合わせてモデルのネットワーク構造を定義することができる．また，モデルの訓練に必要な勾配の計算やパラメータの最適化を行う機能を有しており，ユーザーがそれらのプログラムコードを一から実装する必要がないという利点がある．

既存の深層学習ライブラリとして，Caffee [Jia 14] や Theano [Bergstra 10] などが挙げられるが，これらは静的なネットワーク構造を想定した設計となっていることが多い．すなわち，ネットワーク構造は入力データや途中の計算結果に依らず事前に決定される，または，そのような場合にのみ高速に動作する．手書き文字認識や画像分類のタスクでは，一般的に入力データのサイズは揃えられており，畳み込みニューラルネットワークのようなフィードフォーワード型のネットワーク構造を用いるため，これらのライブラリに適した問題設定であるといえる．

一方，自然言語処理分野では，入力データ（例えば単語列）のサイズがサンプルごとに異なるため，リカレント型や再帰型のニューラルネットワークがよく用いられる．このようなタイプのニューラルネットワークでは，ネットワーク構造が入力サンプルごとに異なるため，あらかじめ完全なネットワーク構造を定義しておくことが難しい場合がある．その結果，入力サンプルのミニバッチ化が困難となったり，GPU による並列計算よりも CPU による逐次計算の方が高速になるといった事例が報告されている [Johnson 11, Ballesteros 15]．

既存の深層学習ライブラリでは，大きなバッチサイズのデータは比較的高速に処理できるが，小さなバッチサイズのデータを何度も処理しなければならない場合では十分に最適化されておらず，実行速度の大幅な低下を招く場合がある．この問題を解決するため，我々は自然言語処理分野のタスクに対しても高速に実行可能な深層学習ライブラリの開発に取り組んだ．特に，科学技術計算に適した高速なスクリプト言語である Julia [Bezanson 15] を用いて深層学習ライブラリの設計と実装を行い，自然言語処理分野の品詞タグ付けタスクで実行速度および精度の評価を行った．Julia は，MIT で開発されている動的なプログラミング言語であり，簡潔な文法と高速な実行速度が特徴である．また，多次元配列，各種の線形代数アルゴリズム，並列計算などを標準的にサポートしており，数値計算のライブラリ開発に適した言語であるといえる．

本研究で開発した深層学習ライブラリ：Merlin.jl [*1] を用いて品詞タグ付けタスクの実行速度を評価した結果，Theano や Chainer [Tokui 15] などの従来の深層学習ライブラリと比較して，バッチサイズによらず高速に実行できることを確認した．本研究で使用したソースコードは，著者らの Web ページで公開する予定である．

Contact: shindo@is.naist.jp

2. 関連研究

これまでに様々な深層学習ライブラリが開発されてきたが，本研究では，自然言語処理タスクのプロトタイピングに適した深層学習ライブラリとして Theano および Chainer を比較対象として選んだ．

2.1 Theano

Theano は，シンボリック計算に基づく Python の数式処理ライブラリである．Theano では，多次元配列（テンソル）を入出力とする数式をシンボルの計算グラフとして表現し，それを事前にコンパイルすることによって計算効率の高いコードを生成する．また，数式の評価や勾配の計算，パラメータの最適化を行う機能が用意されており，深層学習モデルの定義や訓練のために使用することができる．また，Theano は宣言型のパラダイムに基づいた設計を採用しており，事前にネットワーク構造を決定しておくことで計算グラフの全体最適化を行う一方で，ネットワーク構造が入力サンプルや途中の計算結果に依存して決まる動的なモデルの記述にはあまり適していない．

2.2 Chainer

Chainer は，Python 用の深層学習ライブラリである．Chainer では，命令型のパラダイムに基づいた設計を採用しており，関数の適用が先行評価される．そのため，直観的で柔軟なモデルの記述が可能であり，Python の標準的なデータ構造や他のライブラリと連携しやすいという利点がある．しかしながら，現状の Chainer は，CPU での計算が十分に最適化されておらず低速であるという問題がある．

```
# Network definition
f = Sequence(Linear(Float32,300,100),
             ReLU(),
             Linear(Float32,100,10))

# Training
opt = SGD(0.001)
for epoch = 1:10
  for x in train_data
    y = f(Variable(x))
    loss = CrossEntropy(p, y)
    gradient!(loss)
    update!(opt,f)
  end
end
```

図 1: Merlin.jl によるフィードフォーワードニューラルネットワークのコード例．簡単のため，一部の変数定義は省略されている．

3. Julia 言語による深層学習ライブラリ

画像処理分野や自然言語処理分野のタスクでは，一般に複数の入力サンプルをまとめてミニバッチを作成し，ミニバッチ単位でニューラルネットワークの計算を行うことで実行時間の高速化を図っている．特に，GPU を用いた並列処理では，一度に多くのデータを入力する方が計算効率が高い．また，CPU の場合でも，一度に多くのデータをまとめて行列積の計算をする方が一般に計算効率が高くなる．ただし，自然言語処理タスクのように入力サンプル（例えば文）が可変長であったり，階層的・再帰的なネットワーク構造を扱う場合にはミニバッチ化が困難となる可能性がある．データが可変長であるという問題は，同じ長さの文だけでミニバッチ化を行うことや，最大長の単語を基準として，残りの単語はパディングを挿入することによって最大長に合わせるなどの戦略が考えられるが，このようなミニバッチの作成方法はモデルに大きく依存し，パディングによる計算効率の悪化などが避けられない．また，構造学習のタスクでは，ニューラルネットワークによるスコア計算に加えて，最適な出力を探索する処理も必要となるため，複数文を同時に処理することが難しい．

そのような問題に対処するために，我々は Jula を用いた深層学習ライブラリの開発に取り組んだ．Julia を用いる理由は，文法が Python のように簡潔でプロトタイピングが容易であるにもかかわらず，実行速度が非常に高速であるためである．

深層学習ライブラリは，命令型パラダイムと宣言型パラダイムに大別される．命令型パラダイムでは，関数が先行評価されるため，ユーザーの書いた計算手続きと実際の計算手続きが対応しており，直観的で柔軟なニューラルネットワークのモデル定義が可能である．宣言型パラダイムでは，事前にネットワーク構造を定義し，その計算グラフを全体最適化することによって計算速度の高速化を図っている．我々のライブラリ：Merlin.jl では，全体としては命令型に従うが，あらかじめ定義可能な部分ネットワーク構造は宣言型を用いてモデルを構築し，最適化を行うことができる．

図 1 に，Merlin.jl によるフィードフォーワードニューラルネットワークのコード例を示す．まずはじめに，関数 f を定義し，学習データのサンプル x に適用することで出力 y を得る．次に，誤差を計算して勾配を求め，関数 f のパラメータ更新を行っている．図 2 に，リカレントニューラルネットワークのコード例を示す．この例では，三種類の関数（Lookup, Sequence，Linear）は事前に定義するが，ベクトルを連結する関数（Concat）はパラメータを持たないため，学習時に毎回インスタンスを作成して使用することができる．以上のように，Merlin.jl では，モデルに含まれる静的なネットワーク構造（例えば図 1 の Sequence）はあらかじめ定義しておき，それらの部分構造をデコード時や学習時に自由に組み合わせることによって全体のネットワーク構造が決定される．また，各関数はパラメータ以外の内部変数を持たないため，フォーワード計算中に同じ関数のインスタンスを何度でも使用することができる．

∗1 https://github.com/hshindo/Merlin.jl

```
# Network definition
f_in = Lookup(Float32,10000,100)
f_h = Sequence(Linear(Float32,200,100),
               ReLU())
f_out = Linear(Float32,100,10)

# Training
opt = SGD(0.001)
for epoch = 1:10
  for seq in train_data
    for x in seq
      v = Variable(x) |> f_in
      h = [h,v] |> Concat(1) |> f_h
      y = f_out(h)
      loss = CrossEntropy(p, y)
    end
    gradient!(loss)
    update!(opt,f1)
    update!(opt,f2)
  end
end
```

図 2: Merlin.jl によるリカレントニューラルネットワークのコード例．簡単のため，一部の変数定義は省略されている．

4. 深層学習による品詞タグ付け

本研究では，Santos らによって提案されたニューラルネットワークに基づく英語の品詞タグ付けモデル [Santos 14] を深層学習ライブラリで実装し，実行速度とタグ付け精度を評価することとする．英語の品詞タグ付けは，文を入力とし，文に含まれる各単語の品詞を予測するタスクである．我々が実験で用いたデータセットでは，品詞の種類は 45 種類あるため，品詞タグ付けは各単語ごとに 45 クラスの分類問題として定式化できる．

Santos らのモデルの概要を図 3 に示す．彼らのモデルでは，文字から単語，単語から文という二階層の再帰的な畳み込みニューラルネットワークを用いて単語列の特徴ベクトルを抽出する．具体的には，まずはじめに，文中の各単語を構成する文字列から，文字レベルの畳み込みニューラルネットワークによって単語の特徴ベクトルを計算する．これを文字列ベクトルと呼ぶことにする．次に，文字列ベクトルと，単語の埋め込みベクトル（１つの単語に対して１つのベクトルが対応したもの）とを連結して，単語の特徴ベクトルとする．上記の手続きによって文中の各単語に対して単語ベクトルを計算し，それら

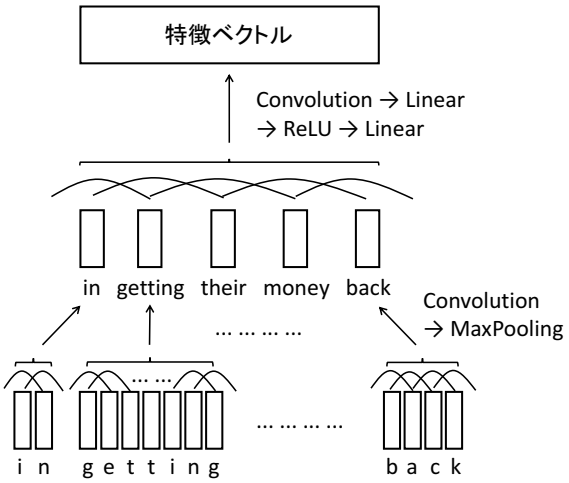

図 3: 深層学習に基づく品詞タグ付けモデル [Santos 14] の概要

を単語レベルの畳み込みネットワークへ入力し，最後に活性化層と線形層（全結合層）を適用することによって単語列の特徴ベクトルを得る．モデルの詳細は，原論文および [進藤 15] を参照されたい．彼らのモデルでは，単語列の特徴ベクトルに対してさらに CRF を適用しているが，今回の実験では単純のため CRF 層は省略したモデルを用いた．

彼らのモデルでは，文字レベルと単語レベルの畳み込みニューラルネットワークが階層的に結合されており，文字レベルの畳み込みニューラルネットワークには可変長の文字列が入力となり，単語レベルの畳み込みニューラルネットワークには可変長の単語列が入力となる．そのため，複数文を一度に処理するためには，ミニバッチ内の単語長および各単語の文字列長を揃える必要がある．

5. 実験

5.1 実験設定

我々の実装した深層学習ライブラリの有効性を検証するため，品詞タグ付けの実験を行った．実験で用いるデータ，モデル，ハイパーパラメータは全て Santos らの研究と同様に設定した．

学習データは，英語 Penn Treebank のセクション 00 から 18 までの約 38000 文を使用し，評価データは，セクション 22 から 24 までの約 5400 文を使用した．また，単語の埋め込みベクトルは 100 次元，文字の埋め込みベクトルは 10 次元，単語の文字列ベクトルは 50 次元，中間層のベクトルは 300 次元，出力層のベクトルは 45 次元である．単語の埋め込みベクトルは，word2vec[*2] を用いて，英語の Gigaword コーパス [*3] に含まれる NewYork Times データから，100 次元のベクトルを事前に学習したものを初期値として用いた．パラメータの最適化は，確率的勾配降下法（SGD）を用いて行った．その他の詳細な設定については，原論文 [Santos 14] を参照されたい．

比較手法として，Theano v0.8，Chainer v1.7.1 を用いて同じモデルを構築し，学習およびテストの実行速度を計測した．実験に用いた計算機は，Intel® Core™ i7 1.7Ghz DualCore, メモリは 8GB である．また，Julia では OpenBLAS がデフォルトでインストールされるため，Theano および Chainer では，Numpy のインストール時に OpenBLAS を指定してコンパイルを行った．また，Merlin.jl では BLAS の計算以外ではシングルスレッドで動作するようにして実行速度を計測した．Theano は OpenMP の設定を OFF にした．以上の設定を行った結果，全てのライブラリで，実験中の CPU 使用率は概ね 190%前後であった．

*2 https://code.google.com/p/word2vec/
*3 https://catalog.ldc.upenn.edu/LDC2011T07

	WordCNN [sec]	WordCNN+CharCNN [sec]
Merlin.jl	1.65	3.88
Theano	1.88	11.30
Chainer	13.49	131.35

表 1: テストデータに対するデコード時間の比較．WordCNN は，Santos らのモデルをさらに簡略化し，単語レベルの畳み込みニューラルネットワークのみを用いて品詞タグ付けを行うモデルである．WordCNN+CharCNN は，図 3 のように単語レベルと文字レベルの畳み込みニューラルネットワークを階層的に結合したモデルである．

5.2 結果

5.2.1 デコード時間の評価

まずはじめに，デコード時間の評価を行った．デコード時間は，テストデータに対して forward 計算のみを行った場合の計算時間である．また，デコードの処理は 1 文単位で行った．結果を表 1 に示す．表に示されているように，Merlin.jl は，1 文単位で処理する場合には高速であり，文字レベルの畳み込みニューラルネットワークを導入しても速度の低下が起こりにくいことがわかった．

WordCNN は単純なフィードフォーワード型のニューラルネットワークであるため，Theano も Merlin.jl とほぼ同じ時間でデコードできていることがわかる．Chainer は，バッチサイズが小さい場合には速度が遅いことが知られており，最も遅い結果になったと考えられるが，ミニバッチサイズを大きくすることで速度向上が期待できる．ミニバッチサイズの影響については，次の学習時間の実験で評価を行う．また，WordCNN に CharCNN を加えることによって，ネットワーク構造が単純なフィードフォーワードニューラルネットワークではなくなるため，実装によっては大幅に速度の低下が起こる場合があることがわかった．

ただし，それぞれの深層学習ライブラリで用意されている関数は完全に同じではなく，end-to-end の実行時間を評価する際に全く平等な条件でベンチマークテストを行うことは困難である．また，Theano や Chainer では，我々の実装が最適であるとは限らず，コードを改善することによって実行速度の大幅な高速化が実現できる可能性があることに留意されたい．したがって，Merlin.jl が必ずしも最速であると結論づけることはできないが，単純な WordCNN や，CharCNN を加えた再帰的な構造を持つモデルにおいても，Merlin.jl は他ライブラリと遜色のない速度で計算が実行できることがわかった．

5.2.2 モデル学習時間の評価

次に，モデルの訓練に必要な学習時間の計測を行った．学習時間は，学習データの forward 計算，backward 計算およびパラメータの更新時間を含めた結果である．図 4 に WordCNN の学習時間，図 5 に WordCNN+CharCNN の学習時間の結果を示す．図中では，縦軸に 1 epoch あたりの実行時間，横軸にミニバッチサイズ（文）を示してある．実験で用いたデータには 1 文あたり 20 - 30 単語が含まれているため，1 文単位の処理は一度に複数の単語を処理していることに相当するが，

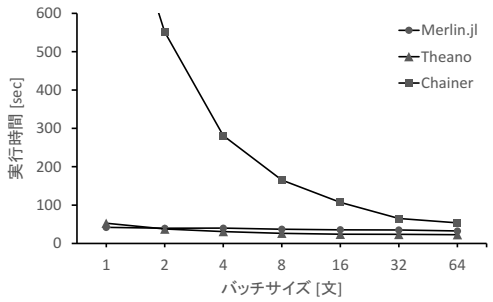

図 4: WordCNN の実験結果.

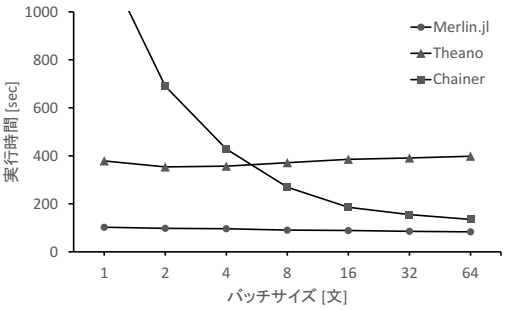

図 5: WordCNN+CharCNN の実験結果.

さらにミニバッチサイズを増やすことによって計算時間がどのように変化するかを評価するため，ミニバッチサイズを複数文へ増やした場合の計測も行った．複数文をミニバッチにする際には，文長（単語数）が同じもののみをまとめてバッチを作成した．

図 4 で示すように，WordCNN の場合には，Merlin.jl と Theano では，バッチサイズを増加させることによる計算時間の短縮はあまり効果がなかった．ただし，Chainer では，複数文をミニバッチとすることで速度が大幅に向上した．モデルが再帰的な構造を持つ場合には，複数文を常にミニバッチ化できるとは限らないので，Merlin.jl や Theano のように，バッチサイズが小さい場合でも高速に動作することは利点であるといえる．

次に，WordCNN に CharCNN を加えてモデルで実験を行った．この場合，Theano および Chainer では，各単語ごとに文字列ベクトルを逐次的に計算するという実装では非常に低速であったため，前処理としてミニバッチに含まれる全ての単語長（単語の文字数）をパディングを用いて統一した．Merlin.jl では，パディングを使わずに，単語ごとに文字列ベクトルを逐次的に計算する処理でも十分に高速であったため，上記の前処理は行っていない．図 5 で示すように，CharCNN を加えた場合には Merlin.jl と Theano とで大きく実行速度に差が見られた．これは，ネットワーク構造が複雑になり，入力も可変長であることに起因すると考えられる．Chainer では，WordCNN と同様にミニバッチサイズを大きくすることによって計算速度が大きく向上した．WordCNN と同様に，Merlin.jl ではバッチサイズを大きくしない場合においても高速に動作することが明らかとなった．これは，Chainer と同様に命令型の設計を採用しているため可変長の入力に対処しやすいことと，Julia 言語の高速な実行速度によるものだと考えられる．

6. おわりに

本研究では，Julia 言語を用いた高速な深層学習ライブラリの実装を行い，他のライブラリと実行時間の比較を行った．その結果，我々の Merlin.jl は，自然言語処理タスクのようなパラメータサイズの小さな問題設定においても，ミニバッチサイズによらず高速に計算ができることがわかった．今後は，GPU の計算時間の評価も行う予定である．

参考文献

[Ballesteros 15] Ballesteros, M., Dyer, C., and Smith, N. A.: Improved Transition-Based Parsing by Modeling Characters instead of Words with LSTMs, in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, No. September, pp. 349–359 (2015)

[Bergstra 10] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y.: Theano: a CPU and GPU Math Expression Compiler, in *Proceedings of the Python for Scientific Computing Conference ({SciPy})*, pp. 1–7 (2010)

[Bezanson 15] Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B.: Julia: A fresh approach to numerical computing, *arXiv*, pp. 1–37 (2015)

[Jia 14] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T.: Caffe: Convolutional Architecture for Fast Feature Embedding, in *Proceedings of the ACM International Conference on Multimedia*, pp. 675–678 (2014)

[Johnson 11] Johnson, M.: Parsing in Parallel on Multiple Cores and GPUs, in *Proceedings of the Australasian Language Technology Association Workshop*, pp. 29–37 (2011)

[Santos 14] Santos, C. D. and Zadrozny, B.: Learning Character-level Representations for Part-of-Speech Tagging, in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, No. 2011, pp. 1818–1826 (2014)

[Tokui 15] Tokui, S., Oono, K., Hido, S., and Clayton, J.: Chainer: a Next-Generation Open Source Framework for Deep Learning, in *Proceedings of Workshop on Machine Learning Systems in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)* (2015)

[進藤 15] 進藤 裕之, 松本 裕治：畳み込みニューラルネットワークを用いた複単語表現の解析, 研究報告自然言語処理（NL）, pp. 1–7 (2015)

謝辞

本研究は，JST，CREST「社会脳科学と自然言語処理による社会的態度とストレスの予測」の支援を受けたものである．