ディープラーニングで用いられる機械学習ライブラリの評価

劉　　忠達[1]・川村　　暁[2]・吉田　等明[3]

Evaluation of Machine Learning Libraries

Zhongda LIU[1], Satoshi KAWAMURA[2] and Hitoaki YOSHIDA[3]

[1]*Faculty of Science and Engineering, Ishinomaki Senshu University*
[2]*Super-Computing and Information Sciences Center, Iwate University*
[3]*Faculty of Education, Iwate University*

1. 概要

近年、深層学習（Deep Learning）の登場により、機械学習が再びブームになっている。また、機械学習に関連する研究が盛んに行われている。これに伴い、多くの機械学習ライブラリが公開され、注目を集めている。

本稿では、機械学習を利用しようとするユーザの視点から、Linux、Mac OSX、Windows といった汎用 OS がサポートされている５つのライブラリをとりあげる。これらのライブラリを評価するため、手書き数字の画像認識のデータセットである MNIST[1] を用いて、学習に必要な時間および評価用データの分類結果の認識率を比較する。学習時間と認識率により、各ライブラリの実用性について評価する。

2. ライブラリの紹介

2.1 Caffe[2]

Caffe はカリフォルニア大学バークレー校で開発されたディープラーニング・フレームワークである。現在では、Berkeley Vision and Learning Center（BVLC）を中心とするコミュニティで改良・管理している。

Caffe は C ++で実装され、GPU に対応し、高速に動作する[3]。畳み込みネットワーク（CNN）を得意とする Caffe は、画像認識に特化したライブラリである。設定ファイルによってニューラルネットワークの定義ができ、コーディングをせずにネットワークの実装・変更ができる。さらに、学習済のモデルを配布して実アプリケーションに組み込むことが出来る。

2.2 Deeplearning4j[4]

Deeplearning4j（以下 DL4J）は Java、Scala で記述されたオープソースの分散ディープラーニング・ライブラリである。DL4J は分散型で、GPU や大規模データの分散処理を実現するプラットフォーム Apache Spark などとの連携ができる。

DL4J はディープニューラルネットワークに特化しており、以下の手法がサポートされている。

- 制限付きボルツマンマシン（RBM）
- サポートベクターマシン（SVM）
- 畳み込みネットワーク（CNN）
- 再帰的オートエンコーダー（RAE）
- 再帰型ニューラルネットワーク（RNN）

2.3 Encog Machine Learning Framework[5]

Encog は Java、C/C ++および C# に対応した機械学習ライブラリである。ニューラルネットワークだけではなく、ベイジアンネットワーク、隠れマルコフモデル（HMM）、サポートベクターマシン（SVM）などの様々な学習アルゴリズムをサポートしている。

Encog はニューラルネットワークのためのデータの正規化や処理のためのサポートクラスも実装している。また、GPU に対応しており、高速な処理が期待できる。

[1]石巻専修大学理工学部
[2]岩手大学情報基盤センター
[3]岩手大学教育学部

2.4 Scikit-learn[6]

Scikit-learn（以下 Sklearn）は Python 用のオープンソースライブラリである。サポートベクターマシン（SVM）、ランダムフォレスト（RF）、勾配ブースティング（Gradient Boosting）、K 近傍法（k-NN）などの学習則とともに、回帰モデルの可視化、DBSCAN によるクラスタリング（クラス分類のためのクラスの抽出）などのアルゴリズム、データの前処理、モデルの調整や評価を行うための便利な関数が実装されている[7]。

2.5 TensorFlow[8]

TensorFlow は、Google が開発した人工知能のオープンソースライブラリである。データフローグラフを使用して複雑なニューラルネットワークを分かりやすく記述できる。Gmail、Google 検索や Google 翻訳など Google 社内のプロダクトでも使用されている。畳み込みニューラルネットワーク（CNN）、再帰型ニューラルネットワーク（RNN）などの深層学習アルゴリズムも実装されている。

また、GPU に対応しており、高速な処理を実現することができる。

表 1 ライセンス

	License
Caffe	2-clause BSD
Deeplearning4j	Apache 2.0
Encog	Apache 2.0
Sklearn	BSD
TensorFlow	Apache 2.0

3. ライブラリの比較

3.1 ライセンス

2 章で示したライブラリが採用しているライセンスを表 1 に示す。これらのライブラリは制約が少ないものを採用している。使用や頒布、修正、派生版の頒布、ライセンスの継承に関して制限がない。成果物の利用に関してはソースコードの開示を求めていないため、研究だけではなく商用利用も可能である。

3.2 言語のサポート状況

表 2 に、各ライブラリのサポートする言語を示した。DL4J は Java しかサポートしていない。Sklearn は Python をしかサポートしていない。他のライブラリは C/C ++などの複数の言語に対応している。

表 2 サポートする言語

	C/C++	*Java*	*Python*	*.net*
Caffe	○		○	
DL4J		○		
Encog	○	○		○
Sklearn			○	
TensorFlow	○	○	○	

3.3 実装されている機能の比較

表 3 前処理

	欠測値の対処	*正規化*	*次元削減*
Caffe	○	○	
DL4J	○	○	
Encog	○	○	
Sklearn	○	○	○
TensorFlow	○	○	○

機械学習を行う際に、空白や NaN（Not A Number）などの欠測値の対処、データ範囲を揃えるための正規化、データの特徴を維持したままでより少ない次元に落とし込む次元の削減などの前処理が必要となる。表 3 に各ライブラリの実装状況を示した。全てのライブラリで欠損値への対応およびデータの正規化を実装している。Sklearn と TensorFlow は主成分分析（PCA）などの次元の削減手法も実装している。

表 4 では、各ライブラリが実装する機械学習アルゴリズムを示した。上部はニューラルネットワーク、下部はそれ以外の学習アルゴリズムになる。アルゴリズムの名称は、付録を参照頂きたい。全体的に見ると、機械学習に用いるアルゴリズムの種類に関しては、Sklearn が最も多い。

ニューラルネットワークについて見てみると、Sklearn は多層パーセプトロン（MLP）だけに対

表 4　機械学習アルゴリズム

	C	D	E	S	T
MLP	○	○	○	○	○
CNN	○	○			○
RNN	○	○	○		○
AdaBoost			○	○	
ベイジアンネット		○	○		
決定木				○	
HMM			○	○	
K-NN		○		○	
ナイーブベイズ				○	
RBM		○	○	○	
SVM		○	○	○	○

C：Caffe，D：DL4J，E：Encog，S：Sklearn，
T：TensorFlow

応している。これと対照的に、Caffe、DL4J と TensorFlow は MLP を始め、畳み込みネットワーク、再帰型ニューラルネットワークなどの深層学習で用いられるニューラルネットワークに対応している。

深層学習の学習時間を短縮するため、並列度を上げるために GPGPU を利用する方法がある[9]。現状では、NVIDIA 社の GPU を利用するには CUDA[10] が主に用いられ、NVIDIA 社以外の GPU 等を利用するには OpenCL[11]が主に用いられる。

表 5 に、各ライブラリの対応状況を示した。現時点では、Sklearn は GPGPU の利用ができない。それ以外のライブラリは CUDA をサポートしており、NVIDIA 社の GPU も利用することができる。Caffe と Encog は OpenCL に対応しており、Nividia 以外の GPU も利用可能である。

なお、Nvidia 社の GPU では、OpenCL よりも CUDA の方が高速に動作する傾向がある。

表 5　GPGPUの対応

	CUDA	*OpenCL*
Caffe	○	○
DL4J	○	
Encog	○	○
Sklearn		
TensorFlow	○	

4. 計算機実験

本文では、機械学習ライブラリを評価するために手書き数字の画像データセット MNIST[1]を用いた．MNIST は、0 から 9 までの手書き数字が 28x28 ピクセルの画像から構成される（図 1）。訓練画像が 6 万枚、テスト画像が 1 万枚用意されている。

本稿では、まず、図 2 のように隠れ層を 1 層しか持たない階層型ニューラルネットワーク（MLP）を実装する。MNIST 画像に対応して入力層は 784 ノードを持つ。0 から 9 までの数字に対応して出力層は 10 ノードを持つ。隠れ層は 1000 ノードを持つ。隠れ層に活性化関数 ReLU、出力層に活性化関数 Softmax を用いる。次に、交差エントロピー損失関数を用いて、確率的勾配降下法（Stochastic Gradient Descent, *SGD*）[12] で

図 1　MNIST 画像データの例

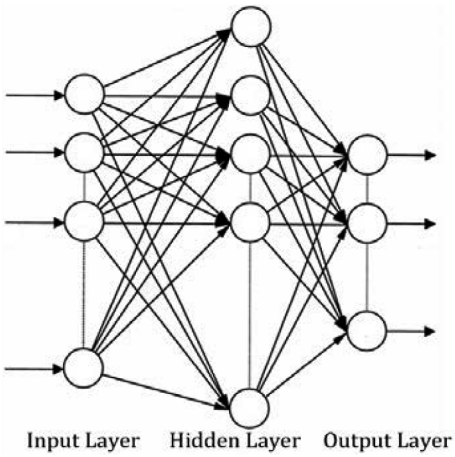


図 2　ニュートラルネットワークモデル

ニューラルネットワークを学習させる。最後に、1万枚のテスト画像を用いて、手書き数字の認識を行う。

4.1 Caffe による実装

Caffe では、図3のようにニューラルネットワーク構成、学習パラメータの設定は prototxt という設定ファイルに記述する。実装が簡単になる。その反面に、用意された layer しか使えない、損失関数の調整などのカストマイズができない。

4.2 DL4J による実装

図4のように DL4J のクラスを用いてニューラルネットワークを簡単に実装できる。活性化関数、損失関数などのパラメータも設定できる。

4.3 Encog による実装

Encog の正式リーリスしたバージョン 3.3 では、確率的勾配降下法がサポートされていない。

```
name: "MlpNet"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include { phase: TRAIN  }
  transform_param {  scale: 0.00390625 }
  data_param {
    source: "examples/mnist/mnist_train_lmdb"
    batch_size: 128
    backend: LMDB
  }
}
... （略）...
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

図3　prototxt ファイアの内容

```
MultiLayerConfiguration conf = new
  NeuralNetConfiguration.Builder()
    .seed(rngSeed)
    .optimizationAlgo(OptimizationAlgorithm.STOCHA
        STIC_GRADIENT_DESCENT)
    .iterations(1)
    .learningRate(0.006)
    .updater(Updater.NESTEROVS).momentum(0.9)
    .regularization(true).l2(1e-4)
    .list()
    .layer(0, new DenseLayer.Builder()
          .nIn(28 * 28)
          .nOut(1000)
          .activation("relu")
          .weightInit(WeightInit.XAVIER)
          .build())
    .layer(1, new OutputLayer.Builder(LossFunction.
           NEGATIVELOGLIKELIHOOD)
           .nIn(1000)
           .nOut(10)
           .activation("softmax")
           .weightInit(WeightInit.XAVIER)
           .build())
    .pretrain(false).backprop(true)
    .build();
MultiLayerNetwork model = new
  MultiLayerNetwork(conf);
```

図4　DL4J での Java ソースコード

本文では、確率的勾配降下法が追加された Encog のテストバージョン（V3.4）を用いてニューラルネットワークを実装した（図5）。DL4J による実装に比べて、Java のソースコード量が減少したが、損失関数の選択などの調整ができない。

4.4 Sklearn による実装

図6のように、Sklearn を用いて非常に短いソースコードでニューラルネットワークを実装できた。しかし、layer 毎で活性化関数の選択、出力層の損失関数の選択などニューラルネットワークのカスタマイズができない。

4.5 TensorFlow による実装

本文では、図7のように Python で TensorFlow を用いてニューラルネットワークを

```
BasicNetwork network = new BasicNetwork();
network.addLayer(new BasicLayer(null, true, 28*28));
network.addLayer(new BasicLayer(new
   ActivationReLU(), true, 1000));
network.addLayer(new BasicLayer(new
   ActivationSoftMax(), false, 10));
network.getStructure().finalizeStructure();
network.reset(rngSeed);
MLDataSet trainingDataset = trainingReader.getData();
final StochasticGradientDescent sgd = new
   StochasticGradientDescent(network, trainingDataset);
sgd.setLearningRate(0.006);
sgd.setMomentum(0.9);
sgd.setUpdateRule(new NesterovUpdate());
sgd.setBatchSize(128);
```

図 5　Encog での Java ソースコード

```
mlp = MLPClassifier(hidden_layer_sizes=(1000,),
   activation='relu', max_iter=15, alpha=1e-4,
   batch_size=128, solver='sgd', verbose=10, tol=1e-4,
   random_state=123, shuffle=True,
   learning_rate_init=.006)
```

図 6　Sklearn での Python ソースコード

```
import tensorflow as tf

x = tf.placeholder("float", [None, 784])
y = tf.placeholder("float", [None, 10])
w_h = tf.Variable(tf.random_normal([784, 1000],
   mean=0.0, stddev=0.05))
w_o = tf.Variable(tf.random_normal([1000, 10],
   mean=0.0, stddev=0.05))
b_h = tf.Variable(tf.zeros([1000]))
b_o = tf.Variable(tf.zeros([10]))

def model(X, w_h, b_h, w_o, b_o):
      h = tf.nn.relu(tf.matmul(X, w_h) + b_h)
      pyx = tf.nn.softmax(tf.matmul(h, w_o) + b_o)
      return pyx

y_hypo = model(x, w_h, b_h, w_o, b_o)
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y *
   tf.log(y_hypo), reduction_indices=[1]))
L2_sqr = tf.nn.l2_loss(w_h) + tf.nn.l2_loss(w_o)
lambda_2 = 1e-4
loss = cross_entropy + lambda_2 * L2_sqr
optimizer = tf.train.MomentumOptimizer(0.006, 0.9,
   use_nesterov=True).minimize(loss)
```

図 7　TensorFlow での Python ソースコード

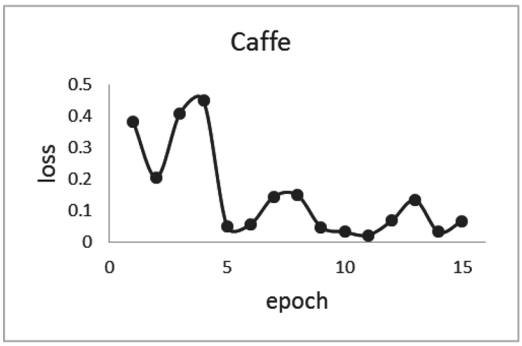
実装した。Sklearn による実装に比べて、ソースコードの量がかなり増えた。TensorFlow は調整可能な項目が多いという特徴がある：様々なニューラルネットワークの実装に対応している、layer 毎で活性化関数の選択が可能、出力層の損失関数の定義。

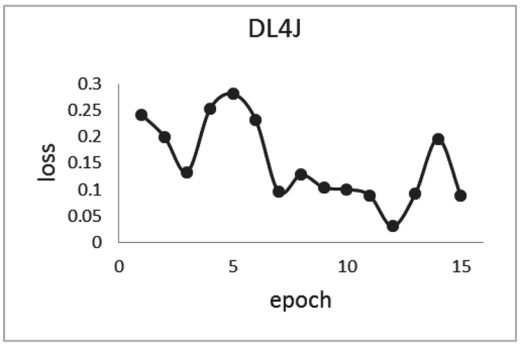
5. 評価

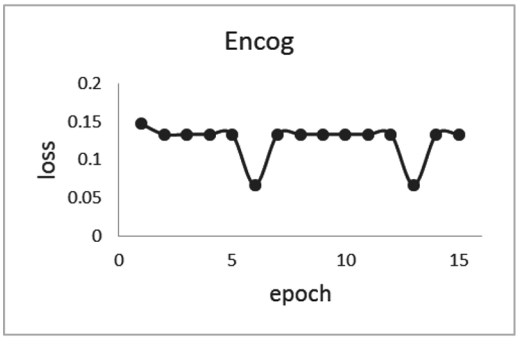
ライブラリを評価するために、手書き文字認識の実験を行った。認識・テスト用のデータとしては、文献［1］のデータを用いた。学習では、各ライブラリに実装したニューラルネットワークで 15 エポック学習する。エポック（Epoch）とは学習の単位を表す。1 エポックの学習は 6 万枚の MNIST 画像データをすべて学習したことに対応する。1 万枚の MNIST テスト画像を用いて、学習済ネットワークの手書き数字の認識を行い、認識精度を確認する。

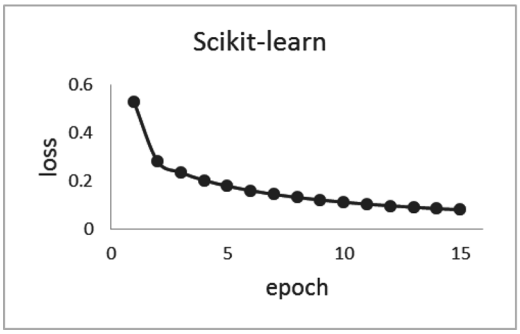
5.1　計算機実験の環境

CPU	Intel Core i7 3.4GHz
メモリ	8GB
OS	Window 10 Ubuntu 14.04（Caffeのみ）









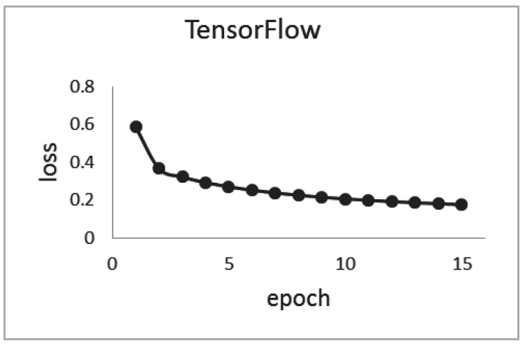


図 8　損失関数の値の推移

5.2　学習

図 8 は各ニューラルネットワークにおける、損失関数の値の推移を表す。確率的勾配降下法によってネットワークの重みパラメータを更新し、訓練画像データに対する損失関数を計算する。

Encog 以外は、学習の回数が進むにつれて損失関数の値が暫減傾向にあり、学習がうまくいっていることが分かる。これに対して Encog では、損失関数の値があまり変わらない、また、次節で示すテスト画像の認識率も悪いため（表 6）、学習自体が失敗したことが分かる。

5.3　評価結果

表 6　学習時間と認識率

	学習時間 (S)	*認識率*
Caffe	171	96.66%
DL4J	175	97.11%
Encog	15	11.29%
Sklearn	270	97.31%
TensorFlow	98	97.24%

学習用データの訓練は計算量が多く計算時間がかかるため、実装したニューラルネットワークを 15 エポック学習するのに要した時間を測定した。これは、6 万枚の学習用の 784 pixel の数字画像（1 つの数字あたり六千枚の画像がある）を 15 エポック学習する時間である。

学習時間と評価用データの認識率を表 6 に示す。Encog が最も学習時間が短くなっているが、これは、表 6 の認識率も低いことから学習に失敗しているためと推察される。Encog を除くと、TensorFlow の学習時間は 98 秒で最も短い。これに対して Sklearn の学習時間は最も長く 270 秒であった。計算時間から、これらの機械学習ライブラリは並列処理などの高速化手法を用いていることが示唆される。特に、DL4J と TensorFlow は Java、Python で実装されているが、並列化処理等を用いているためか処理速度は非常に高速であった。

評価用画像の認識率は、学習に失敗したと思われる Encog 以外は約 97% となった。MNIST の数字画像の分類を行うことが出来たと考えられ

る。

6. まとめ

深層学習等にも対応している機械学習ライブラリの評価を行った。各ライブラリの対応している機能と言語について示した後、数値画像のセットである MNIST を用いて機械学習を行い評価した。

学習に要する時間を計測した結果、6 万枚の画像を 15 エポック学習する時間で、最も遅いライブラリで 270 秒であった。

Encog 以外のライブラリは学習用データの学習に成功し、学習には用いていない評価用データを約 97％で認識・分類できた。

本稿では MNIST の数値画像データだけで評価しているため、他のデータセットでは異なる結果になる可能性があることを指摘し、まとめとする。

参考文献

［1］ Lecun, Yann, Corinna Cortes, and Christopher JC Burges: The MNIST database of handwritten digits, *http://yann. lecun. com/exdb/mnist*

［2］ Caffe: http://caffe.berkeleyvision.org/

［3］ angqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In Proceedings of the 22nd ACM international conference on Multimedia,2014

［4］ Deeplearning4j: https://deeplearning4j.org

［5］ Encog: http://www.heatonresearch.com/encog/

［6］ Scikit-learn: http://scikit-learn.org

［7］ Raschka, Sebastian: *Python machine learning*. Packt Publishing Ltd, 2015.

［8］ TensorFlow: https://www.tensorflow.org/

［9］ Zastrau, David, and Stefan Edelkamp: Stochastic gradient descent with gpgpu. *Annual Conference on Artificial Intelligence*. Springer Berlin Heidelberg, 2012.

［10］ Nvidia, C. U. D. A. : Compute unified device architecture programming guide. 2007.

［11］ Munshi, Aaftab: The opencl specification. *Hot Chips 21 Symposium (HCS)*. IEEE, 2009.

［12］ Qian, Ning: On the momentum term in gradient descent learning algorithms. Neural networks 12.1, 1999

付録（機械学習アルゴリズム一覧）

- AdaBoost　ブースティング・アルゴリズム
- Bayesian network　ベイジアンネットワーク
- Convolutional Neural Network（CNN）　畳み込みニューラルネットワーク
- Density-Based Spatial Clustering（DBSCAN）　密度に基づくクラスタリング手法
- Decision Tree　決定木
- Gradient Boosting　勾配ブースティング
- Hidden Markov Model（HMM）　隠れマルコフモデル
- k-Nearest Neighbor method（k-NN）　K 近傍法
- Logistic Regression　ロジスティック回帰
- Multi-Layer Perceptron（MLP）　多層パーセプトロン
- Naive Bayes　ナイーブベイズ分類器
- Random Forest（RF）　ランダムフォレスト
- Recurrent Neural Network（RNN）　再帰型ニューラルネットワーク
- Restricted Boltzmann Machine（RBM）　制限付きボルツマンマシン
- Stochastic Gradient Descen（SGD）　確率的勾配降下法
- Self-Organizing Map（SOM）　自己組織化マップ
- Support Vector Machine（SVM）　サポートベクターマシーン