ディープラーニングツールを使いこなそう(3)

my recommendations on research & development tools.

山下隆義†

キーワード：ディープラーニング, TensorFlow, MNIST

1. まえがき

TensorFlowは、Googleが公開したディープラーニングフレームワークである(現時点でのバージョンは0.9)¹⁾²⁾。TensorFlowには行列演算やディープラーニングに関連する関数だけでなく、画像を補正したり反転したりするような画像処理の関数なども用意されている。これらを組合せて、目的にあったアルゴリズムを実装することができる自由度の高さが特長となっている。また、実装したプログラムは、組み込み機器から単一の計算機、さらには大規模な分散環境で実行することもできる。TensorFlowの基本的な考え方は、全体を有向グラフとして表現し、そのグラフのノードにシンボルとして定義した変数や処理を割当てていく。そして、セッションを生成し、セッションに作成したグラフを指定して実行するような記述方法となっている。TensorFlowのコアな部分はC++で記述されて高速化が図られている。TensorFlowの関数のインターフェースには、C++版とともにpython版も用意されている。ここでは、python版を中心に説明する。

1.1 TensorFlowのインストール

TensorFlowのインストール

Ubuntu (CPU のみの場合)

```
sudo pip install --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.6.0-cp27-none-linux_x86_64.whl
```

Ubuntu (GPU を利用する場合)

```
sudo pip install --upgrade https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.6.0-cp27-none-linux_x86_64.whl
```

TensorFlowを利用するには、python(バージョン2.7またはバージョン3.3以上)とpip、GPUを利用する場合はCUDA Toolkit 7.0とcuDNN V4をあらかじめインストールしておく。TensorFlowは、以下のようにpipコマ

ンドでインストールができる。また、TensorFlowをGitHubに公開されているソースコードからインストールすることもできる(<https://github.com/tensorflow/tensorflow>)。

1.2 TensorFlowによる学習～ソフトマックス回帰～

TensorFlowの使い方は、英語のチュートリアルが公式ページに公開されており、初心者のためのチュートリアルが「MNIST for ML Beginners」³⁾にある。本稿では、このチュートリアルをもとに、ソフトマックス回帰による手書き文字認識について説明する。手書き文字認識に用いるMNISTデータセット⁴⁾のダウンロードとTensorFlowで扱う形式に変換するプログラムが公開されているので、あらかじめinput_data.py⁵⁾を作成しておく。

次に、図1のようなソフトマックス回帰のプログラムを作成する。TensorFlowで用意されている関数を利用するために、tensorflowをインポートする。これをtfとする。また、先ほど説明したinput_data.pyもインポートしておく。最初に、MNISTデータセットを読み込んで、学習データと評価データに分ける。この処理の詳細はinput_data.pyに記述されている。学習するモデルに入力する学習データはplaceholder関数、モデルのパラメータとなる結合重みやバイアスは、Variable関数を利用して領域を用意する。placeholderには、その変数の型と次元数を指定する。この型と次元数は宣言した後で、基本的には変更しないように注意する。Variable関数にはそのサイズを指定する。ここでは、結合重み W に入力次元数と出力次元数を与えている。出力はソフトマックス関数を利用して求める。ソフトマックス関数には、入力 x と結合重み W の内積をmatmulで求めた値とバイアス b の値を与えている。教師信号も学習データと同様にplaceholder関数により領域を用意しておく。placeholder関数内のNoneは、この時点ではサイズが不定の場合に利用する。そして、誤差を求める処理をGradientDescentOptimizerを利用で行う。ここまでは、学習データと教師信号、モデルのパラメータのサイズ、それらを利用して出力を求める式と更新方法を定義であり、まだ学習処理を実際には行っていない。ここから、まずinitialize_all_variables関数でパラメータの初期化を行う。そして、学習するセッションを作

† 中部大学 工学部 情報工学科

"Let's Start Deep Learning with Efficient Tools (3)" by Takayoshi Yamashita (Department of Computer Science, College of Engineering, Chubu University, Aichi)

```
#TensorFlowのライブラリをインポート
import tensorflow as tf
import input_data

# MNISTデータ読み込み、学習データと評価データに分ける
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

#学習データx, 結合重みW, バイアスbの領域を用意
x = tf.placeholder("float", [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

#xとWの内積をソフトマックス関数に与えて出力yを求める
y = tf.nn.softmax(tf.matmul(x, W) + b)

#教師信号yの領域を用意
y_ = tf.placeholder("float", [None, 10])

#クロスエントロピーを求める
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))

#勾配法によりクロスエントロピー誤差を最小化する
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

#すべてのパラメータを初期化する
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

#学習の繰り返し処理
for i in range(1000):
    #学習データ(ミニバッチ)を選択
    batch_xs, batch_ys = mnist.train.next_batch(100)
    #学習処理
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

#評価を行い、教師信号と一致するか確認
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
#精度を求める
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels})
```

図1 ソフトマックス回帰

成する。for文で指定された回数(1,000回)だけ繰り返し学習処理を行う。学習処理はnext_batchで学習サンプルを選択し、run関数に選択した学習サンプルと教師信号を与えて学習を行う。

評価結果は、argmax関数で出力と教師信号が一致しているかどうか確認し、一致している個数を取得する。一致している個数をもとに、精度を求めている。このファイルをsoftmax_regression.pyとし、以下のように実行する。

ソフトマックス回帰の実行

```
$ python softmax_regression.py
```

実行すると91%程度の精度が得られていることがわかる。

1.3 TensorFlowによる学習

～畳み込み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による手書き文字認識について説明する。複数の畳み込み層と全結合層を持つ場合、毎回同じようなコードを書くことはコードが冗長になり、バグが生まれたり、可読性が悪くなる。一般的なプログラミングと同様に、何度も行う畳み込み層や全結合層に関連する処理の関数化を行い、コードの簡素化をはかる。

図2に畳み込みニューラルネットワークのコードの最初の部分を示す。まず、結合重みとバイアスの初期化を行う関数weight_variable(shape)とbias_variable(shape)

```
import input_data
import tensorflow as tf
import time

#結合重みの初期化
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

#バイアスの初期化
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

#畳み込み
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

#プーリング
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')

#MNISTデータセットの読み込み
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
#データセット、教師信号の領域を用意
x = tf.placeholder("float", shape=[None, 784])
y_ = tf.placeholder("float", shape=[None, 10])

#データセットを形式を変更(バッチサイズx28x28xチャンネル数)
x_image = tf.reshape(x, [-1, 28, 28, 1])
```

図2 畳み込みニューラルネットワーク (その1)

を定義する。truncated_normal関数は、指定された標準偏差(stddev)をもとに乱数を生成して出力する関数である。ここでは、それらをVariable関数に与えている。

畳み込み層の構成を定義する関数がconv2d(x, W)である。引数で与えられたxが入力データ、Wがフィルタとなる。これらをconv2d関数に与える。それぞれ4次元で定義するが、入力データは[バッチサイズ、高さ、幅、チャンネル数]の順に並んでおり、フィルタは[フィルタ高さ、フィルタ幅、入力チャンネル数、出力チャンネル数]の順に並んでいる。畳み込み処理を行う間隔をstride、パディングの有無をpaddingに指定する。strideは、入力データと同様の4次元である。もし、strideを2にしたい場合は、[1,2,2,1]とする。パディングありの場合は、paddingをSAME、なしの場合はVALIDを指定する。

プーリングの関数はmax_pool_2x2(x)に定義する。引数のxは4次元の入力データで、[バッチサイズ、高さ、幅、チャンネル数]の順に並んでいる。プーリングのサイズをksize、間隔をstride、パディングの有無をpaddingに指定する。ksizeとstrideは4次元で入力データと同様に並びは[バッチサイズ、高さ、幅、チャンネル数]の順となっている。

次に、データセットに関連する処理を記述する。前節のソフトマックス回帰と同様に、input_data.pyの関数を利用してMNISTデータセットを読み込む。そして、入力データx、教師信号yの領域をplaceholder関数を利用して用意する。入力データxは、[バッチサイズ(None)、データ次元(784)]の2次元なので、reshape関数を利用して4次元([-1, 28, 28, 1])に変換する。

```
#畳み込み層1層目
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

#活性化関数およびプーリング
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

#畳み込み層2層目
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

#活性化関数およびプーリング
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

#全結合のパラメータ領域を用意
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

#全結合層
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

#ドロップアウト
keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

#全結合のパラメータ領域を用意
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

#ソフトマックス関数
y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

#誤差関数
cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
```

図3 畳み込みニューラルネットワーク (その2)

ネットワークの構成を定義する部分を図3に示す。まず、畳み込み層1層目のパラメータの領域 W_{conv1} と b_{conv1} を用意する。ここで、さきほど定義した `weight_variable` 関数と `bias_variable` 関数を利用している。`weight_variable` 関数には、[フィルタ高さ、フィルタ幅、チャンネル数、フィルタ数]の4次元のテンソル、`bias_variable` 関数には、[バイアス数]の値を与えている。そして、`conv2d` 関数に入力データ x_{image} と畳み込みのパラメータ W_{conv1} 、およびバイアス b_{conv1} を与えて、畳み込み処理を行い、その結果を活性化関数である `relu` 関数に渡している。その結果が h_{conv1} となっている。 h_{conv1} はさらに `max_pool_2x2` に与えられて、プーリングを行った結果が h_{pool1} となっている。畳み込み2層目も同様に、パラメータ領域の用意した後、畳み込み結果を活性化関数に渡し、そしてプーリングを行う。ここでも定義した関数を利用しているが、入力チャンネル数と出力チャンネル数が異なることに注意する。

全結合層のパラメータ領域も `weight_variable` 関数と `bias_variable` 関数を利用して用意する。全結合層のパラメータは[入力ユニット数、出力ユニット数]の2次元である。`weight_variable` 関数は次元数が異なっても利用できるようになっている。前層のプーリング結果は4次元なので、全結合層に入力できるように `reshape` 関数を利用して2次元に変換する。この2次元は[バッチサイズ、入力データサイズ]の並びになっている。全結合層の演算は `matmul`

```
#学習方法
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

#評価方法
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

#学習するためのセッションを生成
sess = tf.Session()

#パラメータの初期化
sess.run(tf.initialize_all_variables())

st = time.time()

#学習の繰り返し処理
for i in range(1000):
    #学習データセット(ミニバッチ)を選択
    batch = mnist.train.next_batch(50)
    #学習処理
    _, loss_value = sess.run([train_step, cross_entropy],
                             feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

    #評価
    if i % 100 == 0:
        acc = sess.run(accuracy,
                        feed_dict={x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0})
        print "Test accuracy at step %s: %s" % (i, acc)

print "elapsed time %f [s]" % (time.time() - st)

#評価
acc = sess.run(accuracy,
                feed_dict={x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0})
print "Test accuracy %s" % acc
```

図4 畳み込みニューラルネットワーク (その3)

関数を利用して行う。演算結果を `relu` 関数に渡して、その結果を h_{fc1} とする。ドロップアウトを行うために、値を保持するかどうかを決める各ユニットの確率 `keep_prob` の領域を `placeholder` 関数で用意する。その確率と全結合層の結果を引数として `dropout` 関数に与える。出力層のパラメータ領域は、1つ前の全結合層と同様に用意する。そして、最後に前層の結果と出力層のパラメータによる演算結果を `softmax` 関数に与えて、出力値を求める。誤差関数にはクロスエントロピー誤差を定義する。

ここまでで定義したネットワークの構成を利用して学習を行う。このコードを図4に示す。最初に学習方法を決める。ここでは、Adamを利用する。`AdamOptimizer` の引数には学習率を与えて行う。

評価は、`argmax` 関数で確率が最大となった出力クラスと教師信号を取得し、それらが一致しているかを `equal` 関数で判定する。`correct_prediction` は、すべての評価データに対する結果がベクトルとして保持しているので、その平均を求めて精度としている。

学習を行うために、まずセッションを `Session` 関数を用いて生成する。そして、すべてのパラメータを初期化する `initialize_all_variables` 関数をセッションの `run` 関数に与えて実行する。学習の繰り返し処理は `for` 文の中で行う。まず、50枚の学習データをミニバッチとして `next_batch` 関数で選択する。実行したい内容はセッションの `run` 関数に与えて、繰り返し行う。ここでは、Adamを利用して誤差を逆

伝播するtrain_stepと誤差を求めるcross_entropyを実行している。実行に用いる学習データをfeed_dictに与える。また、ドロップアウトの確率は、keep_probに指定する。run関数の戻り値は、実行した内容の戻り値になる。ここでは、2つの内容を実行しており、train_stepの戻り値は受け取らず、cross_entropyの戻り値をloss_valueに代入している。loss_valueの値をprint関数を利用して画面出力すれば、学習状況を確認することができる。学習の繰り返し回数は2万回であるが、100回ごとに評価データセットに対する精度を求めている。学習が終了すると、最後にもう一度テストデータに対する評価が行われ、99.2%程度の精度になる。

TensorFlowによる実装は、ネットワークを構成する要素と誤差関数をシンボルとして定義し、誤差関数を学習方法、出力を評価方法に与える記述方法になる。後はそれらを学習の繰り返し処理の中で呼び出すことで学習と評価を行うことができる。TensorFlowのサンプルコードには、CIFAR-10データセットの画像認識やILSVRC2014で優勝したGoogLeNet、Recurrent Neural Networkによる英文の文字予測や英語－フランス語の翻訳など、さまざまなコードが含まれている。大規模なネットワークを実装したい場合は、これらのサンプルコードを参考にすると良い。

1.4 TensorBoardによるネットワークの可視化

TensorFlowには、TensorBoardという学習するネットワーク構成や学習状況を可視化するツールがある。TensorBoardを利用して可視化するためには、可視化したい内容をファイルに出力するようにコードを追加する。図5に畳み込みニューラルネットワークのコードに可視化するためのコードを追加した最初の部分を示す。可視化のために追加したコードを赤字で示している。最初にGraphクラスのas_default関数を利用してグラフを生成する。グラフにはネットワーク構成や学習状況を記録させることができる。また、入力データと教師信号に名前をつけることができ、ここでは、それぞれInputとGroundTruthとしている。

図6は、ネットワーク構成と誤差関数のコードである。ここでは、出力のソフトマックス関数にOutputという名前をつける。誤差関数には、学習状況を出力するscalar_summary関数を利用してクロスエントロピー誤差を出力させるようにする。

学習のメインコードを図7に示す。学習方法と評価方法について、名前をそれぞれtrain、testとつけておく。評価結果であるaccuracyをscalar_summary関数を利用して出力するようにする。学習状況を出力するための設定として、最初にmerge_all_summaries関数を利用して、設定したすべての出力を統合させる。また、学習状況を保存するSummaryWriterクラスを生成する。SummaryWriterクラスには、出力するディレクトリ(MNIST_data)と出力す

```
import input_data
import tensorflow as tf
import time

#結合重みの初期化
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

#バイアスの初期化
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

#畳み込み
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

#プーリング
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')

#MNISTデータセットの読み込み
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
with tf.Graph().as_default():
    #データセット、教師信号の領域を用意
    x = tf.placeholder("float", shape=[None, 784], name="Input")
    y_ = tf.placeholder("float", shape=[None, 10], name="GroundTruth")

    #データセットを形式を変更(バッチサイズx28x28xチャネル数)
    x_image = tf.reshape(x, [-1, 28, 28, 1])
```

図5 可視化コードを追加した畳み込みニューラルネットワーク(その1)

```
#畳み込み層1層目
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

#活性化関数およびプーリング
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

#畳み込み層2層目
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

#活性化関数およびプーリング
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

#全結合のパラメータ領域を用意
W_fc1 = weight_variable([7 * 64, 1024])
b_fc1 = bias_variable([1024])

#全結合層
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

#ドロップアウト
keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

#全結合のパラメータ領域を用意
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

#ソフトマックス関数
with tf.name_scope("Output") as scope:
    y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

#誤差関数
with tf.name_scope("xentropy") as scope:
    cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
    ce_summ = tf.scalar_summary("cross entropy", cross_entropy)
```

図6 可視化コードを追加した畳み込みニューラルネットワーク(その2)

るネットワーク構成を指定する。ここでは、クラス生成時にネットワーク構成を保存するようにしているが、

```
# 学習方法
with tf.name_scope("train") as scope:
    train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

# 評価方法
with tf.name_scope("test") as scope:
    correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_1,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    accuracy_summary = tf.scalar_summary("accuracy", accuracy)

# 学習状況出力するための設定
summary_op = tf.merge_all_summaries()
summary_writer = tf.train.SummaryWriter('MNIST_data', graph_def=sess.graph_def)

# 学習するためのセッションを生成
sess = tf.Session()

# パラメータの初期化
sess.run(tf.initialize_all_variables())

st = time.time()
# 学習の繰り返し処理
for i in range(1000):
    # 学習データセット(ミニバッチ)を選択
    batch = mnist.train.next_batch(50)
    # 学習処理
    _, loss_value = sess.run([train_step, cross_entropy],
                             feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
    if i % 100 == 0:
        result = sess.run([summary_op, accuracy],
                           feed_dict={x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0})
        summary_str = result[0]
        acc = result[1]
        summary_writer.add_summary(summary_str, i)
        print("Test accuracy at step %s: %s" % (i, acc))
    print "elapsed time %f [s]" % (time.time() - st)

# 評価
acc = sess.run(accuracy,
               feed_dict={x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0})
print "Test accuracy %s" % acc
```

図7 可視化コードを追加した畳み込みニューラルネットワーク (その3)

graph_defに指定しなくても良い。ネットワーク構成を保存する場合は、SummaryWriterクラスのadd_graph関数を利用する。

学習の繰り返し処理において、100回ごとにrun関数を実行する。run関数には、学習状況出力するsummary_opと評価を行うaccuracyを与えている。この2つの処理を実行し、その結果をresultに代入する。summary_strには1つ目の処理のsummary_opの結果、accにはaccuracyの結果が代入される。そして、SummaryWriterクラスのadd_summary関数を利用して、summary_strの内容を出力する。scalar_summary関数を利用すると、設定したクロスエントロピー誤差と精度を学習状況として出力できる。

学習が終わったら、保存したネットワーク構成と学習状況を確認する。SummaryWriterクラス生成時に指定したディレクトリに、イベントファイルが保存されている。ここでは、MNIST_dataを保存先に指定したので、以下のように実行する。--logdirで指定するディレクトリはフルパスなので注意する。

TensorBoardの起動

```
$ tensorboard -logdir /path/to/MNIST_data/Starting
TensorBoard on port 6006 (You can navigate to http://localhost:6006)
```

TensorBoardを起動したら、ブラウザを利用してアクセスする。ブラウザにhttp://localhost:6006と入力すると、図8のようなメイン画面が表示される。ここで、accuracyをクリックすると、図9のような繰り返し回数ごとの精度の変化を確認することができる。一方、cross entropyをクリックすると、繰り返し回数ごとのクロスエントロピーの変化を確認することができる。学習のログを保存する際に、TensorFlowで用意されているクラスを活用することで、TensorBoard上でまとめて確認することができるのは便利な機能の一つである。

次に、ネットワーク構成を可視化する。メイン画面の右上にあるGRAPHをクリックすると、図10のようなネットワーク構成を可視化する画面が表示される。ネットワークのノードをクリックすることで、そのノードの属性や入出力関係を確認することができる。また、定義したネットワークが意図した通りの構成になっているかも確認できる。

図8 TensorBoardのメイン画面

図9 TensorBoardによる精度の確認

図10 TensorBoardによるネットワーク構成の確認

2. むすび

本稿では、TensorFlowのインストール方法から、多層パーセプトロン、畳み込みニューラルネットワークの学習および評価方法について紹介した。また、TensorBoardを利用した可視化方法についても紹介した。TensorFlowは、後発のディープラーニングツールにでありながら、利

用者数は増大しており、サンプルコードも充実している。画像認識以外にも自然言語処理や時系列データへの応用も可能である。それらの情報は、WebサイトやGithubを参考にしてほしい。

(2016年8月10日受付)

〔文 献〕

- 1) <https://www.tensorflow.org>
- 2) M. Abadi, et al.: "TensorFlow: Large-scale machine learning on heterogeneous distributed systems", arXiv preprint arXiv:1603.04467 (2016)
- 3) <https://www.tensorflow.org/versions/r0.9/tutorials/mnist/beginners/index.html>
- 4) Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: "Gradient-based learning applied to document recognition", Proceedings of the IEEE, 86, 11, pp.2278-2324 (Nov. 1998)
- 5) https://tensorflow.googleusercontent.com/tensorflow/+master/tensorflow/examples/tutorials/mnist/input_data.py

やまおか たかよし
山下 隆義 2002年、奈良先端科学技術大学院大学博士前期課程修了。同年、オムロン(株)入社。2011年、中部大学情報工学専攻博士後期課程修了。2014年、同大学工学部情報工学科講師。画像認識・ディープラーニングに関する研究に従事。工学博士。