

図3 Qt Designer新規作成画面

●ステップ4…画面を作る

PyQt4とQt Designerを使って画面を作ります。文献(2)のウェブ・サイトを参考にしました。

Qt Designerを起動してください。図3のように「Main Window」を選択して「作成」ボタンを押します。

図4のように左のウィジェットボックスからMainWindowへ、ドラッグ＆ドロップでGUIの部品を配置していきます。「Push Button」を5つ、結果を表示するための「Text Edit」を1つ並べます。

次に右のオブジェクトインスペクタ画面で部品を選択します。これでプロパティエディタの「objectName」の値を入力することで、オブジェクトの名前を変更できます。各ボタンの機能が分かりやすいように画面にならって名前を変更してください。

さらに、プロパティエディタの「text」の値を変更することで、ボタンの上に表示されるテキストを変更できます。各ボタンの機能が分かりやすいように画面にならって名前を変更してください。

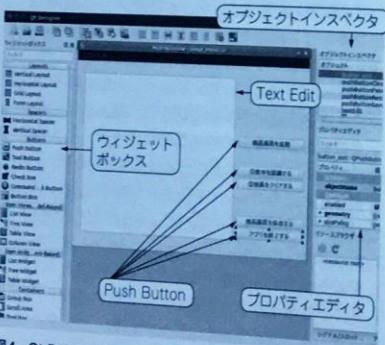


図4 Qt DesignerのMainWindow作成画面

次に、左上の保存ボタンにより、ファイル名「deep_minist.ui」として保存してください。deep_minist.uiを作った後、convert_qt.pyを実行して、deep_minist.pyを生成し、Pythonで使えるようにします。

```
$ python convert_qt.py
```

作成した画面を使うためのモジュールdeep_minist.pyが自動生成されます。以上でメイン画面の作成は終わりです。

●ステップ5…マウス・カーソルで描画できる画面を作る

PyQt4ライブラリには画面を作るためのQWidgetクラスがあります。

文献(3)のウェブ・サイトを参考にQWidgetクラスを拡張した描画画面である、PaintedWidgetクラスを作ります。

作成するモジュール・ファイルはwidget_painted.pyです。これにより真っ白なPaintedWidget画面を用意し、マウスでドラッグしている間に線を描画できます。

●ステップ6…TensorFlowを用いて手書き数字の認識処理を行う関数を作る

model_creator.pyを参考に手書き数字の認識処理を行う関数を作ります。作成するモジュールはmnist_recognizer.pyです。

▶main.pyの変更点[文献(4)に全ソースコード掲載]

23行目: import mnist_recognizer

このモジュールはimportされた時点で、あらかじめ作られたモデル(ckptファイル)を読み込みます。そして、28*28画像を引き数にmnist_recognizer()関数を使うことにより、認識した数字を返すことができます。

以下にmnist_recognizer.pyのコードの特に重要な部分を示します。

▶mnist_recognizer.py
82(行目): saver = tf.train.Saver()
84:saver.restore(sess, "model.ckpt")

ckptファイルを読み込みます。この行の前でmodel_creator.pyと全く同じモデルを定義する必要があります。

86: def mnist_recognizer(image):
28*28の画像を引き数に、このmnist_recognizerメソッドを呼び出すと、戻り値に認識した結果である数値(0~9)を返します。

●ステップ7…これまで作成したモジュールを呼び出すメイン・プログラムを作る

メインのプログラムを作成します。「②数字を認識する」ボタンが押されるとMainWindowクラスのrecognize_number()関数の処理が行われます。

以下にmain.pyのコード特に重要な部分を示します。

▶main.py[文献(4)に全ソースコード掲載]

22(行目): import deep_mnist

ステップ3で作成した画面を取り込みます。

```
92: def recognize_number(self):  
99: self.image = cv2.flip(self.  
    image, 0)  
101: self.image = ndimage.  
    rotate(self.image, 270)  
102: # cv2.imwrite("out_put.bmp",  
    self.image)
```

99行目と101行目で自分の描いた画像を上下回転と時計回り回転を行っています。PaintedWidgetクラスで扱う画像とNumPyで扱う画像はx軸とy軸が入れ替わっている関係にあるためです。ここではマウスで数字を描画したときに得た画像のピクセルをNumPyで扱うために並べ直しています。

103: # 膨張処理

```
104: image_dilation = cv2.  
    dilate(self.image, kernel,  
    iterations = 1)  
105: cv2.imwrite("dilation.bmp",  
    image_dilation)
```

106: # 縮小処理

```
107: self.mnist_image = cv2.  
    resize(image_dilation, mnist_size)  
108: cv2.imwrite("mnist.bmp", self.  
    mnist_image)
```

```
109: # 2値化で強調。BINARY_INVで反転し、白  
    背景で黒文字にする  
110: ret, thre_image = cv2.  
    threshold(self.mnist_image, 5,  
    255, cv2.THRESH_BINARY_INV)
```

ペイントした画像をMNISTの画像のように縮小する前処理を行います。この処理ではモデルが認識しやすいように画像を加工することが求められます。今回は膨張処理、縮小処理、2値化による強調処理を行っています。この作業により、今回用いたMNISTデータセットと同じような数字の画像が得られます。少しボケて、太い文字になります。

```
111: cv2.imwrite("thre_image.  
    bmp", thre_image)
```

前処理が終った後、111行目で画像を保存してい

ます。数字を認識するたびに保存しているので、どんな画像になっているか見てみてください。

```
162: form.pushButtonPaintImage.  
    clicked.connect(lambda: form.  
    paint_image())
```

```
163: form.pushButtonSaveImage.clicked.  
    connect(lambda: form.saved())
```

```
164: form.pushButtonRecognizeNumber.  
    clicked.connect(lambda: form.  
    recognize_number())
```

```
165: form.pushButtonClear.clicked.  
    connect(lambda: form.clear_  
    points())
```

162行目から165行目の記述をすることにより、作成した画面ボタンとMainWindowクラス内の関数を結びつけています。これにより、ボタンが押されると対応した関数が呼び出され処理が行われます。分かりやすく書くと次のようになります。

```
form.[ボタン名].clicked.connect(lambda:  
    form.[MainWindow内の関数名]())
```

●ステップ8…実行して動作を確認する

ターミナルで以下のコマンドを実行し、動作を確認してください。

```
$ python main.py
```

* * *

今日は簡単なGUIアプリケーションを作成しました。気軽に機械学習を試せるので周りの反応も上々です。みなさんもぜひ周りの方に試してもらってみてください。

次はチュートリアルのモデルではなく、自分でTensorFlowモデルを作成してGUIに実装したいと思っています。顔認識の画像処理、音響処理など…機械学習、夢が広がります。

参考文献

- (1) Deep MNIST for Experts.
<https://www.tensorflow.org/versions/r0.11/tutorials/mnist/pros/index.html>
- (2) python + PyQt4 + Qt DesignerでGUIアプリケーション。
<http://domnikki.hateblo.jp/entry/2016/04/07/005006>
- (3) PyQtでのグラフィックスその1(QWidgetに直接描画する)。
<http://bravo.hatenablog.jp/entry/2016/02/07/084048>
- (4) 著者提供のプログラム。
<https://github.com/jintaka1989/DeepMnist GuiAppForMagazine/>

たかぎ・さとし

第2特集

ネットから入手できる画像データセットで試す

第4章

ネット

レッスン4… ちょっと本格的なAI画像認識

山本 大輔

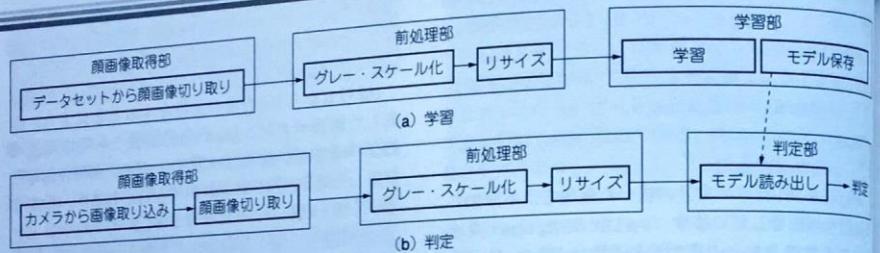


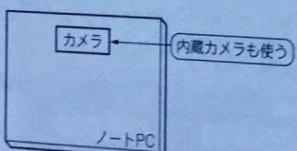
図1 AI画像(顔)認識処理のフロー

ディープ・ラーニングは、特に画像認識の場面で成果をあげています。今回はTensorFlow⁽¹⁾を使った顔画像認識に挑戦します。顔認識を応用すると、画像に映っている人や物が、データベースに登録されている人や物かどうかを自動で判断できます。

応用は無限にあります。例えば以下が挙げられます。

- これから会う人がタイプかどうか
- 自分のタイプの人/そうでない人の画像を多数学習させます。
- 目の前の魚が新鮮かどうか
- 新鮮な魚とそうでない魚の画像を学習させます。
- 見たことがない観光地を自動選別
- 訪れたことがある場所の画像を学習させます。
- 自分には分からぬ双子の兄弟を判別
- 兄の画像/弟の画像を学習させます。

今回は公開されているデータセットを利用し、ジョージ・W・ブッシュ氏に似ている人を探してみます。ブッシュ氏は2001年～2009年に米国の大統領に就任していた人です。

図2 カメラ内蔵PCで実験できる
MacBook Pro, Core i7, 2.8GHz, RAM: 16GB/バイト

概要

● 处理のフロー

今回の顔認識プログラムで行う機能は5つに分かれます。処理フローを図1に示します。また、ハードウェア構成を図2に示します。

本稿のおおまかな流れは以下です。

- ステップ1…顔画像取得：顔画像データを収集します。
- ステップ2…画像の前処理：顔抽出を行います。
- ステップ3…学習モデル構築：学習をどのようなルゴリズムで行うか決めます。
- ステップ4…学習の実行：学習済みモデルを生成します。
- ステップ5…判定：学習済みモデルを取り込んで、新たに取得した画像が誰なのか推定します。

● 顔認識プログラムのディレクトリ構成

顔認識プログラムの構成を図3に示します。

- extract_face.py…顔領域の検出と切り取りを行うコード
- model.py…ニューラル・ネットワークのモデルを定義したコード
- train.py…学習のコード
- detect_face.py…顔認識のコード
- util.py…学習・評価部で使う共通的なコード
- data…データを保存するディレクトリ

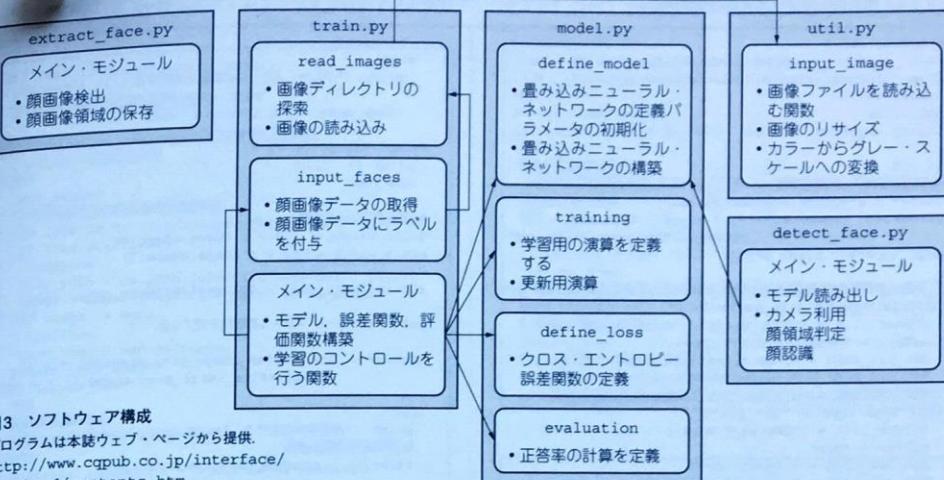


図3 ソフトウェア構成

プログラムは本誌ウェブ・ページから提供。
<http://www.cqpub.co.jp/interface/download/contents.htm>

トリ配下に顔写真の画像(.jpg)が配置されています。

今回のディープ・ラーニングはブッシュ氏を「ブッシュ氏である」と学習し、また、他の画像は「ブッシュ氏ではない」と学習します。他の画像には同データセットのTony Blair氏、Hugo Chaves氏を選択しました。これらのデータを使って、入力された画像がどの程度ジョージ・ブッシュ氏に近いのか、いわゆる「ブッシュ度」を計測します。

● 手順2…顔領域の検出と切り取り

画像の中から顔の部分だけを切り出します。その際にはOpenCVを利用します。OpenCV⁽⁴⁾は画像処理によく用いられるライブラリです。OpenCVのダウンロードは「<http://opencv.org/downloads.html>」から可能です。OpenCVのライブラリにはエッジ検出、特徴量検出、画像の変換を行うアフィン変換などがあります。

OpenCVを使った顔検出コードはリスト1です。こ

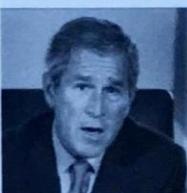


図4 今回使用した画像データセットには有名人も含まれている



図5 図4のブッシュ氏を学習させるときはブッシュ氏じゃない画像も必要

第2特集 Google人工知能 TensorFlow初体験

リスト1 顔画像の取り込みと切り抜き処理

```
# coding:utf-8
from __future__ import absolute_import
from __future__ import unicode_literals
import cv2
import os
import argparse

id = 0
# コマンドライン引数の定義
parser = argparse.ArgumentParser()
parser.add_argument("-t", "--type", type=str,
                    choices=["lfw_pos", "lfw_neg"])
args = parser.parse_args()

# 対象人物の画像ではない python extract_face.py -t lfw_neg
if args.type == "lfw_neg":
    folder_list = ["Tony_Blair", "Hugo_Chavez"]
    suffix = "neg"
    SRC_DIR_PATH = "./data/lfw/"
    DST_DIR_PATH = "./data/faces/negative"
# 対象人物の画像 python extract_face.py -t lfw_pos
elif args.type == "lfw_pos":
    folder_list = ["George_W_Bush"]
    suffix = "pos"
    SRC_DIR_PATH = "./data/lfw/"
    DST_DIR_PATH = "./data/faces/positive"

# 顔画像検出器を初期化する
cascade_path = "INPUT YOUR FACE MODEL PATH"
cascade = cv2.CascadeClassifier(cascade_path)
# ファイルセットを作成
for folder in folder_list:
    src_file_list = os.listdir(SRC_DIR_PATH + folder)
    for file in src_file_list:
        # 不要なファイルの判定処理を行わない
        if file.startswith(".") or file.endswith(".txt"):
            continue

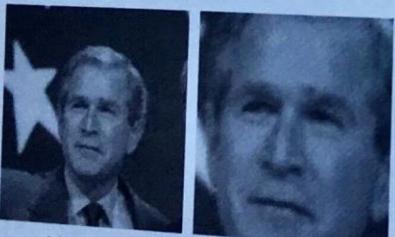
        try:
            # 画像の読み込み
            image = cv2.imread(os.path.join(SRC_DIR_PATH +
                                             folder, file))
            # 画像を読み込めなかった場合に処理を飛ばす
            if image is None:
                continue

            if (len(image.shape)) == 2:
                image_gray = image
        except Exception as e:
            print(e)
```

このコードのポイントは次の2つです。

▶①顔画像検出器の初期化

OpenCVには学習済みの顔検出用のモデルが含まれており、これを利用することで簡単に顔の部分を検出できます。学習済みモデルはCascadeClassifierクラスのコンストラクタの引き数にモデル・ファイル



(a) 切り抜き前
(b) 切り抜き後
図6 画像中から顔の部分を切り抜く

のファイル・パスを指定すると使えます。今回は顔検出モデルであるhaarcascade_frontalface_alt.xmlを使いました。

▶②顔画像の検出

顔画像の検出は、detectMultiScaleメソッドに引き数として判定したい画像を与えると、返り値として顔画像の矩形領域を取得できます(図6)。検出した顔画像を最後に保存します。

今回の顔認識プログラムでは、1枚の画像に1人しか写っていないことを前提とします。そのため1枚の画像で2カ所以上顔領域を検出した場合はノイズデータとし、利用しません。

ステップ2…画像の前処理

● 画像の読み込みと加工

画像をファイルから読み込み、適切なラベル付けを

```
continue
else:
    print("グレー・スケールの変換を開始する.")
    path.join(SRC_DIR_PATH, file))
    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

except Exception as e:
    print(e, file)
    continue

# 画像のスケール調整
scale_height = 512.0 / image.shape[0]
scale_width = 512.0 / image.shape[1]

resize_image_gray = cv2.resize(image_gray, (512,
                                             512))
print("グレー・スケールの変換を完了する.")

minsize = (int(resize_image_gray.shape[0] * 0.1),
            int(resize_image_gray.shape[1] * 0.1))

try:
    print("顔画像検出を行う.")
    # ②顔画像を検出する
    facerect = cascade.detectMultiScale(
        resize_image_gray, scaleFactor=1.1,
        minNeighbors=1, minSize=minsize)
    print("顔画像検出が完了する。")
    if len(facerect) == 0 or len(facerect) > 1:
        print("顔領域検出の結果が0もしくは2カ所以上見つかりました。")
        continue
    for rect in facerect:
        min_height = int(rect[0] / scale_height)
        min_width = int(rect[1] / scale_width)
        max_height = int(rect[2] / scale_height) +
                     min_height
        max_width = int(rect[3] / scale_width) + min_
                     width

        # 画像を保存する
        cv2.imwrite(os.path.join(DST_DIR_PATH,
                               "image_{}_{}.jpg".format(suffix, str(id))),
                    image[min_height:max_height, min_width:max_
                     width])
        id += 1
except Exception as e:
    print(e)
```

リスト2 train.py指定のディレクトリから画像を読み込むコード

```
def read_images(directory_path):
    """
    ディレクトリから画像を読み込む
    :param directory_path: ディレクトリのパス
    :return: 画像のリスト
    """

    face_images = []
    # ディレクトリの中の画像を読み込む
    for file in os.listdir(directory_path):
        if file.startswith("."):
            continue
        file_path = os.path.join(directory_path, file)
        image = input_image(file_path, IMAGE_SIZE)
        if image is None:
            continue
        face_images.append(image)
    return face_images

def inputs_face():
    """
    顔画像を読み込む
    :return: 画像のパスとラベル一覧
    """

    positive_datapath = "./data/faces/positive"
    negative_datapath = "./data/faces/negative"
    # 教師画像(positive)を読み込む
    face_images = read_images(positive_datapath)
    # ラベルを1とする
    labels = [1 for _ in range(len(face_images))]
    # 教師画像(negative)を読み込む
    negative_face_images = read_images(
        negative_datapath)
    face_images += negative_face_images
    # ラベルを0とする
    labels += [0 for _ in range(len(
        negative_face_images))]

    # 0-1へ画像をスケーリングする
    face_images = np.array(face_images, dtype=
                           np.float32) / 255.0
    return face_images, np.array(labels)
```

リスト3 util.py画像の読み込みと加工を行うコード

```
# coding:utf-8
from __future__ import absolute_import
from __future__ import unicode_literals
import cv2
import numpy as np
def input_image(file_path, size):
    """
    画像の入力を行う
    :param file_path: ファイルパス
    :return: 画像
    """

    img = cv2.imread(file_path)
    if len(img.shape) == 2:
        return None
    # グレー画像に変換
    gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # 画像をリサイズする
    resize_image = cv2.resize(gray_image, (size, size))
    # 28 × 28 → 28 × 28 × 1
    expand_image = np.expand_dims(resize_image, 2)
    return expand_image
```

ステップ3…学習環境の構築

● TensorFlowのインストール

学習環境を構築するためにTensorFlowをインストールします。インストールする対象がOSやハードウェアによって異なります。今回はCPUを使って顔認識を行いますが本来、ディープ・ラーニングは、GPU環境のほうが高速に計算できます。TensorFlowのGPU対応版はインストール媒体を利用する(https://www.tensorflow.org/versions/r0.12/get_started/os_setup.html)ことで構築できます。GPU環境で動作させるには、CUDAが必要です。CUDAはNVIDIA社のGPU向けの統合開発環境です。CUDAのインストール手順はNVIDIA社のウェブ・サイトを参照して下さい。

● 学習の全体像

初めに、学習の全体像を紹介します。図7に学習のフローチャートを示します。学習は大きく分けて2つの処理に分かれます。1つ目は学習環境の構築(本ステップ3で説明)、もう1つは実際に学習を行う処理(ステップ4で説明)です。

まずは、学習の定義内容を説明します。ディープ・ラーニングのモデルを学習する場合に決めなければならないことは3つあります。

1つ目はモデルです。ディープ・ラーニングの具体的なモデルを定義して、計算することが必要です。

2つ目は誤差関数です。教師データとどれだけ差分があるかを定義することが必要です。

3つ目はパラメータの更新方法です。ニューラル・ネットワークはパラメータを更新することで、学習

第2特集 Google人工知能 TensorFlow初体験ガイド

コラム 必須アイテムの紹介…ニューラル・ネットワークの学習状況可視化ソフト 山本 大輝

学習部の紹介に入る前に、TensorBoard⁽⁵⁾を紹介します。TensorBoardはTensorFlowの可視化ツールです。TensorFlowをインストールすると含まれます。TensorBoardは図9のような可視化が可能で、TensorBoardはTensorFlowで作成した複雑なモデルや学習の過程、入力したデータ(画像、オーディオなど)をウェブ・アプリケーションで可視化できます。表Aに本プログラムで可視化する対象の説明を示します。

学習時にニューラル・ネットワークの無数にあるパラメータの状態を把握するのは困難です。TensorBoardを使うことで学習途中の誤差関数や正答率、ニューラル・ネットワークの状況の可視化ができます。今回はTensorBoardの使い方を顔認ができます。

し、精度を高めることができます。更新方法も幾つか選択することができ、その更新方法を定義します。詳細は後述します。

これらを使って学習します。TensorFlowでどのように学習を実現しているかを説明します。この箇所がフローチャートのループ部に該当します。また、学習結果をファイルに保存する方法も併せて紹介します。

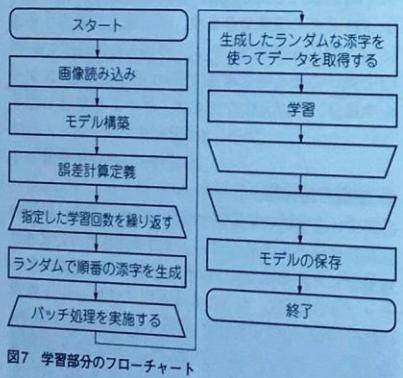
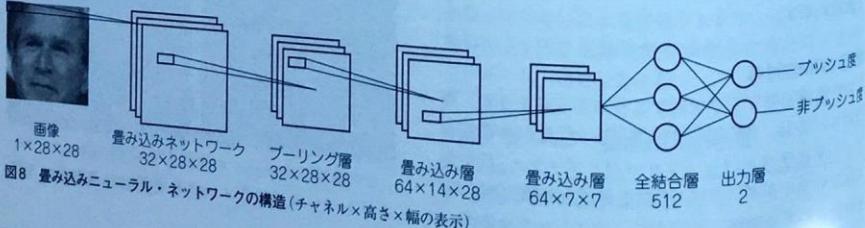


図7 学習部分のフローチャート



識プログラムの作成とともに紹介します。TensorBoardは次のコマンドで起動します。

`tensorboard --logdir=./logs`
`--logdir`はTensorFlowのログの保存先です。

TensorBoardを起動したら「<http://localhost:6006>」にブラウザでアクセスします。

表A TensorBoardのヘッダ項目

ヘッダ	説明
GRAPHS	構成したニューラル・ネットワークのモデルを可視化できる
EVENTS	数値情報を可視化できる 例：学習係数、誤差、正答率など
IMAGES	画像の可視化

①モデルを定義する

学習用ニューラル・ネットワークを定義します。プレースホルダ(Placeholder)と呼ばれるデータを格納する箱を用意します。そして、プレースホルダを使って、計算式を構築します。計算する際には具体的な値を投入します。

顔認識のモデルは畳み込みニューラル・ネットワーク(Convolutional Neural Network)として定義します。畳み込みニューラル・ネットワークは主に画像認識の分野で使われています。畳み込みニューラル・ネットワークは「畳み込み」や「ブーリング」と呼ばれる処理を交互に繰り返し行います。今回作成する畳み込みニューラル・ネットワークを図8に示します。

各ニューロンの計算式を指定することで、モデルの定義を行います。リスト4の前半はパラメータの初期化を行っています。リスト4の後半はプレースホルダに対して、畳み込みの計算を行う処理、活性化関数ブーリングによる計算の適用を行っています。これらを何回か繰り返し、最後にソフトマックス関数を定義して、出力します。

図9はTensorBoardで表示したモデルを示しています。

リスト4 model.py モデルの定義を行うコード

```

def define_model(images):
    """
    ニューラルネットワークのモデルを定義する。
    :return: 計算後の結果
    """
    #畳み込みニューラルネットワークのパラメータの初期化を行う。
    conv1_weights = tf.Variable(
        tf.truncated_normal([5, 5, NUM_CHANNELS, 32],
                            stddev=0.1,
                            seed=SEED, dtype=tf.float32))
    conv1_biases = tf.Variable(tf.zeros([32], dtype=
                                    tf.float32))
    conv2_weights = tf.Variable(tf.truncated_normal(
        [5, 5, 32, 64], stddev=0.1,
        seed=SEED, dtype=tf.float32))
    conv2_biases = tf.Variable(tf.constant(0.1, shape=[64], dtype=tf.float32))
    fcl_weights = tf.Variable(
        tf.truncated_normal([IMAGE_SIZE // 4 * IMAGE_SIZE // 4 * 64, 512],
                           stddev=0.1,
                           seed=SEED,
                           dtype=tf.float32))
    fcl_biases = tf.Variable(tf.constant(0.1, shape=[512], dtype=tf.float32))
    fc2_weights = tf.Variable(tf.truncated_normal([512, NUM_LABELS],
                                                  stddev=0.1,
                                                  seed=SEED,
                                                  dtype=tf.float32))
    fc2_biases = tf.Variable(tf.constant(0.1, shape=[NUM_LABELS], dtype=tf.float32))
    #プレースホルダに対して計算を行う。
  
```

②誤差関数を定義する

誤差関数は教師となるデータとどの程度差分があるかを数値的に表現しています。誤差関数の値が小さくなるようにニューラル・ネットワークのパラメータを更新することで精度が上がります。

誤差関数はニューラル・ネットワークで解きたい問題によって変わります。数値を求めるような問題(例:売上予測)は最小2乗誤差や平方最小2乗誤差が使われます。また、分類問題にはクロス・エントロピー誤差関数を使います。今回のような誰の顔画像かを識別するのは、分類問題です。顔認識アプリケーションはクロス・エントロピー誤差関数を利用しており、リスト5のコードに定義しました。

TensorFlowにクロス・エントロピー誤差関数はsoftmax_cross_entropy_with_logitsとして実装されています。

③パラメータの更新方法を定義する

パラメータの更新方法について紹介します。ニューラル・ネットワークは重みとバイアスと呼ばれるパラメータを持っています。学習はこのパラメータを更新し、教師との差分(誤差関数の数値)を減らします。

ニューラル・ネットワークのパラメータ更新方法は多くの研究者が研究しています。例えば、最急降下法と呼ばれる手法で最適化を行います。更新方法として、確率的こう配降下法(SGD)やAdam, Adadeltaなど

```

with tf.variable_scope("conv1"):
    conv = tf.nn.conv2d(images,
                       conv1_weights,
                       strides=[1, 1, 1, 1],
                       padding='SAME')
    relu = tf.nn.relu(tf.nn.bias_add(conv,
                                     conv1_biases))
pool = tf.nn.max_pool(relu,
                      ksize=[1, 2, 2, 1],
                      strides=[1, 2, 2, 1],
                      padding='SAME')
with tf.variable_scope("conv2"):
    conv = tf.nn.conv2d(pool,
                       conv2_weights,
                       strides=[1, 1, 1, 1],
                       padding='SAME')
    relu = tf.nn.relu(tf.nn.bias_add(conv,
                                     conv2_biases))
pool = tf.nn.max_pool(relu,
                      ksize=[1, 2, 2, 1],
                      strides=[1, 2, 2, 1],
                      padding='SAME')
pool_shape = pool.get_shape().as_list()
with tf.variable_scope("fc1"):
    reshape = tf.reshape(
        pool,
        [pool_shape[0], pool_shape[1] * pool_shape[2] * pool_shape[3]])
    hidden = tf.nn.relu(tf.matmul(reshape,
                                 fcl_weights) + fcl_biases)
return tf.matmul(hidden, fc2_weights) + fc2_biases
  
```

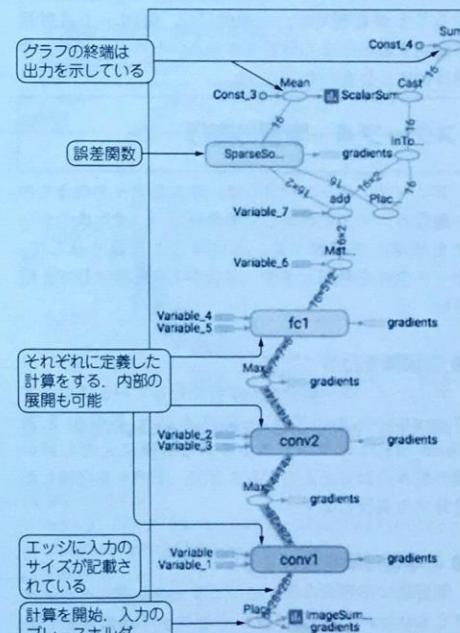


図9 顔認識プログラムの「GRAPHS」画面

第2特集 Google人工知能 TensorFlow初体験ガイド

リスト5 model.py誤差関数の定義を行うコード

```
def define_loss(logits, labels):
    """
    クロス・エントロピー誤差の定義
    :param logits: 入力値
    :param labels: ラベル
    :return: 誤差平均
    """
    labels = tf.to_int64(labels)
    # クロス・エントロピー誤差の計算
    cross_entropy_loss = tf.nn.sparse_softmax_cross_entropy_with_logits(logits, labels)
    # 計算したクロス・エントロピー誤差の平均
    cross_entropy_loss_mean = tf.reduce_mean(cross_entropy_loss)
    return cross_entropy_loss_mean
```

がTensorFlowには用意されています。パラメータの更新を行う計算はリスト6のtraining関数を参照してください。学習係数(learning_rate)は更新方法の中で特に重要なハイパ・パラメータです。ハイパ・パラメータとは人間が設定するパラメータのことを指します。この学習係数の数値が低すぎることは、この学習係数の数値が高すぎる場合は止しません。また、学習係数の数値が高すぎる場合は、学習がなかなか終りません。そのため、適度な学習係数を設定することがあります。そのため、適度な学習係数を設定することが必要です。一般的には、0.001～0.01付近を設定するとうまく学習できるといわれています。今を設定するときも学習できるといわれています。今は0.01としました。

ステップ4…学習の実行

ディープ・ラーニングでは、膨大なデータのうち的一部分のデータを利用して学習を行い、また次のデータを利用して学習する、ということを繰り返して、データ全体を利用します。リスト7を参考にしてください。

①演算を行う

ディープ・ラーニングの学習を行っているメソッドはsession.runです。このメソッドの引き数feed_dictにプレースホルダと実際に入力したい値の組み合わせを入力することで、モデルを定義した計算式を実行します。

②学習結果を保存する

学習後に学習済みディープ・ラーニングのパラメータをSaverのsaveメソッドでファイルに保存します。また、TensorBoardを使うことにより、学習の途中経過を確認できます。

「EVENTS」は数値情報のサマリを表示します。本頃認識プログラムは誤差情報をscalar_summary

リスト6 model.py学習を行うためのコード

```
def training(loss, learning_rate):
    """
    パラメータ更新方法を定義する
    :param loss: 計算した誤差
    :param learning_rate: 学習係数
    :return:
    """
    batch = tf.Variable(0, dtype=tf.float32)
    optimizer = tf.train.GradientDescentOptimizer(
        learning_rate=learning_rate).minimize(
            loss, global_step=batch)
    return optimizer

def evaluation(logits, labels):
    """
    評価
    :param logits: 計算結果
    :param labels: ラベル
    :return: 評価
    """
    # 正しければ1、誤っていれば0
    correct = tf.nn.in_top_k(logits, labels, 1)
    # 正解した数を計算する
    return tf.reduce_sum(tf.cast(correct, tf.int32))
```

に与えています。そのため、「EVENTS」は誤差の遷移を可視化できます。図10が誤差の推移を表したもので、横軸がバッチ単位の学習回数[回]、縦軸が誤差の値です。図10から誤差が減っており、学習が進行していることが読み取れます。

ステップ5…判定

最後に学習済みのモデルを使って、顔認識プログラムを作成します。顔認識プログラムのソースコードはリスト8に示します。

●学習済みモデルの読み込み

まずは、学習済みのモデルを取得します。このモデルの読み込みにはsession内部にあるSaver.restoreメソッドを使います。このメソッドの引数に学習時にモデルの保存をした先のファイルパスを指定します。後は自動的に読み込まれます。

今回の出力は、入力の顔画像をどのクラスに分類すべきかの度合いを出力します。ソフトマックス関数を使い、全部のクラスを合計すると1になるように調整します。学習時に使ったdefine_model関数の威力に対して、softmaxを使います。ここから完成したモデルを使って顔認識アプリケーションを作成します。処理の流れは図11に記載します。

●顔画像の取得

顔認識アプリケーションはカメラを利用して画像を取得します。今回は手持ちのMacBook Proに標準搭載されているカメラを使いました。そしてその画像を評価し、最後に判定した色を付与します。これを顔

リスト7 学習用コードtrain.py

```
# coding:utf-8
import tensorflow as tf
import numpy as np
import time
import os
from model import define_model, define_loss, training, evaluation
from util import input_image
BATCH_SIZE = 16
NUM_CHANNELS = 1
IMAGE_SIZE = 28
SEED = 82
NUM_LABELS = 2
EPOCHS = 100
def main():
    faces, label = input_faces()
    n_class = len(list(set(label)))
    N_faces = len(faces)
    print(n_class, N_faces)
    with tf.Graph().as_default():
        # 入力画像のプレースホルダ
        images = tf.placeholder(tf.float32, [BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, 1])
        # ラベル 0: 1:
        labels = tf.placeholder(tf.int64, shape=[BATCH_SIZE,])
        # モデルの定義
        model = define_model(images=images)
        # 誤差関数
        loss = define_loss(model, labels=labels)
        # 誤差を保存。TensorBoardで可視化できる。
        tf.scalar_summary("loss", loss)
        # 学習の演算を定義する。
        train_op = training(loss, learning_rate=0.01)
        # 評価方法を定義。正解数を計算する
        evaluation_op = evaluation(model, labels)
        # TensorBoardへの記載
        tf.image_summary('images', images, max_images=100)
        summary = tf.merge_all_summaries()
        saver = tf.train.Saver()
        # 变数初期化を行う。
        session = tf.Session()
        session.run(tf.initialize_all_variables())
        summary_writer = tf.train.SummaryWriter("./log", session.graph)
        count = 0
        # 学習を行う。何周学習を行うか
        for epoch in range(EPOCHS):
            perm = np.random.permutation(N_faces)
            loss_value = 0.0
            # バッチ学習を行う。
            for index, step in enumerate(range(0, N_faces - BATCH_SIZE, BATCH_SIZE)):
                start_time = time.time()
                batch_start = step
                batch_end = step + BATCH_SIZE
                feed_dict = {
                    images: faces[perm[batch_start:batch_end]],
                    labels: label[perm[batch_start:batch_end]]}
                # ①演算を行う。
                _, loss_value, eval_value = session.run([train_op, loss, evaluation_op], feed_dict=feed_dict)
                duration = time.time() - start_time
                count += 1
                print('Step %d: loss = %.2f (%.3f sec)' % (step + epoch * N_faces // BATCH_SIZE, loss_value, duration))
                summary_str = session.run(summary, feed_dict=feed_dict)
                summary_writer.add_summary(summary_str, index + epoch * N_faces // BATCH_SIZE)
                summary_writer.flush()
            # 学習が一周完了した段階で出力する。
            print(epoch, loss_value)
            # ②学習結果を保存する。
            saver.save(session, "model.ckpt")
if __name__ == '__main__':
    main()
```

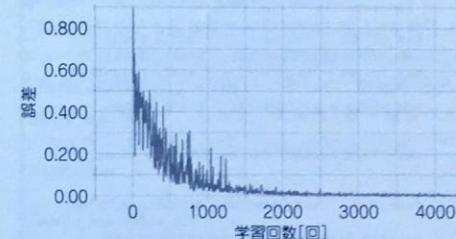


図10 学習の収束具合は学習回数と誤差との関係で確認できる

(3) RSオンライン。

http://jp.rs-online.com/web/general_Display.html?id=raspberry

(4) OpenCV。

<http://opencv.jp/>

(5) TensorBoard。

<https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tensorboard>

やまもと・ひろき

リスト8 detect_face.py評価用コード

```
# coding:utf-8
from __future__ import absolute_import
from __future__ import unicode_literals
import tensorflow as tf
import numpy as np
from model import define_model
import cv2

BATCH_SIZE = 1
NUM_CHANNELS = 1
IMAGE_SIZE = 28

cap = cv2.VideoCapture(0)
cascade_path = "INPUT YOUR FACE MODEL PATH"
cascade = cv2.CascadeClassifier(cascade_path)

with tf.Graph().as_default():
    # 入力画像のプレースホルダ
    images = tf.placeholder(tf.float32, [BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, 1])

    # モデル定義
    model = define_model(images=images)
    # 出力定義
    softmax_op = tf.nn.softmax(model)
    # セーバーの初期化
    saver = tf.train.Saver()
    # 変数初期化を行う
    with tf.Session() as session:
        # モデルの読み出し
        saver.restore(session, "./model.ckpt")

while True:
    response, frame = cap.read()
    image_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    scale_height = 512.0 / image_gray.shape[0]
    scale_width = 512.0 / image_gray.shape[1]
    resize_image_gray = cv2.resize(image_gray, (512, 512))

    minsize = (int(resize_image_gray.shape[0] * 0.1),
               int(resize_image_gray.shape[1] * 0.1))
    facerect = cascade.detectMultiScale(
        resize_image_gray, scaleFactor=1.1,
        minNeighbors=1, minSize=minsize)

    # 画像が1枚の時に実行する
    if len(facerect) == 1:
        for rect in facerect:
            min_height = int(rect[0] / scale_height)
            min_width = int(rect[1] / scale_width)
            max_height = int(rect[2] / scale_height) + min_height
            max_width = int(rect[3] / scale_width) + min_width
            print(min_height, max_height, min_width, max_width)

            face_img = image_gray[min_height:max_height,
                                  min_width:max_width]
            resized_face_img = cv2.resize(face_img, (IMAGE_SIZE, IMAGE_SIZE))

            feed_dict = {
                images: np.array([np.expand_dims(
                    resized_face_img, 2)], dtype=np.float32)
            }
            # 結果
            value = session.run(softmax_op,
                                feed_dict=feed_dict)
            print(value)
            # 矩形の色指定を行う
            if value[0][0] > 0.5:
                color = (255, 0, 0)
            else:
                color = (0, 0, 255)
            cv2.rectangle(frame, (min_width, min_height),
                          (max_width, max_height), color, 10)
            k = cv2.waitKey(1)

            cv2.imshow("face camera", frame)
```

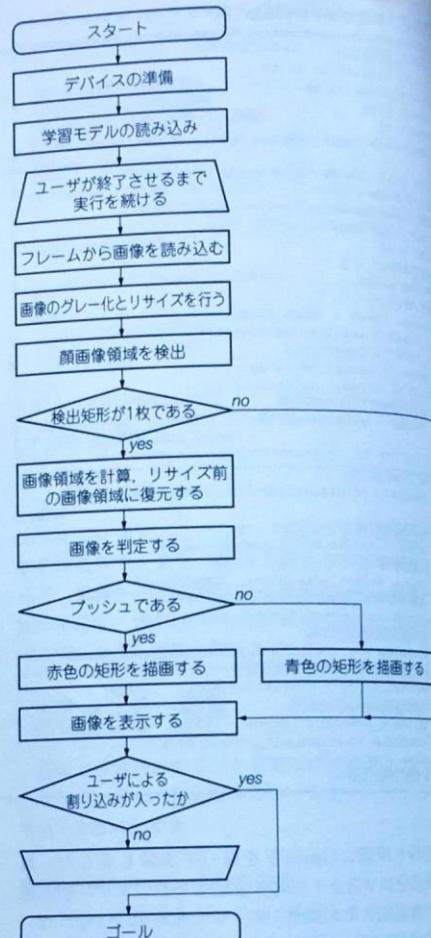


図11 判定のフローチャート



図12 ブッシュ氏を認識できた

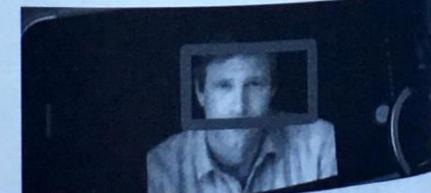


図13 ブッシュ氏ではないと認識できた

レッスン5…
3大人工知能ライブラリ+a

牧野 浩二、西崎 博光

ここ数年、ディープ・ラーニング向けのフレームワーク/ライブラリ注1が複数の会社や団体から提供されています。ソフトウェアの研究では、作成したものを公開して使ってもらうという文化が根付いており、多くの場合はディープ・ラーニングの発展や人材育成、分野のそぞろを広げるなどが目的とされています。本稿では、ディープ・ラーニング向けのフレームワーク/ライブラリについて、特徴や使い勝手、インストール手順などを整理しました(表1)。

なお、ここで紹介するフレームワーク/ライブラリは、自分のPC上にインストールして使うことはもちろん、クラウド・サーバ上に設けたインスタンス(CPUやメモリ)にもインストールできます。

ディープ・ラーニングは主にLinuxでの開発がメインになっていますので、Linuxへのインストールは簡単にできるようになっています。逆にWindowsへのインストールは難しいものが多いです。そこでWindows PCにVirtualBoxをインストールし、その中でUbuntu 16.04を動かして、それぞれのディープ・

● 特徴…ユーザーが多く情報が多い

提供されている視覚化ツールTensorBoardを使うと、ニューラル・ネットワークの学習過程やデータの流れを簡単に確認できます(図1)。そして、複雑なモデルの構築も可能です。しかも、大規模(数十億レベル)な学習データを扱えます。TensorBoardとは、プロ

注1：フレームワークとライブラリの使い分けが難しいです。各社、自社のものをフレームワークとしていたりライブラリとしています。以下、各社の表記に合わせました。

表1 ディープ・ラーニング向けのフレームワーク/ライブラリ

名 称	供給元	使い勝手	特 徴	何に向くのか	公式ホームページ
TensorFlow	Google社	中～上級者向け	ネット上に情報が多く勉強しやすい	大規模なネットワーク	https://www.tensorflow.org/
Caffe	カリフォルニア大学 バークレー校	上級者向けの ライブラリ	画像処理の研究者の中では デファクト・スタンダード	画像処理	http://caffe.berkeleyvision.org/
Chainer	Preferred Networks社	初心者にとって 使いやすい	あらゆるニューラル・ネットワークの構造に柔軟に対応できる	自然言語処理や音声処理	http://chainer.org/
Keras	https://keras.io/	初心者にも 扱いやすい	拡張性が高く、新しいモジュールの実装が簡単に行える	手軽に試してみたいとき にすぐに実装できる	https://keras.io/
Theano	モントリオール大学	少し試したい 人には不向き	Theanoをベースにした機械学習ライブラリが幾つか開発	ディープ・ラーニングの 理論を勉強してゼロから実装したい人	http://deeplearning.net/software/theano/

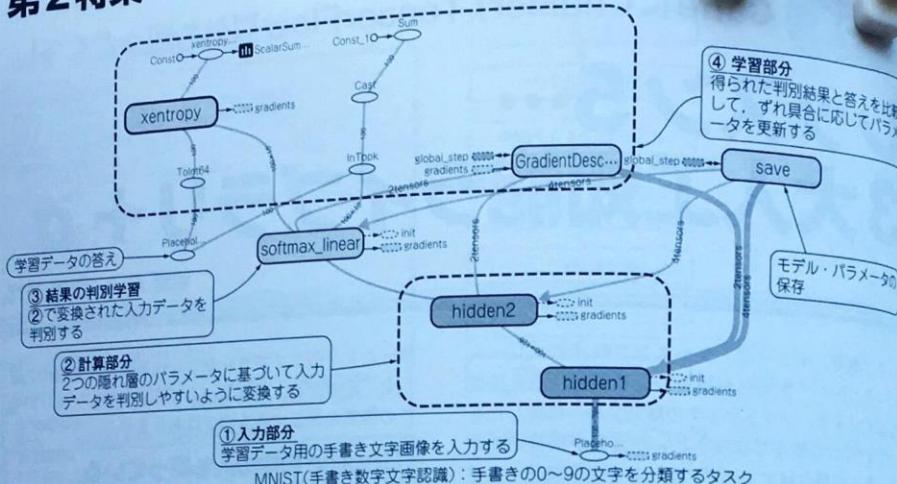


図1 TensorFlowは作ったアルゴリズムを可視化できるため開発/勉強しやすい

プログラミングしたニューラル・ネットワークの構造やネットワークに流れるデータの動きをグラフで可視化(これを計算グラフといいます)するためのツールです。

Googleが公開しているだけあって利用者が多く、GitHubなどでコードがたくさん公開されているため、ネット上に情報が多く、勉強しやすいという特徴があります。

TensorFlowにはModel Zooがあります(<https://github.com/tensorflow/models>)。

Model Zooとは、だれでも利用可能な学習データのセットです。よくできたデータ・セットなので、多くの研究者のテストに使われ、その性能測定に利用されています。

● 何に向くのか…大規模から小規模まで

大規模なネットワーク、大量の学習データを扱ってみたいとき向きます。

モバイル端末や小型端末での機械学習向けには、Mobile TensorFlowというライブラリがあります。これを使うとAndroidやiOS上で動作させることができます。

● 対象OS/プラットフォーム

OSとしてはLinux、Mac OS X、Windowsが挙げられます^{注2}。プラットフォームとしてはAndroid、iOS、ラズベリー・パイがあります。

^{注2}: 対象OSは公式ウェブ・ページで書かれているものと、試したもののが混在しています。そのため、必ずしも動作するとは限りません。

● 言語…PythonとC++

言語はPython 2.7または3.3以上、他にC++が使えます。

● インストール^{注3}

▶手順

公式ホームページ上で「GET STARTED」をクリックし、左側の「Pip Installation」をクリックすると、詳細が書いてあります。

▶コマンド

```
$ sudo apt-get install python-pip
$ sudo pip install --upgrade
https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-
0.11.0-cp27-none-linux_x86_64.whl
```

● 使い勝手…ディープ・ラーニングの知識は必要
ある程度ディープ・ラーニングを学んでいないと、スイスイとコードが書けません。しかし、比較的容易に、かつ柔軟なニューラル・ネットワークを記述できます。計算グラフの理解がやや難しいです。

画像処理では事実上の業界標準 Caffe

Caffeは、カフェと読みます。Yangqing Jia氏(現在はFacebook)が、カリフォルニア大学バークレー校

^{注3}: TensorFlowのインストールについては第1章で解説しています(編集部)。

の博士課程のときに立ち上げたプロジェクトがきっかけで開発が始まりました。画像処理の研究者の中では事実上の業界標準になっているディープ・ラーニングのフレームワークです。

供給元はカリフォルニア大学バークレー校で、開発者はthe Berkeley Vision and Learning Center, UC Berkeleyとなっています。

● 特徴…テキストや時系列データは対象外

Caffeを用いた研究成果(モデル)が公開されているサイト「Model Zoo」があります(http://caffe.berkeleyvision.org/model_zoo.html)。

画像処理のコミュニティでよく利用されているので、画像処理に関する情報は多くあります。最新の研究成果(画像処理分野)がすぐに試せるという特徴があります。

画像処理でのディープ・ラーニングの草分け的なフレームワークであるため、多くの人が使っています。そのため画像処理のサンプル・プログラムも多く、参考になるサイト(ブログ)も多くあります(ただし日本語の解説は少ない)。

なお、画像処理のディープ・ラーニング・フレームワークとして開発されており、テキストや時系列データを扱うことは想定されていません⁽¹⁾。

● 対象OS

Linux、Mac OS X、Windowsです。

● 言語

Python、C++、MATLABを使います。

● インストール

入手先は<http://caffe.berkeleyvision.org>です。

▶手順

公式ホームページにおいて、「Installation instructions」をクリックし、「Ubuntu installation」をクリックすると詳細が書いてあります。コマンドラインで以下を実行します。

・ライブラリのインストール

```
$ sudo apt-get install libprotobuf-dev
libleveldb-dev libsnappy-dev
libopencv-dev libhdf5-serial-dev
protobuf-compiler
$ sudo apt-get install --no-
install-recommends libboost-all-dev
• gitのインストール
$ sudo apt install git
• Caffeのダウンロード
```

```
$ git clone https://github.com/
BVLC/caffe.git
```

- caffe以下のpythonディレクトリへの移動

```
$ cd caffe/python
```

- Caffeのインストール

```
$ for req in $(cat requirements.
txt); do pip install $req; done
```

- caffeディレクトリへの移動

```
$ cd ..
```

- 線形代数ライブラリのインストール

```
$ sudo apt-get install libatlas-
base-dev
```

```
$ sudo apt-get install libopencv-
dev
```

- Makefile.configファイルのコピー

```
$ cp Makefile.config.example Make
file.config
```

- Makefile.configファイルの編集(viが起動する)

```
$ vi Makefile.config
```

— viでの作業(ここから) —

- Makefile.config中の下記のコメント行を有効化

```
# CPU_ONLY := 1
```

- 下記の行をコメント・アウト

```
CUDA_DIR := /usr/local/cuda
```

- 下記のコメント行を有効化

```
# CUDA_DIR := /usr
```

- Makefile.config中の下記を変更

変更前:

```
INCLUDE_DIRS := $(PYTHON_INCLUDE) /
usr/local/include
```

変更後:

```
INCLUDE_DIRS := $(PYTHON_INCLUDE) /
usr/local/include /usr/include/
hdf5/serial
```

— viでの作業(ここまで) —

- Makefileファイルの編集(viが起動する)

```
$ vi Makefile
```

— viでの作業(ここから) —

- Makefile中の下記を変更

変更前:

```
LIBRARIES += glog gflags protobuf
boost_system boost_filesystem m
hdf5_hl hdf5
```

変更後:

```
LIBRARIES += glog gflags protobuf
boost_system boost_filesystem m
hdf5_serial_hl hdf5_serial
```


Appendix 3 話題アルゴリズムの理屈をまとめにまとめておきました

「ディープ・ラーニング」 アルゴリズムあんちょこ

足立 悠



図1 ディープ・ラーニングの注目度
Googleのサービス「Google Trend」で注目度を可視化。執筆時点(2016年12月)を100とする

TensorFlowにはディープ・ラーニング(深層学習)のAPIが含まれるとお伝えしました。ここでは、そのディープ・ラーニングについて紹介します。

ディープ・ラーニングはAI技術の1つです。AI技術はブームと冬の時代を繰り返しながら発展を続けてきました。

文献(1)によると、1960年代の第1次AIブームでは「探索・推論」型の技術が中心でした。ハノイの塔アルゴリズムがこれに当たります。1980年代の第2次AIブームでは「知識表現」型の技術が中心でした。エキスパート・システムなどが生まれました。2000年代以降の第3次AIブームでは「機械学習(特徴抽出)」型の技術が中心となりました。

過去のブームでは、やがて冬の時代を迎えていたものの、第3次以降は現在に至るまで失速することなく、急速に普及しています。

● 注目度

Google社のサービス「Google Trend」で、キーワード「deep learning」と指定すると、ユーザがキーワードに対しどの程度興味を持っているかを時系列で追うことができます。図1のようにここ3年くらいで注目度が高まっています。

Google Trendとは、Google検索エンジンで検索されたデータをもとに、指定したキーワードの注目度を可視化して提供するサービスです。

第3次のブーム以降、AIが冬の時代を迎えることなく急速に発展を遂げている理由として、大量の情報を処理できるインフラを入手できることが挙げられます。しかしそれ以上に技術的なフレークスルーは、ディープ・ラーニングを使えば自動的に特徴を抽出できることです。

予習…機械学習とは

● 人間が処理できない量のデータを扱える

ディープ・ラーニングは機械学習のアルゴリズムの1つです。では、そもそも機械学習とは何なのでしょうか。機械学習とは、「機械にデータを解析させ、データに潜む規則性(ルール)やパターンを見出し、アルゴリズムを発展させていく処理」を指します。データ量が少なければ、人手でルールやパターンを見出しきるかもしれません(例えば100行のデータを1行ずつ目視する)。しかし、センサ・ログを始め現実のデータは、大規模かつ複雑になりつつあり、人間が処理できるレベルを超えてます。そこで、機械に処理させることで、データから効率良く、効果的な知識を見出せます。

現実には、センサ・ログのような数値形式のデータ(構造化データ)、アンケートの自由記述のようなテキスト形式のデータ(非構造化データ)など、さまざまな形式のデータが存在します^{注1}。

そして目的と入力データ形式に応じた「機械学習の手法」を選択し、ルールやパターンを表現するモデルを作成し、さらにより精度を高めるために学習させます。

● アルゴリズムあれこれ

機械学習には次の手法が一般によく使われます。

- 2つに分類
- 3つ以上に分類
- 回帰分析
- 時系列分析
- グループ分け
- パターン発見

「分類(2つ)(3つ以上)」と「回帰分析」は教師あり学習と呼ばれ、予測分析に使われます。「グループ分け(クラスタリング)」と「パターン発見」は教師なし学習と呼ばれ、データの特徴を把握することに使われます。

注1: 画像や音声は非構造化データです。従って特徴を抽出しこれを入力パラメータとするのが一般的です。特徴の抽出には2値化やヒストグラム、周波数帯への変換などが使われます。この特徴抽出を自動で行ってくれるのがディープ・ラーニングの良さです。

Appendix 3 「ディープ・ラーニング」アルゴリズムあんちょこ

電圧	回路	圧力	振動	状態
2015/4/23 6:00	167,423	489,0057	80,96488	42,43256 正常
2015/4/23 19:00	148,655	395,5561	83,07978	42,89457 正常
2015/4/26 0:00	171,8131	471,2578	107,3949	34,3593 正常
2015/4/26 12:00	159,4023	479,8812	73,05121	42,61927 正常
2015/4/28 11:00	178,7248	66,69078	37,36849 正常	
2015/5/3 23:00	178,7248	66,69078	37,36849 正常	
2015/6/21 11:00	146,1204	389,7095	110,06807	29,84947 正常
2015/6/24 12:00	166,4518	465,1385	101,9375	33,65188 正常
2015/6/24 15:00	148,2442	394,9826	103,0276	34,06044 正常
2015/6/24 18:00	157,7862	440,9727	86,66038	38,45552 故障H
2015/6/24 21:00	144,0945	409,3802	106,7209	57,45499 故障H
2015/6/25 0:00	136,7139	470,7812	86,53166	45,28678 正常

図2 センサ・ログのサンプル

▶ 予測に使われるアルゴリズム「分類」

「分類」はデータのカテゴリを判別し予測します。例えばセンサ・ログから正常な機械/故障した機械を分類するパターンを作成し、将来故障しそうな機械を予測することに使われます。センサ・ログが図2の形式の場合、機械の故障予測を考えてみましょう。

機器には電圧や圧力などのセンサが取り付けられており、時々刻々とセンサ値、そして状態(正常/故障)とそのレベル)がデータとして蓄積されています。破線で囲っているデータの最右列「状態」には値がありますが、実線で囲っているデータの最右列「状態」には値がありません。

状態列の値は予測対象(正解となる値)です。正解を持つデータ(破線で囲ったデータ)を使って学習し、学習済みモデルを作成します。この学習を「教師あり機械学習」と呼びます。作成したモデルを、正解となる値が分からないデータ(実線で囲ったデータ)に適用すれば、状態列の値を予測できます(図3)。

一般に、分類に使えるアルゴリズムには以下のものがあります。

- 決定木(Decision Tree)
- k 近傍法(k-NN)
- ランダム・フォレスト
- サポート・ベクタ・マシン(SVM)
- ニューラル・ネットワーク
- ディープ・ラーニング

▶ 「分類」以外のアルゴリズム

分類以外の各アルゴリズムの詳細は、表1をご覧ください。

一般によく使われる教師あり/なし学習の手法と、具体的にどのように活用できるのか事例を掲載しています。教師あり・予測型の「ニューラル・ネットワーク」は、ディープ・ラーニングの基礎となるアルゴリズムです。

▶ ディープ・ラーニングの位置付け

ディープ・ラーニングの日本語表記は「深層学習」であり、機械学習とは別の技術に思われるかもしれません。表1の教師あり・予測型「ニューラル・ネットワーク」を改良したアルゴリズムです。ディープ・ラーニングは、ディープ・ニューラル・ネットワーク

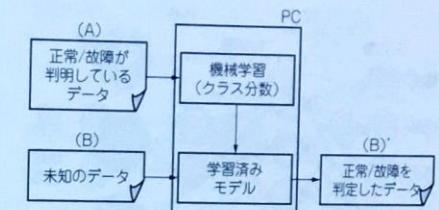


図3 機械学習のアルゴリズムを用いるとOK/NGデータを分類できるようになる

とも呼ばれます。従って以下のような主従関係になります。

機械学習

「ニューラル・ネットワーク」「ディープ・ラーニング」

機械学習には「分類」「回帰」「グループ分け」「パターン発見」に属する多くのアルゴリズムが存在します。ニューラル・ネットワークは「分類」「回帰」のアルゴリズムの1つです。ディープ・ラーニングはニューラル・ネットワークを発展させたものであり、「分類」「回帰」の学習が可能です。

誕生まで

● ニューラル・ネットワークの限界

ニューラル・ネットワークは、「回帰」「分類」型の教師あり機械学習アルゴリズムです。正解を持つデータを用いてモデルを構築するため予測に使えます。ニューラル・ネットワークのモデルを可視化した

表1 機械学習で用いられるアルゴリズムの一例

種類	分析アルゴリズム	事例
機械学習 (教師なし、知識発見)	主成分分析(PCA)	顧客セグメンテーション
	相関分析	購買パターン抽出
	アソシエーション分析	インフルエンサーの特定
	コレスポンデンス分析	ポジネガ分析
	階層型クラスタリング	
	k-means クラスタリング	
機械学習 (教師あり、未来予測)	ネットワーク分析	
	k 近傍法	
	決定木	
	ランダム・フォレスト	機械・設備の故障予測
	ナイーブ・ベイズ	顧客の解約防止
	線形回帰/重回帰	売上予測
機械学習 (教師あり、未来予測)	ロジスティック回帰	不正検知
	ニューラル・ネットワーク	画像/音声認識
	サポート・ベクタ・マシン	
	ディープ・ラーニング	ここ