

レッスン1…TensorFlowの開発環境

渡邊 雄

TensorFlowのインストール方法について、公式ページ^{注1}に紹介されているもののうち、主なものを紹介します。

TensorFlowのCPU用とGPU用は別物で、使いたい方を明示的にインストールする必要があります。今はCPU用のインストールを紹介します。

動作環境

● 重要…TensorFlowがサポートしているOS

r0.11まではMacとUbuntu/Linux(64ビット)だけをネイティブでサポートしていましたが、2016年11月29日に最新バージョンのr0.12がリリースされ、Windows(64ビット)へも直接インストールできるようになりました。また、サポート外のOSでもDockerという仮想化ソフトウェアを使うことで、ほとんどのプラットホームで利用できます。

注1: https://www.tensorflow.org/versions/r0.11/get_started/os_setup.html

注2: 第1特集 第2章～第7章のデスクトップPCとノートPCはこのパターン。ラズパイはC++ライブラリとしてソースコードからビルドしているため他のパターンにも含まれない。(編集部)

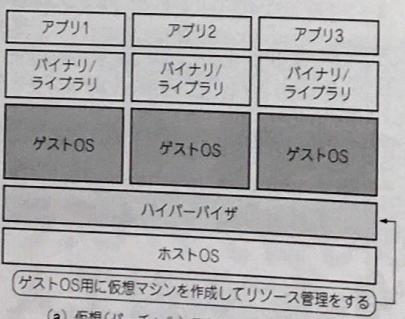


図1 仮想化技術を実現するソフトウェア構成

- 本章で紹介する開発環境のインストール方法
 - パターン1…TensorFlowが直接サポートしていないOSのときはDocker環境で
 - パターン2…TensorFlowが直接サポートしているOSのときはPython環境で^{注2}
 - パターン3…TensorFlowが直接サポートしているOSかつ独立した環境にインストールするときはAnaconda
 - インストール後…ブラウザから簡易的にPythonのコーディングを行うためのアプリケーション Jupyter Notebook

パターン1…TensorFlow用仮想Docker環境

DockerはWindows、Mac、Linux系OS、クラウドサーバなど、多様なプラットフォームをサポートしています。TensorFlowが直接サポートしていないOSでTensorFlowを使いたい場合は、この方法でインストールする必要があります。以下DockerをWindowsのマシンにインストールする前提で説明します。

● TensorFlowを動かすために使うオープンソース Docker

Docker^{注3}は「コンテナ」という仮想化を実現するた

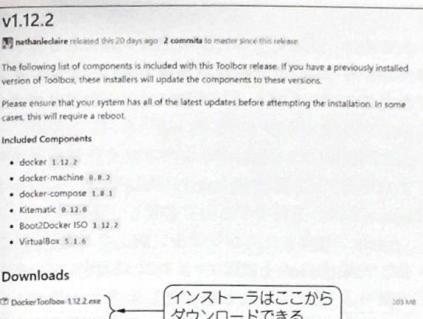
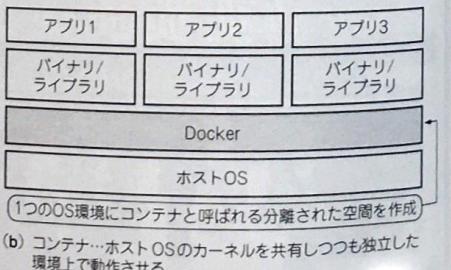


図2 Docker Toolboxのインストーラ(DockerToolbox-X.XX.X.exe)はGitHubの最新リリース・ページから取得できる



図3 最新リリース・ページはDockerのウェブ・サイトからリンクされている

たいと思います。

Docker Toolboxは、ローカル・マシンでDockerを利用するためのコンポーネントが全て含まれております。一括でそれらをインストールできます。インストーラ(DockerToolbox-X.XX.X.exe)は、GitHubの最新リリース・ページから取得できます(図2)。最新リリース・ページはDockerのウェブ・サイト^{注3}からリンクされています(図3)。2016年12月7日現在、v1.12.3が最新です。

インストーラを実行し、ウインドウの指示に従いインストールを行います。

● Dockerの動作確認

インストールが完了したらDocker Quickstart Terminal(図4)を起動し、以下のコマンドを実行してください。

\$ docker run hello-world

正しく動作しているようであれば次のようなメッセージが表示されるはずです。

Status: Downloaded newer image for hello-world:latest Hello from

注5: <https://www.docker.com/products/docker-toolbox>

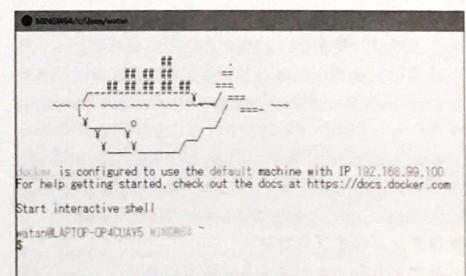


図4 Docker Quickstart Terminalの起動画面

Docker.
This message shows that your installation appears to be working correctly.

この動作確認ではhello-worldという初期動作確認用のイメージをダウンロードし、コンテナを作成しています。

▶結局VirtualBoxを利用している?
インストール時に気づいた方もいると思いますが、Docker ToolboxはOracle VM VirtualBoxというバーチャル・マシンをインストールし、利用しています。これはDocker自体がLinuxの技術を組み合わせて実装されたもので、Linux系OS上でないと動作しないためです。つまりWindowsマシンでDockerを使う場合は、バーチャル・マシンでエミュレートしたLinux上でDockerが動作することになります。

● Docker上にTensorFlowをインストールする
Dockerがインストールできたら、いよいよTensorFlowをインストールします。インストールをすると言っても、Dockerイメージをダウンロードするだけなのでとても簡単です。

▶4つのDockerイメージ

TensorFlowは4つのDockerイメージを公開しています。

- gcr.io/tensorflow/tensorflow:CPU実行バイナリ
 - gcr.io/tensorflow/tensorflow:latest-devel:CPU実行用バイナリ+ソースコード
 - gcr.io/tensorflow/tensorflow:latest-gpu:GPU実行用バイナリ
 - gcr.io/tensorflow/tensorflow:latest-devel-gpu:GPU実行用バイナリ+ソースコード
- これらはいずれもTensorFlowを利用した開発に必要なコンポーネント(Python, Pythonライブラリ, Jupyter Notebookなど)を全て含んでいるため、これらのイメージをダウンロードするだけで基本的な開発環境が整ってしまいます。

▶ダウンロードのコマンド

ここでは一番オーソドックスなCPU実行用バイナリのイメージをダウンロードすることにします。イメージのダウンロードは、Docker Quickstart Terminalからたった1行のコマンドで行うことができます。

```
docker run -it -p 8888:8888 -p 6006:6006 --name tf gcr.io/tensorflow/tensorflow /bin/bash
```

▶コマンドのオプション

コマンドのフォーマットは以下です。

```
$ docker run [オプション] イメージ名
```

[コマンド]

docker runはイメージのダウンロードとそのイメージからのコンテナ生成を一括で実行します。ここで指定しているオプションの意味は以下の通りです。
-it: インタラクティブ・モード
-p: 公開するポートのバインディング(ローカル)(コンテナ), 8888はJupyter Notebook, 6006はTensor Boardを利用するのに必要
--name: 生成されるコンテナに別名を付与

また./bin/bashを初期コマンドとして設定しています。イメージgcr.io/tensorflow/tensorflowはデフォルトでrun_jupyter.shというシェルが起動と同時に実行されるよう設定されており、初期コマンドを明示しないと、このシェルが実行されます。

上記のコマンド例では生成されるコンテナにtfという別名を付与しています。コマンドを実行するとイメージのダウンロードが始まり、ダウンロードが完了した時点でそのイメージを元にコンテナが起動します。イメージのダウンロードは各イメージの初回だけ実行され、2回目以降は既にダウンロード済みのイメージを使用します。ただし、2回目以降の起動にdocker runコマンドを使用すると、新しく別のコンテナ(インスタンス)が立ち上がってしまいます。初回に起動したコンテナを再起動する場合は、先ほど設定した別名を指定する形で、

```
$ docker start -i tf
```

のように実行します。また、[Ctrl+c]で実行中のコンテナを終了できます。

▶-vオプションを使えばローカルのディレクトリをコンテナ内のディレクトリにマウントできる

ローカルのディレクトリをコンテナ内のディレクトリにマウントしたい場合は、-vオプションを使うことができます。以下のコマンドはユーザ・ホームディレクトリ直下のprojectsというフォルダをコンテナのルート直下projectsにマウントしています。

```
$ docker run -it -p 8888:8888 -p 6006:6006 -v $(pwd)/projects:/projects --name tf gcr.io/tensorflow/tensorflow /bin/bash
```

\$(pwd)はカレント・ディレクトリで、Docker Quickstart TerminalのデフォルトはWindowsユーザーのホーム・ディレクトリとなっています(例:C:/Users/watanabe)。echo \$(pwd)を実行すると実際のディレクトリが確認できます。

コンテナ内の/notebooksにはサンプル・コードが含まれています。このディレクトリをマウント後に指定してしまうと、サンプル・コードが削除されてしまうため、サンプル・コードを参照したい人はマウント後に別のディレクトリを指定するようにしましょう。

● TensorFlowの動作確認

上記docker runの実行が完了するとコンテナが立ち上がっている状態になっているので、Docker Quickstart Terminalがコンテナ内のOSにアクセスし、待機している状態になっています。ここでコマンドを打つとコンテナ内のOSに対して司令を出すことになります。TensorFlowの動作確認ができるようPythonを起動します。

\$ python

Pythonが起動するとターミナルの先頭が\$から>>>へ切り替わります。以下のプログラムが動作することを確認しましょう。先頭に>>>の表示がない行がプログラムからの出力です。

```
>>> import tensorflow as tf
>>> hello = tf.constant('Hello
World!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
Hello World!
>>> a = tf.constant(9)
>>> b = tf.constant(27)
>>> print(sess.run(a + b))
36
>>>
```

● Virtual Machineの仕様調整

先ほど少し触れたとおり、Docker ToolboxはOracle VM VirtualBoxを利用して動作します。Dockerはdefaultという名称のVirtual Machineを自動的に作成、利用しますが、初期設定ではCPUが1コア、メモリが1GBになっています。ちょっと触ってみる程度のプログラミングであればこれで問題ありませんが、少し本格的なディープ・ラーニングのニューラル・ネットワークを実装、トレーニングする場合には、Oracle VM VirtualBoxの画面からCPU、メモリを増設するといいでしょう。Virtual Machineの設定を変更するには、Docker Quickstart Terminalを一度シャットダウンする必要があります。

パターン2…直接サポートしているOSのPython環境

Pythonのpip(パッケージ管理システム)を利用して直接、TensorFlowをインストールできます。r012からサポートされたWindowsへのインストールもこの方法で行うことができます。

● Python開発環境のインストール

まだPythonの開発環境をインストールしていない場合は、事前にインストールする必要があります。

▶ Ubuntu/Linux(64ビット)

```
$ sudo apt-get install python-pip
python-dev
```

▶ Mac OS X

```
$ sudo easy_install pip
$ sudo easy_install --upgrade six
```

▶ Windows

Anaconda (Python 3.5系)^{注6}、またはPython 3.5^{注7}を各ウェブ・サイトからダウンロードし、インストールします。サポートされているWindowsは64ビットだけで、Pythonの系統は3.5だけです。

● TensorFlowのインストール

Pythonのバージョンに応じて以下のコマンドを実行します。

▶ Ubuntu/Linux(64ビット)

• Python 2.7の場合

```
$ export TF_BINARY_URL=https://
storage.googleapis.com/tensorflow/
linux/cpu/tensorflow-0.11.0rc2-
cp27-none-linux_x86_64.whl
$ sudo pip install --upgrade $TF_
BINARY_URL
```

• Python 3.4の場合

```
$ export TF_BINARY_URL=https://
storage.googleapis.com/tensorflow/
linux/cpu/tensorflow-0.11.0rc2-
cp34-cp34m-linux_x86_64.whl
$ sudo pip3 install --upgrade $TF_
BINARY_URL
```

• Python 3.5の場合

```
$ export TF_BINARY_URL=https://
storage.googleapis.com/tensorflow/
linux/cpu/tensorflow-0.11.0rc2-
cp35-cp35m-linux_x86_64.whl
$ sudo pip3 install --upgrade $TF_
BINARY_URL
```

▶ Mac OS X

• Python 2.7の場合

```
$ export TF_BINARY_URL=https://
storage.googleapis.com/tensorflow/
mac/cpu/tensorflow-0.11.0rc2-py2-
none-any.whl
$ sudo pip install --upgrade $TF_
BINARY_URL
```

^{注6}: <https://www.continuum.io/downloads>

^{注7}: <https://www.python.org/downloads/release/python-352/>

- Python 3.4. または3.5の場合


```
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-0.11.0rc2-py3-none-any.whl
$ sudo pip3 install --upgrade $TF_BINARY_URL
```

▶ Windows

- Python 3.5


```
$ pip install --upgrade https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-0.12.0rc0-cp35-cp35m-win_amd64.whl
```

Windows環境でAnacondaを利用してインストールしようとすると、以下のようなエラーが発生する可能性があります。

```
Cannot remove entries from nonexistent file /Users/watan/anaconda/envs/tensorflow/lib/python3.5/site-packages/easy-install.pth
```

これはsetuptoolsというライブラリとTensorFlowの互換性に問題があるために起こるようなので、そのような場合にはsetuptoolsを更新すると解消します。

```
pip install --upgrade -I setuptools
```

● TensorFlowに必要なライブラリ

TensorFlowの動作に必要なライブラリがインストールされていない場合、またはバージョンが古い場合は、それらのライブラリが自動的にインストール、または更新されます。

既存の環境を保持したい場合は、次に紹介するAnacondaを利用すると、環境を切り分けてインストールできます。2016年11月1日現在、最新のバージョンr0.11では、以下のライブラリが指定されています。

NumPy 1.11.0以上

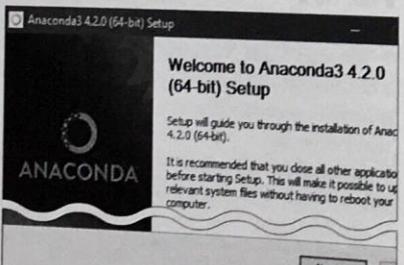


図5 ウィンドウの指示に従いAnacondaのインストールを行う

Six 1.10.0以上

- protobuf 3.1.0
- SciPy 0.15.1以上
- wheel 0.26以上 (Python 3だけ)
- wheel (Python 2だけ)
- mock 2.0.0以上 (Python 2だけ)

インストールが完了したら、パターン1で紹介した「●TensorFlowの動作確認」を実行し、正常にインストールされていることを確認しましょう。

パターン3…直接サポートしているOSでPythonパッケージを使う

AnacondaはPythonディストリビューションの上で、Python開発環境と主要なライブラリを一括でインストールできます。condaという独自のパッケージ管理システムを持っており、複数の異なるライブラリのセットを切り分けて管理できます。

ライブラリ間には複雑な依存関係があるため、新しいライブラリを1つインストールすると他のライブラリがインストールされたり、既存のライブラリが壊されたりしますが、環境を切り分けておくことで、特定のアプリケーション用の開発環境を維持できます。

環境の切り分けはアプリケーション・レベルで行われ、バーチャル・マシンのような仮想環境を使うわけではありません。TensorFlowは使っているマシンに直接インストールすることになりますので、pipのインストール同様、TensorFlowがサポートするOSを対象です。

● Anacondaのインストール

Anacondaの公式ページ⁸から、プラットフォームに応じたインストーラをダウンロードしましょう。2016年12月7日現在最新のバージョンは4.2.0で、Python 2.7. またはPython 3.5が選択できるようになっています。インストーラをダウンロードしたら実行し、ウインドウの指示に従いインストールを行います(図5)。

● TensorFlow用の環境作成

Pythonのバージョンに応じて以下のコマンドを実行します。PYTHON_VERSIONを2.7. または3.5に変更してください。

```
$ conda create -n tensorflow
python=PYTHON_VERSION
```

● 環境のアクティベーション

作成したtensorflowを以下のコマンドでアクティベートします。環境がアクティブになると、ターミナルにします。

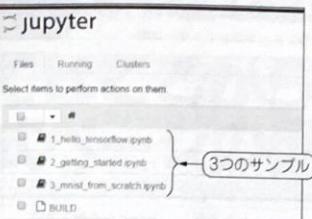


図6 コンテナ内の/notebooksでJupyter Notebookを起動してホーム画面にアクセスすると3つのサンプル・コードが確認できる

先頭が(tensorflow)\$に切り替わります。

```
$ source activate tensorflow
```

● TensorFlowのインストール

環境tensorflowにTensorFlowをインストールします。また、Jupyter Notebookを利用するためIPythonも合わせてインストールします。インストールが完了したら、パターン1で紹介した「●TensorFlowの動作確認」を実行します。以下のコマンドで正常にインストールされていることを確認しましょう。

```
(tensorflow)$ conda install -c conda-forge tensorflow
(tensorflow)$ conda install ipython
```

● 環境のディアクティベーション

環境tensorflowから抜けるときは以下のコマンドを実行します。ターミナルの先頭が\$に戻ります。

```
(tensorflow)$ source deactivate
```

主に使用するPython環境

Jupyter Notebookはブラウザ上でプログラミング&実行を可能してくれるブラウザ・ベースのアプリケーションです。Python以外にもR, Julia, Scalaなど、さまざまなプログラミング言語に対応しています。Notebookという名前の通りテキストや図も含むことができるため、レポート作成やコードの共有に適しています。以降の章ではJupyter Notebookを通じてTensorFlowを利用ていきますので、ここで簡単な使用方法を説明します。

● 起動

ターミナルからjupyter notebookを実行すると、カレント・ディレクトリにマウントする形でJupyter Notebookが起動します。ブラウザから「localhost:8888」にアクセスするとJupyter

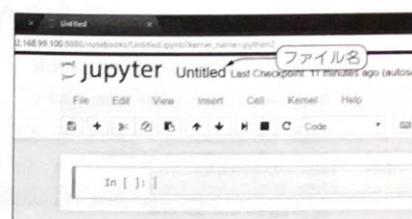


図7 ホーム画面右上の「New」ドロップダウン・リストからPython 2またはPython 3を選択すると自動的に新規ファイルが作成されてブラウザの新規タブ(またはウインドウ)に表示される

Notebookのホーム画面が表示されます。

Dockerを利用している場合は「コンテナのIPアドレス:8888」にアクセスします。コンテナのIPアドレスはDocker Quickstart Terminal起動直後、くじらの絵の右下に表示されています(図4)。コンテナ内の/notebooksでJupyter Notebookを起動し、ホーム画面にアクセスすると3つのサンプル・コードが確認できます(図6)。

● 使い方

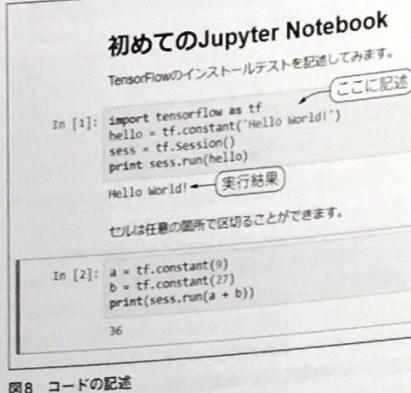
▶ 新規ファイル作成

Jupyter Notebookのホーム画面右上の「New」ドロップダウン・リストから、Python 2. またはPython 3を選択すると、新規ファイルが作成され自動的にブラウザの新規タブ(またはウインドウ)に表示されます(図7)。ファイル名はデフォルトでUntitledとなっていますが、画面左上Jupyterロゴマークの右側に表示されているファイル名をクリックすると編集できます。ホーム画面に戻ると、新規ファイルが1つ追加され、ファイル名左側のノートアイコンが緑になっていることが確認できます。実行中のNotebookはノートアイコンが緑になります。

▶ コードの記述

画面中央の「セル」とよばれる部分にコードまたはテキストを記述していきます。セルにはタイプがあり、コードを記述する場合は「Code」。テキストを記述する場合は「Markdown」を選択します。セルのタイプは画面上部のメニュー「Cell→Cell Type」から選択できます。セル・タイプが「Code」の場合は、セルの左側にIn []が表示されます。また、新しいセルを挿入したい場合は、メニュー「Insert」からできます。

タイプ「Code」のセルは、[Ctrl+Enter](実行後カーソルをそのセルに留める)、または、[Shift+Enter](実行後カーソルを次のセルへ移す)でそのセルを実行できます。実行の出力結果が標準出力へ出力されるプログラムの場合、そのセルの下に実行結果が出力されま



(図8) タイプ「Markdown」のセルは、[Ctrl+Enter]、または[Shift+Enter]を押下することで、テキストを Markdown として表示させることができます。

Notebookにコードまたはテキストを記述する場合、Notebookは「Edit」モードになっています。[Esc]を押下することで「Command」モードに切り替えることができます。

▶ショートカット・キー

それぞれのモードでさまざまなショートカット・キー(表1)が用意されていますので、これらを活用するとコーディングを効率的に行えるようになります。

● プログラム例

TensorFlowがインストールされているかテストしてみます。

```
import tensorflow as tf
hello = tf.constant('Hello World!')
sess = tf.Session()
print(sess.run(hello))
Hello World!
セルは任意の箇所で区切ることができます。
a = tf.constant(9)
b = tf.constant(27)
print(sess.run(a + b))
出力は次のようにになります。
36
```

わたくし

キー	動作
F	検索、置換
Ctrl+Shift+P	コマンド・パレットを開く
Enter	エディット・モードへ移行
Shift+Enter	セルを実行し、下のセルへ進む
Ctrl+Enter	セルを実行する
Alt+Enter	セルを実行し、下へセルを挿入する
Y	セルをコード用セルへ変換
M	セルをマークダウン用セルへ変換
R	セルをプレーンセルへ変換
I	セルを見出し1へ変換
2	セルを見出し2へ変換
3	セルを見出し3へ変換
4	セルを見出し4へ変換
5	セルを見出し5へ変換
6	セルを見出し6へ変換
K	上のセルを選択
Up	上のセルを選択
Down	下のセルを選択
J	下のセルを選択
Shift+K	選択セルを上方向へ拡大
Shift+Up	選択セルを上方向へ拡大
Shift+Down	選択セルを下方向へ拡大
Shift+J	選択セルを下方向へ拡大
A	上にセルを挿入
B	下にセルを挿入
X	選択されたセルを切り取り
C	選択されたセルをコピー
Shift+V	上にセルを貼り付け
V	下にセルを貼り付け
Z	セル削除を取り消し
D.D	選択されたセルを削除
Shift+M	選択されたセルの結合(一つのセルのみ選択している場合は下のセルと結合)
Ctrl+S	保存、チェックポイント作成
S	保存、チェックポイント作成
L	行番号表示、非表示のスイッチ
O	選択されたセルのアウトプット表示、非表示のスイッチ
Shift+O	選択されたセルのアウトプットのスクロールバー表示、非表示のスイッチ
H	ショートカット・キーの表示
LI	カーネル中断
0.0	カーネル再起動(確認ダイアログあり)
Esc	ページ(ヘルプの表示窓)を閉じる
Q	ページ(ヘルプの表示窓)を閉じる
Shift+Space	ノートブックのスクロール・アップ
Space	ノートブックのスクロール・ダウン

Appendix 1 専門用語&英語が苦手な人のために

全てはここから! 公式ページの歩き方

足立 悠

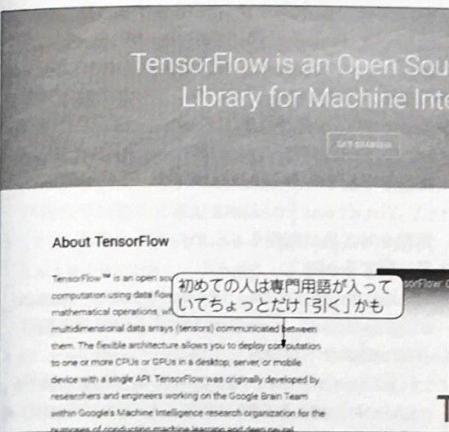


図1 全てはここから! Google公式のTensorFlowサイト
英語である上に専門用語なのでとっつきにくい
<https://www.tensorflow.org/>

TensorFlowはGoogle社が開発したオープンソースのディープ・ラーニング・ライブラリです。商用非商用を問わず使用できます(Apache 2.0 ライセンス)。

図1にGoogle公式のTensorFlowサイトを示します。TensorFlowの概要、インストール方法、使い方、使用できる関数、チュートリアルなどを確認できます。

各項目ごとに詳しい内容が記載されていますが、なにぶん英語であることと、難解な専門用語が多数使われていることから、とっつきにくいようです^{注1}。

本稿ではこれからTensorFlowを活用していく方向けに、差し当たって重要なポイントをピックアップして解説します。

お宝多数! 公式サイトの構成

Google社公式のTensorFlowサイト(<https://www.tensorflow.org/>)の構成は以下です。

注1:筆者らはスタートアップ・セミナを開催しています。
<http://www.ksk-anl.com/event>

(トップ)	
— GET STARTED	… (1)
— TUTORIALS	… (2)
— HOW TO	… (3)
— MOBILE	… (4)
— API	… (5)
— RESOURCES	… (6)
— ABOUT	… (7)

● (1) GET STARTED

これからTensorFlowを使い始める場合は、まずこのページを参照しましょう。

(GET STARTED)

- | — Introduction : はじめに
- | — Download and Setup : 入手と設定方法
- | — Basic Usage : コードの書き方

● (2) TUTORIALS

TensorFlowを使って開発できる19種類のアプリケーションについて紹介しています。

(TUTORIALS)

- | — Basic Neural Networks
手書き文字画像MNISTを使ったニューラル・ネットワーク・モデルの構築方法です。
- | — MNIST For ML Beginners
手書き文字画像MNISTを使った畳み込みニューラル・ネットワーク(CNN)モデルの構築方法です。
- | — TensorFlow Mechanics 101
作成したネットワーク・モデルの概観や学習状況を確認できる「TensorBoard」の使い方です。
- | — tf.contrib.learn Quickstart
あやめの花のデータ(花弁やがくの長さなどの数値)を使った分類モデルの構築方法です。
- | — Overview of Linear Models with tf.contrib.learn
TensorFlowにおける線形モデルの考え方です。
- | — Linear Model Tutorial
国勢調査の収入データを使ったロジスティック回帰モデルの構築方法です。

- | - Wide and Deep Learning Tutorial
国勢調査の収入データを使ったロジスティック回帰/ニューラル・ネットワーク/この2つを組み合わせたモデル、の3つのモデルを構築する方法です。
- | - Logging and Monitoring Basics with tf.contrib.learn
あやめの花のデータ(花弁や額の長さなどの数値)を使った分類モデルの精度をMonitor APIとTensorBoardで確認する方法です。
- | - Building Input Functions with tf.contrib.learn
TensorFlowにおける入力関数の作成方法です。
- | - Creating Estimators in tf.contrib.learn
評価器(予測の精度)の作成方法です。
- | - TensorFlow Serving
構築したモデルを別システムに適用する方法です。
- | - Convolutional Neural Networks
CIFAR-10画像データセットによるCNNモデルの構築方法です。
- | - Image Recognition
ImageNet画像データセットによるCNNモデルの構築方法です。
- | - Vector Representations of Words
単語ベクトル(word2vec)モデルの構築方法です。
- | - Recurrent Neural Networks
出現する単語を予測するための再帰型ニューラル・ネットワーク(Recurrent Neural Network, RNN)モデルの構築方法です。
- | - Sequence-to-Sequence Models
機械翻訳に使われる配列モデル(seq2seq)の構築方法です。
- | - SyntaxNet: Neural Models of Syntax
TensorFlowにおける自然言語処理フレームワーク「Syntax」の使い方です。
- | - Mandelbrot Set
マンデルブロ集合の可視化方法です。
- | - Partial Differential Equations
偏微分方程式のシミュレーション実施方法です。
まずは、上の3つ「MNIST For ML Beginners」、「MNIST For ML Beginners」、「TensorFlow Mechanics 101」を実行してみましょう。チュートリアルにはサンプルコードが付いています。

● (3) HOW TO

GPUでの実装、分散処理方法などを紹介しています。チュートリアルに慣れたら、自分でカスタマイズしてみましょう。

● (4) MOBILE

TensorFlowを使って構築したモデル・アプリケーションを「Android」、「iOS」、「Raspberry Pi」で実行する方法について紹介しています。

● (5) API

TensorFlowで使用できるPython、C++のAPIを一覧で紹介しています。

ここでは、基本的なチュートリアル3種「MNIST For ML Beginners」、「MNIST For ML Beginners」、「TensorFlow Mechanics 101」で使うものに絞って紹介します。

▶ TensorFlowの処理に関するAPI

• `tf.Session()`

処理を実行するためのクラスです。

• `tf.InteractiveSession()`

処理を対話的に実行するためのクラスです。

▶ 層に関するAPI

• `tf.nn.dropout(x, keep_prob, noise_shape=None, seed=None, name=None)`
ドロップアウトを計算します。

• `tf.placeholder(dtype, shape=None, name=None)`
入出力データの格納を定義します。

▶ 重みやバイアスなど変数に関するAPI

• `tf.global_variables_initializer()`
変数を初期化します。

• `tf.truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)`
変数を初期化(乱数発生)します。

• `tf.Variable(<initial-value>, name=<optional-name>)`
行列要素の格納を定義します。

▶ 活性化関数に関するAPI

• `tf.nn.relu(features, name=None)`
ReLU関数を定義します。

• `tf.nn.softmax(logits, dim=-1, name=None)`
ソフトマックス関数を定義します。

▶ 基本的な行列計算に関するAPI

• `tf.argmax(input, axis=None, name=None, dimension=None)`
変数の最大値を返します。

• `tf.cast(x, dtype, name=None)`
変数の型変換を実行します。

• `tf.constant(value, dtype=None, shape=None, name='Const', verify_shape=False)`

定数の行列要素を生成します。

• `tf.equal(x, y, name=None)`
2つの変数の値が等しいかどうか比較します。

• `tf.matmul(a, b, transpose_a=False, transpose_b=False, adjoint_a=False, adjoint_b=False, a_is_sparse=False, b_is_sparse=False, name=None)`
2つの行列の積を計算します。

• `tf.truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)`
切断正規分布によるランダムな値を出力します。

• `tf.reduce_mean(input_tensor, axis=None, keep_dims=False, name=None, reduction_indices=None)`
行列の平均値を計算します。

• `tf.reduce_sum(input_tensor, axis=None, keep_dims=False, name=None, reduction_indices=None)`
行列の合計値を計算します。

• `tf.reshape(tensor, shape, name=None)`
行列の形式を変更します。

▶ CNNアルゴリズム実装に関するAPI

• `tf.nn.conv2d(input, filter, strides, padding, use_cudnn_on_gpu=None, data_format=None, name=None)`
畳み込み演算を定義します。

• `tf.nn.max_pool(value, ksize, strides, padding, data_format='NHWC', name=None)`
最大プーリングを定義します。

▶ 学習に関するAPI

• `tf.train.GradientDescentOptimizer(learning_rate)`
勾配降下法による計算をします。

• `tf.train.AdamOptimizer(learning_rate)`
Adamアルゴリズムによる計算をします。

以上が最初のチュートリアルで使用するAPIです。TensorFlowでは基本的にユーザ自身で行列計算を実装してモデルを構築する必要があります。

● (6) RESOURCES

ホワイトペーパーやコミュニティ、よくある質問などを紹介しています。

The screenshot shows the TensorFlow website's 'Basic Neural Networks' tutorial. It includes sections like 'What's New', 'Basic Neural Networks', 'MNIST For ML Beginners', 'Deep MNIST For Experts', and 'TensorFlow Mechanics 101'. A callout box highlights the 'Basic Neural Networks' section.

図2 TensorFlowサイトのチュートリアル

TensorFlow公式サイトには、画像や自然言語含め19種類(2016年11月末時点)のチュートリアルが用意されている。これからTensorFlowを使い始める方は、チュートリアル一番上から順に試していくと良い。

● (7) ABOUT

クレジットや謝辞を紹介しています。

以上がサイトの概要です。どのページに何の情報が載っているかを簡単に紹介しました。詳細を知りたい場合は各ページを参照してください。

● サンプルを通して慣れる

まずは、先にご紹介したTensorFlow公式サイトのチュートリアルをご確認ください(図2)。手書き数字(0～9)の画像データセットMNIST(Mixed National Institute of Standards and Technology database)^{注2}を使って、基本的なニューラル・ネットワーク・モデルや畳み込みニューラル・ネットワーク・アルゴリズムによるモデル学習・構築をできます。画像の他、自然言語のチュートリアルも用意されています。

あだち・はるか

注2: <https://www.tensorflow.org/tutorials/>

レッスン2…TensorFlow プログラミングの基礎知識

渡邊輝

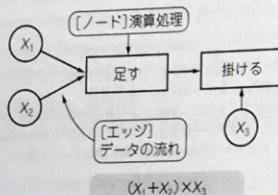


図1 TensorFlowの基本概念「グラフ」のイメージ

データがエッジ(矢印線)によって流れノードでデータに対しての演算処理が実行される。エッジは必ず一方通行で両方向になることはない。その性質から有向非巡回グラフとも呼ばれる

ここでは前章で構築した開発環境を利用して、コーディングをしながらTensorFlowの基本への理解を深めていきます。

●準備…開発環境の立ち上げ

Dockerで開発環境を作成した方は、Docker Quickstart Terminalを立ち上げ、TensorFlow開発用コンテナを起動し(docker startコマンド)、コンテナOSにアクセスしたらJupyter Notebookを起動してください(jupyter notebookコマンド)。

マシンに直接インストールした方は、ターミナルまたはコマンド・プロンプトからJupyter Notebookだけを起動します。ブラウザからJupyter Notebookへアクセスし、サンプル・コーディング用の新規ファイルを作成したら準備完了です。

●各ノードをエッジでつながりたいものがグラフ

TensorFlowは「グラフ」という概念に基づいて算術演算を実装するよう設計されています。従ってTensorFlowを利用する場合には、「グラフ」とは何か、それをどのように実装、実行するのかを理解しなければなりません。

グラフを構成する要素はノードとエッジです(図1)。ノードは丸や四角で表現され、特定の演算処理を実装します。エッジはノードをつなぐ矢印線でデータの流れを表現します。1つのノードの観点から

表1 TensorFlowのデータ型一覧

データ型	Pythonの型	説明
DT_FLOAT	tf.float32	32ビット浮動小数点
DT_DOUBLE	tf.float64	64ビット浮動小数点
DT_INT8	tf.int8	8ビット符号あり整数
DT_INT16	tf.int16	16ビット符号あり整数
DT_INT32	tf.int32	32ビット符号あり整数
DT_INT64	tf.int64	64ビット符号あり整数
DT_UINT8	tf.uint8	8ビット符号なし整数
DT_UINT16	tf.uint16	16ビット符号なし整数
DT_STRING	tf.string	変数長のバイオ配列(テンソルの各要素がバイオ配列)
DT_BOOL	tf.bool	真偽値
DT_COMPLEX64	tf.complex64	32ビット浮動小数点を2つ使う複素数(それぞれ実数部分と虚数部分を表現する)
DT_COMPLEX128	tf.complex128	64ビット浮動小数点を2つ使う複素数(それぞれ実数部分と虚数部分を表現する)
DT_QINT8	tf.qint8	量子化された処理に使われる8ビット符号あり整数
DT_QINT32	tf.qint32	量子化された処理に使われる32ビット符号あり整数
DT_QUINT8	tf.quint8	量子化された処理に使われる8ビット符号なし整数

見ると、自分自身に向かれたエッジは入力を意味し、外向きのエッジは演算後の出力を意味します。つまり、グラフとは一連の演算処理です。データがエッジによって流れを制御され、ノードに実装された演算処理によって次々と姿を変えていくことになります。TensorFlowはグラフを構築するための部品を提供します。

●グラフ内に流せるデータはテンソルという構造をとる

グラフ内に流すことのできるデータはテ

コラム TensorFlowには1000のAPIが用意されている

渡邊輝

ここまでTensorFlowの基本概念を理解するため初步的な演算処理のコードを見てきましたが、実際にTensorFlowが提供するAPIは基本的な算術演算からディープ・ラーニング(ニューラル・ネットワーク)のコンポーネントまで、実にさまざまなもの

が存在します。最新バージョンのr0.12では、合計44カテゴリ、約1000のAPIが公開されています注A。

注A : https://www.tensorflow.org/versions/r0.12/api_docs/python/index.html

リスト1 単純な行列演算を実行するグラフを構築する

```
import tensorflow as tf

##### グラフ構築 #####
# 1x2の定数マトリックスを出力するノードをグラフに追加
matrix1 = tf.constant([[3., 3.]])

# 2x1の定数マトリックスを出力するノードをグラフに追加
matrix2 = tf.constant([[2., 2.]])

# matrix1とmatrix2を入力とし、その積を算出するノードをグラフに追加
product = tf.matmul(matrix1, matrix2)

# ----- この時点ではまだ処理は実行されていない -----
##### グラフ実行 #####
# セッションを作成
sess = tf.Session()

# グラフ実行
result = sess.run(product)
print(result)
# ==> [[ 12.]]

# セッションをクローズ
sess.close()
```

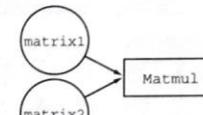


図2 リスト1を図にしたもの

リスト1の場合、仮に構築と実行が同時に行われるようなしきみであればC++からPythonへのデータの受け渡しが3回行われることになりますが、TensorFlowはそれを1回で実現しています。演算量の少ないプログラムでは言語間のデータ授受を意識する必要性を感じないかもしれません。ディープ・ラーニングのように膨大な量の演算を実行するケースになると、そのメリットが顕著になります。

●セッションと実行

リスト1にあったように、グラフを実行するためにセッションを作成し、run()メソッドをコールする必要があります。tf.Session()でセッションを生成していますが、引き数にグラフ名を指定しない場合、デフォルトのグラフに対するセッションが生成されます。セッションは指定したグラフの処理(ノード)に対して実行環境を提供します。演算が完了したらリソースを開放するためにクローズする必要があります。

実際にはリスト2のようにwith句でセッションを記述することが多くなります。with句でセッションを記述することで、どの処理がどのセッションで実行されるかがコード上明確になるというメリットと、with句が終了した時点でセッションが自動的にクローズされるというメリットがあります。

リスト2 with句でセッションを記述することで処理がどのセッションで実行されるかが明確になる

```
with tf.Session() as sess:
    matrix1 = tf.constant([[3., 3.]])
    matrix2 = tf.constant([[2., 2.]])
    product = tf.matmul(matrix1, matrix2)
    sess.run(product)
```

リスト3 変数は必ず使用前に初期化を行う

```
# 変数生成。0で初期化
state = tf.Variable(0, name="counter")

# 状態「state」に1を足すノードを追加
one = tf.constant(1)
new_value = tf.add(state, one)
update = tf.assign(state, new_value)

# 初期化ノードを追加
init_op = tf.initialize_all_variables()

# グラフを起動し実行
with tf.Session() as sess:
    # 初期化実行
    sess.run(init_op)
    # 変数「state」の初期値を出力
    print(sess.run(state))
    # 1を足す処理を3回実行
    for _ in range(3):
        sess.run(update)
        print(sess.run(state))

# output:
# 0
# 1
# 2
# 3
```

● 変数と初期化

グラフは変数を扱うことができ、「状態」を保持できます。変数はグラフへ入力するテンソルやグラフから出力されるテンソルを保持したり、ディープ・ラーニングなどの機械学習モデルをトレーニングする際にそのパラメータを保持したりします。ただし、変数と言ってもあくまでもグラフのコンポーネントであり、厳密に言うと状態が参照できるテンソルを出力するノードということになります。変数は必ず使用前に初期化を行う必要があります。定義段階で指定した型は不変です（リスト3）。

● 取り出し（Fetch）

グラフから演算結果を取り出す（Fetchする）ために、演算結果を取得したいノードを引き数に渡しSessionオブジェクトのrun()メソッドをコールする必要があります。ノードは幾つでも引き数として渡すことが可能です。リスト4では2つのノードを引き数として渡しているため、それぞれのノードの演算結果が返却されています。

取り出す対象の演算結果に必要な全ての演算は一度だけ実行されます。リスト4のinput2とinput3の足し算はmulとintermedの両方で必要な演算ですが、1回だけ実行され、その演算結果を両方のノードで持ります。

● 送り込み（Feed）

グラフにテンソルを送り込む（Feedする）と同時に、グラフを実行することも可能です。実行時にデータ

リスト4 グラフから演算結果を取り出す

```
input1 = tf.constant([3.0])
input2 = tf.constant([2.0])
input3 = tf.constant([5.0])
intermed = tf.add(input1, input2)
mul = tf.mul(input1, intermed)

with tf.Session() as sess:
    result = sess.run([mul, intermed])
    print(result)
    # => array([ 21.], dtype=float32), array([ 7.],
    #           dtype=float32)
```

リスト5 グラフにテンソルを送り込む

```
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
output = tf.mul(input1, input2)

with tf.Session() as sess:
    print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))
    # => [array([ 14.], dtype=float32)]
```

タを指定することで、より汎用性の高い演算処理を実装できます。リスト5のようにtf.placeholder()によってテンソルを送り込むプレースホルダをグラフ内に作成する方法が最もよく使われます。

実行時に送り込むテンソルはrun()メソッドの引き数feed_dictに指定します。Dictionary型の変数を用い、キーにテンソル名、値に送り込むデータを指定します。

実装したい演算内容に応じて検索をしながら利用する必要がありますが、ディープ・ラーニング・ネットワークの構築では、以下のカテゴリのAPIがよく使われます。

Building Graphs: グラフ構築

Variables: 変数

Tensor Transformations: テンソル変換

Math: 基礎算術

Images: 画像処理

Neural Network: ニューラル・ネットワーク

Training: モデルのトレーニング

APIを利用することで複雑な演算処理を実装するところからは開放されますが、そのAPIがどのような演算を行なうのかを理解しなければ利用するのは難しいでしょう。特に多次元配列（=Tensor）の行列演算処理は、ディープ・ラーニングの基本コンポーネントになるため、理解をすることが必要不可欠です。

わたなべ・てる

レッスン3… TensorFlow初体験

渡邊 輝

ここではTensorFlowを初めて触るという方のために、簡単なニューラル・ネットワークを実装し、分類モデルを構築してみます。分類する対象は手書き数字で「0」から「9」までの画像データです。画像データはピクセルごとの数字の集合として表現される非構造化データですが、そのような複雑な数字の集合から画像の意味する数字を判別する分類モデルを構築します。

手書きの数字は同じ数字でもそれぞれの数字が独自の形態を持っています。また、書き手によって書き方、癖が異なるため、1つの数字に対するデータでもピクセル・レベルの数字データとして見ると、大きなばらつきを持った集合となります。このようなばらつきを持ったデータに対して、従来のプログラミング的なアプローチで人間が認識する意味（この場合数字0～9のいずれか）に分類しようとすると、ほぼ無限に繰り返されるIF文のようなプログラムとなり、複雑にもかかわらず分類精度のあまり高くないものになってしまいます。

ニューラル・ネットワークのような機械学習モデルにデータの特徴（ばらつきの傾向）を学習させることで、そのようなプログラミングをすることなく高い精度での判別が可能となります。



図1 ディープ・ラーニング・トレーニング用の画像データ
MNIST (<http://yann.lecun.com/exdb/mnist/>)
機械学習分類モデルのベンチマークとしてよく用いられる

本稿の内容はTensorFlow公式ページ注2で紹介されているものの意訳+追記になります。

ステップ1 データのダウンロード

● 入手先

前章までで解説した開発環境Jupyter Notebookを起動します。TensorFlowのチュートリアル用に用意されたinput_dataという関数を使ってMNISTデータを読み込みます。初めてコードを実行する場合はデータのダウンロードから実行されます。

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

注1: <http://yann.lecun.com/exdb/mnist/>
注2: <https://www.tensorflow.org/versions/r0.11/tutorials/mnist/beginners/index.html>

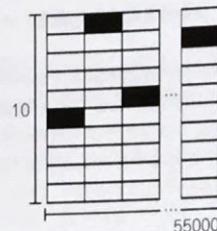


図5 トレーニング用のラベル・データ全体

出力された確率のうち一番高い確率の数字を予測結果として採用します。ソフトマックス関数は、1つの画像に対して0～9それぞれの数字である確率を出力します。また、その確率の合計は常に1になるよう正規化されます。

計算処理は大きく2つのステップから構成されます。画像データから特定の分類クラス(数字)に対する証跡を算出するステップと、その証跡を確率へ変換するステップです。

● 数字*i*に対する証跡を算出するステップ

数字*i*に対する証跡を算出するステップは以下のように単純な線形回帰式の合計になります。

$$\text{証跡}_i = \sum_j W_{ij} x_j + b_i$$

784あるピクセル・データ x_j にウェイト W_i をかけ、バイアス b_i を足し込んだ値を証跡としています。あるピクセル・データが特定の数字に分類される根拠を強める場合はポジティブな値、その逆の場合にはネガティブな値がウェイトに設定されます。バイアスは入力データからは分離独立した形で確率を調整するために用いられます。

● 証跡を確率へ変換するステップ

この証跡を元に確率を算出するのが2つ目のステップです。

$$y = \text{softmax}(\text{証跡})$$

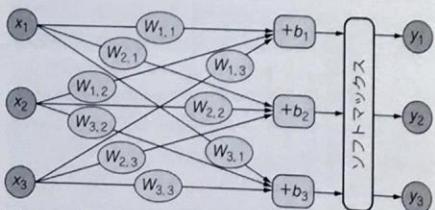


図6 ソフトマックス処理の集合をモデル化するとニューラル・ネットワークになる

ソフトマックス関数が活性化関数として使われ、0～9それぞれの数字に対する確率を出力します。証跡を確率へ変換していると考えるとよいでしょう。ソフトマックス関数は以下のように定義されます。

$$\text{softmax}(x) = \text{normalize}(\exp(x))$$

これを展開された形で表現すると以下のようになります。

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

● ソフトマックス処理の集合がニューラル・ネットワーク

上記ソフトマックスの仕組みを全ての分類クラスまとめたものが今回構築するニューラル・ネットワークになります(図6)。

左側の x_j ノードが入力層で、画像データを入力する部分になり、各 x_j が画像データの1ピクセルに当たります。また、 y_i が出力層で、0～9の各数字に対する確率を出力します。

図6にはシンプルに説明するために x が3つしかありませんが、今回のケースでは784個の x が入力となり、10個に集約される構成(b が10個)になります。各ピクセルにウェイト W をかけ合わせ、バイアス b を足し込んだものを10個の集約ノードごとに集計し、ソフトマックス関数へ渡します。ソフトマックス関数は画像が数字*i*である確率 y_i を出力します。実際の計算は以下のように行列演算を行うことで、効率良く実行されます。

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

ニューラル・ネットワークは入力層と出力層の間に複数の中間層を挟む(=ディープ・ラーニング)ことができますが、今回のチュートリアルでは入力層と出力層だけのシンプルなニューラル・ネットワークを構築します。つまり今回のケースでは単純な10個のソフトマックス処理の集合がニューラル・ネットワークということになります。

TensorFlowによる ニューラル・ネットワークの構築

それでは上記のようなニューラル・ネットワークを実際にTensorFlowで実装してみましょう。

● ライブリのインポート

TensorFlowを使うときのお決まりごとです、`as tf`としておくことで、TensorFlowライブラリを`tf`として参照できるようにしておきます。

```
import tensorflow as tf
```

値を学習するプロセスを構築します。

パラメータWとbの最適な値を学習するプロセスを構築

● まずは損失関数を定義

最適な W と b を学習するためにモデルの出力する確率 y と正解ラベル y' の差を表現した損失関数を定義します。トレーニング・データを繰り返しモデルに入力し、その出力と正解ラベルとの誤差を最小化するよう、少しづつ W と b を調節します。この W と b の調節プロセスを「トレーニング」と呼んでいます。

損失関数には交差エントロピーを用います。交差エントロピーは以下の式で定義されます。

$$H_y(y) = -\sum_i y'_i \log(y_i)$$

モデルの出力する確率 y と正解ラベル y' は画像ごとにそれぞれ長さ10の配列であることを思い出してください。確率は0～1の値をとる連続変数、正解ラベルは0、または1の値をとる離散変数です。交差エントロピーは各要素の差が損失として積み上がるよう定義されています。

● 正解ラベルのプレースホルダ

損失関数を定義するため、正解ラベル y' をグラフへ入力するプレースホルダを作成します。 y' をソースコード上 $y_$ で表現します。

$$y_ = \text{tf.placeholder(tf.float32, [None, 10])}$$

● 交差エントロピー

交差エントロピー $= \sum_i y'_i \log(y_i)$ をTensorFlowライブラリを利用して記述すると以下のようにになります。定義式をそのままソースコードに反映するような形で記述できます。

$$\text{cross_entropy} = \text{tf.reduce_mean}(-\text{tf.reduce_sum}(y_ * \text{tf.log}(y), reduction_indices=[1]))$$

計算を実行順に記載すると以下のようになります。

①`tf.log`を利用し、モデルがOutputする確率 y の対数を算出します。

②①で算出した対数と正解ラベル $y_$ を要素ごとに掛け合わせます。

③`tf.reduce_sum`を利用し、②で算出した要素ごとの損失を画像ごとにまとめます。`reduction_indices=[1]`は、配列の2次元目をなくすように足し算することを意味しています。

④`tf.reduce_mean`を利用し、トレーニングの平均のバッチに用いられた画像すべての損失の平均を算出します(厳密に言うとこのステップは交差

エントロピーの定義式に含まれていませんが、損失関数の意味自体に変わりはありません。

● 損失の最小化

損失関数を定義したら、それを最小化するプロセスを記述します。本来このような最小化のプロセスは、定義された式の微分をとり、その微分式が減少するようWとbとを調整しなければなりませんが、TensorFlowはそのプロセスを1つのAPIにまとめてくれています。

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

`tf.train.GradientDescentOptimizer`という最適化APIを利用し、交差エントロピー`cross_entropy`を最小化するノードを作成しています。このAPIはGradient Descent(確率的こう配降下法)というアルゴリズムを利用して最適化を実行します。0.5は学習率(Learning Rate)と呼ばれる最適化のパラメータで、この値が大きいほどトレーニング・バッチごとのWとbの調整幅が大きくなります。本来であればとても煩雑な最適化のプロセスが、これもまた、たった数行で記述できてしまいました。ここにもTensorFlowの便利さを垣間見ることができます。

仕上げ…手書き文字認識をやってみる

ここまでニューラル・ネットワークとそのパラメータを最適化する損失関数、そしてパラメータの最適化のプロセスをグラフ内に構築しました。ここからいよいよこれらの実行へ移ります。

● 変数の初期化

グラフを実行する際には必ず変数を初期化する必要がありますので、初期化ノードを作成します。

```
init = tf.initialize_all_variables()
tf.initialize_all_variables()を利用する
と、現在グラフに定義された全ての変数を一度に初期化できます。初期化自体もグラフを実行する必要があります。前章で説明したようにtf.Session()でセッションを生成し、sess.run()でグラフを実行します。
```

```
sess = tf.Session()
sess.run(init)
```

● トレーニングの実行

トレーニング・データ`mnist.train`から100ずつ画像データ、正解ラベルを取り出し、それぞれ`batch_xs`, `batch_ys`へ格納します。モデルの損失最小化

プロセス`train_step`と、これらのトレーニングデータを引き数に渡し、`sess.run()`をコールすることでモデルのトレーニングが実行されます。引き数`feed_dict={x: batch_xs, y: batch_ys}`は、ニューラル・ネットワーク構築時に作成したプレースホルダ`x`と`y`に`batch_xs`と`batch_ys`を入力することを意味します。`for`ループで100画像分のトレーニングを1000回繰り返します。

```
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y: batch_ys})
```

● モデル・パラメータの確認

最後にトレーニング後のモデルの精度を確認します。

トレーニングの実行そのものは何も出力しませんが、トレーニングそのものは実行され、その過程で`W`と`b`は最適化されています。試しに`b`の中身を確認してみましょう。

```
print(sess.run(b))
```

と入力すると、出力は、

```
[ -0.36774942  0.33787405
 0.11726006 -0.23839895  0.02524511
 1.2930851 -0.1073177   0.62908971
 -1.46908319 -0.22000468]
```

`b`は`tf.zeros`で初期化していますので、更新されていることが確認できます。

● 正答率算出

ニューラル・ネットワークが outputする各画像に対する10個の確率のうち一番確率の高いものをモデルの分類結果として、それが正解ラベルと一致するかをチェックします。

```
correct_prediction = tf.equal(tf.argmax(y, dimension=1), tf.argmax(y_, dimension=1))
```

`tf.argmax()`は引き数`dimension`で指定された次元(0 = 第1次元, 1 = 第2次元)のうち、最大の値が格納されたインデックスを返却します。ここではインデックス = 予測 / 正解ラベルとなり、`tf.equal()`は予測と正解が一致した場合には1を、異なる場合には0を返却します。

正解、不正解を表現した1, 0の配列`accuracy`を小数点浮動小数型`tf.float32`に変換し平均をとることで、正答率が算出できます。

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

正答率算出プロセスとテスト・データを入力にしてグラフを実行すると、モデルのテスト・データにおける正答率が出力されます。

```
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels}))
```

出力は、

0.9198

となります。ただし最適化プロセスはランダムな要素があるため数字は若干前後します。簡単なニューラル・ネットワークとトレーニング・プロセスを構築し実行しただけですが、テスト・データに対して92%弱の精度で分類ができました。最後にセッションをクローズし、チュートリアルを終了します。

```
sess.close()
```

TensorFlowを使いこなせるようになるためには

ここまでチュートリアルを読み進められて、TensorFlowライブラリが複雑な演算処理を隠蔽している、少量のコーディングでそれらの演算処理を実装可能にしていることを実感していただけたかと思います。また、それと同時にAPIの背後にある数学や行列演算の複雑さにへきえきした人も多いのではないでしょうか。

今回紹介できたのはほんの一端のAPIだけですが、これら以外にも何百というAPIが存在し、それぞれにその背景となる知識が存在します。また、GitHubでソースコードが公開されているため、世界中の人々が日々開発を進め進化し続けています。

他のさまざまなライブラリと比較すると前提となる知識の敷居が高く使いにくいと感じる方も多いと思いますので、最後に筆者が考える「TensorFlowが使えるようになるためコツ」を紹介して締めくくりたいと思います。

● その1…行列演算に慣れ親しむ

TensorFlowが行う演算処理は行列演算が基本です。ベクトルやマトリックスを掛け算したときにどのような配列がoutputされるのかを具体的にイメージできるようになるとグラフ内のデータの流れが把握しやすくなります。

● その2…実装例から学ぶ

ドキュメントを読んで何百とあるAPIを単純に学習していくのは効率的ではありません。GitHubで公開されているソースコードなど、具体的にTensorFlowを利用しているソースコードを読み、実際的な使い方から学ぶとよいでしょう。

● その3…オンラインコースを受講する

筆者は先日Creative Applications of Deep Learning with TensorFlowというオンライン・コースを受講しました。

<https://www.kadenze.com/courses/creative-applications-of-deep-learning-with-tensorflow/info>

ディープ・ラーニングを利用して画像や音声データで何か芸術的なことをしてみるというコンセプトのコースで、TensorFlowやディープ・ラーニングの基本概念と実際的な使い方をバランス良く学ぶことができました。最近はMOOC(Massive open Online Course)が盛んになってきていることもあり、検索をすればたくさんの選択肢が見つかるはずです。

● その4…自分がやりたいことをどう実装できるか考える

ある程度使い方の感覚をつかんだら、自分でやりたいことを実装してみるとよいでしょう。「やりたいこと」ベースに考え方を動かすと、学習スピードが格段にアップします。筆者はカメレオン・ネットなるものを先日作ってみました。

<https://github.com/watanabe8760/chameleon-net>

ディープ・ラーニングを利用して画像のスタイルを他の画像へ移転する技術のちょっとした応用です。画像のスタイル移転そのものはある程度確立されたディープ・ラーニングの技術ですが、それをカメレオンのように変化させる画像を出力するというひとひねりを入れるだけで結構苦労し、その過程でたくさん学びました。

● その5…前提となる知識は必要になった時点で勉強する

昨今世間にきわめてAIのコアとなるコンポーネントは数学的、統計的演算処理であり、TensorFlowはそのプラットフォームとなるライブラリです。そのような広大なナレッジ・ベースのライブラリを利用すると、その過程で知らないことやあまりなじみのない概念に遭遇します。そのようなときにイベント・ドリブンで学習をすることで、少しずつ確実に知識を蓄積できます。先に知識だけを考えると、その果てしなしさにあきらめてしまうということになりかねないので、必要になった時点で学習をするというのがポイントです。

わたなべ・てる