Pythonで体験する深層学習 2章(輪講用資料)

Pythonで体験する深層学習 2章輪講用の資料です。

1980年にGuido van Rossumが開発を始めた。 Pythonはシンプルで一貫性のあるプログラミング言語です。 Pythonコミュニティでは、可読性と一貫性が重視されています。

2.2 Python2かPython3か

Python2系とPython3系に互換性はない。 新たにPythonを始めるのであればPython3系で良い。 しかし、Python2系のコードは残っている。過去の遺産という意味では、Python2系も無視できない。本書ではPython2系を前提に記載した。

(発表者によるコメント 始まり)

Python2系

Python2系の最後のバージョン（2.7）は、開発を終了しており 大きな修正が入ることはない。 しかし、2.7のサポートは 2020 年まで延長されており、 Python2系とPython3系が混在している状況はまだ続くと思われる。

Python3系

Python3系は2008年にリリースされた。 現在積極的に開発が行われれているのはPython3系である。 メジャーなライブラリは、Python3系に対応されておりほぼ問題ない状態と言える。

(発表者によるコメント　終わり)

2.4 コーディングスタイル

可読性を高め、保守を容易にするために一貫したコーディングスタイルと文書を残すことには価値がある。

コーディングスタイルで有名なものとしてPEP8等がある。

(発表者によるコメント 始まり)

デファクトスタンダードとなる。PEP8を読みましょう。

PEP8 日本語訳 http://pep8-ja.readthedocs.io/ja/latest/

PEP8 チェックツールを使う pip install pep8でインストール可能です。 チェックツールを使用して自動でチェックを行いましょう。

(発表者によるコメント　終わり)

2.5 PythonとNumPy略説

2.5.2 コンテナ

Pythonにはリスト、辞書、集合、タプルというコンテナが標準実装されている。 複数の項目を一つにまとめる際、リストはかぎかっこ[]、辞書と集合はブレースカッコ{}、タプルは通常のカッコである。 辞書と集合の区別は各項目がキーとして属性値の対がコロン;で構成されていれば辞書である。

(発表者によるコメント 始まり)

- リスト：配列　値の変更が可能(mutable)
- 辞書：他言語のマップ、ハッシュ
- 集合：Set 重複する要素をもたない、順序づけられていない要素の集まり
- タプル：値の変更ができない（immutable） 変数や実態の順序に意味がある場合にはタプルが適用される。 リストとタプルのどちらでも良い場合、タプルを使用したほうが良い。

(発表者によるコメント　終わり)

2.5.5 NumPy

NumPyはPythonにおける科学技術計算の核となるライブラリーである。 多次元配列オブジェクトと配列処理用ツールから構成されている。後に述べる通り、柔軟な配列の操作が畳み込みニューラルネットワークの処理には欠かせない。

(発表者によるコメント 始まり)

なぜNumPyを使うのか？

1. 数値計算では大量の処理が発生するが、NumPyを使うことで実行時間を速くできる。（forループを使って計算するとPythonのリストを使うより遅くなるので、できるだけ、NumPy内に閉じて処理を行ったほうが良い。）
2. 大きさの違うarray同士で計算することが可能（ブロードキャスト） これにより複雑な計算を簡易に行うことができる。

(発表者によるコメント　終わり)

教科書参照

[5]ブロードキャスト

(発表者によるコメント 始まり)

ブロードキャストとは、大きさの異なる配列同士で計算を可能にする仕組みです。

例えば以下の例1だと1を足すことで、自動的に拡張されnp.array([1, 2]) + np.array([1, 1])を行ったものとして扱います。

```
#例1
>>> import numpy as np
>>> np.array([1, 2]) + 1
array([2, 3])
```

以下の例2だとnp.array([5, 6])を足すことで、自動的に拡張されnp.array([[1, 2], [3, 4]]) + np.array([[5, 5], [5, 5]])を行ったものとして扱います。

```
#例2
>>> import numpy as np
>>> np.array([[1, 2], [3, 4]]) + 5
array([[6, 7],
       [8, 9]])
```

以下の例3だとnp.array([2, 3])を足すことで、自動的に拡張されnp.array([[1, 2], [3, 4]]) + np.array([[2, 3], [2, 3]])を行ったものとして扱います。

```
#例3
>>> import numpy as np
>>> np.array([[1, 2], [3, 4]]) + np.array([2, 3])
array([[3, 5],
       [5, 7]])
```

(発表者によるコメント　終わり)

2.6 Pythonによる深層学習フレームワーク

2.6.1 Caffe

CaffeはC++, CUDAで書かれた深層学習の処理系である。 Python, C++及びMATLABから利用可能である。

Caffeは訓練済みモデルを提供しており、Caffeのコミュニティで訓練済みのパラメータファイルを動物園モデル(Model Zoo)と呼び共有している。 AlexNet, GoogLeNet, VGG, SPPを始めとする学習モデルが利用可能である。 学習ずみのcaffemodel（動物園モデルのファイル名の拡張子は.caffemodelである）を用いれば深層学習の成果を試すことができる。

[1]操作手順

1. データを収集する。
2. データの平均を計算し保存する
3. 画像の分類を行う。
4. 必要に応じて学習経過を見る。

教科書参照

2.6.2 Theano

本項では、Theanoのコードを理解する上でのポイントを概説する。 Theano上に構築されたパッケージとしてBlocks, deepy, Keras, Lasagne, Nolearn, Pylearn2などがある。

(発表者によるコメント 始まり)

> TheanoとはPythonにおける数値計算用のライブラリである. 行列演算などを行う関数を提供しており, numpy/scipyの代替パッケージと思えばいいだろう.
>
> 大きな特徴は

1. 実行時におけるC++コードの生成とコンパイル
2. GPUサポート
3. 解析的な微分のサポート (x^2の微分=2xという操作を自動でやってくれる)

> これによってMultilayer Perceptronを実行するベンチマークでは, Theanoが>numpyより1.8倍, Matlabより1.6倍速いという結果がこちらの資料で示されている. >(もちろん速度差はケースバイケースである) http://d.hatena.ne.jp/saket/20121207/1354867911より参照

読み方はテアノです。

(発表者によるコメント 終わり)

[1]Theano処理手順

1. Theanoの共有変数として結合係数行列を定義する。
2. 各層の変換関数を記述する。
3. 自動微分機能などで勾配を定義する。
4. thano.functionで各変数の依存関係を表現する。

(発表者によるコメント 始まり)

[2]テンソル変数

テンソル変数は階数（次元）と要素の型を併せて、変数の型を定義している。

型の詳細は以下参照

http://deeplearning.net/software/theano/library/tensor/basic.html#libdoc-basic-tensor

(発表者によるコメント　終わり)

テンソル変数の型数は4次まで定義されている。 画像処理を考えれば、1枚の画像は3次元配列（縦横のピクセル2次元画像と色チャネルで3次）となる。画像認識には複数枚の訓練画像が必要となる。この場合に用いられる画像数を1次元として考慮すれば、4次元配列が必要となる。深層学習における層間の処理とは4次元配列を各層で変換し、最終的な認識を得ることに相当する。したがって、深層学習における各層の流れはテンソル変数を角層ごとに処理することである、と極言することができる。このような理由から、グーグルは自らの機械学習パッケージをTensorFlowと名づけたのであろう。

[3]関数theano.function

theano.functionすなわちTheanoにおける関数とは、古典的言語における関数とは異なる。頻用される関数はTheanoにも用意されている。 これらの関数とtheano.functionは異なる。thano.functionとは、定義ずみの変数やグラフ連結関係から第1引数で指定される入力変数と、第2引数で指定される出力変数との関係をコンパイルする演算を指す。

theano.functionが評価されるとtheano.functionの実行にやや遅延が感じられる。これはtheano.functionの命令をCのコードに変換してコンパイルを行うからである。

[4]自動微分theano.grad

(発表者によるコメント 始まり)

Theanoの目玉機能の一つがこの微分機能です。自動微分と呼ばれる「式を分析して微分した式を求める」ことができます。 例えば、

```
x, y = T.dscalars("x", "y") # まとめて宣言する書き方
z = (x+2*y)**2
```

という式を x について微分したら、 dz/dx = 2(x+2*y) になりますが、それを

```
gx = T.grad(z, x)
```

で式変換できます。 yについての微分ならば、 dz/dy = 4(x+2*y) ですが、それを gy = T.grad(z, y) で式変換できます。 実際に値を求めるときは、やはり関数化が必要で、

```
fgy = theano.function([x,y], gy)
>>> fgy(1,2)
array(20.0)
```

などとします。 http://qiita.com/mokemokechicken/items/3fbf6af714c1f66f99e9より引用

(発表者によるコメント　終わり)

2.6.3 Chainer

Chainerはソースコードの可読性に優れ、製作者の力量が高い事がわかる。 またマニュアルとチュートリアルも簡潔で明瞭な記述がなされている。

(発表者によるコメント 始まり)

> なぜ今新しいフレームワーク？
>
> Deep Learning のフレームワークとしては Caffe, Theano/Pylearn2, Torch7 の 3 つが人気です。これらはフィードフォワードなネットワークを書くことが基本的な目標として開発されています。ですが、最近では Deep Learning の進展に伴い、より複雑なネットワークを柔軟に書けることの必要性が高まっています。そこで、この中でも特に自由度が高い Theano をベースに、新しいフレームワークがたくさん模索されています（例：Blocks, Keras, Lasagne, deepy など）。
>
> これらのフレームワークを含め、既存の実装のほとんどは、一度ニューラルネット全体の構造をメモリ上に展開して、その処理を順に見てその通りに順伝播・逆伝播を実行するというアプローチを取っています。これは、独自のミニ言語に対するインタープリタを実装しているようなものです。例えば Caffe ならば Protocol Buffer で定義されたスキーマがミニ言語に対応します。Torch7 の場合には、コンテナと呼ばれる特殊なモジュールが制御構造の役割を果たします。Theano はより柔軟な定義が可能ですが、ループを書くために scan と呼ばれる特殊な機能を使います。このアプローチにおいて、より複雑な計算フローをサポートしようと思ったら、基本的にはこのミニ言語を拡充していく必要があり、学習コストや記述コストは高くなっていきます。基本的には今後、ニューラルネットの構造はさらに複雑化していくことを考えると、この展開は好ましくありません。
>
> Chainer はこれとは異なるアプローチを取ります。Python をベースとしていますが、Theano は使いません。制御構造はすべて Python のものがそのままつかえます。Chainer は、実際に Python のコードを用いて入力配列に何の処理が適用されたかだけを記憶しておき、それを誤差逆伝播の実行に使います。このアプローチは、複雑化していく Deep Learning の研究・開発速度を保つために必要だと考えており、私たちが新しいフレームワークの開発に乗り出した理由です。
>
> https://research.preferred.jp/2015/06/deep-learning-chainer/より引用

(発表者によるコメント 終わり)

[1]Chainerの処理の概要

1. Linkを使ってChainを定義する。
2. OptimizerにChainを設定する
3. forward関数を定義する
4. データセットを読み込み、訓練用と評価用に分ける。

5. 訓練ループを回す。
6. 適度な頻度で評価ループを回す。

したがって、標準的なニューラルネットワークの学習を素直に実装していると考えられる。

Chainer専用の変数Chainer.Variableがある。 Theanoとの比較では

```
>>> import chainer
>>> import theano
>>> import numpy as np
>>> X = np.ndarray((3, 5))
>>> x_chainer = chainer.Variable(X)
>>> x_theano = theano.tensor.dmatrix()

>>> print(x_chainer.label)
(3, 5), float32
>>> print(x_theano.type)
TensorType(float3, matrix)
```

上の例は比較のためであり、ChainerとTheanoとを同時に使うことは現実的ではない。

[3]LinkとChain

(発表者によるコメント 始まり)

多くのニューラルネットの構造は複数のLinkを含みます。例えば、多層パーセプトロンは複数の線形層から構成されます。複数のLinkを組み合わせることで、この複雑な手順をパラメータによって記述できます。

```
l1 = L.Linear(4, 3)
l2 = L.Linear(3, 2)
def my_forward(x):
    h = l1(x)
    return l2(h)
```

ここで、Lはchainer.linksモジュールのことです。この方法で処理手順をパラメータによって定義すると再利用が難しいです。よりPythonらしい方法はLinkと手順をクラスにまとめることです：

```
class MyProc(object):
    def __init__(self):
        self.l1 = L.Linear(4, 3)
        self.l2 = L.Linear(3, 2)

    def forward(self, x):
        h = self.l1(x)
        return self.l2(h)
```

もっと再利用しやすいようにするには、パラメータ管理、CPU/GPUの移行サポート、頑強で柔軟性のある保存/読み込み、などの機能がほしいです。これらの特徴はすべて、ChainerのChainクラスがサポートします。Chainクラスの特徴を利用するためには、単純に上記のクラスをChainクラスのサブクラスとして定義するだけです：

```
class MyChain(Chain):
    def __init__(self):
        super(MyChain, self).__init__(
            l1=L.Linear(4, 3),
            l2=L.Linear(3, 2),
        )

    def __call__(self, x):
        h = self.l1(x)
        return self.l2(h)
```

より複雑なつながりがより単純なLinkによって構築されるかを示します。l1とl2のようなLinkはMyChainの子供Linkとして呼び出されます。Chainそれ自身はLinkを継承しています。それは、MyChainオブジェクトを子供Linkとして、より複雑なつながりを定義することができることを意味します。

よいパラメータの値を得るためにOptimizerクラスによって最適化がなされます。OptimizerはLinkに与えられた数値最適化アルゴリズムを走らせます。多くのアルゴリズムがoptimizersモジュールに実装されています。ここでは最も単純なもの、確率的勾配降下法をつかいましょう：

```
model = MyChain()
optimizer = optimizers.SGD()
optimizer.setup(model)
```

http://www.iandprogram.net/entry/chainer_japaneseより引用

(発表者によるコメント 終わり)

2.6.5 Tensorflow

ポイントはセッション管理（チェックポイントなど）とテンソルボード（tensorboard）である。

TensorFlowではニューラルネットワークが推定すべき記憶範囲を変数(tensorflow.Variable)、推定する必要がない記憶範囲をプレースホルダー(tensorflow.placeholder)と呼ぶ。したがって、入力画像はプレースホルダーである。一方、結合係数（行列）やバイアス（ベクトル）は変数となる。Theanoのマニュアルではプレースホルダと変数とを区別せずに用いられていた。Variableを定義して管理するのはTheanoやChainerと共通する。

2.6.7 scikit-learn

本書で取り上げた他のフレームワークとは異なるが、機械査定のPython実装としてscikit-learnがある。scikit-learnにも多層パーセプトロンが用意されているようだ。ただし、多層パーセプトロンの導入はscikit-learn 0.18である。2016年2月現在安定版である0.17には含まれていない。

(発表者によるコメント 始まり)

2016年9月に0.18.0はリリースされており、こちらに多層パーセプトロンが含まれている。

http://scikit-learn.org/stable/modules/neural*networks*supervised.html#multilayer-perceptron

(発表者によるコメント 終わり)

scikit-learnを用いれば、ニューラルネットワークにかぎらず機械学習の既存のアルゴリズムとの比較が容易になる。

(発表者によるコメント 始まり)

上記で言いたいことはscikit-learnでは下記コードのようにアルゴリズムに関わらず、インターフェイスが整えられている。これはscikit-learnが行ったことであり、このおかげで以下のような記載が容易に行え、アルゴリズムの比較が容易に行えるということである。

```
for name, estimator in ESTIMATORS.items():
    estimator.fit(X_train, y_train)
    y_test_predict[name] = estimator.predict(X_test)
```

http://scikit-learn.org/stable/auto*examples/plot*multioutput*face*completion.htmlより引用 (発表者によるコメント 終わり)

自作のアルゴリズムをscikit-learnから利用可能なアルゴリズムと比較するにはfit(X, y)とpredict(X)を自作する必要がある。