「いちばんやさしい ディープラーニング 入門教室」
正誤表

Windows環境でのパスの表記について

本書では、Windowsのパスを表記する場合、Windows環境内のパスについては「¥」で区切っています。これはバックスラッシュ（\）と同じ意味です。

一方、Pythonのソースコード内にWindowsのパスを記述する場合は、スラッシュ（/）で記述する必要がありますので、そのように表記しています。

例）P.155「CNN+Plots-onelayer.py」の16行目

```
'C:/Program Files (x86)/Graphviz2.38/bin'
```

※ソースコードでは、スラッシュで区切ります

例）P.156「POINT」下段の太字部分

```
'C:¥Program Files (x86)¥Graphviz2.38¥bin'
```

※Windows環境内のパスを表記する場合は、¥（\）で区切っています

以上をまとめると、
Windows環境内では¥またはバックスラッシュ（\）で表記していますが、ソースコード内ではスラッシュ（/）で記述する必要があることに、ご注意ください。

Windows環境の方は以下のコードを追記してください

上記と関連して、Windowsユーザーの方は、以下のコードを追記お願いいたします。

◎P.196「CNN+Train-dropout.py」20行目に
◎P.216「RNN+Plots-LSTM.py」13行目に
◎P.221「RNN+Plots-GRU」13行目に

↓以下の3行を追記してください

```
# Windows の場合は以下を追加
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
```

※以上、wivern.com様のご指摘により、訂正させていただきました（2018年10月19日）

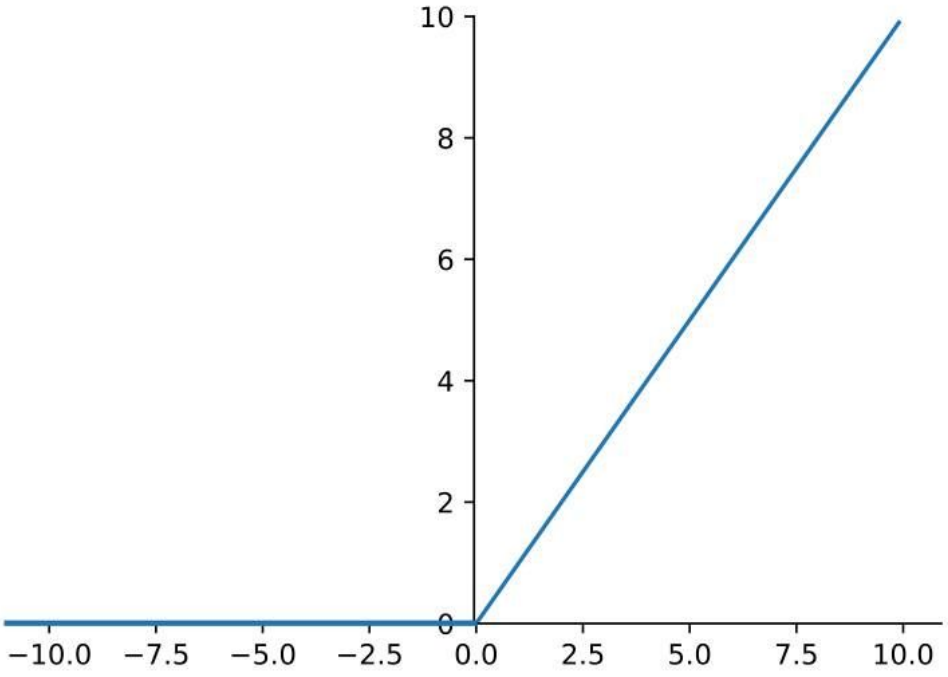
初版 第1刷　正誤表（2018年5月29日現在の情報です）

赤い文字＝間違った表記
青い文字＝正しい表記

◎P.26　表1-4-1
誤：Θ　θ　ツェータ
正：Θ　θ　シータ

◎カバー右袖、奥付の著者プロフィール（康先生分）
誤：2015年4より徳島大学…
正：2015年4月より徳島大学…

◎P.101　図4-2-3
ReLU関数のグラフの横軸、左側（負の領域）にx軸に沿って青線が必要。



※以上は、第2刷で訂正済みです

初版 第2刷　正誤表（2018年12月19日現在の情報です）

◎P.30　本文「微分と偏微分」見出し以下
※以下のように訂正します
==========
微分というのは、ある変数の小さな変化に対応する関数の変化の割合や変化量を意味しています。
微小な変化をδ、大きな差分をΔ、微小な差分（微分）をdとし、関数 *f*(x) を考えたときの微分を以下のように表します。
==========

◎P.63　コード9行目
誤：# リストの長さ、最大、最小を調べて表示
正：# タプルの長さ、最大、最小を調べて表示

◎P.63　コード11行目
誤：7 10 6
正：7 10 7

◎P.64　コード9～11行目が不要
誤

```
>>> def hello(who, aisatsu):      # 複数の引数付き関数を定義
...     print(aisatsu + ', ' + who + '!')
...
>>> hello('Python', 'Konichiwa')  # 関数helloを呼び出し
Konichiwa, Python!
>>> def keisan(x, y):
...     return x, y, x + y
...
>>> e, f, g = keisan(3, 6)
>>> def keisan(x, y):            # 足し算をする関数keisanを定義
...     return x, y, x + y       # 引数x, yとx + yの計算結果を返す
...
>>> e, f, g = keisan(3, 6)       # x, yとx + yの計算結果をe, f, gに入れる
>>> print(e, f, g)               # e, f, gの中身をprint文で表示
3 6 9
>>> _
```

正

```
>>> def hello(who, aisatsu):      # 複数の引数付き関数を定義
...     print(aisatsu + ', ' + who + '!')
...
>>> hello('Python', 'Konichiwa')  # 関数helloを呼び出し
Konichiwa, Python!
>>> def keisan(x, y):            # 足し算をする関数keisanを定義
```

```
...     return x, y, x + y        # 引数x, yとx + yの計算結果を返す
...
>>> e, f, g = keisan(3, 6)       # x, yとx + yの計算結果をe, f, gに入れる
>>> print(e, f, g)               # e, f, gの中身をprint文で表示
3 6 9
>>> _
```

※関数が重複しているため、1つを削除して、コメントを移動しています

◎P.49　Pythonのバージョンを調べるコマンド

誤：> python -version [Enter]

正：> python --version [Enter]

※半角ハイフンが2つ必要です

◎P.98　図4-1-4　入力x1の重み

誤：重みv11

正：重みw11

◎P.121　数式（3）

誤：$= -1.0 \log_e (0.67) - t_{0,1} \log_e (0.33)$

正：$= -1 \cdot \log_e (0.67) - 0 \cdot \log_e (0.33)$

◎P.121　数式（6）

誤：$-1.0 \log_e (0.12) - t_{0,1} \log_e (0.88)$

正：$0 \cdot \log_e (0.67) - 1 \cdot \log_e (0.33)$

◎P.97　数式4.1.1

誤

$$z = f(\mu) = f\left(\sum_{i,j}^{n} w_{ij} - h\right)$$

↓

正

$$z = f(\mu) = f\left(\sum_{i=1}^{n} w_i \, x_i - h\right)$$

◎P.128　数式4.7.2
誤

$$\text{更新式（傾き）} = \frac{\delta y}{\delta x} = \frac{\delta y}{\delta z}\frac{\delta z}{\delta x}$$

↓
正

$$\text{更新式（傾き）} = \frac{\delta y}{\delta x} = \frac{\delta y}{\delta z}\frac{\delta z}{\delta x} \approx \frac{\Delta y}{\Delta z}\frac{\Delta z}{\Delta x}$$

◎P.137　本文　下から二行目

誤：この計算式の結果、$\sigma(y)j$ で、全部足すと1（100%）になるような

正：この計算式の結果、$\sigma(y)_j$ で、全部足すと1（100%）になるような

※jは下付き文字になります

◎P.147　図5-1-2
誤

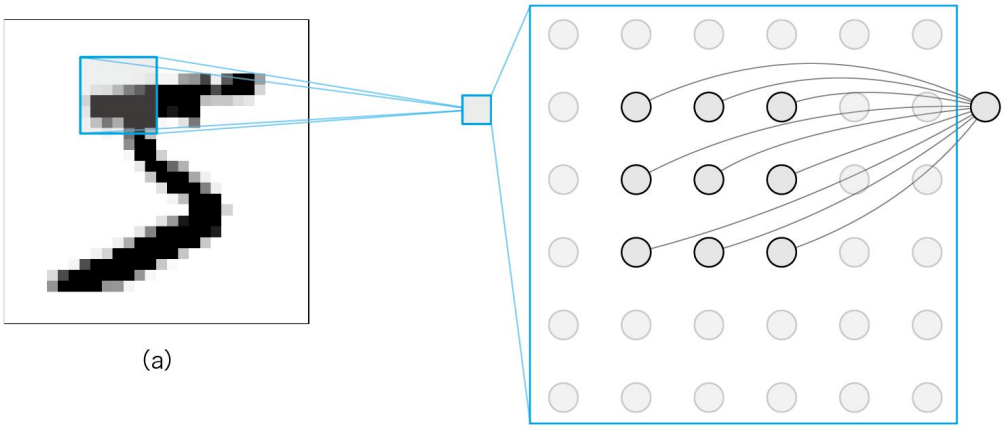


図5-1-2　CNNの局所的なコネクション

↓
正

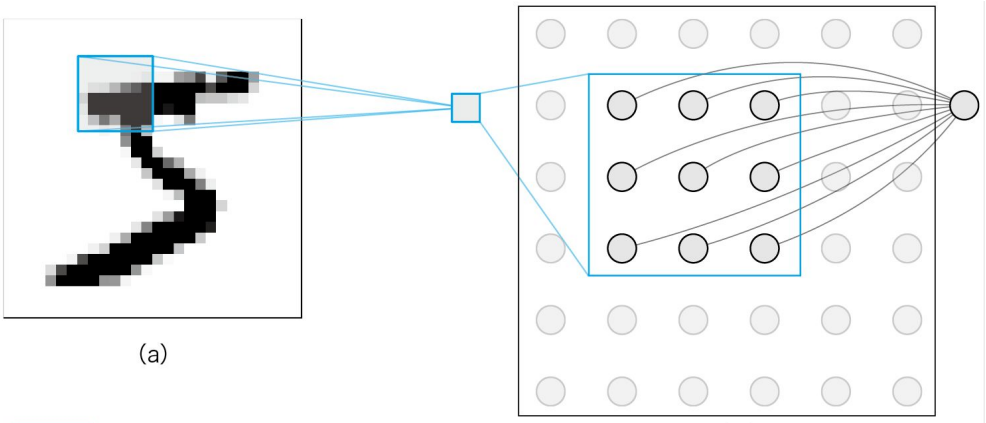


図5-1-2　CNNの局所的なコネクション

◎P.163　本文
※以下のように訂正します。
==========
図5-4-5 で、左上の 3 つの四角は 1層目の3個のニューロンにより学習され、2層目の1個目のニューロンへ入力される局所パターンです。
その 右側の四角は、このニューロンに手書き文字の「5」を入力した場合のニューロンからの出力イメ ージです。
1層目が3個のニューロンで構成され、受容野を用いて3枚の入力データから局所パターンを学習し、その3つの局所パターンを組み合わせていますが、1個のニューロンは、元画像の中の左向きの「くの字」に対して非常に敏感なニューロンを構成していることがわかります。

同様に、左下の3つの四角は、1層目の3個のニューロンにより学習され、2層目の2個目のニューロンへ入力される局所パターンです。
右下の 四角はその出力イメージです。この出力から、元画像の中の 135 度の角度の線分を表す画像の特徴(edge)に敏感であることがわかります。

2層目の畳み込み層では、それぞれのニューロンが縦横 3×3の形状で深さ2の受容野によって畳み込み演算を行います。
このとき入力データの 3×3の領域を対象とし、局所パターンによって要約しますが、これは図5-4-6で表すように、元画像の 5×5の領域を要約していること に相当します。
これは、1層目のニューロンが、入力画像から意味のある局所パターンを認識して要約することで、2層目のニューロンに向けて、より使いやすい情報の形で受け渡していることを意味します。これが、深層学習の重要な特徴というわけです。
==========

◎P.213　本文最終行
誤：Σ関数には、勾配消失問題と呼ばれる副作用が生じます。
正：σ関数には、勾配消失問題と呼ばれる副作用が生じます。

◎P.214　本文内数式（図6-2-1の下）
誤：LSTMニューラルネットワークでは、要素ごとの積 $t_t \times h_{t-1}$ が……
正：LSTMニューラルネットワークでは、要素ごとの積 $f_t \times h_{t-1}$ が……

◎P.220　リセットゲート本文4行目
※本文をより理解しやすいように訂正します
誤：y_t の計算には影響しません。このとき、残りの y_{t-1} の残りの要素についても触れることはありません。
正：y_{t-1} の残りの要素については触れないため、y_t の計算には影響しません。