『ゼロから作る Deep Learning
Python で学ぶディープラーニングの理論と実装』
読書ノート

斎藤康毅（さいとう こうき。1984-）による 2016 年 9 月 28 日の本である。

ソースコード: https://github.com/oreilly-japan/deep-learning-from-scratch

deep-learning-from-scratch/.gitignore に.DS_Store とある。これは何？　macOS のフォルダの状態データ。

Python の文法チェッカーとしては `flake8` コマンドがある。

p. 2

ディープラーニングのフレームワークの例: Caffe、TensorFlow、Chainer、Theano。

p. 20

『入門 Python 3』、『Python によるデータ分析入門』という本がある。『Scipy Lecture Notes』というウェブサイトがある。.pdf ファイルで全 690 ページ。

p. 21

Frank Rosenblatt (1928-1971)。論文『The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain』(1958) がある。全 23 ページ。

> Prefiguring Cyberculture: An Intellectual History (2004-09-17):
>
> Shortly afterwards (July 11, 1971) Frank Rosenblatt, its guiding inspiration, died in a boating accident that some say was a suicide.
>
> But within the AI community, early connectionism had already met its demise (see, for example, Dreyfus and Dreyfus 1988, 24).
>
> In the absense of funding, and of the kinds of concrete successes in embodying biological principles in non-biological systems that might have generated new alliances and new sources of funding, the union promised between organisms and computers failed to materialize.
>
> Neural net models, which had been the heart of Rosenblatt's effort, had to wait another 15 years to be revived; interactionist models of development have had to wait even longer.

Evidently, the soil of American science in the 1960s was not quite right for this largely European transplant to take root.

Even so, as Hubert and Stuart Dreyfus write (1988, 24), “blaming the rout of the connectionists on an antiholistic bias is too simple.”

In a rather expansive sweep, they fault the influence of an atomistic tradition of Western philosophy, from Plato to Kant; von Foerster (1995), by contrast, blames both the funding policies of American science (the short fuding cycle, the emphasis on targeted research) and the excessively narrow training of American students.

(Historians have yet to have their say on the matter.)

p. 22

$$y = \begin{cases} 0 & \text{if} \quad w_1 x_1 + w_2 x_2 \leq \theta \\ 1 & \text{if} \quad w_1 x_1 + w_2 x_2 > \theta \end{cases}$$

p. 23

AND ゲートが、例えば $(w_1, w_2, \theta) = (0.5, 0.5, 0.7)$ で実現できると記載がある。matplotlib でプロットしてみる。

```
import numpy as np
import matplotlib.cm
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d


def f(x1, x2):
    return 0.5*x1 + 0.5*x2


x1 = np.linspace(0, 1, 100)
x2 = np.linspace(0, 1, 100)
X1, X2 = np.meshgrid(x1, x2)
Y = f(X1, X2)

fig = plt.figure()
ax = plt.axes(projection=mpl_toolkits.mplot3d.Axes3D.name)
ax.set_title("y = 0.5*x1 + 0.5*x2")
ax.set_xlabel('x1')
```

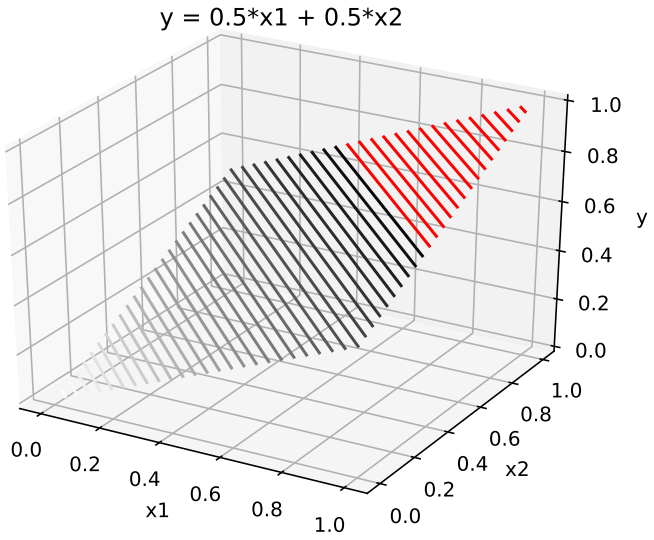

図 1　$(w_1, w_2, \theta) = (0.5, 0.5, 0.7)$ による AND ゲート

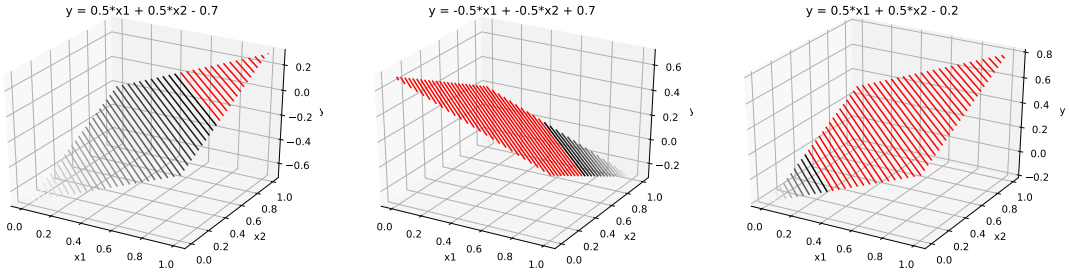

図 2　バイアスを用いた AND、NAND、OR ゲート

```
ax.set_ylabel('x2')
ax.set_zlabel('y')
cmap = matplotlib.cm.get_cmap('binary')
cmap.set_over('r')
ax.contour3D(X1, X2, Y, 50, cmap=cmap, vmax=0.7)
fig.savefig("output.pdf", bbox_inches='tight')
plt.show()
```

p. 26

$$y = \begin{cases} 0 & \text{if} \quad b + w_1 x_1 + w_2 x_2 \leq 0 \\ 1 & \text{if} \quad b + w_1 x_1 + w_2 x_2 > 0 \end{cases}$$

p. 42

$$y = h(b + w_1x_1 + w_2x_2)$$

$$h(x) = \begin{cases} 0 & \text{if} \quad x \leq 0 \\ 1 & \text{if} \quad x > 0 \end{cases}$$

これは binary step 関数とか、ヘヴィサイドの階段関数と呼ばれるようである。

p. 45

sigmoid function。シグモイドとは、ギリシャ語の文字シグマのようにうねってるということ。

$$h(x) = \frac{1}{1 + e^{-x}}$$

p. 67

softmax function.

$$y_k = \frac{e^{a_k}}{\sum_{i=1}^{n} e^{a_i}}$$

p. 73

MNIST Database のデータをダウンロードなどしてくれる mnist.py を本書が提供している。実行時の表示を以下にメモしておく。

```
$ python3 mnist.py
Downloading train-images-idx3-ubyte.gz ...
Done
Downloading train-labels-idx1-ubyte.gz ...
Done
Downloading t10k-images-idx3-ubyte.gz ...
Done
Downloading t10k-labels-idx1-ubyte.gz ...
Done
Converting train-images-idx3-ubyte.gz to NumPy Array ...
Done
Converting train-labels-idx1-ubyte.gz to NumPy Array ...
Done
```

```
Converting t10k-images-idx3-ubyte.gz to NumPy Array ...
Done
Converting t10k-labels-idx1-ubyte.gz to NumPy Array ...
Done
Creating pickle file ...
Done!
```

p. 75

学習済みパラメータを使って MNIST Database のテスト用データを判定させる。正答率 0.9352 が得られる。第 0 層から第 3 層までを用いる 3 層のニューラルネットワークを用いる。各層のニューロンの個数は、$28 \times 28 = 784$、50、100、$10 = |\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}|$ であり、合計で $784 + 50 + 100 + 10 = 944$ 個だ。ここで隠れ層のニューロンの個数 50 と 100 は自由らしいが、学習済みパラメータを使うので動かすわけにいかない。なお、学習済みデータはそれなりに大きい。どのくらいの大きさだろうか。

第 1 層の第 0 ニューロンを考えてみる。すると、バイアス 1 つと、784 個の入力が入ってくる。よって重みの個数は $784 + 1 = 785$ だろう。第 1 層の 50 個のニューロンについてこれは同様に言えるから、合計で、$50 \times 785 = 39250$ 個の重みがあると考えられる。次に第 2 層の第 0 ニューロンを考えてみる。すると入力の個数は 51 個だろう。第 2 層の 100 個のニューロンについてこれは同様に言えるから、合計で、$100 \times 51 = 5100$ 個の重みがあると考えられる。第 3 層の第 0 ニューロンについて考えてみる。すると入力の個数は 101 だろう。第 3 層の 10 個のニューロンについてこれは同様に言えるから、合計で、$10 \times 101 = 1010$ 個の重みがあると考えられる。こうして得られた値の総和を求めると、$39250 + 5100 + 1010 = 45360$ 個の重みがあるということになる。

p. 76 のコードに次の美しくないコードを挿入してこの個数 (45360) を確認した。`print(sum(map(len, [W1.flatten(), W2.flatten(), W3.flatten(), b1.flatten(), b2.flatten(), b3.flatten()])))`

また、同じコードに `f.write(str(network))` などと挿入して書き出したテキストファイルのサイズは 895,218 バイトであったが、これを重みの個数 45360 で割ると、19.73584656084656 が得られる。1 つの重み当たりテキストで 20 文字弱を占めているということになり、大きな矛盾はない。

ニューロンが 944 個でシナプスが 45360 個だと考えられるだろう。$\frac{944}{45360} = 0.020811...$ であり、$\frac{45360}{944} = 48.050847...$ である。つまり 1 個のニューロンにつき 50 個弱のシナプスが存在する。これはもちろん、ネットワークの構成によるのだろう。なおおそらく、MNIST は最小の課題の一つだろう。ニューロンとシナプスが約 1000 個と約 50000 個だと思ってしまおうか。

テスト用データの個数は 10000 個で、自分の環境でそれを処理するのは 2 秒ほどかかった。つまり、0.2 ms で 1 個処理でき、1 ms で 5 個処理できる。学習との計算量の比率はかなり異なると思っていいのだろう。多くの用途において、学習済みデータを用いた推論は、スマートフォンや組み込み機器のハードウェアで処理できるという印象は感じられた。

なお、本書で提供されている学習済みパラメータの.pkl ファイルである ch03/sample_weight.pkl は 181,853 バイトだ。素朴にテキスト化したところ上記の通り 895,218 バイトだったわけである。$\frac{895,218}{181,853} = 4.922756...$ で 5 倍弱になっている。しかも精度が落ちているかもしれない。深層学習などを扱う上では、このようなデータを格納しておく必要性はしばしば生じそうだ。.pkl は実用的にもよい選択肢なのだろうか。.zip にしてみる

とそれぞれ、169,644 バイトと 297,814 バイトであった。もし.pkl 表現に後方互換性がなかったら、テキストファイルを.zip にすることは大きな損ではないかもしれない。.json 形式が互換性が高いだろう。.csv 形式が人気があるようだ。思うに、.csv 形式は行で貪欲に完結しているのは長所だろう。

p. 88

人気のある損失関数である 2 乗和誤差 (mean squared error)。2 で割るのは、微分したときに係数が消える利点のため。

$$E = \frac{1}{2}\sum_k (y_k - t_k)^2$$

また別の人気のある損失関数である交差エントロピー誤差 (cross entropy error)。

$$E = -\sum_k t_k \log y_k$$

N 個の訓練データの損失関数の和。

$$\begin{aligned} E &= \frac{1}{N}\sum_n \left(-\sum_k t_{nk} \log y_{nk} \right) \\ &= -\frac{1}{N}\sum_n \sum_k t_{nk} \log y_{nk} \end{aligned}$$

p. 97

微分。前方差分による。

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

p. 101

サンプルプログラム ch04/gradient_1d.py のコードスニペットについて考える。

```
def tangent_line(f, x):
    """関数 f の入力 x における接線を返す。"""
    d = numerical_diff(f, x)  # x における微分係数を得る
    print(d)                  # 0.1999999999990898
    y = f(x) - d*x            # 高さは x のとき f(x) だが 0 のときは dx 減る
    return lambda t: d*t + y  # y = ax + b ならぬ dt + y を返す
```

p. 107

勾配法。

$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

p. 108

学習率が大きいからといって発散するものだろうか。またそもそも、勾配は方向を知るのにのみ使って、その大きさは用いなくてもよいのではないか。むしろ平坦な場所ほど大きく移動したい気もする。

p. 118

のコードで学習が実行される。ここまでの本文では誤差逆伝搬法の内容の説明はないが、効果は同じということで、コードでは誤差逆伝搬法による方法がデフォルトでオンになっている。それでも少し時間がかかる。誤差逆伝搬法を使わない場合には 100 倍くらいかかるだろうか。

どこでどの程度の時間がかかっているのだろう。巨大な行列の乗算が繰り返されるが、そもそも、行列の積を素朴に求めるときの計算量は何だろうか。$L \times M$ 行列 A と $M \times N$ 行列 B の積 C を考えてみようか。C に存在する LN 個の要素それぞれは、A と B のそれぞれ M 個の要素について、各々を掛け、足し合わせたものである。よって、$LN(M + (M - 1))$ だと考えられる。大雑把に見れば、LMN だ。つまり、行列 A、B の辺の長さがどれも計算量に同様に作用している。

第 0 層入力層に $28 \times 28 = 784$ 個のニューロンがあり、第 1 層隠れ層に 50 個のニューロンがあるとする。すると第 0 層と第 1 層を結ぶ辺は第 1 層のニューロン 1 個につき 784 個あるから合計で $50 \times 784 = 39,200$ 個だろう。行列の積として考えると、1×784 行列 A と 784×50 行列 B の積 C が 1×50 行列なのだと言える。C の各要素は 784 要素の積和なので全体の計算量が $1 \times 784 \times 50$ という感じだろう。そして、実際に入力されるのはミニバッチである。

複数の行列の積 $ABCDEFG$ のようなものを考えてみる。結合法則により乗算の順序は任意である。小さいものからやったほうが計算量が小さい気がする。しかし、ニューラルネットワークは sigmoid 関数などの活性化関数で分断されているので、関係はないと思われる。

活性化関数では累乗の計算がある。累乗の計算量は $O(\log n)$ だったろう。とりあえず考えずにおこう。ミニバッチの大きさを 100 と考える。行列として書くと、100×78 どこでどの程度の時間がかかっているのだろう。巨大な行列の乗算 4 行列 A と 784×50 行列 B と 50×10 行列 C から 100×10 行列 D を得ると考えられる。まず第 1 層の計算のために、$100 \times 784 \times 50 = 3,920,000$ 回四則演算して 100×50 行列を得る。次に第 2 層を算出するために、$100 \times 50 \times 10 = 50,000$ 回四則演算して 100×10 行列を得る。よって 1 個のミニバッチを predict() するのに $3,920,000 + 50,000 = 3,970,000$ 回四則演算すると思われる。

MNIST の訓練データは 60,000 個あるから、ミニバッチの大きさを 100 としたときには、$\frac{60,000}{100} = 600$ 回が 1 epoch をなすのだろう。1 epoch あたりの四則演算の回数は、$3,970,000 \times 600 = 2,382,000,000$ だと概算される。勾配などの計算を考えずにである。すでに 10^9 のオーダーであるから、1 分間かかるくらいの感じだろうか。本書のサンプル train_neuralnet.py においては、epoch=600 で大きさ 100 のミニバッチを 10,000 回実行しているから、epoch=$\frac{10,000}{600} = 16.\bar{6}$ である。さほど時間はかからず手元の環境で 45 秒間ほどである。なお、numpy モジュールは複数の CPU コアを自動的に活用しているようである。

数値差分により勾配を得る方法、それは重みつまり辺のすべてについて上下に動かし、そのたびに predict() するのだから、大きな計算量になるとは想像できる。

例えば本書サンプル train_neuralnet.py で言えば、iters_num=10,000 回ミニバッチを処理するたびに TwoLayerNet#numerical_gradient() しているが、TwoLayerNet#numerical_gradient() はそれぞれの重み

のセットについて numerical_gradient() を呼び、numerical_gradient() は $2 \times 3 = 6$ 回 TwoLayerNet#loss() を呼び、loss() は predict() を呼んでいる。

iter_num=1 として predict() が実行された回数を数えてみると 79,523 である。辺の数が $39,200 + 500 = 39,700$ であったから、$2 \times 39,700 = 79,400$ がほぼ近い。バイアスも数え入れて数え直そうか。$50 \times 785 = 39,250$、$10 \times 51 = 510$、$39,250 + 510 = 39,760$、$2 \times 39,760 = 79,520$ となる。残りの 3 つについては、TwoLayerNet の外から loss() が 1 回、accuracy() が 2 回呼ばれているので、数が合うと考えられる。

p. 141

ReLU (rectified linear unit):

$$y = \begin{cases} x & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x \leq 0 \end{cases}$$

その微分：

$$\frac{\partial y}{\partial x} = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x \leq 0 \end{cases}$$

p. 144

シグモイド関数の微分。

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

これを、xy と e^x と $x + y$ と $\frac{x}{y}$ に分けて考える。

$$\frac{\partial L}{\partial y}(-y^2) \times 1 \times e^{-x} \times (-1) = \frac{\partial L}{\partial y} y^2 e^{-x}$$

シグモイド関数の微分が得られた。これをさらに整理する：

$$\begin{aligned}
\frac{\partial L}{\partial y} y^2 e^{-x} &= \frac{\partial L}{\partial y} \left(\frac{1}{1 + e^{-x}} \right)^2 e^{-x} \\
&= \frac{\partial L}{\partial y} \frac{1}{1 + e^{-x}} \frac{e^{-x}}{1 + e^{-x}} \\
&= \frac{\partial L}{\partial y} \frac{e^{-x}}{(1 + e^{-x})^2} \\
&= \frac{\partial L}{\partial y} \left(\frac{1 + e^{-x}}{(1 + e^{-x})^2} - \frac{1}{(1 + e^{-x})^2} \right) \\
&= \frac{\partial L}{\partial y} \left(\frac{1}{1 + e^{-x}} - \frac{1}{(1 + e^{-x})^2} \right) \\
&= \frac{\partial L}{\partial y} \frac{1}{1 + e^{-x}} \left(1 - \frac{1}{1 + e^{-x}} \right) \\
&= \frac{\partial L}{\partial y} y(1 - y)
\end{aligned}$$