作って学ぶ
ニューラルネットワーク

— 機械学習の基礎から追加学習まで —

博士（工学） 山内　康一郎 【著】

コ ロ ナ 社

はじめに

最近の人工知能技術の基本はニューラルネットワークである。ニューラルネットワークはもとは神経細胞（ニューロン）のモデルであり，古くから行われてきた神経生理学的研究と計算論的な神経細胞とそのネットワークのモデル化の研究の賜物である。

現在の人工知能技術は，人に近づき，一部では人を超えて人が太刀打ちできないレベルにある。その学習方式には大きく分けてオフライン学習法とオンライン学習法とが存在する。オフライン学習は「十分に学んだ後で仕事をする」方式である。一方，オンライン学習は「学びながら仕事をする」方式である。実際にはオンライン学習にもさまざまな方式があるが，その多くは「学びながら仕事をする」というよりはオフライン学習を簡潔な計算で近似していると表現したほうが適切かもしれない。

いずれにしても，あらかじめ学習用のデータが十分に存在するならばオフライン学習を使うことができて，確実な学習をさせることができる。しかし，現実には学習データは五月雨式に降ってくるほうが多いのではないだろうか。そのような場合に人工知能に学習させるには，学習済みの知識を保持しながら新しい知識を学習させる必要があり，そうしたときに後者のオンライン学習法を使うことになる。だが，オンライン学習法は基本的には過去の学習データを勘案せず，今与えられているデータだけを使って学習する方式のため，過去の記憶をすっかり忘れてしまう「破滅的忘却」問題が生じてしまうのである。

現在の機械学習（人工知能が学習する技術）の最前線では，これを克服するために多くの研究がなされている。それらの研究が今後の機械学習システムの基幹技術となっていくであろうことは想像に難くない。本書ではこの技術の概要を初心者であっても理解できるように構成した。

最近の人工知能は大規模なニューラルネットワークである。この巨大なネットワークを効率よく構築し，さらに効率よく学習させるために Python を基本とするライブラリ群が多用されている。このライブラリは利便性を高める一方，その内部構造は難解であり，その基本的な仕組みを理解する者は少数であるといわざるを得ない。

質のよい人工知能を構築するには，その構造・メカニズムを深く理解することが重要である。そのために必要なのは，その数理的な知識のみならず，実際にプログラミングをしてニューラルネットワークを構築する経験であろう。これらがあって初めて，既存のライブラリをスムーズに活用できると確信する。

このような観点から，本書ではつぎのような工夫をした。

- ニューラルネットワークの理論を理解することを読者の目的として設定した。
- ニューラルネットワークのメカニズムを説明する過程で，それを理解するのに必要な数学的知識をその場で学べるようにした。
- Python のプログラムをニューラルネットワーク専用のライブラリを使用せずに構築することで，理論と実際の動作とを結びつけられるようにした。
- 以上を踏まえたうえで，4 章で追加学習に関する理論を解説するとともに，これを PyTorch を使って記述し，これまで学んできた内容をどのようにライブラリで記述できるのかを学べるようにした。

プログラミングを通して読者の疑問が明らかとなり，ニューラルネットワークについてさらに理解を深めるきっかけになれば幸いである。

2020 年 8 月

山内 康一郎

目　　次

1. 人工知能とは

2. 機械学習の基礎

3. ニューラルネットワーク

4. 追加学習

```
16                  self.datasetSize += 1 #ラベルの一致するデータの個数を数える
17          self.mydata = torch.tensor([self.datasetSize,28,28]) #テンソル型の入力データ配列を用意する
18          self.mydata = torch.zeros(self.datasetSize, 28, 28) #上で用意したテンソル型データを初期化
19          for p in range(self.datasetSize): #ラベルの一致する入力データセットを作成する
20              each_input, each_label = self.dataset.__getitem__(self.indicies[p])
21              self.mydata[p] = each_input
22          self.data = self.mydata.numpy() #入力データは Tensor から numpy array に変換する必要がある
23
24      def __len__(self): #データセットのサイズを返すメソッド
25          return self.datasetSize
26
27      def __getitem__(self, idx): #おのおののデータを返すメソッド
28          out_data = self.data[idx]
29          out_label = self.label[idx]
30          if self.transform:
31              out_data = self.transform(out_data) #ラベルデータの変換
32          return out_data, out_label
```

1 行目では ( ) の中に torch.utils.data.Dataset が記述されており，このクラスが torch.utils.data.Dataset を継承することを意味している。このクラスはオリジナルの MNIST データセットを呼び出しており，コンストラクタで与えられた文字のリスト（TargetLabels）に従ってデータを収集する。例えば，TargetLabels=[0, 1] ならば，このクラスは数字の'0' と'1' の学習サンプルのみ読み込むことになる。**ソースコード 4.2** にその実行部分を示す。

ソースコード **4.2**　追加学習の実行

```
1           Acc = []
2           for n in range(0, self.num_division):
3               target_labels = []
4               for l in range(n*self.label_step, (n+1)*self.label_step):
5                   target_labels.append(l) #読み込む文字を target_labels に追加
6
7               ##学習用・テスト用データをロードする##
8               self.train_dataset = MyMNIST(target_labels, './data', transforms.
                  ToTensor())
9               for i in range(100): #100 回繰り返す
10                  train_loader = torch.utils.data.DataLoader(dataset=self.
                      train_dataset, batch_size=100, shuffle=True)
11                  err, acc = self.model.learning(train_loader) #ミニバッチ学習
12                  print("%s : err=%s acc=%s" %(i, err, acc))
```

```
13            testerr, AccMatrix, class_size = self.model.evaluation(test_loader) #
                  テスト用サンプルでの誤差計測
14            print("Accuracy matrix = %s" %(AccMatrix))
15            for c in range(10):
16                print("%s : %s " %(c, (float)(AccMatrix[c][c].item()/class_size[c
                  ].item())))
17            Acc.append(testerr)
18        ##誤差の一覧表示##
19        for n in range(0, len(Acc)):
20            print("Acc[%s]=%s" %(n, Acc[n]))
```

4, 5行目でリスト target_labels にこのラウンドで追加学習させる文字をセットし，8行目でこのリストを MyMNIST() に渡すことで，この文字を読み込んでいる。このようにして 100 エポックの学習を終了させ，改めて'0'～'9' のテストサンプルで認識率をテストする。**表 4.1** はその結果である。

表 4.1 オリジナルのパーセプトロンに新しい文字を追加学習させた結果

追加学習の回数 ＼ テストサンプル	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
0	0.998	0.998	0.021	0.225	0.020	0.016	0.00	0.007	0.07	0.09
1	0.433	0.040	0.953	0.969	0.003	0.006	0.00	0.002	0.010	0.003
2	0.067	0.013	0.162	0.015	0.997	0.991	0.00	0.054	0.054	0.009
3	0.087	0.001	0.039	0.294	0.563	0.087	0.993	0.972	0.049	0.0
4	0.018	0.040	0.087	0.054	0.015	0.017	0.267	0.058	0.902	0.930

このように誤った箇所を重点的に学習させれば，すべてのサンプルに対して正解できる可能性が高まるであろうと期待される。ところがこの表からわかるように，予想に反して追加的に学習させた後の認識率は極めて悪く，事前学習したクラスのデータをほぼすべて間違えていることがわかる。これを一般に**破滅的忘却**（catastrophic forgetting）と呼び，ニューラルネットワークの大きな問題点として有名である。この問題を解決するために，これまでにも多くの研究者がさまざまな追加学習理論を提案してきた[19]。

これらについて解説する前に，このような問題が発生する理由について**図 4.1** を用いて解説しておこう。ニューラルネットワークの重みベクトルは，識別境界面を形成する。一つの識別境界面は入力空間を二分するが，それぞれのニュー

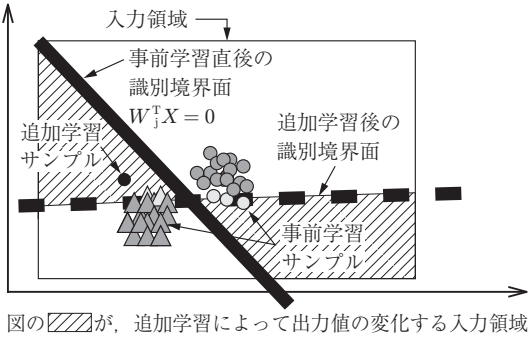

図 4.1　破滅的忘却が発生する理由

ロンモデルによって重みベクトルが異なるため，その分割位置が異なる。仮に一つの重みパラメータを更新したとすると，識別境界面が変化するため，入力領域の多くの部分に対してそのニューロンの出力値が変化する。このニューロンの出力がつぎの層に影響を与えることを考慮すると，最終層の出力値は入力領域の大部分の入力ベクトルに対して出力値が変化してしまう。これが破滅的忘却の原因となる。

このような問題を解決する手段としてかつてよく採られた手法は，一つのニューロンが出力を出す領域を絞り込む戦略である。このために radial basis function と呼ばれるニューラルネットワークを使用した研究が多く行われていた。このネットワーク（の多く）はガウスカーネル関数を基底関数としてそれらの重みつきの和によって任意の関数近似を行う学習機械である。一つのニューロンが出力を出す領域はその基底関数のセントロイド（基底関数が最も大きな出力を出す入力ベクトル）を中心とする限られた領域になるため，その基底関数に関係するパラメータを変更することによるニューラルネットワークの出力値の変化領域も，そのカーネル関数の出力を出す入力領域に限られることになる。したがってこの手法は通常のニューラルネットワークに比べると，追加的な学習に伴う忘却をかなり防ぐことができる。現在においてもカーネル法を使った学習理論がこの系統を受け継いでおり，オンラインで学習する場合によく使われている[20)~24)]。だがこのような動径基底関数を使う場合の欠点の一つは変数

選択が難しいという点である†。

4.2 再学習を行わせる手法

4.2.1 手法および実装例

破滅的忘却を防ぐ方法の中で最も確実な手法は，過去のサンプルのすべてを用意して新しいサンプルと織り交ぜて再度学習させることである。この場合ニューラルネットワークの横にバッファを用意する必要がある（**図 4.2**）。新規サンプルの提示方法は 1 個ずつ与える場合と，複数個ずつに与える場合がある。ここでは一般的によく行われる方式として，複数個ずつ新規サンプルを与える方式を考えよう。T 回目の追加学習時に与える新規サンプル集合を $\chi_{\text{new}}^{(T)} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{m}$ とすると，T 回目の追加学習時のバッファ内部のサンプル集合 $\chi_{\text{buffer}}^{(T)}$ は $\chi_{\text{buffer}}^{(T)} = \chi_{\text{buffer}}^{(T-1)} \cup \chi_{\text{new}}^{(T)}$ で表される。ニューラルネットワークは $(\boldsymbol{x}_p, \boldsymbol{y}_p) \in \chi_{\text{buffer}}^{(T)}$ を学習する。すなわち，ニューラルネットワークのパラメータベクトル $\boldsymbol{\theta}$ は，損失関数が $L(\boldsymbol{\theta}, \chi_{\text{buffer}}^{(T)})$ を最小化するように更新される。しかしながらこの手法では T が大きくなるにつれて $|\chi_{\text{buffer}}^{(T)}|$ も大きくなり，学習に要する時間および計算量も増えていくため，現実的な手法とはいい難い。

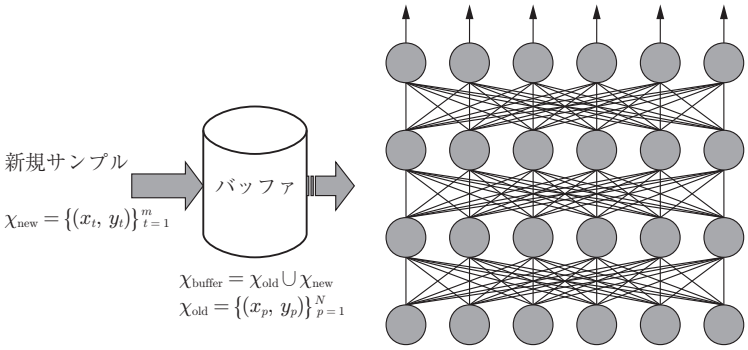

図 4.2 過去のサンプルすべての再学習を行う追加学習法

† ガウスカーネル関数の分散共分散行列も学習時に更新すればこの問題はある程度解決できる。だがこれを導入すると一つのニューロンの出力を出す入力領域が大きく変化し，大きな忘却を起こすケースもある。

お わ り に

本書はニューラルネットワークの仕組みを解説するとともに，Python を使ったプログラミング手法を解説した。さらに，Python 上で構築されているライブラリ：PyTorch を使って，各種追加学習手法について代表的な三つの手法の構築を行った。なお，本書で紹介したソースコードは，筆者が構築したものであり，ほんの一例に過ぎない。さらに実行効率のよいコードも十分にありえるので，後は読者自ら工夫し改良して頂ければと思う。

これらの手法はいまだ発展途上にあり，今後さらなる優秀な手法が提案されていくことであろう。読者の中から新たな手法を提案する研究者が現れたならこのうえない喜びである。

また，本書で使用したライブラリ PyTorch は日々更新されていくと考えられる。読者の皆さんにはその最新事情をつねに把握し，自らそれに追従していく必要があると考える。

本書で作成したプログラムソースについては，コロナ社の本書の書籍詳細ページ（https://www.coronasha.co.jp/np/isbn/9784339029116/，本書カバーそでに QR コードあり）にて公開されているので参考にされたい。

謝辞　本書の執筆にあたって多くの助言をいただいたコロナ社に感謝の意を表明します。また，私がプログラミングした Python コードの重大なバグを発見してくれた中部大学工学部の大学院生若原 涼君，および Indian Institute of Technology Guwahati からのインターンシップ生 Abhilash Reddy 君に感謝します。

索　　引

—— 著 者 略 歴 ——

1989年　名古屋工業大学工学部電気情報学科卒業
1991年　名古屋工業大学大学院工学研究科博士前期課程修了（電気情報工学専攻）
1994年　大阪大学大学院基礎工学研究科博士後期課程修了（生物工学専攻）
　　博士（工学）
1994年　名古屋工業大学助手
2000年　北海道大学大学院助教授
2006年　北海道大学大学院准教授
2009年　中部大学教授
　　現在に至る

作って学ぶニューラルネットワーク

―機械学習の基礎から追加学習まで―

Neural Network to Learn through Programming

2020年10月13日　初版第1刷発行　★

検印省略

著　者　山内　康一郎
発行者　株式会社　コロナ社
　　代表者　牛来真也
印刷所　三美印刷株式会社
製本所　有限会社　愛千製本所

112–0011　東京都文京区千石 4–46–10
発行所　株式会社　コロナ社
CORONA PUBLISHING CO., LTD.
Tokyo Japan
振替 00140–8–14844・電話(03)3941–3131(代)
ホームページ　https://www.coronasha.co.jp

ISBN 978–4–339–02911–6　C3055　Printed in Japan　（柏原）