第 1 章

実装に用いたソースコード

1.1　動作環境

実装は以下の環境のもとで行った。なお Keras は TensorFlow に付属のものを用いた。

- CPU : 2.4 GHz クアッドコア Intel Core i5
- メモリ : 16 GB 2133 MHz LPDDR3
- GPU : Intel Iris Plus Graphics 655 1536 MB
- OS : macOS Catalina 10.15.2
- Python : 3.6.6
- TensorFlow : 1.11.0
- Keras on TensorFlow : 2.1.6

1.2　1 隠れ層のフィードフォーワード・ニューラルネットワークのソースコード

ソースコード 1.1　1 隠れ層のニューラルネットワーク

```
1  import tensorflow as tf
2
3  # (1) データのインポート
4  from tensorflow.keras.datasets import mnist
5  (x_train, y_train), (x_test, y_test) = mnist.load_data()
6
7  # (2)データの変形
8  import numpy as np
9  from tensorflow.python.keras.utils import np_utils
10 # (2-1)
11 x_train = x_train.reshape(60000, 784) # (2-1-1)
12 x_train = x_train.astype('float32') # (2-1-2)
13 x_train = x_train / 255 # (2-1-3)
14 num_classes = 10 # (2-1-4)
15 y_train = np_utils.to_categorical(y_train, num_classes) # (2-1-5)
16 # (2-2)
```

```
x_test = x_test.reshape(10000, 784)
x_test = x_test.astype('float32')
x_test = x_test / 255
y_test = np_utils.to_categorical(y_test, num_classes)

# (3) ネットワークの定義
np.random.seed(1) # (3-1)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
model = Sequential() # (3-2)
model.add(Dense(100, input_dim=784, activation='relu')) # (3-3)
model.add(Dense(10, activation='softmax')) # (3-4)
model.compile(loss='categorical_crossentropy',
              optimizer='sgd', metrics=['accuracy']) # (3-5)

# (4) 学習
num_epochs = 100
batchsize = 1000 # (4-1)

import time # (4-2)
startTime = time.time() # (4-3)
history = model.fit(x_train, y_train, epochs=num_epochs, batch_size=batchsize,
                    verbose=1, validation_data=(x_test, y_test)) # (4-4)
score = model.evaluate(x_test, y_test, verbose=0, batch_size=batchsize) # (4-5)

# (5) 計算結果の表示
print('Test␣loss:', score[0])
print('Test␣accuracy:', score[1])
print("Computation␣time:{0:.3f}␣sec".format(time.time() - startTime))
print(model.summary())
```

まず (1) で MNIST データセットのデータをインポートしている。x_train は 60000 × 28 × 28 の配列変数で、それぞれの要素には画像のグレースケールを表す 0 から 255 までの整数値で格納されている。y_train はサイズが 60000 の 1 次元配列で、それぞれの要素には 0 から 9 まで正解のラベルが格納される。同様に 10000 個のテストデータが (x_test,y_test) に格納される。

(2) でデータの配列を変形した。まず Python の拡張モジュール NumPy と np_utils をインポートしておいた。(2-1-1) で 60000 × 28 × 28 の配列を 60000 × 784 の配列に変換した。x_train を int 型（整数）から float 型（32 ビットの不動少数点数）に変換し (2-1-2)、0 から 1 までの実数に変換した (2-1-3)。分類するクラスの数を num_classes=10 として (2-1-4)、y_train をワンホットベクトルに変換した (2-1-5)。同じ処理をテストデータについても行った (2-2)。

(3) でニューラルネットワークの定義をした。まず NumPy による乱数を固定するために、seed 値を固定した (3-1)。これをすることで再現性のある分析や処理を行うことができる。model を Sequential() で定義した (3-2)。パーセプトロンが信号を次のパーセプトロンにつないでゆき、これをつなげていくというモデルが Sequential である。隠れ層のユニット数を 100, 活性化関数には ReLU を使うことをで指定した (3-3)。出力層のユニット数が 10 で、活性化関数にソフトマックス関数を使うことを (3-4) で指

定した。最後にコスト関数には交差エントロピーを、最適化アルゴリズムには確率的勾配法（SGD）を採用した (3-5)。また metrics=['accuracy'] とすることで、テストデータでネットワークの性能を測る際の汎化性能を、テストデータにおける正解率（accuracy）でみた。

(4) でニューラルネットワークに学習をさせ、その結果を計算させた。エポック数を 100, ミニバッチに用いる訓練データの数を 1000 にし、あとで調節できるようにそれぞれ num_epochs と batchsize という変数に格納しておいた (4-1)。プログラムの実行時間を計測するために、time モジュールをインポートしておき (4-2)、実行開始時刻を startTime とした (4-3)。エポックごとの学習の評価値を表示させるために verbose=1 とし、学習を行った (4-4)。最後にテストデータでのコスト関数と汎化誤差（ここでは正解率）を計算させた (4-5)。score は 2 成分からなる配列で、第 0 成分にテストデータでのコスト関数、第 1 成分に正解率が格納される。

(5) で学習結果を表示させた。テストデータでのコスト関数を Testloss、正解率を Testaccuracy、実行時間を Computationtime で表示させた。print(mode.summary()) でモデルの層に対するパラメータの数を表示させた。

また節??では、Python の for 文を用いソースコード 1.1 の (3-3) から (3-5) の部分をソースコード 1.2 のように変更した。

ソースコード 1.2　隠れ層を増やすために加えた変更

```
num_hidden_layers = 2
a = 8
num_first_hidden_units = num_classes * a ** num_hidden_layers
model.add(Dense(num_first_hidden_units, input_dim=784, activation='relu'))
for k in range(2, num_hidden_layers + 1):
    num_hidden_units = num_classes * a ** (num_hidden_layers + 1 - k)
    model.add(Dense(num_hidden_units, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='sgd', metrics=['accuracy'])
```

1.3　1 隠れ層の畳み込みニューラルネットワークのソースコード

ソースコード 1.3　1 隠れ層の畳み込みニューラルネットワーク

```
import tensorflow as tf

# (1) データのインポート
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# (2) データの変形
import numpy as np
from tensorflow.python.keras.utils import np_utils
# (2-1)
x_train = x_train.reshape(60000, 28, 28, 1) # (2-1-1)
x_train = x_train.astype('float32')
```

```
13 x_train /= 255
14 num_classes = 10
15 y_train = np_utils.to_categorical(y_train, num_classes)
16 # (2-2)
17 x_test = x_test.reshape(10000, 28, 28, 1)
18 x_test = x_test.astype('float32')
19 x_test /= 255
20 y_test = np_utils.to_categorical(y_test, num_classes)
21
22 # (3) ネットワークの定義
23 np.random.seed(1)
24 from tensorflow.keras.models import Sequential
25 from tensorflow.keras.layers import Conv2D, MaxPooling2D
26 from tensorflow.keras.layers import Activation, Flatten, Dense, Dropout
27 import time
28
29 model = Sequential()
30 model.add(Conv2D(1, (3, 3), # (3-1)
31                  padding='same', # (3-2)
32                  input_shape=(28, 28, 1), activation='relu'))
33 model.add(Flatten()) # (3-3)
34 model.add(Dense(10, activation='softmax'))
35 model.compile(loss='categorical_crossentropy',
36               optimizer='sgd', metrics=['accuracy'])
37
38 # (4) 学習
39 startTime = time.time()
40
41 num_epochs = 20
42 batchsize = 1000
43 history = model.fit(x_train, y_train, batch_size=batchsize, epochs=num_epochs,
44                     verbose=1, validation_data=(x_test, y_test))
45 score = model.evaluate(x_test, y_test, verbose=0)
46
47 # (5) 計算結果の表示
48 print('Test␣loss:', score[0])
49 print('Test␣accuracy:', score[1])
50 print("Computation␣time:{0:.3f}␣sec".format(time.time() - startTime))
51 print(model.summary())
```

フィードフォーワード・ニューラルネットワークの学習のときと同様、(1) で MNIST データセットのデータをインポートし、(2) でデータの配列を変形した。ただし画像のデータをベクトルに展開せず、(2-1-1) で $60000 \times 28 \times 28 \times 1$ のまま使用した点が異なる。テストデータも同様に処理した (2-2)。

(3) でニューラルネットワークの定義をした。model は Sequential() とした。1 枚の 3×3 のフィルターを学習するパラメータとし (3-1)、出力のサイズが変わらないようにパディングを追加した (3-2)。活性化関数は ReLU とした。この層の出力サイズは $28 \times 28 \times 1$ だが、出力層に入力するのにこれを 784

のベクトルに変形した (3-3)。出力層の活性化関数はソフトマックス関数、コスト関数は交差エントロピー、最適化アルゴリズムは確率的勾配降下法（SGD)、汎化性能を正解率とした。

あとはフィードフォーワード・ネットワークの実装と同様、(4) で学習、(5) で学習の結果を表示させた。