情報工学実験 II レポート

Pytorch を用いたディープラーニング入門

人間情報システム工学科 4 年　45 番

山口惺司

実験日：	2024 年 12 月 18 日 2024 年 1 月 8 日
締切：	2025 年 1 月 15 日
提出日：	2025 年 1 月 15 日

評価項目	やった/一部やった/やっていない/何をやったかの概要
基本的プログラムを正常に動作させた	やった
選択した課題を１つ行った 行った内容：	基本的なプログラムを動かせた
追加的課題：	ミニバッチの数を 100，500，1000，5000，10000、エポック数を 10，20，30，40，50 にしたときの判別制度の違いを比較した
追加的課題：	

背景と実験目的

概要：

Pytorch と呼ばれる有力なディープラーニングのパッケージを用いて、最も基礎的な分類問題を解くプログラムを扱ってみることで、ニューラルネットワークの概念を理解し、ディープラーニングに繋がる入門的知識を理解する。

目標：

最も基礎的な分類問題を扱い、元重基本的なニューラルネットワークを構成し、ディープラーニングへとつながる手法で解くことができる。

課題 1　　（各章題はゴシック体。本文は明朝体）

【問題】

テーマの内容の学習問題を実行し、試行錯誤してみる。

【アルゴリズム・解き方】

1. データセットを用意し、訓練データとテストデータに分割する。
2. データをミニバッチ処理する
3. 最適化手法を用いてネットワークを作成する
4. 作成したネットワークを train 関数で学習する
5. test 関数で正解数、精度を取得する

【実行結果】

バッチ数を[100, 500, 2500, 12500, 62500], エポック数を 10 で固定した時の出力をそれぞれ図 1～5 に示す。

またバッチ数を 62500 で固定、エポック数を[20, 30, 40, 50]にしたときの出力をそれぞれ図 6～9 に示す。

```
epoch 0 : loss 173.55834078416228
epoch 1 : loss 63.94455514661968
epoch 2 : loss 41.50013287598267
epoch 3 : loss 29.703751460649073
epoch 4 : loss 21.800097821978852
epoch 5 : loss 17.9219986818498
epoch 6 : loss 13.882630031206645
epoch 7 : loss 12.493573541461956
epoch 8 : loss 9.507569077832159
epoch 9 : loss 8.765088661348273
correct: 9767
total: 10000
accuracy: 0.9767
```

図 1 バッチ数 100, エポック数 10

```
epoch 0 : loss 62.56294696033001
epoch 1 : loss 23.04151540249586
epoch 2 : loss 15.637311555445194
epoch 3 : loss 11.764017604291439
epoch 4 : loss 8.901687242090702
epoch 5 : loss 7.108125327154994
epoch 6 : loss 5.770860617980361
epoch 7 : loss 4.615906152874231
epoch 8 : loss 3.619429607875645
epoch 9 : loss 3.0116408225148916
correct: 9795
total: 10000
accuracy: 0.9795
```

図 2 バッチ数 500, エポック数 10

```
epoch 0 : loss 29.6895948946476
epoch 1 : loss 8.5655317902565
epoch 2 : loss 6.654867112636566
epoch 3 : loss 5.601213097572327
epoch 4 : loss 4.796360969543457
epoch 5 : loss 4.162790283560753
epoch 6 : loss 3.6345842331647873
epoch 7 : loss 3.2507170885801315
epoch 8 : loss 2.948467642068863
epoch 9 : loss 2.619794048368931
correct: 9665
total: 10000
accuracy: 0.9665
```

図 3 バッチ数 2500, エポック数 10

```
epoch 0 : loss 10.914264440536499
epoch 1 : loss 8.40133786201477
epoch 2 : loss 5.156432569026947
epoch 3 : loss 3.1505094170570374
epoch 4 : loss 2.349150210618973
epoch 5 : loss 1.9901646077632904
epoch 6 : loss 1.774524986743927
epoch 7 : loss 1.6259336471557617
epoch 8 : loss 1.5100322663784027
epoch 9 : loss 1.4169001281261444
correct: 9212
total: 10000
accuracy: 0.9212
```

図 4 バッチ数 12500, エポック数 10

```
epoch 0 : loss 2.305121660232544
epoch 1 : loss 2.2545671463012695
epoch 2 : loss 2.2002322673797607
epoch 3 : loss 2.1326112747192383
epoch 4 : loss 2.0475823879241943
epoch 5 : loss 1.9444929361343384
epoch 6 : loss 1.8256832361221313
epoch 7 : loss 1.6948142051696777
epoch 8 : loss 1.5561513900756836
epoch 9 : loss 1.4135299921035767
correct: 7697
total: 10000
accuracy: 0.7697
```

図 5 バッチ数 62500, エポック数 10

```
epoch 0 : loss 2.3025012016296387
epoch 1 : loss 2.25024676322937
epoch 2 : loss 2.1948940753936768
epoch 3 : loss 2.1255922317504883
epoch 4 : loss 2.039212465286255
epoch 5 : loss 1.9358315467834473
epoch 6 : loss 1.8172122240066528
epoch 7 : loss 1.6863670349121094
epoch 8 : loss 1.5474265813827515
epoch 9 : loss 1.4053854942321777
epoch 10 : loss 1.2649537324905396
epoch 11 : loss 1.1308258771896362
epoch 12 : loss 1.00765860080719
epoch 13 : loss 0.8986019492149353
epoch 14 : loss 0.8050161600112915
epoch 15 : loss 0.7268673777580261
epoch 16 : loss 0.6621514558792114
epoch 17 : loss 0.6091479063034058
epoch 18 : loss 0.5655999183654785
epoch 19 : loss 0.529478132724762
correct: 8620
total: 10000
accuracy: 0.862
```

図 6 バッチ数 62500, エポック数 20

```
epoch 0 : loss 2.3044159412384033
epoch 1 : loss 2.2510886192321777
epoch 2 : loss 2.1933412551879883
epoch 3 : loss 2.1211910247802734
epoch 4 : loss 2.0324926376342773
epoch 5 : loss 1.9277634620666504
epoch 6 : loss 1.8090219497680664
epoch 7 : loss 1.6785062551498413
epoch 8 : loss 1.5392454862594604
epoch 9 : loss 1.396734356880188
epoch 10 : loss 1.2569488286972046
epoch 11 : loss 1.1241945028305054
epoch 12 : loss 1.0018019676208496
epoch 13 : loss 0.893258273601532
epoch 14 : loss 0.7997719645500183
epoch 15 : loss 0.7208045125007629
epoch 16 : loss 0.6561277508735657
epoch 17 : loss 0.6030967831611633
epoch 18 : loss 0.5595098733901978
epoch 19 : loss 0.5240723490715027
epoch 20 : loss 0.4943402409553528
epoch 21 : loss 0.46971186995506287
epoch 22 : loss 0.44865673780441284
epoch 23 : loss 0.4308737814426422
epoch 24 : loss 0.4154701828956604
epoch 25 : loss 0.40227606892585754
epoch 26 : loss 0.39075982570648193
epoch 27 : loss 0.38078755140304565
epoch 28 : loss 0.3718320429325104
epoch 29 : loss 0.3637542128562927
correct: 9011
total: 10000
accuracy: 0.9011
```

図 7 バッチ数 62500, エポック数 30

```
epoch 0 : loss 2.3007757663726807
epoch 1 : loss 2.246591567993164
epoch 2 : loss 2.1887288093566895
epoch 3 : loss 2.1165659427642822
epoch 4 : loss 2.0262067317962646
epoch 5 : loss 1.9178473949432373
epoch 6 : loss 1.7943235635757446
epoch 7 : loss 1.6597613096237183
epoch 8 : loss 1.519080638885498
epoch 9 : loss 1.3766494989395142
epoch 10 : loss 1.2369531393051147
epoch 11 : loss 1.1051559448242188
epoch 12 : loss 0.985225260257721
epoch 13 : loss 0.8789844512939453
epoch 14 : loss 0.7877845168113708
epoch 15 : loss 0.7121244668960571
epoch 16 : loss 0.64991295337677
epoch 17 : loss 0.5990026593208313
epoch 18 : loss 0.5577190518379211
epoch 19 : loss 0.5237220525741577
epoch 20 : loss 0.49571582674980164
epoch 21 : loss 0.47200822830200195
epoch 22 : loss 0.45218977332115173
epoch 23 : loss 0.4349249601364136
epoch 24 : loss 0.420102596282959
epoch 25 : loss 0.4069238603115082
epoch 26 : loss 0.3953341543674469
epoch 27 : loss 0.3847563862800598
epoch 28 : loss 0.3752500116825104
epoch 29 : loss 0.3664630949497223
epoch 30 : loss 0.3585067689418793
epoch 31 : loss 0.351092666387558
epoch 32 : loss 0.34426888823509216
epoch 33 : loss 0.3379625082015991
epoch 34 : loss 0.33200469613075256
epoch 35 : loss 0.32646799087524414
epoch 36 : loss 0.321125328540802
epoch 37 : loss 0.316058874130249
epoch 38 : loss 0.31115642189979553
epoch 39 : loss 0.3064550459384918
correct: 9145
total: 10000
accuracy: 0.9145
```

図 8 バッチ数 62500, エポック数 40

```
epoch 0 : loss 2.3011019229888916
epoch 1 : loss 2.2533533573150635
epoch 2 : loss 2.199401617050171
epoch 3 : loss 2.1286449432373047
epoch 4 : loss 2.037757635116577
epoch 5 : loss 1.928623080253601
epoch 6 : loss 1.8050518035888672
epoch 7 : loss 1.6708815097808838
epoch 8 : loss 1.5294886827468872
epoch 9 : loss 1.3847769498825073
epoch 10 : loss 1.2421321868896484
epoch 11 : loss 1.1068207025527954
epoch 12 : loss 0.9830066561698914
epoch 13 : loss 0.8738434314727783
epoch 14 : loss 0.7801000475883484
epoch 15 : loss 0.7019545435905457
epoch 16 : loss 0.6379507780075073
epoch 17 : loss 0.5860605835914612
epoch 18 : loss 0.5447275042533875
epoch 19 : loss 0.5109310746192932
epoch 20 : loss 0.4834236800670624
epoch 21 : loss 0.4606795012950897
epoch 22 : loss 0.4416053593158722
epoch 23 : loss 0.42540261149406433
epoch 24 : loss 0.41150733828544617
epoch 25 : loss 0.39920032024383545
epoch 26 : loss 0.3885800540447235
epoch 27 : loss 0.37880218029022217
epoch 28 : loss 0.3700726330280304
epoch 29 : loss 0.3619018495082855
epoch 30 : loss 0.3543650507926941
epoch 31 : loss 0.3473009765148163
epoch 32 : loss 0.34068843722343445
epoch 33 : loss 0.33449316024780273
epoch 34 : loss 0.32867419719696045
epoch 35 : loss 0.3232157528400421
epoch 36 : loss 0.3180248439311981
epoch 37 : loss 0.31308290362358093
epoch 38 : loss 0.308322548866272
epoch 39 : loss 0.30369341373443604
epoch 40 : loss 0.29924705624580383
epoch 41 : loss 0.2948778569698334
epoch 42 : loss 0.29062870144844055
epoch 43 : loss 0.28649020195007324
epoch 44 : loss 0.28247788548469543
epoch 45 : loss 0.27861490845680237
epoch 46 : loss 0.2748735249042511
epoch 47 : loss 0.27126485109329224
epoch 48 : loss 0.2677608430385596
epoch 49 : loss 0.2643364369869232
correct: 9249
total: 10000
accuracy: 0.9249
```

図 9 バッチ数 62500, エポック数 50

【考察】

図 1~5 の実行結果、図 5~9 の実行結果をグラフ化し、それぞれ図 10, 11 に示す。

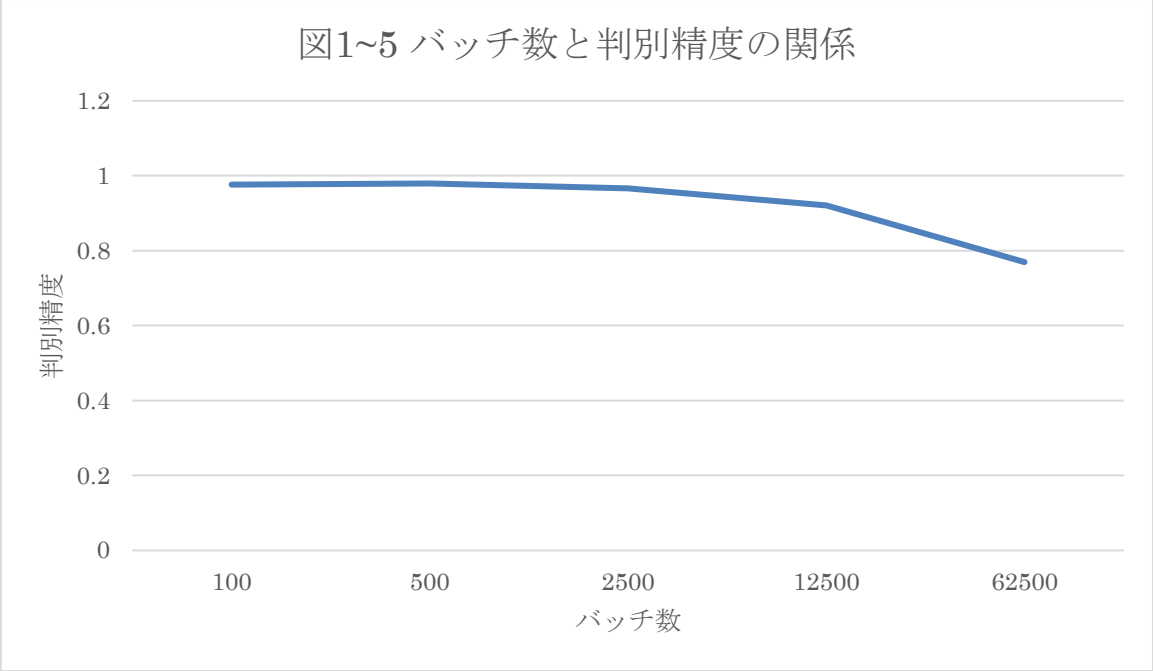

図 10 図 1~5 バッチ数と判別精度の関係グラフ

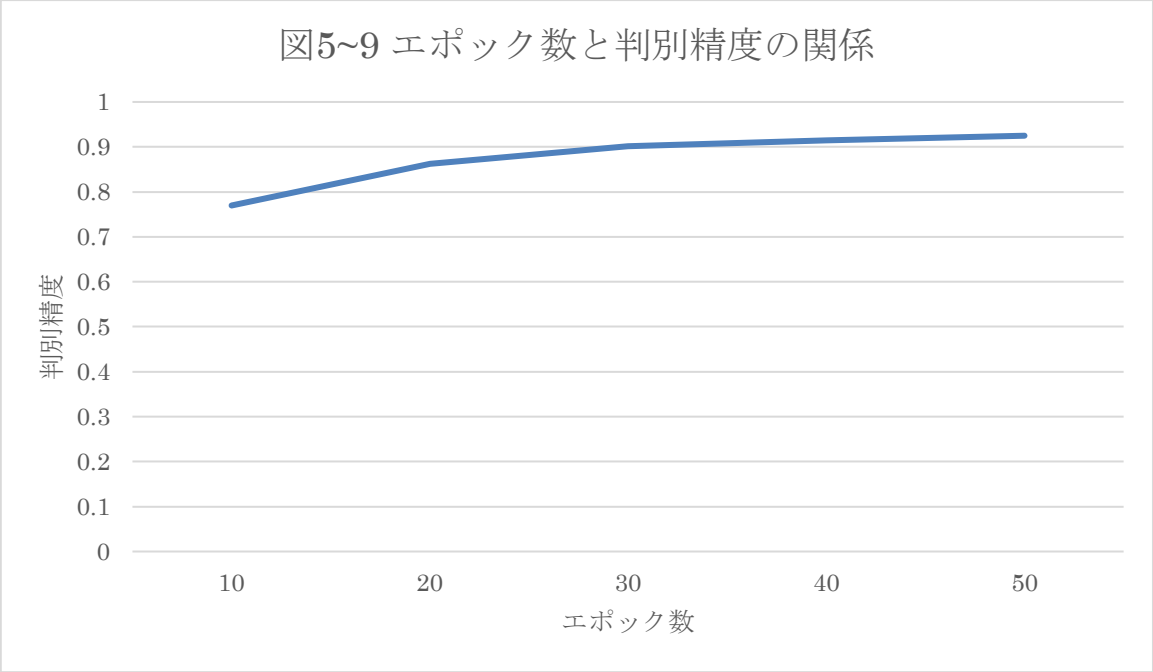

図 11 図 5~9 バッチ数と判別精度の関係グラフ

図 11 を見るとバッチ数を増やしていくと判別精度が落ちていくことが分かる。
図 12 を見るとエポック数を増やしていくと判別精度が上がっていくことがわかる。
これはエポック数を増やすことによって学習回数が増えるためである。

まとめ

今回の実験ではディープラーニングの基礎を学んだが、バッチ数やエポック数を変更してデータを集め、分析することで、その違いを目で見て理解できたためよかった。

付録

【プログラムソース】

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision as tv

MODEL_NAME = "mnist.model"
BATCHSIZE = 500
EPOCH = 10
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

train_dataset = tv.datasets.MNIST(root="./",
                                  train=True,
                                  transform=tv.transforms.ToT
ensor(),
                                  download=True)

test_dataset = tv.datasets.MNIST(root="./",
                                 train=False,
                                 transform=tv.transforms.ToTe
nsor(),
                                 download=True)

train_loader =
torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=BATCHSIZ
E,
                                           shuffle=True)
```

```
test_loader =
torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=BATCHSIZE
,
                                          shuffle=False)

class MNIST(nn.Module):
    def __init__(self):
        super(MNIST,self).__init__()
        self.l1 = nn.Linear(784, 300)
        self.l2 = nn.Linear(300, 300)
        self.l3 = nn.Linear(300, 10)
    def forward(self,x):
        h = F.relu(self.l1(x))
        h = F.relu(self.l2(h))
        y = self.l3(h)
        return y

def train(train_loader):
    model = MNIST().to(DEVICE)
    optimizer = torch.optim.Adam(model.parameters())
    model.train()
    for epoch in range(EPOCH):
        total_loss = 0
        for images, labels in train_loader:
            images = images.view(-1, 28*28).to(DEVICE)
            labels = labels.to(DEVICE)
            optimizer.zero_grad()
            y = model(images)
            loss = F.cross_entropy(y, labels)
            total_loss += loss.item()
            loss.backward()
            optimizer.step()
        print("epoch", epoch, ": loss", total_loss)
    torch.save(model.state_dict(), MODEL_NAME)

def test(test_loader):
    total = len(test_loader.dataset)
```

```
    correct = 0
    model = MNIST().to(DEVICE)
    model.load_state_dict(torch.load(MODEL_NAME,
weights_only=True))
    model.eval()
    for images, labels in test_loader:
        images = images.view(-1, 28*28).to(DEVICE)
        labels = labels.to(DEVICE)
        y = model(images)
        pred_labels = y.max(dim=1)[1]
        correct += (pred_labels == labels).sum()
    print("correct:", correct.item())
    print("total:", total)
    print("accuracy:", correct.item()/total)

train(train_loader)
test(test_loader)
```