**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。

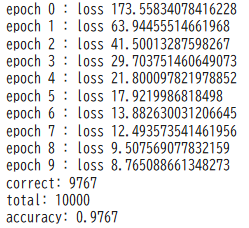
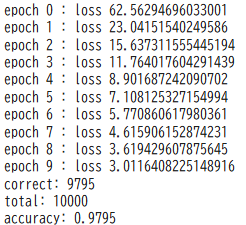
 

図1 バッチ数100, エポック数10 図2バッチ数500, エポック数10

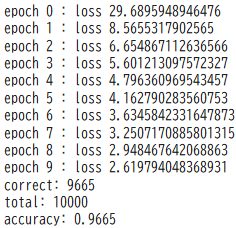
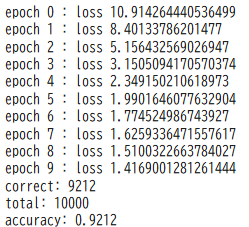
 

図3 バッチ数2500, エポック数10 図4バッチ数12500, エポック数10

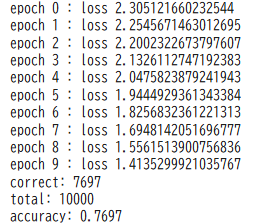


図5バッチ数62500, エポック数10

テキスト

自動的に生成された説明新聞の記事

自動的に生成された説明

図6 バッチ数62500, エポック数20 図7バッチ数62500, エポック数30

新聞記事の一部

中程度の精度で自動的に生成された説明

図8 バッチ数62500, エポック数40

新聞の記事

自動的に生成された説明

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.ToTensor(),

                                  download=True)

test\_dataset = tv.datasets.MNIST(root="./",

                                 train=False,

                                 transform=tv.transforms.ToTensor(),

                                 download=True)

train\_loader = torch.utils.data.DataLoader(dataset=train\_dataset,

                                           batch\_size=BATCHSIZE,

                                           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)