# 实验一 手写数字识别

## 一、实验目的

1.掌握卷积神经网络基本原理；

2.掌握PyTorch（或其他框架）的基本用法以及构建卷积网络的基本操作；

3.了解PyTorch（或其他框架）在GPU上的使用方法。

## 二、实验要求

1.搭建PyTorch（或其他框架）环境；

2.构建一个规范的卷积神经网络结构；

3.在MNIST手写数字数据集上进行训练和评估，实现测试集准确率达到98%及以上；

4.按规定时间在课程网站提交实验报告、代码以及PPT。

## 三、实验原理（以PyTorch为例）

1.PyTorch基本用法：

使用 PyTorch, 必须了解PyTorch:

（1）使用动态图（Eager Execution）而非旧版本TensorFlow的静态图（Graph Execution）来表示计算任务。

（2）使用张量 （tensor）表示数据，可以说numpy的替代品。

（3）通过变量 (Variable) 维护状态，简单封装了 Tensor，并记录对 tensor 的操作记录用来构建计算图。Variable 主要包含三个属性：

* data：保存 Variable 所包含的 tensor。
* grad：保存 data 对应的梯度，grad 也是 Variable，而非 tensor，它与 data 形状一致。
* grad\_fn：指向一个 Function，记录 Variable 的操作历史，即它是什么操作的输出，用来构建计算图。

（4）autograd 包是 PyTorch 中所有神经网络的核心，该 autograd 软 件包为 Tensors 上的所有操作提供自动微分。

（5）torch.Tensor 是包的核心类。如果将其属性 .requires\_grad 设置为 True，则会开始跟踪针对 tensor 的所有操作。完成计算后，可以调用 .backward() 来自动计算所有梯度。该张量的梯度将累积到 .grad 属性中。要停止 tensor 历史记录的跟踪，您可以调用 .detach()，它将其与计算历史记录分离，并防止将来的计算被跟踪。

（6）如果计算导数，可以调用 Tensor.backward()。

（7）神经网络可以通过 torch.nn 包来构建。神经网络基于自动梯度 (autograd)来定义一些模型。一个 nn.Module 包括层和一个方法 forward(input) ，它会返回输 出(output)。

2.卷积神经网络：

典型的卷积神经网络由卷积层、池化层、激活函数层交替组合构成，因此可将其视为一种层次模型，形象地体现了深度学习中“深度”之所在。

（1）卷积操作

卷积运算是卷积神经网络的核心操作，给定二维的图像I作为输入，二维卷积核K， 卷积运算可表示为：

(1)

给定5×5输入矩阵、3×3卷积核，相应的卷积操作如图1所示。



图1 卷积运算

在使用TensorFlow等深度学习框架时，卷积层会有padding参数，常用的有两种选择，一个是“valid”，一个是“same”。前者是不进行填充，后者则是进行数据填充并保证输出与输入具有相同尺寸。

构建卷积或池化神经网络时，卷积步长也是一个很重要的基本参数。它控制了每个操作在特征图上的执行间隔。

（2）池化操作

池化操作使用某位置相邻输出的总体统计特征作为该位置的输出，常用最大池化（max-pooling）和均值池化（average-pooling）。池化层不包含需要训练学习的参数，仅需指定池化操作的核大小、操作步长以及池化类型。池化操作示意如图2所示。



图2 池化操作

（3）激活函数层

卷积操作可视为对输入数值进行线性计算发挥线性映射的作用。激活函数的引入，则增强了深度网络的非线性表达能力，从而提高了模型的学习能力。常用的激活函数有sigmoid、tanh和ReLU函数。

## 四、实验所用工具及数据集（以PyTorch为例）

1.工具

PyCharm、PyTorch （下载地址及相关介绍：<https://www.jetbrains.com/pycharm/>，<https://pytorch.org/> ）

2.数据集

MNIST手写数字数据集（下载地址及相关介绍：<http://yann.lecun.com/exdb/mnist/> ）

## 五、实验步骤与方法（以PyTorch为例）

**1.安装实验环境**，包括PyCharm、PyTorch，如果使用GPU版本还需要安装cuda、cudnn；

**2.下载MNIST手写数字数据集**；

**3.编辑代码**

**1）定义读取标签和图像函数**

# 1.1.Function to read label & image

**def** \_read**(**image**,** label**):**

# minist\_dir = os.path.dirname(\_\_file\_\_)+'/MNIST\_data/'

minist\_dir **=** './MNIST\_data/'

**with** gzip**.open(**minist\_dir **+** label**)** **as** flbl**:**

magic**,** num **=** struct**.**unpack**(**">II"**,** flbl**.**read**(**8**))**

label **=** np**.**fromstring**(**flbl**.**read**(),** dtype**=**np**.**int8**)**

**with** gzip**.open(**minist\_dir **+** image**,** 'rb'**)** **as** fimg**:**

magic**,** num**,** rows**,** cols **=** struct**.**unpack**(**">IIII"**,** fimg**.**read**(**16**))**

image **=** np**.**fromstring**(**fimg**.**read**(),** dtype**=**np**.**uint8**).**reshape**(**

**len(**label**),** rows**,** cols**)**

**return** image**,** label

# 1.2.Function to get data from .gz file

**def** get\_data**():**

train\_img**,** train\_label **=** \_read**(**

'train-images-idx3-ubyte.gz'**,**

'train-labels-idx1-ubyte.gz'**)**

test\_img**,** test\_label **=** \_read**(**

't10k-images-idx3-ubyte.gz'**,**

't10k-labels-idx1-ubyte.gz'**)**

**return** **[**train\_img**,** train\_label**,** test\_img**,** test\_label**]**

**2）定义LeNet5网络**

# 1.3.LeNet5

# 32-5+1=28,(28-2)/2+1=14,14-5+1=10,(10-2)/2+1=5,5-5+1=1,

# 1\*120 -> 84=7\*12 -> 10

**class** **LeNet5(**nn**.**Module**):**

**def** \_\_init\_\_**(**self**):**

**super(**LeNet5**,** self**).**\_\_init\_\_**()**

self**.**conv1 **=** nn**.**Conv2d**(**1**,** 6**,** 5**,** padding**=**2**)**

self**.**conv2 **=** nn**.**Conv2d**(**6**,** 16**,** 5**)**

self**.**fc1 **=** nn**.**Linear**(**16 **\*** 5 **\*** 5**,** 120**)**

self**.**fc2 **=** nn**.**Linear**(**120**,** 84**)**

self**.**fc3 **=** nn**.**Linear**(**84**,** 10**)**

**def** forward**(**self**,** x**):**

x **=** F**.**max\_pool2d**(**torch**.**tanh**(**self**.**conv1**(**x**)),** **(**2**,** 2**))**

x **=** F**.**dropout**(**x**,** p**=**0.3**,** training**=**self**.**training**)**

x **=** F**.**max\_pool2d**(**torch**.**tanh**(**self**.**conv2**(**x**)),** **(**2**,** 2**))**

x **=** F**.**dropout**(**x**,** p**=**0.3**,** training**=**self**.**training**)**

x **=** x**.**view**(-**1**,** self**.**num\_flat\_features**(**x**))**

# print('x.size:', x.size()) # [100, 400]

x **=** torch**.**tanh**(**self**.**fc1**(**x**))**

x **=** F**.**dropout**(**x**,** p**=**0.3**,** training**=**self**.**training**)**

x **=** torch**.**tanh**(**self**.**fc2**(**x**))**

x **=** F**.**dropout**(**x**,** p**=**0.3**,** training**=**self**.**training**)**

x **=** self**.**fc3**(**x**)**

**return** x

# Flatten the size of x (BATCH\_SIZE\*16\*5\*5 -> BATCH\_SIZE\*400)

**def** num\_flat\_features**(**self**,** x**):**

size **=** x**.**size**()[**1**:]**

num\_features **=** 1

**for** s **in** size**:**

num\_features **\*=** s

**return** num\_features

**3）定义初始化参数函数**

# 1.4.Function to initialize parameters

**def** weight\_init**(**m**):**

**if** **isinstance(**m**,** nn**.**Conv2d**):**

n **=** m**.**kernel\_size**[**0**]** **\*** m**.**kernel\_size**[**1**]** **\*** m**.**out\_channels

m**.**weight**.**data**.**normal\_**(**0**,** math**.**sqrt**(**2. **/** n**))**

**elif** **isinstance(**m**,** nn**.**BatchNorm2d**):**

m**.**weigth**.**data**.**fill\_**(**1**)**

m**.**bias**.**data**.**zero\_**()**

**4）定义训练和测试函数**

# 1.5.Function to train network

**def** train**(**epoch**):**

model**.**train**()**

**for** batch\_idx**,** **(**data**,** target**)** **in** **enumerate(**train\_loader**):**

**if** use\_gpu**:**

data**,** target **=** data**.**cuda**(),** target**.**cuda**()**

data**,** target **=** Variable**(**data**),** Variable**(**target**.**long**())**

optimizer**.**zero\_grad**()**

outputs **=** model**(**data**)**

# print(data.shape, outputs.shape, target.shape) # [100, 1, 28, 28] [100, 10] [100]

loss **=** criterion**(**outputs**,** target**)**

loss**.**backward**()**

optimizer**.**step**()**

**if** **(**batch\_idx**+**1**)** **%** 100 **==** 0**:**

**print(**'Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'**.format(**

epoch**,** **(**batch\_idx**+**1**)** **\*** **len(**data**),** **len(**train\_loader**.**dataset**),**

100. **\*** **(**batch\_idx**+**1**)** **/** **len(**train\_loader**),** loss**.**data**))**

# 1.6.Function to test network

**def** test**():**

model**.eval()**

test\_loss **=** 0

correct **=** 0

**for** data**,** target **in** test\_loader**:**

**if** use\_gpu**:**

data**,** target **=** data**.**cuda**(),** target**.**cuda**()**

**with** torch**.**no\_grad**():**

data **=** Variable**(**data**)**

# data = Variable(data, volatile=True)

target **=** Variable**(**target**.**long**())**

outputs **=** model**(**data**)**

test\_loss **+=** criterion**(**outputs**,** target**).**data

pred **=** outputs**.**data**.max(**1**,** keepdim**=True)[**1**]**

correct **+=** pred**.**eq**(**target**.**data**.**view\_as**(**pred**)).**cpu**().sum()**

test\_loss **/=** **len(**test\_loader**.**dataset**)**

**print(**'\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'**.format(**

test\_loss**,** correct**,** **len(**test\_loader**.**dataset**),**

100. **\*** correct **/** **len(**test\_loader**.**dataset**)))**

**5）定义一些常量参数**

# 2.1.Set some parameters

# use\_gpu = torch.cuda.is\_available()

use\_gpu **=** **False**

BATCH\_SIZE **=** 100

kwargs **=** **{**'num\_workers'**:** 0**,** 'pin\_memory'**:** **True}**

**6）调用函数生成批量加载器dataloader**

# 2.2.Prepare data of train\_img, train\_label, test\_img, test\_label

train\_img**,** train\_label**,** test\_img**,** test\_label **=** get\_data**()**

train\_x**,** train\_y **=** torch**.**from\_numpy**(**

train\_img**.**reshape**(-**1**,** 1**,** 28**,** 28**)).float(),** torch**.**from\_numpy**(**train\_label**.**astype**(int))**

test\_x**,** test\_y **=** torch**.**from\_numpy**(**

test\_img**.**reshape**(-**1**,** 1**,** 28**,** 28**)).float(),** torch**.**from\_numpy**(**test\_label**.**astype**(int))**

train\_dataset **=** TensorDataset**(**train\_x**,** train\_y**)**

test\_dataset **=** TensorDataset**(**test\_x**,** test\_y**)**

train\_loader **=** DataLoader**(**dataset**=**train\_dataset**,**

shuffle**=True,** batch\_size**=**BATCH\_SIZE**,** **\*\***kwargs**)**

test\_loader **=** DataLoader**(**dataset**=**test\_dataset**,**

shuffle**=True,** batch\_size**=**BATCH\_SIZE**,** **\*\***kwargs**)**

# print(len(train\_loader), len(test\_loader)) # 600 100

**7）抽样绘图查看读取数据是否正确**

# 2.3.Print the shape(100\*1\*28\*28) of data & labels(100) in each batch , and show the data as gray img

**for** i**,** **(**data**,** target**)** **in** **enumerate(**train\_loader**):**

**if** i **==** 0**:**

**print(**data**.**shape**,** target**)**

**for** j **in** **range(**BATCH\_SIZE**):**

**if** j **<** 10**:**

plt**.**figure**()**

plt**.**imshow**(**data**[**j**][**0**],** cmap**=**'gray'**)**

plt**.**show**()**

**8）实例化网络并初始权重，定义优化器，定义损失函数**

# 2.4.Instantiation network, set optimizer & criterion, apply weight\_init funcction

model **=** LeNet5**()**

# optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

optimizer **=** optim**.**Adam**(**model**.**parameters**(),** lr**=**0.001**,** betas**=(**0.9**,** 0.99**))**

criterion **=** nn**.**CrossEntropyLoss**(**size\_average**=False)**

model**.**apply**(**weight\_init**)**

**9）训练和测试**

# 2.5.Train & test

ENDEPOCH **=** 99

**for** epoch **in** **range(**0**,** ENDEPOCH**+**1**):**

**print(**'----------------start train-----------------'**)**

train**(**epoch**)**

**print(**'----------------end train-----------------'**)**

# save the parameters of final model

**if** epoch **==** ENDEPOCH**:**

torch**.**save**(**model**.**state\_dict**(),** './model\_params.pkl'**)**

# test each epoch

**print(**'----------------start test-----------------'**)**

test**()**

**print(**'----------------end test-----------------'**)**

# load the parameters of final model and then test the model

model **=** LeNet5**()**

model**.**load\_state\_dict**(**torch**.**load**(**'./model\_params.pkl'**))**

**print(**'----------------start final test-----------------'**)**

test**()**

**print(**'----------------end final test-----------------'**)**

## 六、实验结果

**...**

**----------------**start train**-----------------**

Train Epoch**:** 99 **[**10000**/**60000 **(**17**%)]** Loss**:** 0.695330

Train Epoch**:** 99 **[**20000**/**60000 **(**33**%)]** Loss**:** 12.374598

Train Epoch**:** 99 **[**30000**/**60000 **(**50**%)]** Loss**:** 7.556098

Train Epoch**:** 99 **[**40000**/**60000 **(**67**%)]** Loss**:** 11.868968

Train Epoch**:** 99 **[**50000**/**60000 **(**83**%)]** Loss**:** 7.283059

Train Epoch**:** 99 **[**60000**/**60000 **(**100**%)]** Loss**:** 5.666159

**----------------**end train**-----------------**

**----------------**start test**-----------------**

Test **set:** Average loss**:** 0.0393**,** Accuracy**:** 9883**/**10000 **(**98.83**%)**

**----------------**end test**-----------------**

**----------------**start final test**-----------------**

Test **set:** Average loss**:** 0.0393**,** Accuracy**:** 9883**/**10000 **(**98.83**%)**

**----------------**end final test**-----------------**

实验采用了所有训练数据60000对训练，所有测试数据10000对测试，训练了100次，实现了在测试集数据上手写数字识别准确率98.83%。