实验四：深度学习算法及应用

一、实验名称

实验四：深度学习算法及应用

二、实验目的

1. 了解深度学习的基本原理；
2. 能够使用深度学习开源工具；
3. 应用深度学习算法求解实际问题。

三、实验要求

1. 解释深度学习原理；
2. 对实验性能进行分析；
3. 回答思考题。

四、开发环境

- OS: Windows 10 Pro X64
- Language: Python 3.7
- 百度飞桨及AIstudio平台：https://www.paddlepaddle.org.cn/

五、实验内容及步骤

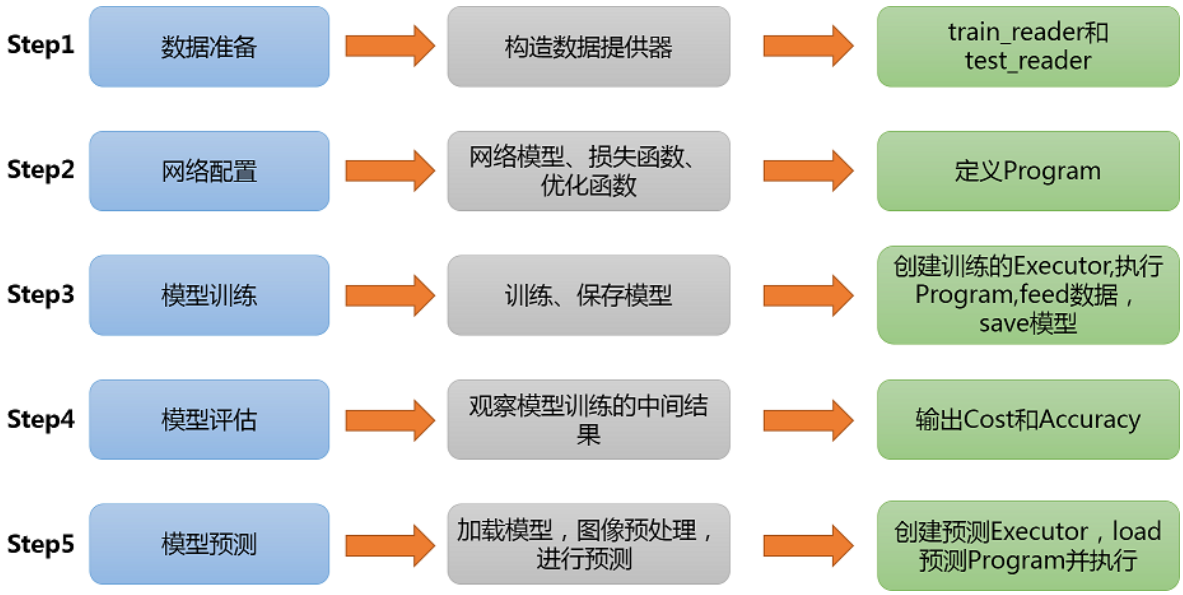
选用一个实验平台，采用开源深度学习工具求解实际人工智能应用问题，例如计算机视觉、自然语言处理、无人驾驶、语音识别等。

实验内容

基于百度飞桨和AIstudio平台，使用多层感知器训练DNN模型，预测手写数字图片。

实验步骤

实验总体过程和步骤如下图



首先导入必要的包

- numpy---------->python第三方库，用于进行科学计算
- PIL---------->Python Image Library,python第三方图像处理库
- matplotlib----->python的绘图库,pyplot:matplotlib的绘图框架
- os---------->提供了丰富的方法来处理文件和目录

In [1]

```
#导入需要的包
import numpy as np
import paddle as paddle
import paddle.fluid as fluid
from PIL import Image
import matplotlib.pyplot as plt
import os
```

Step1 准备数据

(1)数据集介绍

MNIST数据集包含60000个训练集和10000测试集。分为图片和标签，图片是28*28的像素矩阵，标签为0~9共10个数字。

(2)train_reader和test_reader

paddle.dataset.mnist.train()和test()分别用于获取mnist训练集和测试集

paddle.reader.shuffle()表示每次缓存BUF_SIZE个数据项，并进行打乱

paddle.batch()表示每BATCH_SIZE组成一个batch

（3）打印看下数据是什么样的？PaddlePaddle接口提供的数据已经经过了归一化、居中等处理。

In [2]

```
BUF_SIZE=512
BATCH_SIZE=128
#用于训练的数据提供器，每次从缓存中随机读取批次大小的数据
train_reader = paddle.batch(
    paddle.reader.shuffle(paddle.dataset.mnist.train(),
                          buf_size=BUF_SIZE),
    batch_size=BATCH_SIZE)
#用于训练的数据提供器，每次从缓存中随机读取批次大小的数据
test_reader = paddle.batch(
    paddle.reader.shuffle(paddle.dataset.mnist.test(),
                          buf_size=BUF_SIZE),
    batch_size=BATCH_SIZE)

#用于打印，查看mnist数据
train_data=paddle.dataset.mnist.train();
sampledata=next(train_data())
print(sampledata)
```

```
[==================================================]t/train-images-idx3-ubyte.gz
not found, downloading https://dataset.bj.bcebos.com/mnist/train-images-idx3-
ubyte.gz
[==================================================]t/train-labels-idx1-ubyte.gz
not found, downloading https://dataset.bj.bcebos.com/mnist/train-labels-idx1-
ubyte.gz
[==================================================]t/t10k-images-idx3-ubyte.gz
not found, downloading https://dataset.bj.bcebos.com/mnist/t10k-images-idx3-
ubyte.gz
[==================================================]t/t10k-labels-idx1-ubyte.gz
not found, downloading https://dataset.bj.bcebos.com/mnist/t10k-labels-idx1-
ubyte.gz
(array([-1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
```

```
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -0.9764706 , -0.85882354, -0.85882354,
       -0.85882354, -0.01176471,  0.06666672,  0.37254906, -0.79607844,
        0.30196083,  1.        ,  0.9372549 , -0.00392157, -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -0.7647059 , -0.7176471 , -0.26274508,  0.20784318,
        0.33333337,  0.9843137 ,  0.9843137 ,  0.9843137 ,  0.9843137 ,
        0.9843137 ,  0.7647059 ,  0.34901965,  0.9843137 ,  0.8980392 ,
        0.5294118 , -0.4980392 , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -0.6156863 ,  0.8666667 ,
        0.9843137 ,  0.9843137 ,  0.9843137 ,  0.9843137 ,  0.9843137 ,
        0.9843137 ,  0.9843137 ,  0.9843137 ,  0.96862745, -0.27058822,
       -0.35686272, -0.35686272, -0.56078434, -0.69411767, -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -0.85882354,  0.7176471 ,  0.9843137 ,  0.9843137 ,
        0.9843137 ,  0.9843137 ,  0.9843137 ,  0.5529412 ,  0.427451  ,
        0.9372549 ,  0.8901961 , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -0.372549  ,  0.22352946, -0.1607843 ,  0.9843137 ,  0.9843137 ,
        0.60784316, -0.9137255 , -1.        , -0.6627451 ,  0.20784318,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -0.8901961 ,
       -0.99215686,  0.20784318,  0.9843137 , -0.29411763, -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        ,  0.09019613,
        0.9843137 ,  0.4901961 , -0.9843137 , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -0.9137255 ,  0.4901961 ,  0.9843137 ,
       -0.45098037, -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -0.7254902 ,  0.8901961 ,  0.7647059 ,  0.254902  ,
       -0.15294117, -0.99215686, -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
```

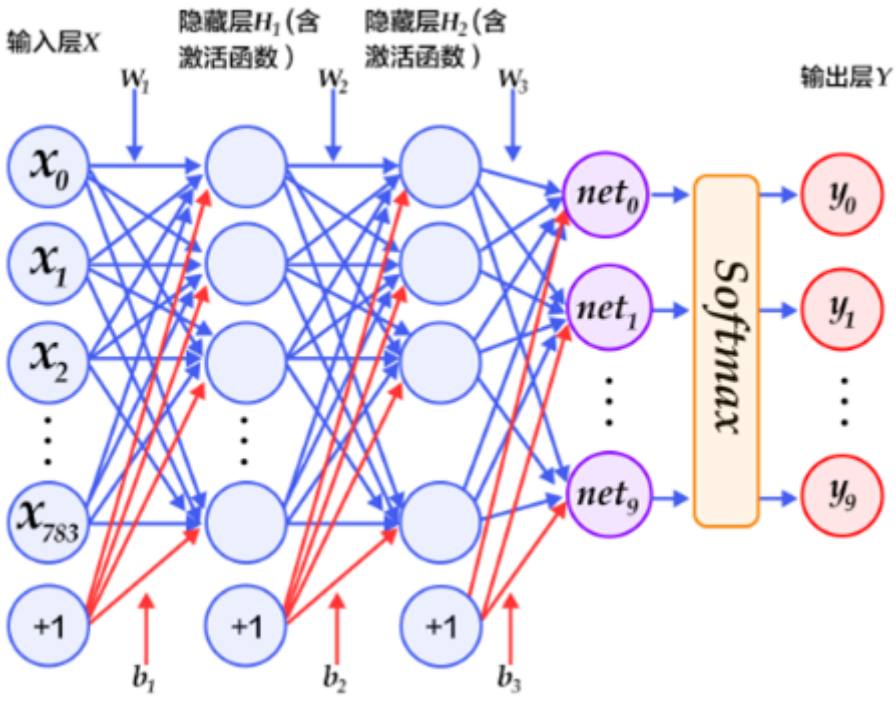
```
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -0.36470586,  0.88235295,  0.9843137 ,  0.9843137 , -0.06666666,
       -0.8039216 , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -0.64705884,
        0.45882356,  0.9843137 ,  0.9843137 ,  0.17647064, -0.7882353 ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -0.8745098 , -0.27058822,
        0.9764706 ,  0.9843137 ,  0.4666667 , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        ,  0.9529412 ,  0.9843137 ,
        0.9529412 , -0.4980392 , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -0.6392157 ,  0.0196079 ,
        0.43529415,  0.9843137 ,  0.9843137 ,  0.62352943, -0.9843137 ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -0.69411767,
        0.16078436,  0.79607844,  0.9843137 ,  0.9843137 ,  0.9843137 ,
        0.9607843 ,  0.427451  , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -0.8117647 , -0.10588235,  0.73333335,  0.9843137 ,  0.9843137 ,
        0.9843137 ,  0.9843137 ,  0.5764706 , -0.38823527, -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -0.81960785, -0.4823529 ,  0.67058825,  0.9843137 ,
        0.9843137 ,  0.9843137 ,  0.9843137 ,  0.5529412 , -0.36470586,
       -0.9843137 , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -0.85882354,  0.3411765 ,  0.7176471 ,
        0.9843137 ,  0.9843137 ,  0.9843137 ,  0.9843137 ,  0.5294118 ,
       -0.372549  , -0.92941177, -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -0.5686275 ,  0.34901965,
        0.77254903,  0.9843137 ,  0.9843137 ,  0.9843137 ,  0.9843137 ,
        0.9137255 ,  0.04313731, -0.9137255 , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
       -1.        , -1.        , -1.        , -1.        , -1.        ,
```

```
        -1.        ,  0.06666672,  0.9843137 ,  0.9843137 ,  0.9843137 ,
         0.6627451 ,  0.05882359,  0.03529418, -0.8745098 , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        ],dtype=float32), 5)
```

Step2 网络配置

（1）定义多层感知器

以下的代码判断就是定义一个简单的多层感知器，一共有三层，两个大小为100的隐藏层和一个大小为10的输出层，因为MNIST数据集是手写0到9的灰度图像，类别有10个，所以最后的输出大小是10。最后输出层的激活函数是Softmax，所以最后的输出层相当于一个分类器。加上一个输入层的话，多层感知器的结构是：输入层->隐藏层->隐藏层->输出层。



In [3]

```
# 定义多层感知器
def multilayer_perceptron(input):
    # 第一个全连接层，激活函数为ReLU
    hidden1 = fluid.layers.fc(input=input, size=100, act='relu')
    # 第二个全连接层，激活函数为ReLU
    hidden2 = fluid.layers.fc(input=hidden1, size=100, act='relu')
    # 以softmax为激活函数的全连接输出层，输出层的大小必须为数字的个数10
    prediction = fluid.layers.fc(input=hidden2, size=10, act='softmax')
    return prediction
```

（2）定义数据层

输入的是图像数据。图像是 28 * 28 的灰度图，所以输入的形状是[1, 28, 28]，如果图像是32*32的彩色图，那么输入的形状是[3, 32, 32]，因为灰度图只有一个通道，而彩色图有RGB三个通道。

In [4]

```
# 输入的原始图像数据，大小为1*28*28
image = fluid.layers.data(name='image', shape=[1, 28, 28], dtype='float32')#单通道，28*28像素值
# 标签，名称为label,对应输入图片的类别标签
label = fluid.layers.data(name='label', shape=[1], dtype='int64')          #图片标签
```

（3）获取分类器

In [5]

```
# 获取分类器
predict = multilayer_perceptron(image)
```

（4）定义损失函数和准确率

这次使用的是交叉熵损失函数，该函数在分类任务上比较常用。

定义了一个损失函数之后，还有对它求平均值，训练程序必须返回平均损失作为第一个返回值，因为它会被后面反向传播算法所用到。

同时我们还可以定义一个准确率函数，这个可以在我们训练的时候输出分类的准确率。

In [6]

```
#使用交叉熵损失函数,描述真实样本标签和预测概率之间的差值
cost = fluid.layers.cross_entropy(input=predict, label=label)
# 使用类交叉熵函数计算predict和label之间的损失函数
avg_cost = fluid.layers.mean(cost)
# 计算分类准确率
acc = fluid.layers.accuracy(input=predict, label=label)
```

（5）定义优化函数

这次我们使用的是Adam优化方法，同时指定学习率为0.001

In [7]

```
 #使用Adam算法进行优化，learning_rate 是学习率(它的大小与网络的训练收敛速度有关系)
optimizer = fluid.optimizer.AdamOptimizer(learning_rate=0.001)
opts = optimizer.minimize(avg_cost)
```

在上述模型配置完毕后，得到两个fluid.Program：fluid.default_startup_program() 与 fluid.default_main_program() 配置完毕了。

参数初始化操作会被写入fluid.default_startup_program()

fluid.default_main_program()用于获取默认或全局main program(主程序)。该主程序用于训练和测试模型。fluid.layers 中的所有layer函数可以向 default_main_program 中添加算子和变量。default_main_program 是fluid的许多编程接口（API）的Program参数的缺省值。例如,当用户program没有传入的时候， Executor.run() 会默认执行 default_main_program 。

Step3 模型训练 & Step4 模型评估

（1）创建训练的Executor

首先定义运算场所 fluid.CPUPlace()和 fluid.CUDAPlace(0)分别表示运算场所为CPU和GPU

Executor:接收传入的program，通过run()方法运行program。

In [8]

```
# 定义使用CPU还是GPU，使用CPU时use_cuda = False,使用GPU时use_cuda = True
use_cuda = False
place = fluid.CUDAPlace(0) if use_cuda else fluid.CPUPlace()
# 获取测试程序
test_program = fluid.default_main_program().clone(for_test=True)
exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())
[]
```

（2）告知网络传入的数据分为两部分，第一部分是image值，第二部分是label值

DataFeeder负责将数据提供器（train_reader,test_reader）返回的数据转成一种特殊的数据结构，使其可以输入到Executor中。

In [9]

```
feeder = fluid.DataFeeder(place=place, feed_list=[image, label])
```

(3)展示模型训练曲线

In [10]

```
all_train_iter=0
all_train_iters=[]
all_train_costs=[]
all_train_accs=[]

def draw_train_process(title,iters,costs,accs,label_cost,lable_acc):
    plt.title(title, fontsize=24)
    plt.xlabel("iter", fontsize=20)
    plt.ylabel("cost/acc", fontsize=20)
    plt.plot(iters, costs,color='red',label=label_cost)
```

```
    plt.plot(iters, accs,color='green',label=lable_acc)
    plt.legend()
    plt.grid()
    plt.show()
```

（4）训练并保存模型

训练需要有一个训练程序和一些必要参数，并构建了一个获取训练过程中测试误差的函数。必要参数有executor,program,reader,feeder,fetch_list。

executor表示之前创建的执行器

program表示执行器所执行的program，是之前创建的program，如果该项参数没有给定的话则默认使用defalut_main_program

reader表示读取到的数据

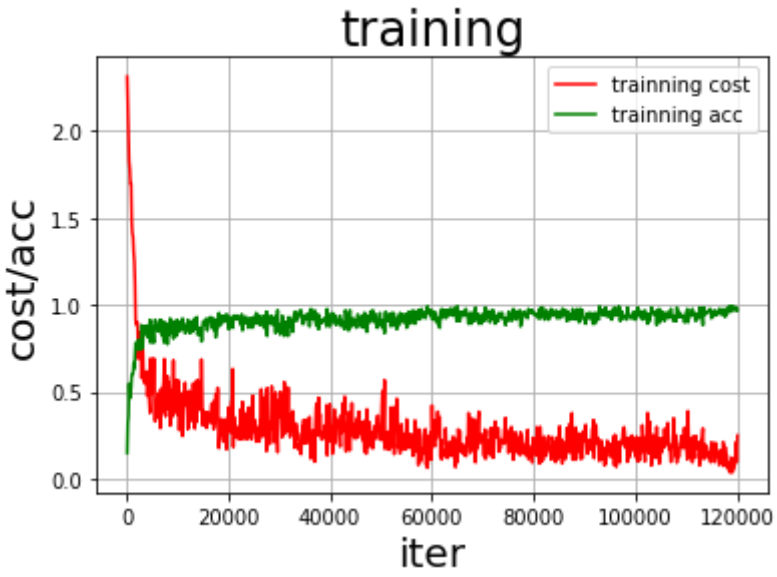
feeder表示前向输入的变量

fetch_list表示用户想得到的变量

In [11]

```
EPOCH_NUM=2
model_save_dir = "/home/aistudio/work/hand.inference.model"
for pass_id in range(EPOCH_NUM):
    # 进行训练
    for batch_id, data in enumerate(train_reader()):                #遍历train_reader
        train_cost, train_acc = exe.run(program=fluid.default_main_program(),#运行主程序
                                        feed=feeder.feed(data),             #给模型喂入数据
                                        fetch_list=[avg_cost, acc])
 #fetch 误差、准确率

        all_train_iter=all_train_iter+BATCH_SIZE
        all_train_iters.append(all_train_iter)

        all_train_costs.append(train_cost[0])
        all_train_accs.append(train_acc[0])

        # 每200个batch打印一次信息  误差、准确率
        if batch_id % 200 == 0:
            print('Pass:%d, Batch:%d, Cost:%0.5f, Accuracy:%0.5f' %
                  (pass_id, batch_id, train_cost[0], train_acc[0]))

    # 进行测试
    test_accs = []
    test_costs = []
    #每训练一轮 进行一次测试
    for batch_id, data in enumerate(test_reader()):                 #遍历test_reader
        test_cost, test_acc = exe.run(program=test_program, #执行训练程序
                                      feed=feeder.feed(data),               #喂入数据
```

```
                                    fetch_list=[avg_cost, acc])
#fetch 误差、准确率
        test_accs.append(test_acc[0])                                   #每个
batch的准确率
        test_costs.append(test_cost[0])                                 #每个
batch的误差


    # 求测试结果的平均值
    test_cost = (sum(test_costs) / len(test_costs))                     #每轮
的平均误差
    test_acc = (sum(test_accs) / len(test_accs))                        #每轮
的平均准确率
    print('Test:%d, Cost:%0.5f, Accuracy:%0.5f' % (pass_id, test_cost,
test_acc))

    #保存模型
    # 如果保存路径不存在就创建
if not os.path.exists(model_save_dir):
    os.makedirs(model_save_dir)
print ('save models to %s' % (model_save_dir))
fluid.io.save_inference_model(model_save_dir,    #保存推理model的路径
                              ['image'],    #推理（inference）需要 feed 的数据
                              [predict],    #保存推理（inference）结果的
Variables
                              exe)             #executor 保存 inference model

print('训练模型保存完成！')
draw_train_process("training",all_train_iters,all_train_costs,all_train_accs,"tr
ainning cost","trainning acc")
Pass:0, Batch:0, Cost:2.31112, Accuracy:0.14844
Pass:0, Batch:200, Cost:0.24132, Accuracy:0.92969
Pass:0, Batch:400, Cost:0.26719, Accuracy:0.92188
Test:0, Cost:0.22510, Accuracy:0.93018
Pass:1, Batch:0, Cost:0.24731, Accuracy:0.92188
Pass:1, Batch:200, Cost:0.12370, Accuracy:0.97656
Pass:1, Batch:400, Cost:0.24292, Accuracy:0.93750
Test:1, Cost:0.15191, Accuracy:0.95263
save models to /home/aistudio/work/hand.inference.model
训练模型保存完成！
```



Step5 模型预测

（1）图片预处理

在预测之前，要对图像进行预处理。

首先进行灰度化，然后压缩图像大小为28*28，接着将图像转换成一维向量，最后再对一维向量进行归一化处理。

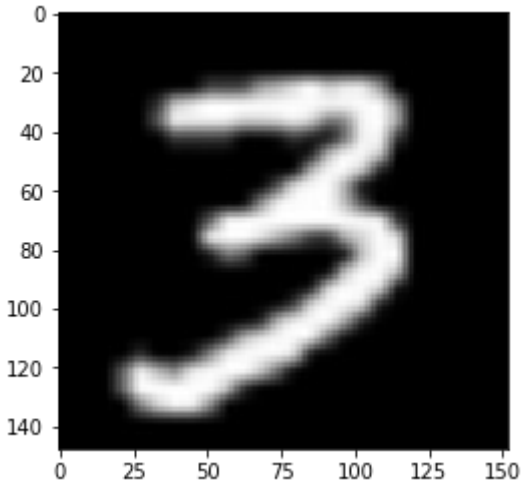
In [12]

```
def load_image(file):
    im = Image.open(file).convert('L')                                  #将RGB转化为灰度图像，L代表灰度图像，像素值在0~255之间
    im = im.resize((28, 28), Image.ANTIALIAS)                           #resize image with high-quality 图像大小为28*28
    im = np.array(im).reshape(1, 1, 28, 28).astype(np.float32)#返回新形状的数组,把它变成一个 numpy 数组以匹配数据馈送格式。
    # print(im)
    im = im / 255.0 * 2.0 - 1.0                                         #归一化到【-1~1】之间
    return im
```

（2）使用Matplotlib工具显示这张图像。

In [13]

```
infer_path='/home/aistudio/data/data1910/infer_3.png'
img = Image.open(infer_path)
plt.imshow(img)    #根据数组绘制图像
plt.show()         #显示图像
```



(3)创建预测用的Executer

In [14]

```
infer_exe = fluid.Executor(place)
inference_scope = fluid.core.Scope()
```

(4)开始预测

通过fluid.io.load_inference_model，预测器会从params_dirname中读取已经训练好的模型，来对从未遇见过的数据进行预测。

In [15]

```
# 加载数据并开始预测
with fluid.scope_guard(inference_scope):
    #获取训练好的模型
    #从指定目录中加载 推理model(inference model)
    [inference_program,                                             #推理Program
     feed_target_names,                                             #是一个str列表，
它包含需要在推理 Program 中提供数据的变量的名称。
     fetch_targets] =
fluid.io.load_inference_model(model_save_dir,#fetch_targets：是一个 Variable 列表，
从中我们可以得到推断结果。model_save_dir：模型保存的路径
                                                  infer_exe)     #infer_exe: 运
行 inference model的 executor
    img = load_image(infer_path)

    results = infer_exe.run(program=inference_program,                #运行推测程序
                  feed={feed_target_names[0]: img},           #喂入要预测的img
                  fetch_list=fetch_targets)                   #得到推测结果，
    # 获取概率最大的label
    lab = np.argsort(results)                                 #argsort函数返回的是
result数组值从小到大的索引值
    #print(lab)
    print("该图片的预测结果的label为: %d" % lab[0][0][-1])     #-1代表读取数组中倒数第一
列
```

```
该图片的预测结果的label为: 3
```

六、算法原理

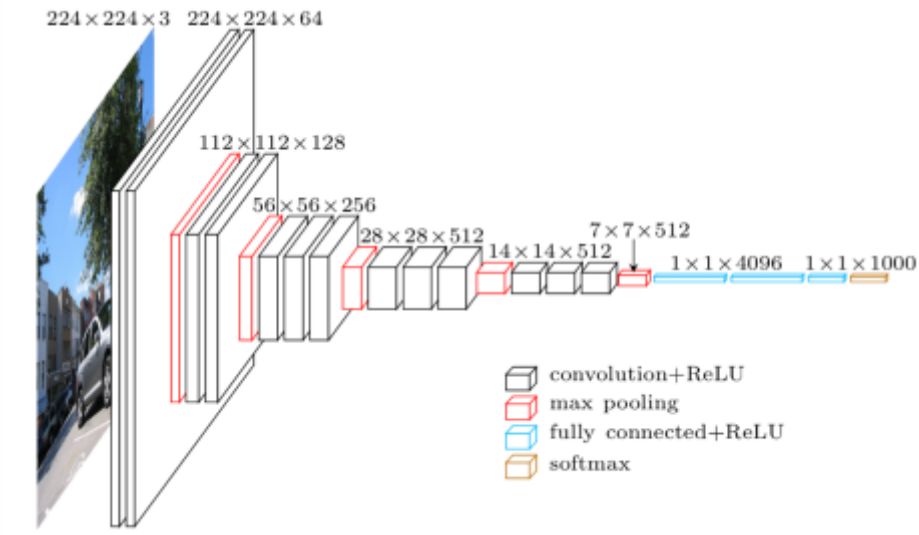
深度学习原理

1. 对神经网络的权重随机赋值，由于是对输入数据进行随机的变换，因此跟预期值可能差距很大，相应地，损失值也很高；
2. 根据损失值，利用反向传播算法来微调神经网络每层的参数，从而降低损失值；
3. 根据调整的参数继续计算预测值，并计算预测值和预期值的差距，即损失值；
4. 重复步骤2,3，直到整个网络的损失值达到最小，即算法收敛。

卷积神经网络实现原理

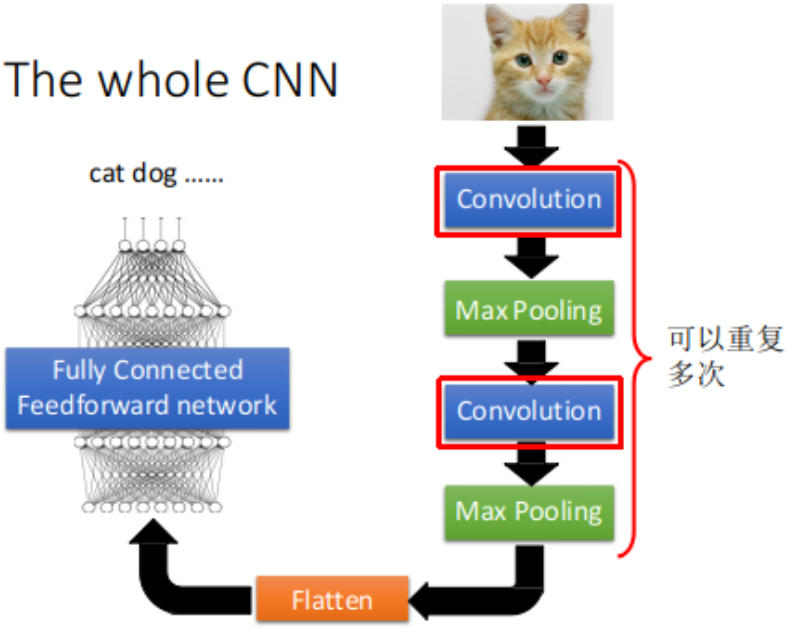
(1) CNN原理

卷积神经网络（CNN）主要是用于图像识别领域，它指的是一类网络，而不是某一种，其包含很多种不同结构的网络。所有CNN最终都是把一张图片转化为特征向量，特征向量就相当于这张图片的DNA。就像VGG网络一样，通过多层的卷积，池化，全连接，降低图片维度，最后转化成了一个一维向量。这个向量就包含了图片的特征（这个特征不是肉眼上的图片特征，而是针对于神经网络的特征）。



之所以用VGG举例，因为它的网络结构非常简洁，清晰，相当好理解，简单介绍一下：

它的输入是一张224x224 的三通道图片，经过两层卷积之后，图片维度不变，通道数增加到了64。
之后那个红色的层是最大池化（max pooling），把图片维度变成了112x112。后续就是不断重复步骤1，2。
当变成1维向量之后，经过全连接（fully connected）加ReLU激活，softmax处理之后，变成了一个包含1000个数字的特征向量。
CNN的工作步骤：卷积，池化，全连接，降低图片维度



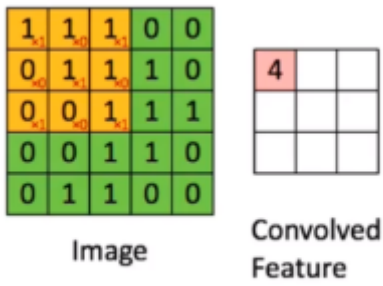
(2) CNN如何训练

1. 卷积神经网络的前向传播过程

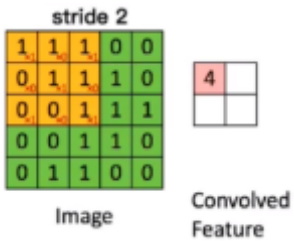
在前向传播过程中，输入的图形数据经过多层卷积层的卷积和池化处理，提出特征向量，将特征向量传入全连接层中，得出分类识别的结果。当输出的结果与我们的期望值相符时，输出结果。

1.1 前向传播中的卷积操作

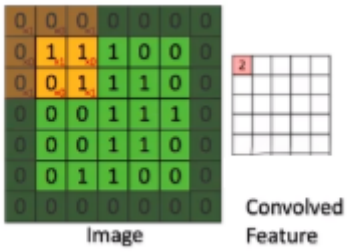
- 用一个小的权重矩阵去覆盖输入数据，对应位置加权相乘，其和作为结果的一个像素点;
- 这个权重在输入数据上滑动，形成一张新的矩阵:



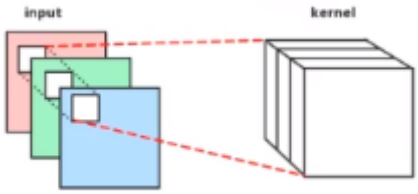
- 这个权重矩阵称为卷积核（convolution kernel）；
- 其覆盖位置称为感受野（receptive field）；
- 参数共享;
- 滑动的像素数量叫做步长（stride）：



- 以卷积核的边还是中心点作为开始/结束的依据，决定了卷积的补齐（padding）方式。上面的图片是valid方式（这种方式新的矩阵维度可能会降低），而same方式则会在图像边缘用0补齐（这种方式图像维度不会降低）：



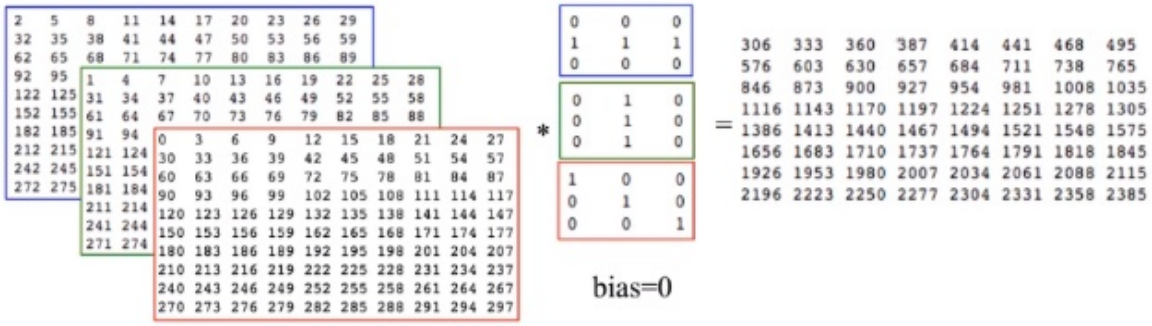
- 如果输入通道不只一个，那么卷积核是三阶的。所有通道的结果累加：



卷积核将某个局部感受野生成一个数值（新的feature map中的一个像素）

$$output_{ij} = \sigma(b + \Sigma_m \Sigma_n \Sigma_d w_{m,n,d} x_{i+m,j+n,d})$$

$$output = X * W + b$$



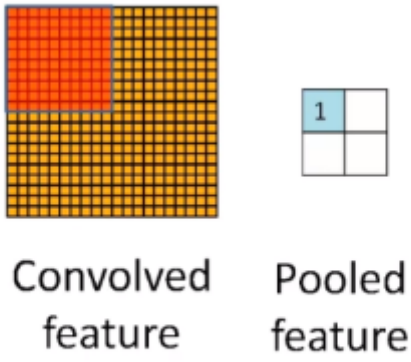
1.2 前向传播中的池化操作

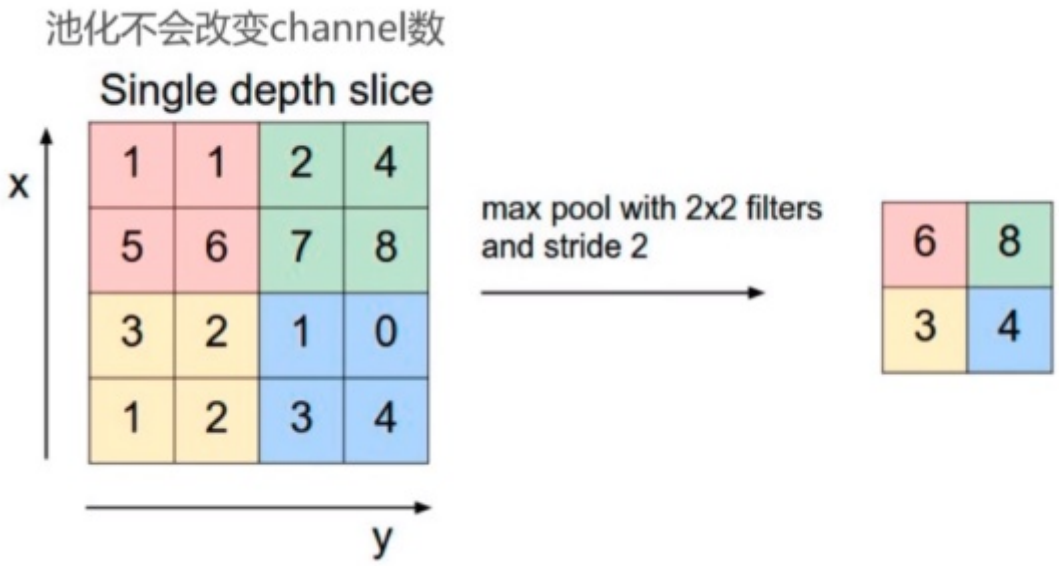
池化又称为降采样（down_sampling），类型：

- 最大池化（max pooling）：在感受野内取最大值输出；
- 平均池化（average pooling）：在感受野内取平均值进行输出；
- 其他如L2池化等。

理解：

- 一个选择框，将输入数据某个范围（矩阵）的所有数值进行相应计算，得到一个新的值，作为结果的一个像素点；
- 池化也有步长和补齐的概念，但是很少使用，通常选择框以不重叠的方式，在padding=0的输入数据上滑动，生成一张新的特征图：





1.3 前向传播中的全连接

特征图进过卷积层和下采样层的特征提取之后，将提取出来的特征传到全连接层中，通过全连接层，进行分类，获得分类模型，得到最后的结果。

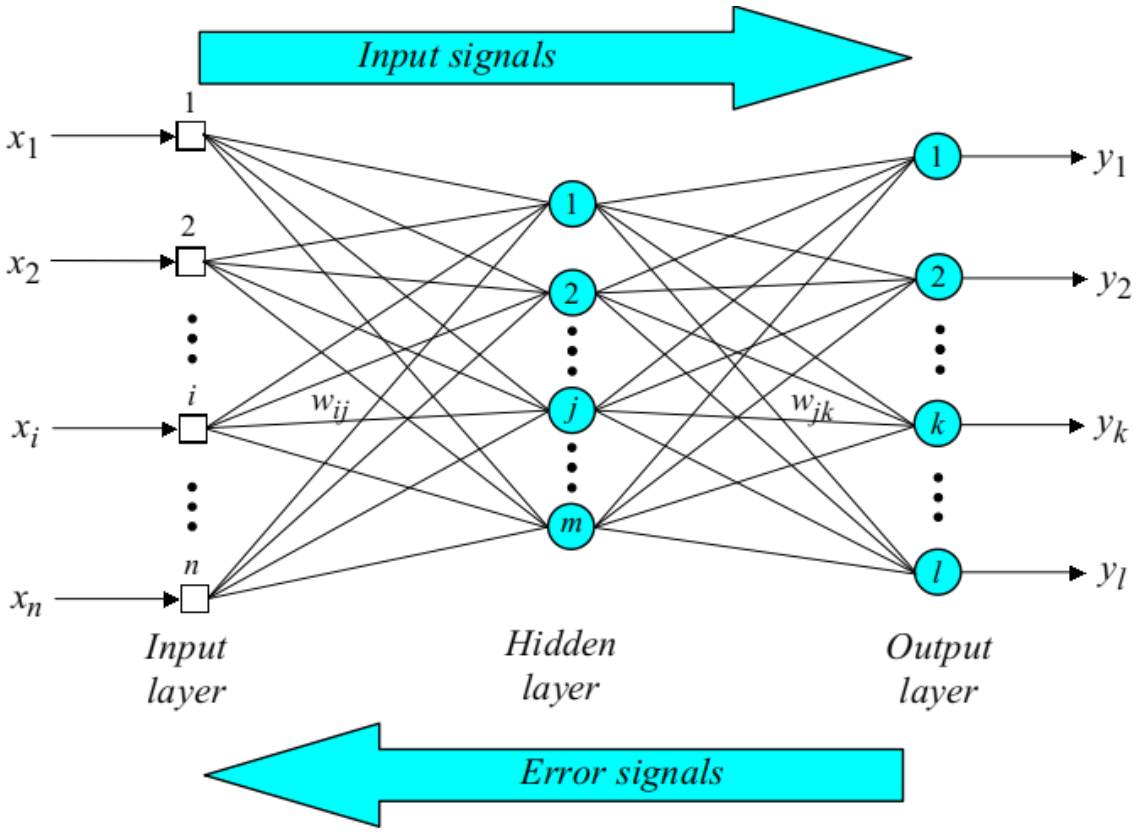
2. 卷积神经网络的反向传播过程

当卷积神经网络输出的结果与我们的期望值不相符时，则进行反向传播过程。求出结果与期望值的误差，再将误差一层一层的返回，计算出每一层的误差，然后进行权值更新。

3. 卷积神经网络的权值更新

卷积层的误差更新过程为：将误差矩阵当做卷积核，卷积输入的特征图，并得到了权值的偏差矩阵，然后与原先的卷积核的权值相加，并得到了更新后的卷积核。

2和3两个部分和神经网络是一致的。



(3) back propagation(反向传播)

算法思想：

训练数据输入到神经网络中，神经网络计算真实输出与实际输出之间的差异，然后通过调节权值来减小这种差异

两阶段:

- 训练数据输入，通过前向层层计算到输出层输出结果；
- 计算真实输出与实际输出之间的错误，通过反向从输出层-隐藏层-输入层的调节权值来减少错误。

算法步骤：

- 第一步，初始化

  设定初始的权值w1,w2,...,wn和阈值θ为如下的一致分布中的随机数

  $$\left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i}\right)$$

  Fi是输入神经元数量总和

- 第二步：计算激活函数值

  根据输入x1§, x2§,..., xn§ 和权值w1,w2,...,wn计算输出y1§, y2§,..., yn§

  - （a）计算隐藏层神经元的输出

  $$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \cdot w_{ij}(p) - \theta_j\right]$$

n是第j个隐藏层神经元的输入数量，sigmoid是激活函数

- ◦ （b）计算输出层神经元的输出

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p)\cdot w_{jk}(p) - \theta_k\right]$$

m是第k个输出神经元的输入数量

- 第三步：权值更新(从后往前)
  - （a）计算输出层的错误梯度

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

计算权值纠正值

$$\delta_k(p) = y_k(p)\cdot\left[1 - y_k(p)\right]\cdot e_k(p)$$

更新输出层的权值

$$\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)$$

- ◦ （b）计算隐藏层的错误梯度

$$\delta_j(p) = y_j(p)\cdot\left[1 - y_j(p)\right]\cdot\sum_{k=1}^{l}\delta_k(p)\,w_{jk}(p)$$

计算权值纠正值

$$\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p)$$

更新隐藏层的权值

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

- 第四步：迭代循环

  增加p值，不断重复步骤二和步骤三直到收敛。

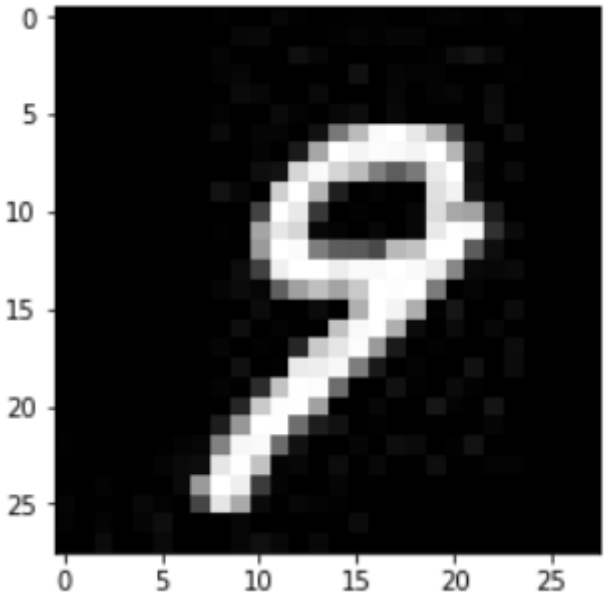
七、实验代码

篇幅所限，在此贴出源码链接。

源码地址

https://gitee.com/gunshi3/artificial-intelligence/tree/master/lab4_code

运行结果




```
# 加载数据并开始预测
with fluid.scope_guard(inference_scope):
    #获取训练好的模型
    #从指定目录中加载 推理model(inference model)
    [inference_program,                                         #推理Program
     feed_target_names,                                         #是一个str列表，它包含需要在推理 Program 中提供数据的变量的名称。
     fetch_targets] = fluid.io.load_inference_model(model_save_dir,#fetch_targets：是一个 Variable 列表，从中我们可以得到推断结果。model_save_dir：模型保
                                                    infer_exe)     #infer_exe: 运行 inference model的 executor
    img = load_image(infer_path)

    results = infer_exe.run(program=inference_program,              #运行推测程序
                  feed={feed_target_names[0]: img},            #喂入要预测的img
                  fetch_list=fetch_targets)                    #得到推测结果,
    # 获取概率最大的label
    lab = np.argsort(results)                                   #argsort函数返回的是result数组值从小到大的索引值
    #print(lab)
    print("该图片的预测结果的label为：%d" % lab[0][0][-1])     #-1代表读取数组中倒数第一列
```

运行时长: 86毫秒　结束时间: 2021-12-23 17:16:07

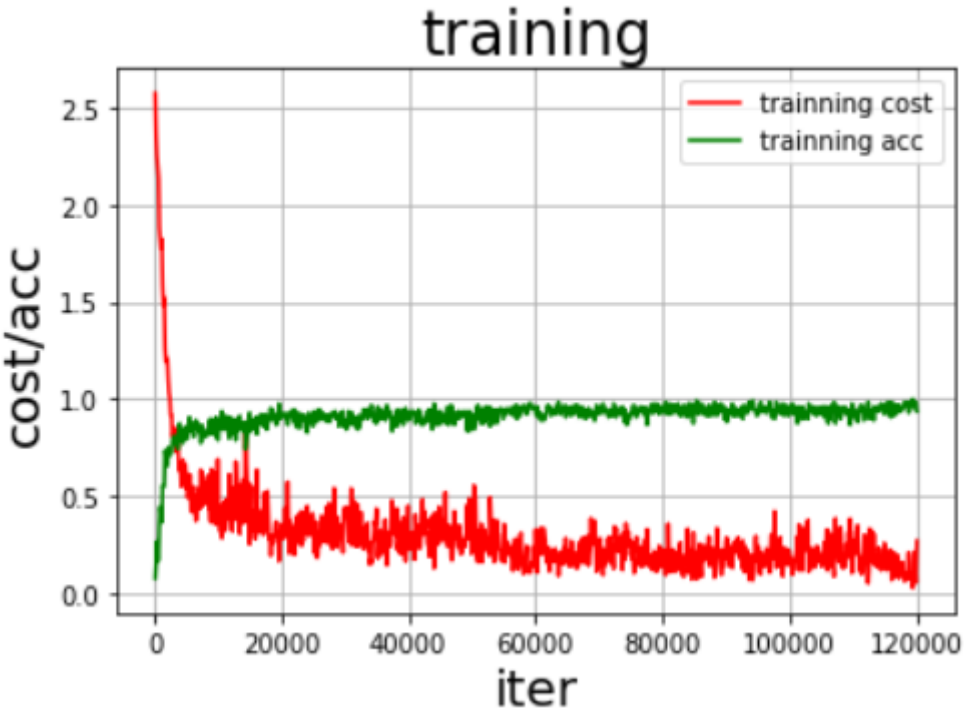
```
该图片的预测结果的label为： 9
```

八、实验结果与分析

1. 模型评估

我们的模型训练效果如何？训练是否收敛？对此，我要求每200个batch打印一次信息（误差、准确率），使用plt对训练准确率train_accs，训练损失train_cost，以及迭代次数train_iters进行绘图，观察绘制结果，反映模型的训练过程。

下面是训练过程：



可以看到整个数据的训练过程中训练准确率train_accs逐渐上升，最终趋近100%，训练损失train_cost逐渐下降，最终趋近0。整体上训练效果还是比较可观的。

2. 预测测试集

为了观察训练结果如何，我对测试集进行了预测，统计了预测结果准确率。

```
Predicted:  8 3 2 7 7 7 8 6 3 9 3 5 1 0 1 8 3 5 5 3 8 9 4 2 7 8 4 1 3 4 6 3 5 4
6 0 4 3 1 1 2 3 2 0 0 7 4 5 8 4 2 9 8 1 5 2 7 2 0 3 7 2 8 9
Accuracy of the network on the  test images: 98.850000 %
Accuracy of 0 : 100.000000 %
Accuracy of 1 : 99.393939 %
Accuracy of 2 : 98.039216 %
Accuracy of 3 : 99.386503 %
Accuracy of 4 : 98.734177 %
Accuracy of 5 : 99.358974 %
Accuracy of 6 : 98.726115 %
Accuracy of 7 : 99.295775 %
Accuracy of 8 : 98.181818 %
Accuracy of 9 : 97.530864 %
```

观察结果发现，0～9共10个数字的预测准确率都在98%左右，其中0的预测精度达到了100%，9的预测结果稍差，在97.5%左右。

九、思考题

深度算法参数的设置对算法性能的影响?

本实验中的CNN的参数设置有如下这些:

- 卷积层: nn.Conv2d

参数·	含义
in_channels	输入信号的通道数.
out_channels	卷积后输出结果的通道数.
kernel_size	卷积核的形状. 例如kernel_size=(3, 2)表示3X2的卷积核，如果宽和高相同，可以只用一个数字表示
stride	卷积每次移动的步长, 默认为1.

参数·	含义
padding	处理边界时填充0的数量, 默认为0(不填充).
dilation	采样间隔数量, 默认为1, 无间隔采样.
groups	输入与输出通道的分组数量. 当不为1时, 默认为1(全连接).
bias	为 True 时, 添加偏置.

- 池化层: nn.MaxPool2d()

参数	含义
kernel_size	最大池化操作时的窗口大小
stride	最大池化操作时窗口移动的步长, 默认值是 kernel_size
padding	输入的每条边隐式补0的数量
dilation	用于控制窗口中元素的步长的参数
return_indices	如果等于 True, 在返回 max pooling 结果的同时返回最大值的索引 这在之后的 Unpooling 时很有用
ceil_mode	如果等于 True, 在计算输出大小时,将采用向上取整来代替默认的向下取整的方式

上面的参数我们不一定需要全部设置，一般我们需要调整的参数是：

CNN的各部分层数，卷积核的形状，卷积核的数量，卷积层的步长以及Padding，池化层的选择，激活函数的选择。

- CNN层数：层数越多，模型的训练效果越好，但是训练时间也会相应增加，所以要结合实际进行设置。
- 卷积核的形状kernel_size：决定了其提取的特征的数目，需要进行调参根据经验才能确定，当前还没有一个确定的算法计算。
- 卷积层的步长和Padding：Padding = 0会使得边缘的特征提取不完全；为了防止过拟合，可以加入Dropout层，将小于阈值的值赋值为0，进行丢弃。除此之外，还可以对优化器的学习率进行修改，较小的学习率会使收敛速率较慢，但是较大的学习率又会导致发散和欠拟合。
- 优化方法选择：本次实验中选择的是Adam优化方法，我们也可以选择 SGD 随机梯度下降优化方法，优化方法的选择也需要根据实际情况来调整。

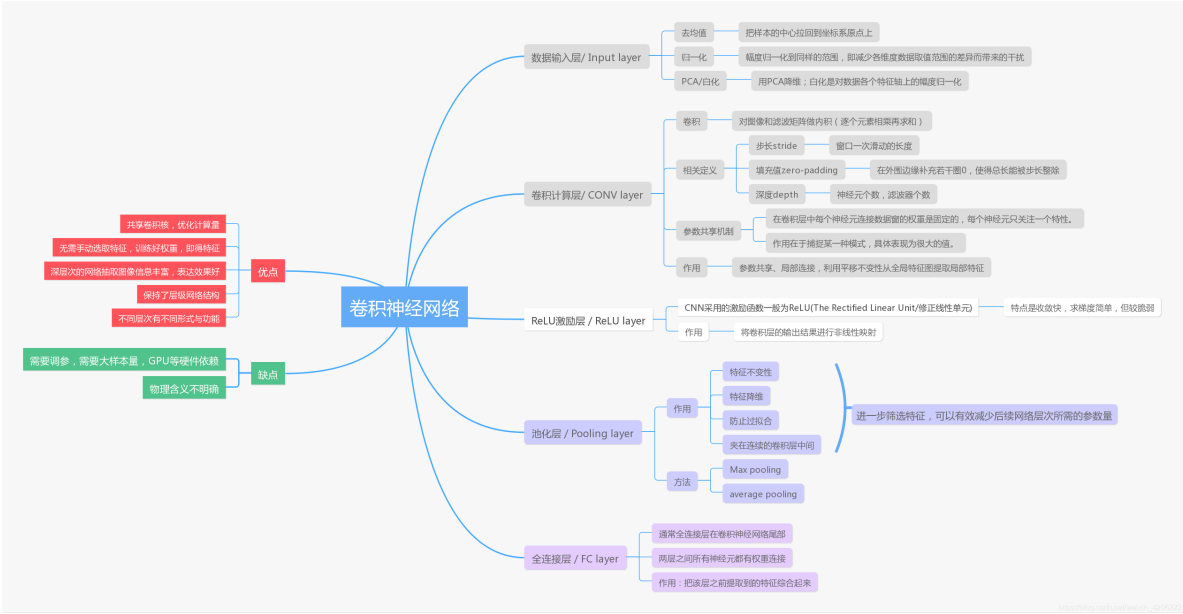
十、实验总结

实验收获

通过这次实验，我了解到了深度学习的基本原理，更是从实验中学习到了深度学习的基本流程、基本的调参方法，通过本次实验，我的收获比较多。本次实验侧重于对深度学习算法的实际应用，对于内在原理，要在日后的学习中逐渐理解掌握。

难点重点讨论

卷积神经网络（Convolutional Neural Network,CNN）是一类包含卷积计算且具有深度结构的前馈神经网络。



CNN整体架构：卷积神经网络是一种多层的监督学习神经网络，隐含层的卷积层和池采样层是实现卷积神经网络特征提取功能的核心模块。该网络模型通过采用梯度下降法最小化损失函数对网络中的权重参数逐层反向调节，通过频繁的迭代训练提高网络的精度。卷积神经网络的低隐层是由卷积层和最大池采样层交替组成，高层是全连接层对应传统多层感知器的隐含层和逻辑回归分类器。第一个全连接层的输入是由卷积层和子采样层进行特征提取得到的特征图像。最后一层输出层是一个分类器，可以采用逻辑回归。

卷积神经网络CNN主要用来识别位移、缩放及其他形式扭曲不变性的二维图形。由于CNN的特征检测层通过训练数据进行学习，所以在使用CNN时，避免了显式的特征抽取，而隐式地从训练数据中进行学习；再者由于同一特征映射面上的神经元权值相同，所以网络可以并行学习，这也是卷积网络相对于神经元彼此相连网络的一大优势。卷积神经网络以其局部权值共享的特殊结构在语音识别和图像处理方面有着独特的优越性，其布局更接近于实际的生物神经网络，权值共享降低了网络的复杂性，特别是多维输入向量的图像可以直接输入网络这一特点避免了特征提取和分类过程中数据重建的复杂度。

十一、参考地址

- 卷积神经网络（CNN）详解
- 课程4-深度学习入门CV-手写数字识别
- VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION
- CSDN裕东方