

华为认证 AI 系列教程

HCIA-AI

# 机器学习实验指导手册

版本：4.0



华为技术有限公司

**版权所有 © 华为技术有限公司 2025。 保留一切权利。**

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## **商标声明**



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## **注意**

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

**华为技术有限公司**

地址：            深圳市龙岗区坂田华为总部办公楼            邮编： 518129

网址：            <https://e.huawei.com>

---

## 华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT技术架构与应用认证、云服务与平台认证两类认证。

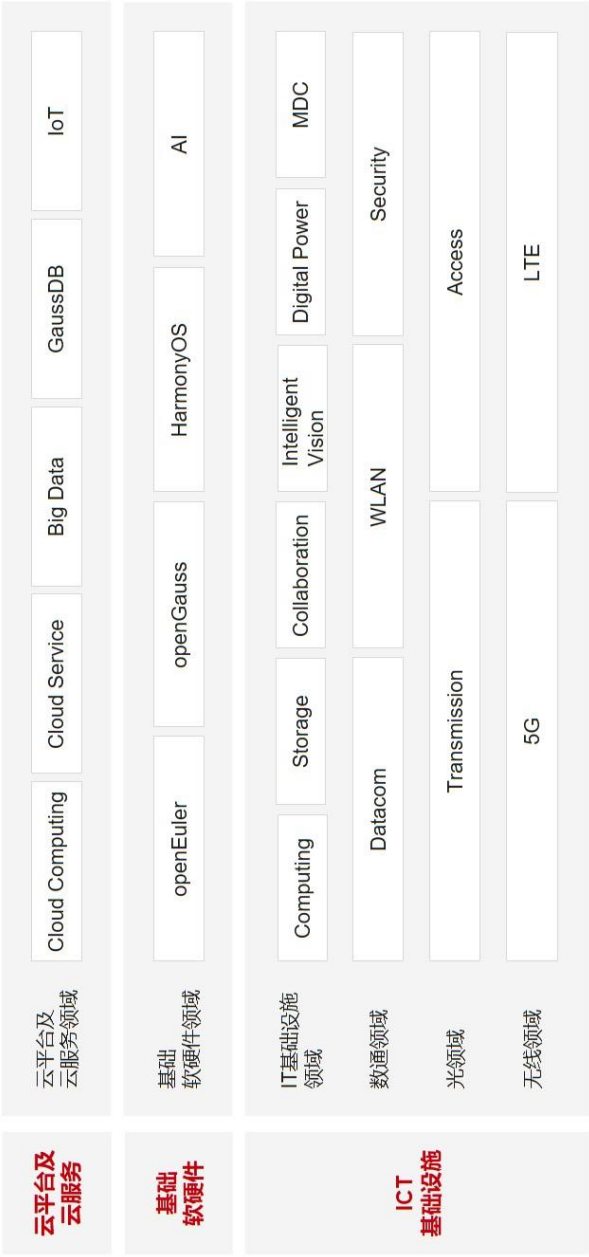
根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIA-AI V4.0 定位于培养和认证具备人工智能基础知识，掌握机器学习、深度学习和大模型等基础算法，可使用AI开发框架进行人工智能产品开发和创新的工程师。

通过HCIA-AI V4.0 认证，将证明您了解人工智能发展历史和前沿应用场景，掌握传统机器学习、深度学习和大模型的相关基础算法；具备利用AI开发框架搭建、训练和部署神经网络的能力；能够胜任人工智能开发工程师和人工智能技术支持等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在人工智能世界的潮头浪尖！



Project Management



Huawei Certified ICT Expert



Huawei Certified ICT Professional



Huawei Certified ICT Associate

## 简介

本书为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-AI 考试的学员或者希望了解机器学习基础知识的读者。

## 内容描述

本实验指导书共包含 6 个小的实验。

- 实验一线性回归，通过 Python 工具包 scikit-learn 实现简单线性回归算法的使用。
- 实验二为线性回归拓展实验，通过 Python 基础工具包 numpy 等从零实现线性回归、梯度下降等算法，加深理解。
- 实验三逻辑回归，通过工具包中的逻辑回归算法实现简单的分类任务。
- 实验四为决策树算法的使用，通过构建决策树实现天气预测，并可视化决策树算法加深理解。
- 实验五为垃圾邮件分类，通过朴素贝叶斯算法实现垃圾邮件分类。
- 实验六为聚类算法 K-means 的使用。

## 读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有 Python 编程基础知识；
- 具备线性代数、概率论等基础数学知识。

## 实验环境说明

本实验环境为 Python3.7（相关环境搭建，请参考环境搭建指南）。

实验数据链接：<https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V4.0/chapter4/ML.zip>

---

# 目录

---

|                     |          |
|---------------------|----------|
| <b>前 言</b>          | <b>3</b> |
| 简介                  | 3        |
| 内容描述                | 3        |
| 读者知识背景              | 3        |
| 实验环境说明              | 3        |
| <b>1 常见机器学习算法实现</b> | <b>5</b> |
| 1.1 实验介绍            | 5        |
| 1.1.1 关于本实验         | 5        |
| 1.1.2 实验目的          | 5        |
| 1.2 代码实现            | 5        |
| 1.2.1 线性回归          | 5        |
| 1.2.2 线性回归算法实现（扩展）  | 7        |
| 1.2.3 逻辑回归          | 10       |
| 1.2.4 决策树           | 12       |
| 1.2.5 朴素贝叶斯         | 14       |
| 1.2.6 K-means 算法实现  | 17       |
| 1.3 思考题             | 22       |
| 1.4 本章总结            | 22       |

---

# 1 常见机器学习算法实现

---

## 1.1 实验介绍

### 1.1.1 关于本实验

本实验包含了常见机器学习算法的使用，通过从零开始构建线性回归算法和基于 scikit-learn 调用实现决策树、朴素贝叶斯、聚类算法，帮助您更加了解机器学习算法的使用流程和功能。

### 1.1.2 实验目的

- 实现从零开始构建线性回归算法。
- 掌握分类、回归算法的使用。
- 掌握聚类算法的使用。
- 掌握机器学习应用实现流程。

## 1.2 代码实现

### 1.2.1 线性回归

#### 步骤 1 引入相关依赖的包

输入：

```
from sklearn.linear_model import LinearRegression#导入线性回归模型
import matplotlib.pyplot as plt#绘图库
import numpy as np
```

#### 步骤 2 构建房价数据集并可视化

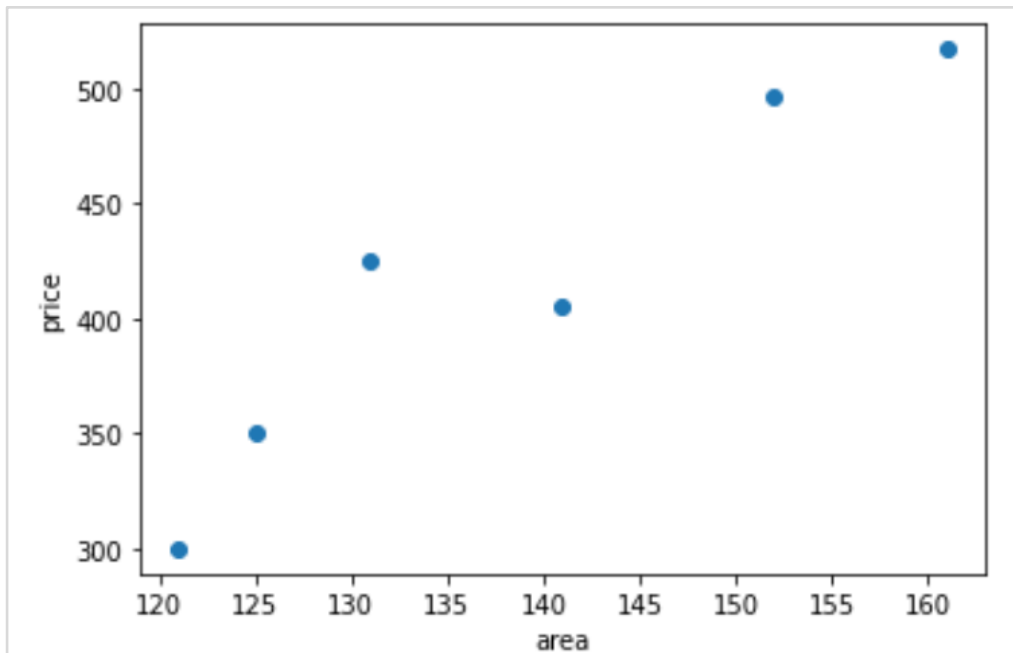
输入：

```
x = np.array([121, 125, 131, 141, 152, 161]).reshape(-1,1)#x 是房屋面积，作为特征
y = np.array([300, 350, 425, 405,496,517])#y 是房屋的
plt.scatter(x,y)
```

---

```
plt.xlabel("area")#添加横坐标面积
plt.ylabel("price")#添加纵坐标价格
plt.show()
```

输出：



### 步骤 3 模型训练

输入：

```
lr = LinearRegression()#将线性回归模型封装为对象
lr.fit(x,y)#模型在数据上训练
```

### 步骤 4 模型的可视化

输入：

```
w = lr.coef_#存储模型的斜率
b = lr.intercept_#存储模型的截距
print('斜率:',w)
print('截距:',b)
```

输出：

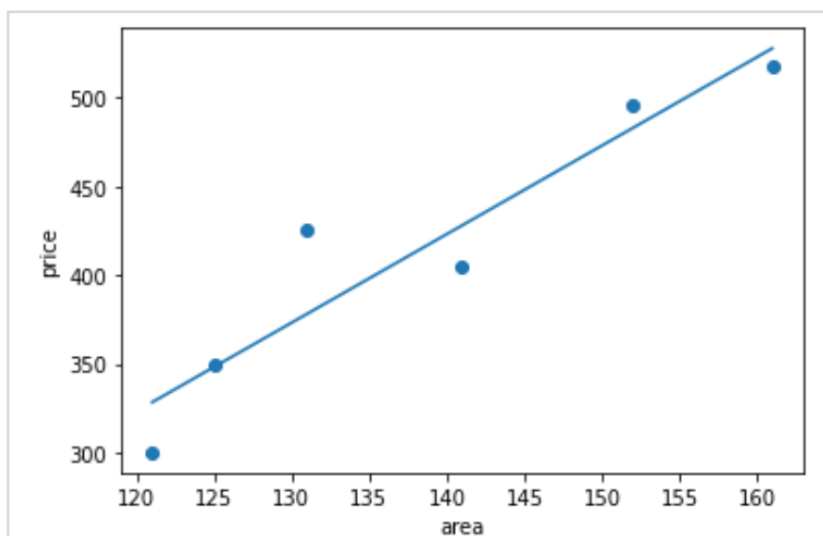
```
斜率: [4.98467124]
截距: -274.8769665187576
```

输入：

```
plt.scatter(x,y)
plt.xlabel("area")#添加横坐标面积
```

```
plt.ylabel("price")#添加纵坐标价格
plt.plot([x[0],x[-1]],x[0]*w+b,x[-1]*w+b])
```

输出：



## 步骤 5 模型预测

输入：

```
testX = np.array([[130]])#测试样本，面积为 130
lr.predict(testX)
```

输出：

```
array([373.13029447])#模型预测获得样本的房价
```

## 1.2.2 线性回归算法实现（扩展）

在“线性回归算法实现”部分，使用的数据集为 lr2\_data.txt，主要是自行模拟的房屋面积与房屋价格数据。该数据集可在实验环境搭建中获取。

### 步骤 1 导入依赖

输入：

```
import numpy as np
import matplotlib.pyplot as plt
```

### 步骤 2 定义函数，计算梯度

输入：

```
def generate_gradient(X, theta, y):
    sample_count = X.shape[0]
    # 计算梯度，采用矩阵计算  $1/m \sum ((h(x^i)-y^i)) x_j^i$ 
```

```
return (1./sample_count)*X.T.dot(X.dot(theta)-y)
```

### 步骤 3 定义函数，用于读取数据集

输入：

```
def get_training_data(file_path):
    orig_data = np.loadtxt(file_path, skiprows=1) #忽略第一行的标题
    cols = orig_data.shape[1]
    return (orig_data, orig_data[:, :cols - 1], orig_data[:, cols-1:])
```

### 步骤 4 定义函数，初始化参数

输入：

```
# 初始化 θ 数组
def init_theta(feature_count):
    return np.ones(feature_count).reshape(feature_count, 1)
```

### 步骤 5 定义函数，实现梯度下降法

输入：

```
def gradient_descending(X, y, theta, alpha):
    Jthetas = [] # 记录代价函数 J(θ)的变化趋势，验证梯度下降是否运行正确
    # 计算损失函数，等于真实值与预测值差的平方。(y^i-h(x^i))^2
    Jtheta = (X.dot(theta)-y).T.dot(X.dot(theta)-y)
    index = 0
    gradient = generate_gradient(X, theta, y) #计算梯度
    while not np.all(np.absolute(gradient) <= 1e-5): #梯度小于 0.00001 时计算结束
        theta = theta - alpha * gradient
        gradient = generate_gradient(X, theta, y) #计算新梯度
        # 计算损失函数，等于真实值与预测值差的平方(y^i-h(x^i))^2
        Jtheta = (X.dot(theta)-y).T.dot(X.dot(theta)-y)
        if (index+1) % 10 == 0:
            Jthetas.append((index, Jtheta[0])) #每 10 次计算记录一次结果
        index += 1
    return theta, Jthetas
```

### 步骤 6 定义函数，可视化损失函数变化曲线

输入：

```
#展示损失函数变化曲线图
def showJTheta(diff_value):
    p_x = []
    p_y = []
    for (index, sum) in diff_value:
        p_x.append(index)
```

```
p_y.append(sum)
plt.plot(p_x, p_y, color='b')
plt.xlabel('steps')
plt.ylabel('loss funtion')
plt.title('step - loss function curve')
plt.show()
```

## 步骤 7 定义函数，可视化数据点及拟合的曲线

输入：

```
#展示实际数据点以及拟合出的曲线图
def showlinercurve(theta, sample_training_set):
    x, y = sample_training_set[:, 1], sample_training_set[:, 2]
    z = theta[0] + theta[1] * x
    plt.scatter(x, y, color='b', marker='x', label="sample data")
    plt.plot(x, z, 'r', color="r", label="regression curve")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('liner regression curve')
    plt.legend()
    plt.show()
```

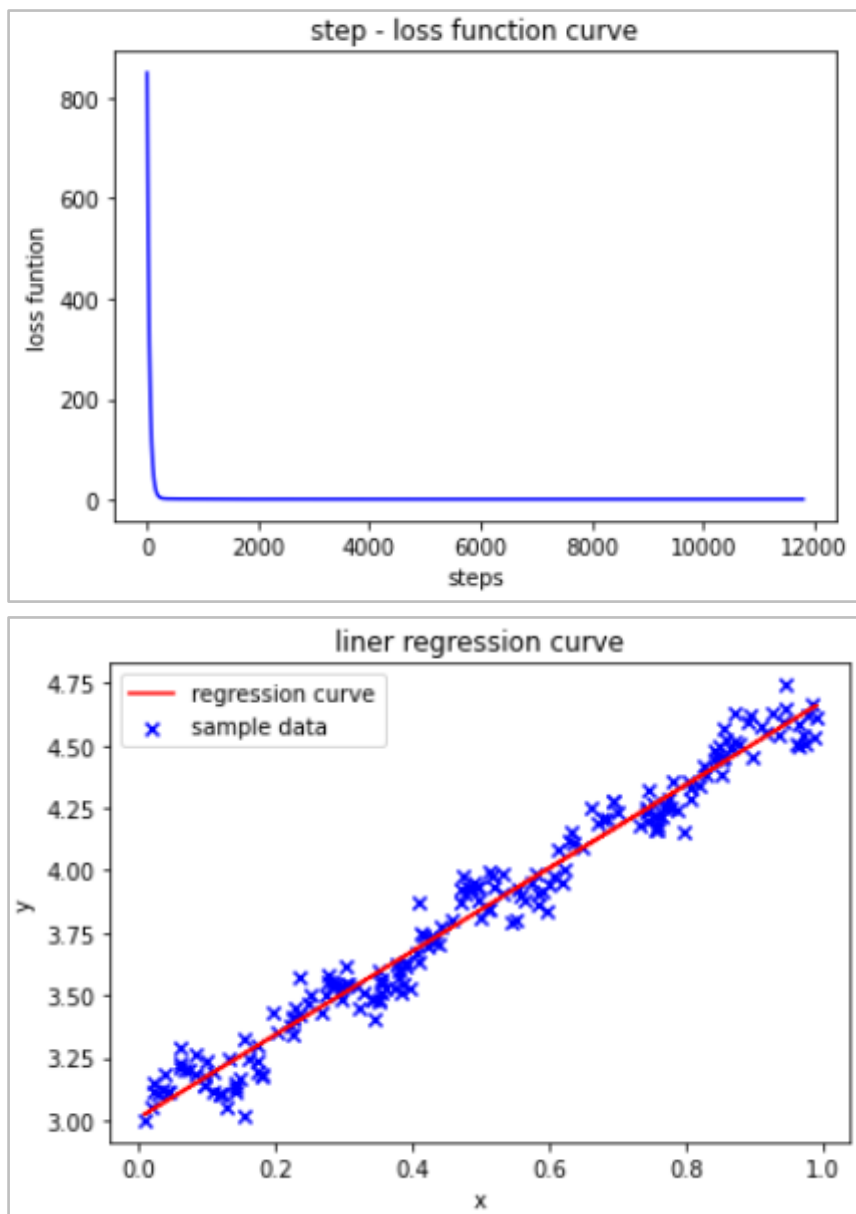
## 步骤 8 最终展示

输入：

```
# 读取数据集
training_data_include_y, training_x, y = get_training_data("./ML/02/Ir2_data.txt")
# 获取数据集数量及特征数
sample_count, feature_count = training_x.shape
# 定义学习步长  $\alpha$ 
alpha = 0.01
# 初始化  $\theta$ 
theta = init_theta(feature_count)
# 获取最终的参数  $\theta$  及代价
result_theta, Jthetas = gradient_descending(training_x, y, theta, alpha)
# 打印参数
print("w:{}".format(result_theta[0][0]), "b:{}".format(result_theta[1][0]))
showJTheta(Jthetas)
showlinercurve(result_theta, training_data_include_y)
```

输出（warning 可以忽略）：

```
w:3.0076279423997594 b:1.668677412281192
```



### 1.2.3 逻辑回归

在逻辑回归部分，使用的数据集为自定义的房屋租金和面积相关的数据集，在实验初始阶段会进行定义。

#### 步骤 1 导入依赖

输入：

```
# 从 sklearn.preprocessing 里导入 StandardScaler。
from sklearn.preprocessing import StandardScaler
# 从 sklearn.linear_model 里导入 LogisticRegression
from sklearn.linear_model import LogisticRegression
```

## 步骤 2 自定义数据集

输入：

```
# x: 每一项表示租金和面积
# y: 表示是否租赁该房间 (0: 不租, 1: 租)
x=[[2200,15],[2750,20],[5000,40],[4000,20],[3300,20],[2000,10],[2500,12],[12000,80],
   [2880,10],[2300,15],[1500,10],[3000,8],[2000,14],[2000,10],[2150,8],[3400,20],
   [5000,20],[4000,10],[3300,15],[2000,12],[2500,14],[10000,100],[3150,10],
   [2950,15],[1500,5],[3000,18],[8000,12],[2220,14],[6000,100],[3050,10]
  ]
y=[1,1,0,0,1,1,1,1,0,1,1,0,1,1,0,1,0,0,0,1,1,1,0,1,0,1,0,1,1,0]
```

## 步骤 3 数据预处理

标准化数据，保证每个维度的特征数据方差为 1，均值为 0。使得预测结果不会被某些维度过大的特征值而主导。

输入：

```
ss = StandardScaler()
X_train = ss.fit_transform(X)
```

查看标准化处理后的数据。

输入：

```
print(X_train)
```

输出：

```
[[-0.60583897 -0.29313058]
 [-0.37682768 -0.09050576]
 [ 0.56003671  0.71999355]
 [ 0.14365254 -0.09050576]
 [-0.14781638 -0.09050576]
 [-0.68911581 -0.49575541]
 [-0.48092372 -0.41470548]
 [ 3.47472592  2.34099218]
 [-0.32269773 -0.49575541]
 [-0.56420055 -0.29313058]
 [-0.89730789 -0.49575541]
 [-0.27273163 -0.57680534]
 [-0.68911581 -0.33365555]
 [-0.68911581 -0.49575541]
 [-0.62665818 -0.57680534]
 [-0.10617796 -0.09050576]
 [ 0.56003671 -0.09050576]
 [ 0.14365254 -0.49575541]
 [-0.14781638 -0.29313058]]
```

```
[-0.68911581 -0.41470548]
[-0.48092372 -0.33365555]
[ 2.64195758  3.15149149]
[-0.21027401 -0.49575541]
[-0.29355084 -0.29313058]
[-0.89730789 -0.69838024]
[-0.27273163 -0.17155569]
[ 1.80918923 -0.41470548]
[-0.59751129 -0.33365555]
[ 0.97642089  3.15149149]
[-0.25191242 -0.49575541]]
```

#### 步骤 4 数据拟合

输入：

```
#调用 Lr 中的 fit 模块训练模型参数
lr = LogisticRegression()
lr.fit(X_train, y)
```

输出：

```
LogisticRegression()
```

#### 步骤 5 数据预测

输入：

```
testX = [[2000,8]]
X_test = ss.transform(testX)
print("待预测的值：",X_test)
label = lr.predict(X_test)
print("predicted label = ", label)
#输出预测概率
prob = lr.predict_proba(X_test)
print("probability = ",prob)
```

输出：

```
待预测的值： [[-0.68911581 -0.57680534]]
predicted label =  [1]
probability =  [[0.41886952 0.58113048]]
```

### 1.2.4 决策树

本实验使用的数据集为 tennis.txt，主要包含 14 个样本，每个样本包含天气相关的特征及是否适合打球。

#### 步骤 1 导入依赖

输入：

```
import pandas as pd
import numpy as np
from sklearn import tree
import pydotplus
```

## 步骤 2 定义函数，生成决策树

输入：

```
#生成决策树
def createTree(trainingData):
    data = trainingData.iloc[:, :-1] # 特征矩阵
    labels = trainingData.iloc[:, -1] # 标签
    trainedTree = tree.DecisionTreeClassifier(criterion="entropy") # 分类决策树
    trainedTree.fit(data, labels) # 训练
    return trainedTree
```

## 步骤 3 定义函数，保存生成的树图

输入：

```
def showtree2pdf(trainedTree, filename):
    dot_data = tree.export_graphviz(trainedTree, out_file=None) #将树导出为 Graphviz 格式
    graph = pydotplus.graph_from_dot_data(dot_data)
    graph.write_pdf(filename) #保存树图到本地，格式为 pdf
```

## 步骤 4 定义函数，用于生成向量化数据

函数中，通过 `pd.Categorical(list).codes` 可以得到原始数据对应的序号列表，从而将类别信息转化成数值信息。

输入：

```
def data2vectoc(data):
    names = data.columns[:-1]
    for i in names:
        col = pd.Categorical(data[i])
        data[i] = col.codes
    return data
```

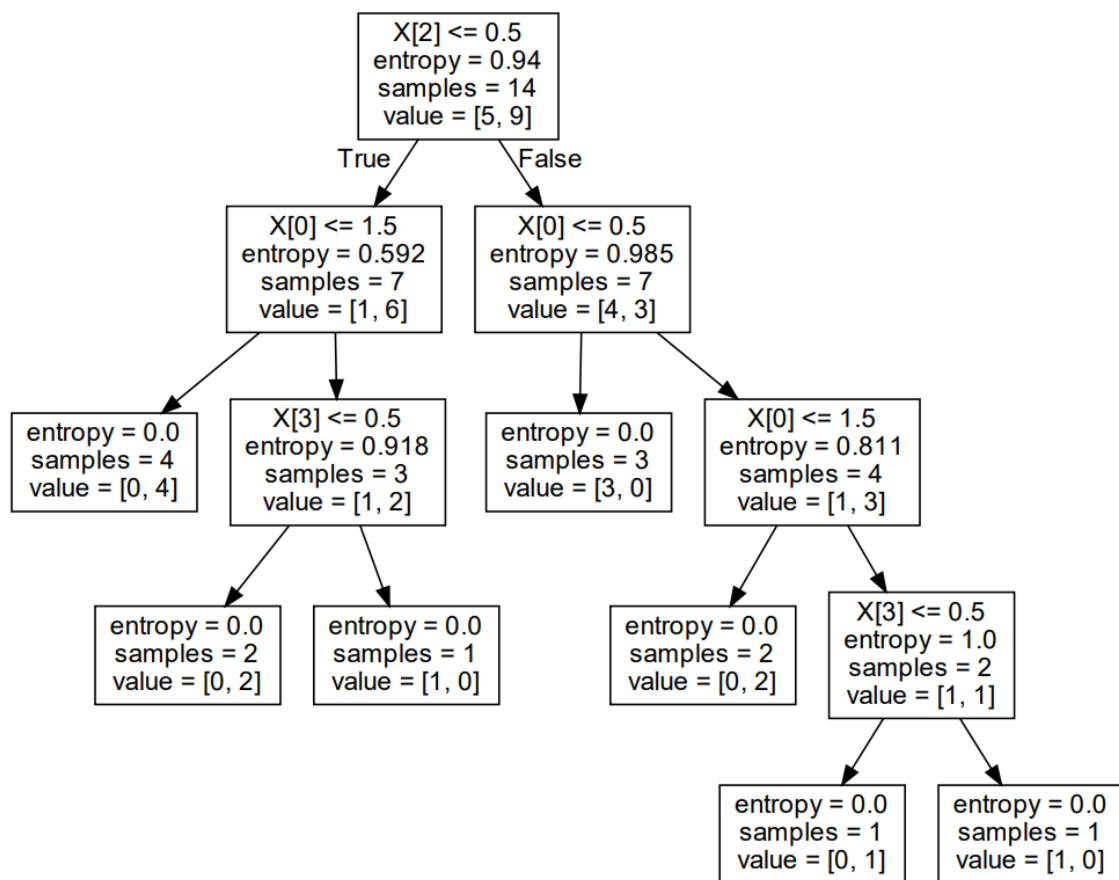
## 步骤 5 调用函数进行预测

输入：

```
data = pd.read_table("tennis.txt", header=None, sep='\t') #读取训练数据
trainingvec = data2vectoc(data) #向量化数据
decisionTree = createTree(trainingvec) #创建决策树
```

```
showtree2pdf(decisionTree,"tennis.pdf") #图示决策树
```

此时在本地生成决策树图，名称为 “tennis.pdf”。



可以看到里面的内容就是决策树的可视化呈现。其中  $X[2]$  就表示第 3 个特征变量：湿度， $X[0]$  则表示第 1 个特征变量：天气， $X[3]$  则表示第 4 个特征变量：风力；entropy 则表示该节点的熵值；samples 则表示该节点中的样本数，比如说第一个节点，也即根节点中的 14 就是训练集中的样本数量；value 则表示不同种类所占的个数，比如说根节点中 value 左边的 5 表示“否”的数量，9 则表示“是”的数量。

新样本预测，输入：

```
testVec = [0,0,1,1] # 天气晴、气温冷、湿度高、风力强
print(decisionTree.predict(np.array(testVec).reshape(1,-1))) #预测
```

输出：

```
['否']
```

## 1.2.5 朴素贝叶斯

通过 jieba 文字分词库对邮件数据集的垃圾邮件和进行文本处理，提取特征。然后调用 sklearn 机器学习库中的朴素贝叶斯算法训练模型，最后推理测试集中邮件是否为垃圾邮件。

## 步骤 1 引入相关依赖的包

输入：

```
from numpy import *
from os import listdir
import codecs #字符转换模块，用于文本的编码和解码
import jieba#中文分词库
import re
from sklearn.naive_bayes import MultinomialNB
from collections import Counter
from itertools import chain #用于串联迭代对象
```

## 步骤 2 构建文本处理函数

输入：

```
def segment2word(doc: str):
#从 stop_list.txt 文件中提取停用词，存储为列表
    stop_words = codecs.open("./ML/04/stop_list.txt", "r", "UTF-8").read().splitlines()
    doc = re.sub('[\t\r\n]', ' ', doc)#去除邮件文本中的缩进，换行等
    word_list = list(jieba.cut(doc.strip())) #用 jieba 进行分词
    out_str = ""
    for word in word_list: #删去邮件文本中的停用词
        if word == ' ' or word == '':#
            continue
        if word not in stop_words:
            out_str += word.strip()
            out_str += ' '
    segments = out_str.strip().split(sep=' ')
    return segments
```

## 步骤 3 构建文本读取函数

输入：

```
def getDatafromDir(data_dir):
    docLists = []
    docLabels = [f for f in listdir(data_dir) if f.endswith('.txt')]#存储每一封邮件的名称
    for doc in docLabels:
        try:
            filepath=data_dir + "/" + doc
            #对训练集的邮件进行文本处理
            wordList = segment2word(codecs.open(filepath, "r", "UTF-8").read())
            docLists.append(wordList)#整合训练集的邮件处理后的结果
        except:
            print("handling file %s is error!!" %filepath)
```

```
return docLists
```

#### 步骤 4 构建数据集

输入：

```
spamDocList=getDatafromDir("./ML/04/email/spam/") #对垃圾邮件进行文本处理
hamDocList = getDatafromDir("./ML/04/email/ham/") #对正常邮件进行文本处理
fullDocList = spamDocList + hamDocList#储存邮件的特征
# 添加标签, 垃圾邮件标记为 1, 正常邮件标记为 0
classList = array([1]*len(spamDocList)+[0]*len(hamDocList))
frequencyDic = Counter(chain(*fullDocList)) # 生成词频映射词典
topWords = [w[0] for w in frequencyDic.most_common(500)] #获取前 500 个最频繁的热词。
vector = []

for docList in fullDocList:
    #统计每封邮件中每个热词出现的频率
    topwords_list = list(map(lambda x: docList.count(x), topWords))
    vector.append(topwords_list)

#生成 vector 作为数据特征
vector = array(vector)
```

#### 步骤 5 模型训练

输入：

```
model = MultinomialNB() #选取多项式贝叶斯为训练模型
model.fit(vector, classList) #vector 为特征, classlist 为标签, 训练贝叶斯模型
```

#### 步骤 6 模型测试

输入：

```
#存储每一封训练集邮件的名称
dataList=[]
test_dir = "./ML/04/email/spam/"
docLabels = [f for f in listdir(test_dir) if f.endswith('.txt')]

#模型推理
for doc in docLabels:
    try:
        filepath = test_dir + "/" + doc
        dataList = segment2word(codecs.open(filepath, "r", "UTF-8").read())
    except:
        print("handling file %s is error!!" % filepath)
```

```
#统计测试集邮件中的热词的词频，提取特征
testVector = array(tuple(map(lambda x: dataList.count(x), topWords)))
testVector_reshape = testVector.reshape(1,-1)

#特征传入模型进行推理
predicted_label = model.predict(testVector.reshape(1, -1))
if(predicted_label == 1):
    print("%s is spam mail" %doc)
else:
    print("%s is NOT spam mail" % doc)
```

输出：

```
ham134.txt is NOT spam mail
ham148.txt is NOT spam mail
ham22.txt is NOT spam mail
ham5.txt is NOT spam mail
spam017.txt is NOT spam mail
spam07.txt is spam mail
spam117.txt is spam mail
spam3.txt is spam mail
spam32.txt is spam mail
spam79.txt is spam mail
```

## 1.2.6 K-means 算法实现

### 步骤 1 导入依赖

make\_blobs 用于产生本次实验所需的数据集，matplotlib 用于数据可视化，KMeans 用于 Kmeans 算法拟合训练。

输入：

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

### 步骤 2 生成数据集

make\_blobs()可以用来生成聚类任务的数据集，它会返回产生的数据集以及相应标签。但是需要注意的是，Kmean 算法是用于处理无标签数据集的，也就是，在实际使用 Kmeans 算法的时候，数据集是不含标签的。此处为方便大家理解，产生的数据集包含了标签信息。

其中，n\_samples 表示样本数量，n\_features 表示每个样本的特征数，也表示数据的维度，centers 表示类别数，random\_state 表示随机生成器的种子，可以固定生成的数据，给定之后，每次生成的数据集就是固定的。

输入：

```
X, y = make_blobs(n_samples=500,n_features=2,centers=4,random_state=1)
```

查看产生的数据集的维度信息。

输入：

```
print("X 的维度为: {}".format(X.shape))
print("y 的维度为: {}".format(y.shape))
```

输出：

```
x 的维度为: (500, 2)
y 的维度为: (500,)
```

可以看到，该数据集一共包含 500 个样本点，每个样本点包含了 2 个特征。

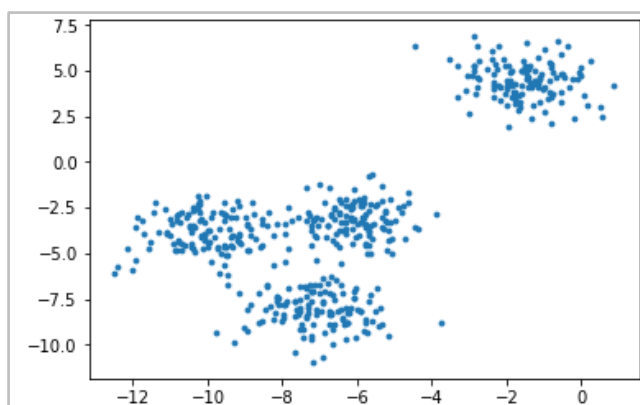
### 步骤 3 绘制散点图

首先，直接绘制该数据集的散点图，此时不考虑该数据集在生成时对应的标签。从图中我们可以看到 4 个不同的簇。

输入：

```
fig, ax1 = plt.subplots(1)
ax1.scatter(X[:, 0], X[:, 1],
            ,marker='o' # 设置点的形状为圆形
            ,s=8 # 设置点的大小
            )
plt.show()
```

输出：



接下来，我们根据生成数据集时对应的标签，按照不同颜色绘制散点图。

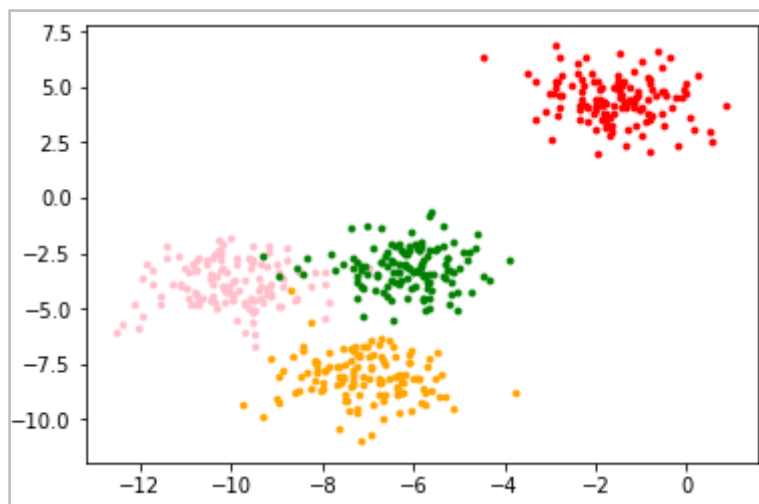
输入：

```
color = ["red", "pink", "orange", "green"]
fig, ax1 = plt.subplots(1)

for i in range(4):
```

```
ax1.scatter(X[y==i, 0], X[y==i, 1] # 根据每个点的标签绘制
            ,marker='o' # 设置点的形状为圆形
            ,s=8 # 设置点的大小
            ,c=color[i]
            )
plt.show()
```

输出：



#### 步骤4 Kmeans 聚类

调用 Sklearn 提供的 sklearn.cluster.KMeans 实现 K-means 聚类。首先我们将数据集聚成 3 类。

输入：

```
n_clusters = 3
cluster1 = KMeans(n_clusters=n_clusters,random_state=3).fit(X)
```

查看聚类后的每一个样本点的标签。

输入：

```
y_pred1 = cluster1.labels_
print(y_pred1)
```

输出：

```
[0 0 2 1 2 1 2 2 2 2 0 0 2 1 2 0 2 0 1 2 2 2 2 1 2 2 1 1 2 2 0 1 2 0 2 0 2
 2 0 2 2 2 1 2 2 0 2 2 1 1 1 2 2 2 0 2 2 2 2 2 1 1 2 2 1 2 0 2 2 2 0 2 2 0
 2 2 0 2 2 2 1 1 2 1 1 2 2 1 2 2 1 0 2 2 1 0 0 2 0 1 1 0 1 2 1 2 2 1 1 2 2
 0 1 2 1 2 1 2 1 2 2 0 0 2 2 2 1 0 0 2 1 2 2 2 2 0 1 2 1 1 2 0 2 1 1 1 2 2
 0 0 2 2 1 0 1 2 2 2 2 2 2 2 2 1 0 0 0 2 1 0 2 2 0 1 2 2 2 2 0 2 2 1 0 0
 2 2 0 0 2 1 1 0 0 2 1 2 0 0 1 0 2 1 2 2 0 2 2 0 2 2 2 2 0 2 2 2 1 2 1 2 0
 2 2 2 2 2 1 2 1 0 2 0 2 1 1 2 0 1 0 2 2 0 0 0 0 2 2 0 2 2 1 1 2 2 1 2 2 2
 1 2 1 2 2 1 2 0 0 2 2 2 2 1 1 2 1 2 0 1 0 1 0 0 1 0 1 1 2 2 2 2 2 2 0 1
 0 0 0 2 2 2 0 2 0 0 2 0 0 2 1 0 2 2 1 1 2 0 1 1 2 0 1 1 2 2 1 2 2 0 0 1 2
 0 2 1 1 2 2 2 0 2 1 1 2 1 1 1 1 0 0 2 1 2 2 0 1 2 1 2 1 2 2 2 1 2 2 0 1 0]
```

```
0000020101121222012120220221102212200
2022020210122122102112222010210002120
222202222202202121222111222021201010
2110220222021220002]
```

打印每个簇的质心。

输入：

```
centroid1 = cluster1.cluster_centers_
print(centroid1)
```

输出：

```
[[ -7.09306648 -8.10994454]
 [ -1.54234022  4.43517599]
 [ -8.0862351  -3.5179868 ]]
```

可视化聚类结果。

输入：

```
color = ["red", "pink", "orange", "gray"]

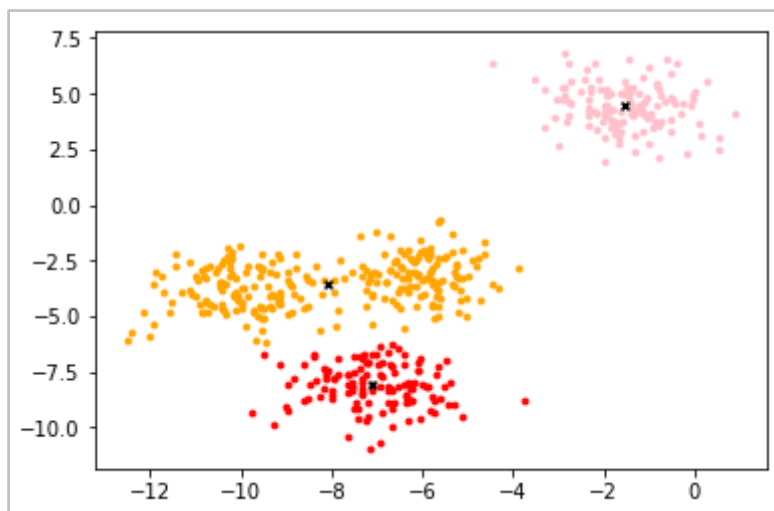
fig, ax1 = plt.subplots(1)

for i in range(n_clusters):
    ax1.scatter(X[y_pred1==i, 0], X[y_pred1==i, 1]
                ,marker='o' #点的形状
                ,s=8 #点的大小
                ,c=color[i]
                )

ax1.scatter(centroid1[:,0],centroid1[:,1]
            ,marker="x"
            ,s=15
            ,c="black")

plt.show()
```

输出：



可以看到，数据集被聚成了 3 个簇，每个簇的质心用黑色 x 表示。

### 步骤 5 再次 Kmeans 聚类

接下来，我们按照上述步骤重新实现 Kmeans 聚类，这次我们聚为 4 类。

输入：

```
n_clusters = 4
cluster2 = KMeans(n_clusters=n_clusters,random_state=0).fit(X)
y_pred2 = cluster2.labels_
centroid2 = cluster2.cluster_centers_
print("质心： {}".format(centroid2))
```

输出：

```
质心： [[ -6.08459039  -3.17305983]
 [ -1.54234022   4.43517599]
 [ -7.09306648  -8.10994454]
 [-10.00969056  -3.84944007]]
```

可视化聚类结果。

输入：

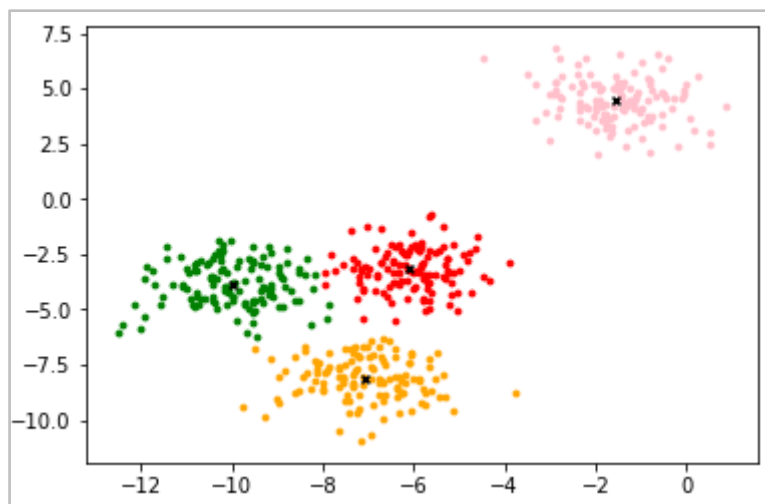
```
color = ["red","pink","orange","green"]

fig, ax1 = plt.subplots(1)

for i in range(n_clusters):
    ax1.scatter(X[y_pred2==i, 0], X[y_pred2==i, 1]
                ,marker='o' #点的形状
                ,s=8 #点的大小
                ,c=color[i]
                )
```

```
ax1.scatter(centroid2[:,0],centroid2[:,1]
            ,marker="x"
            ,s=15
            ,c="black")
plt.show()
```

输出：



可以看到，数据集被聚成了 4 个簇，每个簇的质心用黑色 x 表示。大家可以对比带有标签信息的原始数据散点图和被聚为 4 类后的数据散点图，可以发现，存在较多的样本点被聚到了错误的簇中。

## 1.3 思考题

如何使用 Python 从零实现 K-means 算法？

## 1.4 本章总结

本章主要介绍机器学习实现流程：主要包含导入数据，分割数据，数据标准化，定义模型以及设置相关超参数等方面；基于 sklearn 实现常用机器学习算法，加深学员整体上对机器学习模型的构建和使用的理解。

华为认证 AI 系列教程

HCIA-AI

# 深度学习和人工智能开发框架

## 实验指导手册

版本：4.0



华为技术有限公司

**版权所有 © 华为技术有限公司 2025。 保留一切权利。**

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## **商标声明**



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## **注意**

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

**华为技术有限公司**

地址：            深圳市龙岗区坂田华为总部办公楼            邮编： 518129

网址：            <https://e.huawei.com>

---

## 华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT技术架构与应用认证、云服务与平台认证两类认证。

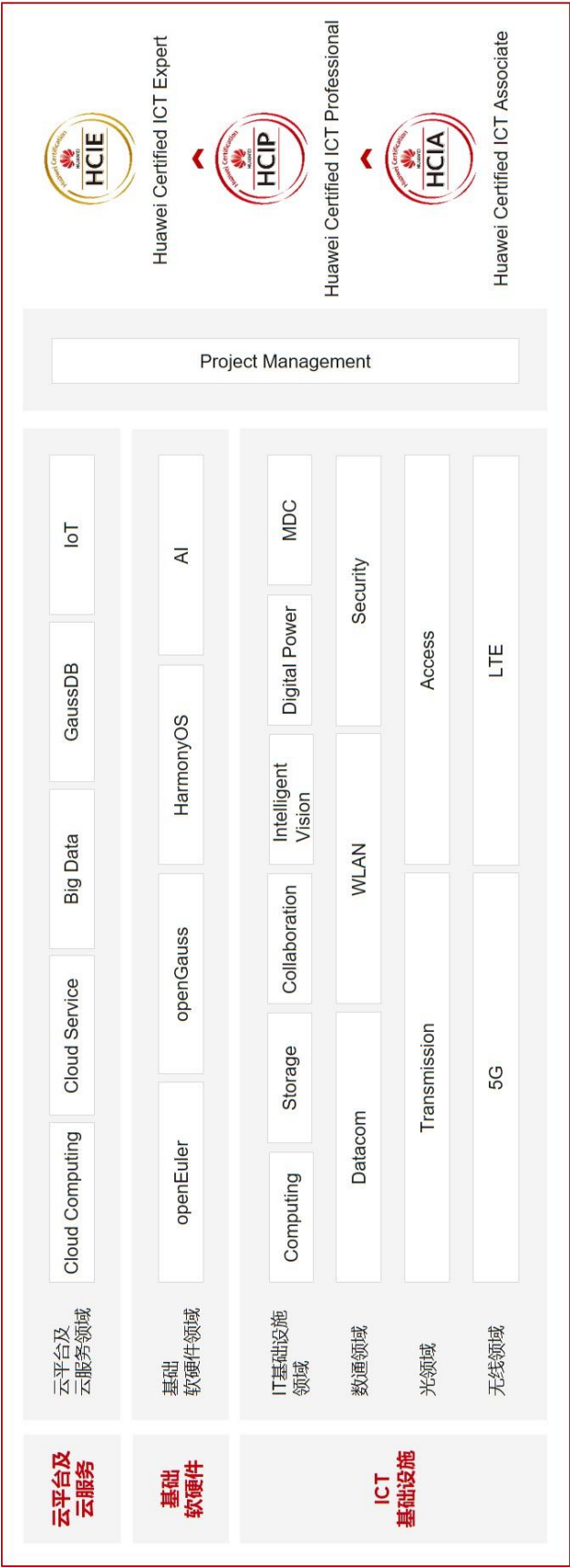
根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIA-AI V4.0 定位于培养和认证具备人工智能基础知识，掌握机器学习、深度学习和大模型等基础算法，可使用AI开发框架进行人工智能产品开发和创新的工程师。

通过HCIA-AI V4.0 认证，将证明您了解人工智能发展历史和前沿应用场景，掌握传统机器学习、深度学习和大模型的相关基础算法；具备利用AI开发框架搭建、训练和部署神经网络的能力；能够胜任人工智能开发工程师和人工智能技术支持等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在人工智能世界的潮头浪尖！



# 前言

---

## 简介

本书为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-AI 考试的学员或者希望了解 AI 基础知识及 PyTorch 或 MindSpore 编程基础的读者。

## 内容描述

本实验指导书共包含三个实验：

- 实验一使用 python 编程语言和基础第三方库实现了三层全连接神经网络，包含网络结构、梯度计算、参数优化，并使用不同优化器进行训练，对比损失函数变化情况。该实验为学员深入了解深度学习整体流程和各个环节的基本实现提供了有效的参考。
- 实验二为 MindSpore 实验，本实验会先介绍 MindSpore 的基本语法和常用模块，在掌握 MindSpore 的使用后，使用 MindSpore 完成 MNIST 手写数字识别、图像分类和情感分析实践。
- 实验三为 PyTorch 实验，本实验会先介绍 PyTorch 的基本使用，在掌握 PyTorch 的使用后，使用 PyTorch 完成图像分类实践。

实验手册中使用了 MindSpore 和 PyTorch 两个框架，学员可根据自己的能力及需求选择其中一个框架进行学习，若学有余力可以以比对的方式同时学习两个框架。

**实验中涉及到的数据及代码文件链接：**

<https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V4.0/chapter4/%E6%95%B0%E6%8D%AE%E9%9B%86%26%E4%BB%A3%E7%A0%81%E6%96%87%E4%BB%B6.rar>

## 读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本的 Python 知识背景。
-

# 目录

|                              |           |
|------------------------------|-----------|
| <b>前 言 .....</b>             | <b>2</b>  |
| 简介 .....                     | 2         |
| 内容描述 .....                   | 2         |
| 读者知识背景 .....                 | 2         |
| <b>1 从 0 实现全连接神经网络 .....</b> | <b>5</b>  |
| 1.1 实验介绍 .....               | 5         |
| 1.1.1 关于本实验 .....            | 5         |
| 1.1.2 文件解析 .....             | 5         |
| 1.1.3 实验的整体流程 .....          | 6         |
| 1.2 训练代码解析 .....             | 7         |
| 1.2.1 代码主程序 .....            | 7         |
| 1.2.2 加载数据集 .....            | 8         |
| 1.2.3 构建网络结构 .....           | 11        |
| 1.2.4 开始训练 .....             | 15        |
| 1.2.5 绘制图形 .....             | 15        |
| 1.2.6 实验结果 .....             | 16        |
| 1.3 思考题 .....                | 16        |
| <b>2 MindSpore .....</b>     | <b>17</b> |
| 2.1 实验介绍 .....               | 17        |
| 2.1.1 关于本实验 .....            | 17        |
| 2.1.2 实验目的 .....             | 17        |
| 2.1.3 实验环境 .....             | 17        |
| 2.2 MindSpore 基础 .....       | 17        |
| 2.2.1 Tensor 介绍 .....        | 17        |
| 2.2.2 数据集加载 .....            | 22        |
| 2.2.3 网络构建 .....             | 25        |
| 2.2.4 模型训练与评估 .....          | 28        |

|                           |           |
|---------------------------|-----------|
| 2.2.5 模型保存与加载 .....       | 30        |
| 2.2.6 自动微分 .....          | 31        |
| 2.2.7 思考题 .....           | 32        |
| 2.3 MNIST 手写体字符实验 .....   | 33        |
| 2.4 ResNet50 图像分类实验 ..... | 39        |
| 2.5 TextCNN 情感分析实验 .....  | 55        |
| <b>3 PyTorch .....</b>    | <b>67</b> |
| 3.1 实验介绍 .....            | 67        |
| 3.1.1 关于本实验 .....         | 67        |
| 3.1.2 实验目的 .....          | 67        |
| 3.1.3 实验环境 .....          | 67        |
| 3.2 PyTorch 基础 .....      | 67        |
| 3.2.1 创建 tensor .....     | 67        |
| 3.2.2 Tensor 的属性 .....    | 69        |
| 3.2.3 Tensor 的方法 .....    | 70        |
| 3.2.4 加载数据集 .....         | 70        |
| 3.3 LeNet .....           | 75        |
| 3.4 ResNet50 图像分类 .....   | 78        |
| 3.5 思考 .....              | 85        |

---

# 1 从 0 实现全连接神经网络

---

## 1.1 实验介绍

### 1.1.1 关于本实验

本实验旨在使用 Python 编程语言和 NumPy 等基础库实现一个多层的全连接神经网络，并使用 MNIST 数据集进行训练和测试，实现对手写数字图像的识别。

实验的目的是不使用任何深度学习框架完成神经网络的构建、训练和测试，通过逐步实现各个部分功能来深入了解神经网络的实现原理及训练和测试的流程。

MNIST 数据集是一个广泛使用的手写数字识别数据集，它包含了大量的手写数字图像，这些图像被用于训练和测试各种图像处理和识别技术，特别是手写数字识别。

MNIST 数据集由 0 到 9 这 10 个数字的手写体图像组成，分为训练集和测试集两部分。训练集包含 60,000 个样本，测试集包含 10,000 个样本。每个样本都是一张 28x28 像素的灰度图像，像素值范围从 0（白色）到 255（黑色）。这些图像是从美国国家标准与技术研究院（NIST）的一个项目中提取出来的，该项目旨在收集大量的手写数字数据，用于光学字符识别（OCR）系统的研究和开发。

### 1.1.2 文件解析

下载完成代码及数据集文件后，解压文件。文件目录如下：

```
|— MNIST
|   |— functions.py
|   |— gradient.py
|   |— layers.py
|   |— load_mnist.py
|   |— multi_layer_net.py
|   |— train_mnist.py
|   |— optimizer.py
|   |— util.py
```

---

```
| |— t10k-images-idx3-ubyte
| |— t10k-labels-idx1-ubyte
| |— train-images-idx3-ubyte
| |— train-labels-idx1-ubyte
```

数据文件介绍：

train-images-idx3-ubyte：训练集图像数据，包含 60,000 张 28x28 像素的灰度图像。

train-labels-idx1-ubyte：训练集图像标签，包含 60,000 个对应的数字标签。

t10k-images-idx3-ubyte：测试集图像数据，包含 10,000 张 28x28 像素的灰度图像。

t10k-labels-idx1-ubyte：测试集图像标签，包含 10,000 个对应的数字标签。

mnist.pkl：该文件是一个经过 Python pickle 模块序列化的文件，它包含了 MNIST 数据集的训练集、验证集。

train\_mnist.py：本次实验代码入口，训练主程序。

load\_mnist.py：包含数据加载函数的实现。

multi\_layer\_net.py：多层神经网络的实现。

layers.py：基本层函数的实现。

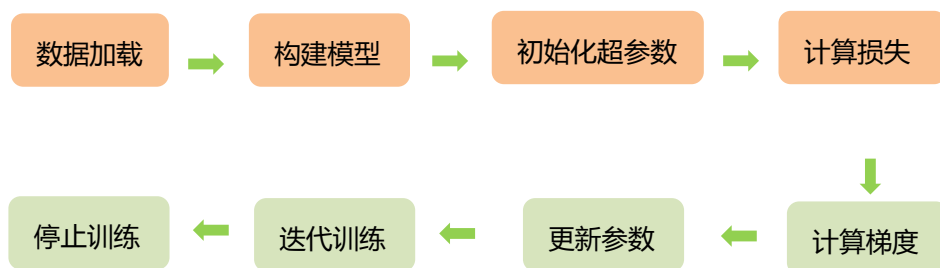
gradient.py：计算梯度函数的实现。

optimizer.py：包含多种优化器的实现。

util.py：包含数据转换函数的实现。

functions.py：多种激活函数的实现。

### 1.1.3 实验的整体流程



## 1.2 训练代码解析

### 1.2.1 代码主程序

train\_mnist.py 为项目主程序，代码如下：

```
# coding: utf-8
import matplotlib.pyplot as plt
from load_mnist import load_mnist
from util import smooth_curve
from multi_layer_net import MultiLayerNet
from optimizer import *

# 0:读入 MNIST 数据=====
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)

train_size = x_train.shape[0]
batch_size = 128
max_iterations = 1000

# 1:进行实验的设置=====
optimizers = {} # 创建字典，存储不同的优化器
optimizers['SGD'] = SGD()
optimizers['Momentum'] = Momentum()
optimizers['AdaGrad'] = AdaGrad()
optimizers['Adam'] = Adam()

networks = {} # 创建存储网络结构的字典
train_loss = {} # 创建存储损失函数的字典
for key in optimizers.keys():
    networks[key] = MultiLayerNet(
        input_size=784, hidden_size_list=[100, 100, 100, 100],
        output_size=10)
    train_loss[key] = []

# 2:开始训练=====
for i in range(max_iterations):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]
```

```

for key in optimizers.keys():
    grads = networks[key].gradient(x_batch, t_batch)
    optimizers[key].update(networks[key].params, grads)

    loss = networks[key].loss(x_batch, t_batch)
    train_loss[key].append(loss)

if i % 100 == 0:
    print("=====" + "iteration:" + str(i) + "=====")
    for key in optimizers.keys():
        loss = networks[key].loss(x_batch, t_batch)
        print(key + ":" + str(loss))

# 3.绘制图形=====
markers = {"SGD": "o", "Momentum": "x", "AdaGrad": "s", "Adam": "D"}
x = np.arange(max_iterations)
for key in optimizers.keys():
    plt.plot(x, smooth_curve(train_loss[key]), marker=markers[key], markevery=100, label=key)
plt.xlabel("iterations")
plt.ylabel("loss")
plt.ylim(0, 1)
plt.legend()
plt.show()

```

## 1.2.2 加载数据集

本实验已经下载 mnist 数据集压缩包。使用 load\_mnist 函数加载数据集，该函数会将数据集加载到 pickle 文件中，后续读取数据集时，只需要加载 pickle 文件即可。

```

# 读入 MNIST 数据
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)

train_size = x_train.shape[0]    # 获取训练图片数量
batch_size = 128                 # 设置 batch_size
max_iterations = 1000            # 设置最大迭代次数

```

### 分析 load\_mnist 函数

打开 load\_mnist.py 文件。

load\_mnist.py 首先声明了如下变量：

```

# key_file 以字典的形式保存了数据集压缩包的名称
key_file = {
    'train_img': 'train-images-idx3-ubyte.gz',
    'train_label': 'train-labels-idx1-ubyte.gz',

```

```
'test_img': 't10k-images-idx3-ubyte.gz',
'test_label': 't10k-labels-idx1-ubyte.gz'
}

# dataset_dir 变量是获取当前代码文件所在的文件夹绝对路径
dataset_dir = os.path.dirname(os.path.abspath(__file__))

# save_file 通过字符串拼接生成.pkl 文件的保存路径
save_file = dataset_dir + "/mnist.pkl"
```

load\_mnist 函数主体如下：

```
def load_mnist(normalize=True, flatten=True, one_hot_label=False):
    """读入 MNIST 数据集
    Parameters
    -----
    normalize : 将图像的像素值正规化为 0.0~1.0
    one_hot_label :
        one_hot_label 为 True 的情况下，标签作为 one-hot 数组返回
        one-hot 数组是指[0,0,1,0,0,0,0,0,0]这样的数组
    flatten : 是否将图像展开为一维数组

    Returns
    -----
    (训练图像, 训练标签), (测试图像, 测试标签)
    """
    # 判断保存数据集的 pickle 文件是否存在，如果不存在，则使用压缩的数据集制作。
    if not os.path.exists(save_file):
        init_mnist()
    # 使用 pickle 文件加载数据集
    with open(save_file, 'rb') as f:
        dataset = pickle.load(f)

    """以下操作是对数据集进行预处理"""
    if normalize:
        for key in ('train_img', 'test_img'):
            dataset[key] = dataset[key].astype(np.float32) # 转换为 np.float32 数据类型
            dataset[key] /= 255.0 # 将像素值转换为 0-1 之间

    if one_hot_label: # 将标签转换为 one-hot 形式
        dataset['train_label'] = _change_one_hot_label(dataset['train_label'])
        dataset['test_label'] = _change_one_hot_label(dataset['test_label'])

    if not flatten: # 判断数据是否需要拉平，拉平即将保持 (784, ) 格式，否则转换为 (28, 28)
        for key in ('train_img', 'test_img'):
```

```
dataset[key] = dataset[key].reshape(-1, 1, 28, 28)

return (dataset['train_img'], dataset['train_label']), (dataset['test_img'], dataset['test_label'])
```

分析 load\_mnist 函数体内调用的其他函数：

**init\_mnist 函数：**

```
def init_mnist():
    dataset = _convert_numpy()    # 将数据集转化为 numpy 格式
    print("Creating pickle file ...")
    with open(save_file, 'wb') as f:    # 以二进制可写方式打开.pkl 文件
        pickle.dump(dataset, f, -1)    # 将数据集写入到.pkl 文件中，完成后自动关闭该文件。
    print("Done!")
```

**\_convert\_numpy 函数：**

```
def _convert_numpy():
    dataset = {}    # 创建一个空字典
    # 将数据集按名称保存在字典中
    dataset['train_img'] = _load_img(key_file['train_img'])
    dataset['train_label'] = _load_label(key_file['train_label'])
    dataset['test_img'] = _load_img(key_file['test_img'])
    dataset['test_label'] = _load_label(key_file['test_label'])

    return dataset
```

**\_load\_label 函数和\_load\_img 函数：**

```
# 加载标签
def _load_label(file_name):
    file_path = dataset_dir + "/" + file_name    # 被写入文件的绝对路径

    print("Converting " + file_name + " to NumPy Array ...")
    with gzip.open(file_path, 'rb') as f:    # 打开压缩的数据集
        """
        np.frombuffer():
        这是一个 numpy 库中的函数，用于从字节缓冲区创建一个 numpy 数组。
        第一个参数是缓冲区，在这个例子中，它是通过 f.read()读取的文件内容。
        f.read():
        f 是已经打开的文件对象。
        f.read()是一个文件对象的方法，用于读取文件的所有剩余内容，并以字节的形式返回。
        np.uint8:
        指定 np.frombuffer()函数读取的缓冲区中的数据应该被解释为 uint8。
        offset=8:
```

这个参数指定了在开始解析数据之前，应该跳过缓冲区开头的字节数。

它告诉函数跳过缓冲区的前 8 个字节，从第 9 个字节开始创建数组。

```

"""
labels = np.frombuffer(f.read(), np.uint8, offset=8)
print("Done")

return labels
# _load_img 函数结构和_load_label 相同，作用是加载数据（图像）。
def _load_img(file_name):
    file_path = dataset_dir + "/" + file_name

    print("Converting " + file_name + " to NumPy Array ...")
    with gzip.open(file_path, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=16)
        data = data.reshape(-1, 784)
        print("Done")

    return data

```

**\_change\_one\_hot\_label 函数：**

```

def _change_one_hot_label(X):
    T = np.zeros((X.size, 10)) # 初始化全 0 矩阵 T，形状为 (X.size,10)
    for idx, row in enumerate(T):
        row[X[idx]] = 1 # 将 one-hot 中对应标签位置置为 1
    # T = np.eye(10)[X] #以上代码可以被这一句代替
    return T

```

## 1.2.3 构建网络结构

```

# 1. 设置多种优化器，对比使用不同优化器 loss 变化
optimizers = {}
optimizers['SGD'] = SGD()
optimizers['Momentum'] = Momentum()
optimizers['AdaGrad'] = AdaGrad()
optimizers['Adam'] = Adam()

# 构建网络模型
networks = {} # 保存网络模型
train_loss = {} # 创建字典保存不同优化器下的 loss 值
for key in optimizers.keys():
    networks[key] = MultiLayerNet(
        input_size=784, hidden_size_list=[100, 100, 100, 100],
        output_size=10)
    train_loss[key] = []

```

## 解析 MultiLayerNet 类：

```
class MultiLayerNet:
    """全连接的多层神经网络

    Parameters
    -----
    input_size : 输入大小 (MNIST 的情况下为 784)
    hidden_size_list : 隐藏层的神经元数量的列表 (e.g. [100, 100, 100])
    output_size : 输出大小 (MNIST 的情况下为 10)
    activation : 'relu' or 'sigmoid'
    weight_init_std : 指定权重的标准差 (e.g. 0.01)
        指定'relu'或'he'的情况下设定“He 的初始值”
        指定'sigmoid'或'xavier'的情况下设定“Xavier 的初始值”
    weight_decay_lambda : Weight Decay (L2 范数) 的强度
    """
    def __init__(self, input_size, hidden_size_list, output_size,
                  activation='relu', weight_init_std='relu', weight_decay_lambda=0):
        # 设置网络
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size_list = hidden_size_list
        self.hidden_layer_num = len(hidden_size_list)
        self.weight_decay_lambda = weight_decay_lambda
        self.params = {}

        # 初始化权重
        self.__init_weight(weight_init_std)

        # 生成层
        activation_layer = {'sigmoid': Sigmoid, 'relu': Relu}
        self.layers = OrderedDict()
        for idx in range(1, self.hidden_layer_num+1):
            self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
                                                       self.params['b' + str(idx)])
            self.layers['Activation_function' + str(idx)] = activation_layer[activation]()

        idx = self.hidden_layer_num + 1
        self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
                                                    self.params['b' + str(idx)])

        self.last_layer = SoftmaxWithLoss()

    def __init_weight(self, weight_init_std):
```

```

"""设定权重的初始值

Parameters
-----
weight_init_std : 指定权重的标准差 (e.g. 0.01)
    指定'relu'或'he'的情况下设定“He 的初始值”
    指定'sigmoid'或'xavier'的情况下设定“Xavier 的初始值”
"""

all_size_list = [self.input_size] + self.hidden_size_list + [self.output_size]
for idx in range(1, len(all_size_list)):
    scale = weight_init_std
    if str(weight_init_std).lower() in ('relu', 'he'):
        scale = np.sqrt(2.0 / all_size_list[idx - 1]) # 使用 ReLU 的情况下推荐的初始值
    elif str(weight_init_std).lower() in ('sigmoid', 'xavier'):
        scale = np.sqrt(1.0 / all_size_list[idx - 1]) # 使用 sigmoid 的情况下推荐的初始值

    self.params['W' + str(idx)] = scale * np.random.randn(all_size_list[idx-1], all_size_list[idx])
    self.params['b' + str(idx)] = np.zeros(all_size_list[idx])

def predict(self, x):
    for layer in self.layers.values():
        x = layer.forward(x)

    return x

def loss(self, x, t):
    """求损失函数

    Parameters
    -----
    x : 输入数据
    t : 正确标签

    Returns
    -----
    损失函数的值
    """
    y = self.predict(x)

    weight_decay = 0
    for idx in range(1, self.hidden_layer_num + 2):
        W = self.params['W' + str(idx)]
        weight_decay += 0.5 * self.weight_decay_lambda * np.sum(W ** 2)

    return self.last_layer.forward(y, t) + weight_decay

```

```
def accuracy(self, x, t):
    y = self.predict(x)
    y = np.argmax(y, axis=1)
    if t.ndim != 1 : t = np.argmax(t, axis=1)

    accuracy = np.sum(y == t) / float(x.shape[0])
    return accuracy

def numerical_gradient(self, x, t):
    """求梯度（数值微分）
    Parameters
    -----
    x：输入数据
    t：正确标签

    Returns
    -----
    具有各层的梯度的字典变量
        grads['W1']、grads['W2']、...是各层的权重
        grads['b1']、grads['b2']、...是各层的偏置
    """
    loss_W = lambda W: self.loss(x, t)
    grads = {}
    for idx in range(1, self.hidden_layer_num+2):
        grads['W' + str(idx)] = numerical_gradient(loss_W, self.params['W' + str(idx)])
        grads['b' + str(idx)] = numerical_gradient(loss_W, self.params['b' + str(idx)])

    return grads

def gradient(self, x, t):
    """求梯度（误差反向传播法）
    Parameters
    -----
    x：输入数据
    t：教师标签

    Returns
    -----
    具有各层的梯度的字典变量
        grads['W1']、grads['W2']、...是各层的权重
        grads['b1']、grads['b2']、...是各层的偏置
    """
    # forward
    self.loss(x, t)
```

```
# backward
dout = 1
dout = self.last_layer.backward(dout)

layers = list(self.layers.values())
layers.reverse()
for layer in layers:
    dout = layer.backward(dout)

# 设定
grads = {}
for idx in range(1, self.hidden_layer_num+2):
    grads['W' + str(idx)] = self.layers['Affine' + str(idx)].dW + self.weight_decay_lambda *
self.layers['Affine' + str(idx)].W
    grads['b' + str(idx)] = self.layers['Affine' + str(idx)].db

return grads
```

## 1.2.4 开始训练

```
# 2:开始训练=====
for i in range(max_iterations):
    batch_mask = np.random.choice(train_size, batch_size) # 生成随机数
    x_batch = x_train[batch_mask] # 使用随机数作为下标, 数选取训练数据
    t_batch = t_train[batch_mask] # 使用随机数作为下标, 数选取对应的标签

    # 优化参数
    for key in optimizers.keys():
        grads = networks[key].gradient(x_batch, t_batch)
        optimizers[key].update(networks[key].params, grads)

        loss = networks[key].loss(x_batch, t_batch)
        train_loss[key].append(loss)

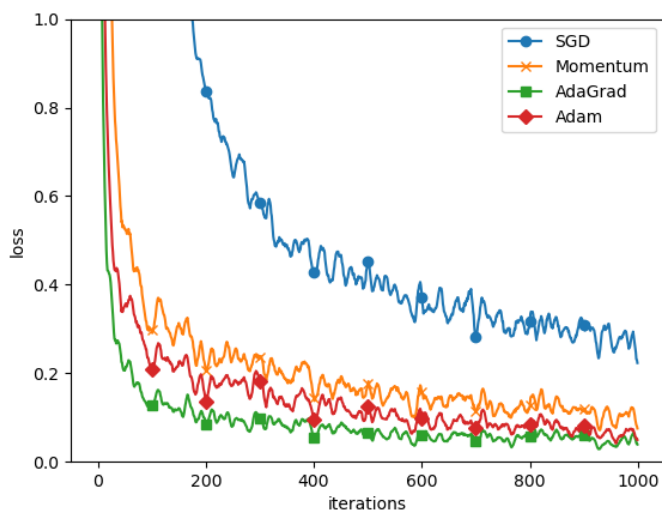
    if i % 100 == 0:
        print( "===== " + "iteration:" + str(i) + "===== ")
        for key in optimizers.keys():
            loss = networks[key].loss(x_batch, t_batch)
            print(key + ":" + str(loss))
```

## 1.2.5 绘制图形

```
markers = {"SGD": "o", "Momentum": "x", "AdaGrad": "s", "Adam": "D"}
x = np.arange(max_iterations)
for key in optimizers.keys():
    plt.plot(x, smooth_curve(train_loss[key]), marker=markers[key], markevery=100, label=key)
```

```
plt.xlabel("iterations")
plt.ylabel("loss")
plt.ylim(0, 1)
plt.legend()
plt.show()
```

## 1.2.6 实验结果



## 1.3 思考题

1. 代码中如何修改学习率？请使用不同的学习率设置策略，观察 loss 曲线。
2. 请在代码中加入测试模块，输出不同阶段测试准确率。

# 2 MindSpore 实验

---

## 2.1 实验介绍

### 2.1.1 关于本实验

本实验主要是介绍 MindSpore 的张量数据结构，通过对张量的一系列操作介绍，可以使学员对 MindSpore 的基本语法有所了解。在了解 MindSpore 基础操作以后，使用 MindSpore 实现 ResNet、TextCNN 等结构，进一步了解 MindSpore 的应用开发过程。

### 2.1.2 实验目的

- 掌握 MindSpore 张量的创建方法。
- 掌握 MindSpore 张量的属性与方法。
- 掌握 MindSpore 的网络搭建过程。

### 2.1.3 实验环境

本实验所需的框架为 MindSpore，建议版本为 2.0 及以上版本。实验可以在个人 PC 上完成也可以登录华为云购买 ModelArts 服务完成。

## 2.2 MindSpore 基础

### 2.2.1 Tensor 介绍

张量（Tensor）是 MindSpore 网络运算中的基本数据结构。张量中的数据类型可参考 [dtype](#)。

不同维度的张量分别表示不同的数据，0 维张量表示标量，1 维张量表示向量，2 维张量表示矩阵，3 维张量可以表示彩色图像的 RGB 三通道等等。

MindSpore 张量支持不同的数据类型，包含 int8、int16、int32、int64、uint8、uint16、uint32、uint64、float16、float32、float64、bool\_，与 NumPy 的数据类型一一对应。

在 MindSpore 的运算处理流程中，Python 中的 int 数会被转换为定义的 int64 类型，float 数会被转换为定义的 float32 类型。

---

### 2.2.1.1 创建 tensor

构造张量时，支持传入 Tensor、float、int、bool、tuple、list 和 NumPy.array 类型，其中 tuple 和 list 里只能存放 float、int、bool 类型数据。

Tensor 初始化时，可指定 dtype。如果没有指定 dtype，初始值 int、float、bool 分别生成数据类型为 mindspore.int32、mindspore.float32、mindspore.bool\_ 的 0 维 Tensor，初始值 tuple 和 list 生成的 1 维 Tensor 数据类型与 tuple 和 list 里存放的数据类型相对应，如果包含多种不同类型的数据，则按照优先级：bool < int < float，选择相对优先级最高类型所对应的 mindspore 数据类型。如果初始值是 Tensor，则生成的 Tensor 数据类型与其一致；如果初始值是 NumPy.array，则生成的 Tensor 数据类型与之对应。

#### 步骤 1 数组创建 Tensor

代码：

```
# 导入 MindSpore
import mindspore
# cell 同时输出多行
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import numpy as np
from mindspore import Tensor
from mindspore import dtype
# 用数组创建张量
x = Tensor(np.array([[1, 2], [3, 4]]), dtype.int32)
x
```

输出：

```
Tensor(shape=[2, 2], dtype=Int32, value=
[[1, 2],
 [3, 4]])
```

#### 步骤 2 用数值创建 Tensor

代码：

```
# 用数值创建张量
y = Tensor(1.0, dtype.int32)
z = Tensor(2, dtype.int32)
y
z
```

输出：

```
Tensor(shape=[], dtype=Int32, value= 1)
Tensor(shape=[], dtype=Int32, value= 2)
```

### 步骤3 用 Bool 创建 Tensor

代码：

```
# 用 Bool 创建张量
m = Tensor(True, dtype.bool_)
m
```

输出：

```
Tensor(shape=[], dtype=Bool, value= True)
```

### 步骤4 用 tuple 创建 Tensor

代码：

```
# 用 tuple 创建张量
n = Tensor((1, 2, 3), dtype.int16)
n
```

输出

```
Tensor(shape=[3], dtype=Int16, value= [1, 2, 3])
```

### 步骤5 用 list 创建 Tensor

代码：

```
# 用 list 创建张量
p = Tensor([4.0, 5.0, 6.0], dtype.float64)
p
```

输出：

```
Tensor(shape=[3], dtype=Float64, value= [4.00000000e+000, 5.00000000e+000, 6.00000000e+000])
```

### 步骤6 继承另一个张量的属性，形成新的张量

代码：

```
from mindspore import ops
oneslike = ops.OnesLike()
x = Tensor(np.array([[0, 1], [2, 1]]).astype(np.int32))
output = oneslike(x)
output
```

输出：

```
Tensor(shape=[2, 2], dtype=Int32, value=
[[1, 1],
 [1, 1]])
```

### 步骤7 输出恒定值 Tensor

代码：

```
from mindspore.ops import operations as ops

shape = (2, 2)
ones = ops.Ones()
output = ones(shape, dtype.float32)
print(output)

zeros = ops.Zeros()
output = zeros(shape, dtype.float32)
print(output)
```

输出：

```
[[1. 1.]
 [1. 1.]]
[[0. 0.]
 [0. 0.]]
```

### 2.2.1.2 Tensor 的属性

张量的属性包括形状（shape）和数据类型（dtype）。

- 形状：Tensor 的 shape，是一个 tuple。
- 数据类型：Tensor 的 dtype，是 MindSpore 的一个数据类型。

代码：

```
x = Tensor(np.array([[1, 2], [3, 4]]), dtype.int32)

x.shape # 形状
x.dtype # 数据类型
x.ndim  # 维度
x.size  # 大小
```

输出：

```
(2, 2)
mindspore.int32
2
4
```

### 2.2.1.3 Tensor 的方法

asnumpy()：将 Tensor 转换为 NumPy 的 array。

代码：

```
y = Tensor(np.array([[True, True], [False, False]]), dtype.bool_)
```

```
# 将 Tensor 数据类型转换成 NumPy
y_array = y.asnumpy()

y
y_array
```

输出：

```
Tensor(shape=[2, 2], dtype=Bool, value=
[[ True,  True],
 [False, False]])

array([[ True,  True],
       [False, False]])
```

### 2.2.1.4 Tensor 的运算

张量之间有很多运算，包括算术、线性代数、矩阵处理（转置、标引、切片）、采样等，下面介绍其中几种操作，张量运算和 NumPy 的使用方式类似。

#### 步骤 1 索引和切片

代码：

```
tensor = Tensor(np.array([[0, 1], [2, 3]]).astype(np.float32))
print("First row: {}".format(tensor[0]))
print("First column: {}".format(tensor[:, 0]))
print("Last column: {}".format(tensor[:, -1]))
```

输出：

```
First row: [0. 1.]
First column: [0. 2.]
Last column: [1. 3.]
```

#### 步骤 2 Tensor 拼接

代码：

```
data1 = Tensor(np.array([[0, 1], [2, 3]]).astype(np.float32))
data2 = Tensor(np.array([[4, 5], [6, 7]]).astype(np.float32))
op = ops.Stack()
output = op([data1, data2])
print(output)
```

输出：

```
[[[0. 1.]
   [2. 3.]]

  [[4. 5.]
   [6. 7.]]]
```

### 步骤3 转换为 Numpy

代码：

```
zeros = ops.Zeros()
output = zeros((2,2), dtype.float32)
print("output: {}".format(type(output)))
n_output = output.asnumpy()
print("n_output: {}".format(type(n_output)))
```

输出：

```
output: <class 'mindspore.common.tensor.Tensor'>
n_output: <class 'numpy.ndarray'>
```

## 2.2.2 数据集加载

MindSpore.dataset 提供 API 来加载和处理各种常见的数据集，如 MNIST, CIFAR-10, CIFAR-100, VOC, ImageNet, CelebA 等。

### 步骤1 加载 MNIST 数据集

加载数据集之前，建议先从以下链接 <http://yann.lecun.com/exdb/mnist/>，下载 MNIST 数据集，并将训练文件和测试文件放到 MNIST 文件夹下面。

代码：

```
import os
import mindspore.dataset as ds
import matplotlib.pyplot as plt

dataset_dir = "./MNIST/train" # 数据集路径
# 从 mnist dataset 读取 3 张图片
mnist_dataset = ds.MnistDataset(dataset_dir=dataset_dir, num_samples=3)
# 查看图像，设置图像大小
plt.figure(figsize=(8,8))
i = 1

# 打印 3 张子图
for dic in mnist_dataset.create_dict_iterator(output_numpy=True):
    plt.subplot(3,3,i)
    plt.imshow(dic['image'][:, :, 0])
    plt.axis('off')
    i += 1
plt.show()
```

输出：



图2-1 MNIST 数据集样本

## 步骤 2 自定义数据集

对于目前 MindSpore 不支持直接加载的数据集，可以构造自定义数据集类，然后通过 GeneratorDataset 接口实现自定义方式的数据加载。

代码：

```
import numpy as np
np.random.seed(58)

class DatasetGenerator:
    #实例化数据集对象时，__init__函数被调用，用户可以在此进行数据初始化等操作。
    def __init__(self):
        self.data = np.random.sample((5, 2))
        self.label = np.random.sample((5, 1))
    #定义数据集类的__getitem__函数，使其支持随机访问，能够根据给定的索引值 index，获取数据集中的数据并返回。
    def __getitem__(self, index):
        return self.data[index], self.label[index]
    #定义数据集类的__len__函数，返回数据集的样本数量。
    def __len__(self):
        return len(self.data)
    #定义数据集类之后，就可以通过 GeneratorDataset 接口按照用户定义的方式加载并访问数据集样本。
dataset_generator = DatasetGenerator()
dataset = ds.GeneratorDataset(dataset_generator, ["data", "label"], shuffle=False)
#通过 create_dict_iterator 方法获取数据
for data in dataset.create_dict_iterator():
    print('{}{}'.format(data["data"], '{}'.format(data["label"])))
```

输出：

```
[0.36510558 0.45120592] [0.78888122]
[0.49606035 0.07562207] [0.38068183]
[0.57176158 0.28963401] [0.16271622]
[0.30880446 0.37487617] [0.54738768]
[0.81585667 0.96883469] [0.77994068]
```

## 步骤 3 数据增强

MindSpore 提供的数据集接口具备常用的数据处理方法，如：打乱、设置 batch 等。用户只需调用相应的函数接口即可快速进行数据处理。

下面的样例先将数据集随机打乱顺序，然后将样本两两组成一个批次。

代码：

```
ds.config.set_seed(58)

# 随机打乱数据顺序，buffer_size 表示数据集中进行 shuffle 操作的缓存区的大小。
dataset = dataset.shuffle(buffer_size=10)
# 对数据集进行分批，batch_size 表示每组包含的数据个数，现设置每组包含 2 个数据。
dataset = dataset.batch(batch_size=2)

for data in dataset.create_dict_iterator():
    print("data: {}".format(data["data"]))
    print("label: {}".format(data["label"]))
```

输出：

```
data: [[0.36510558 0.45120592]
       [0.57176158 0.28963401]]
label: [[0.78888122]
        [0.16271622]]
data: [[0.30880446 0.37487617]
       [0.49606035 0.07562207]]
label: [[0.54738768]
        [0.38068183]]
data: [[0.81585667 0.96883469]]
label: [[0.77994068]]
```

代码：

```
import matplotlib.pyplot as plt

from mindspore.dataset.vision import Inter
import mindspore.dataset.vision.c_transforms as c_vision

DATA_DIR = './MNIST/train'
#取出 6 个样本
mnist_dataset = ds.MnistDataset(DATA_DIR, num_samples=6, shuffle=False)
# 查看数据原图
mnist_it = mnist_dataset.create_dict_iterator()
data = next(mnist_it)
plt.imshow(data['image'].asnumpy().squeeze(), cmap=plt.cm.gray)
plt.title(data['label'].asnumpy(), fontsize=20)
plt.show()
```

输出：

---

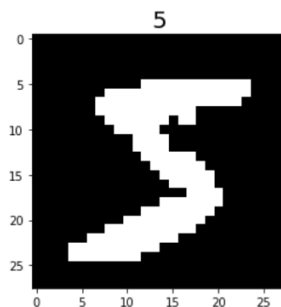


图2-2 数据样本

代码：

```
resize_op = c_vision.Resize(size=(40,40), interpolation=Inter.LINEAR)
crop_op = c_vision.RandomCrop(28)
transforms_list = [resize_op, crop_op]
mnist_dataset = mnist_dataset.map(operations=transforms_list, input_columns=["image"])
mnist_dataset = mnist_dataset.create_dict_iterator()
data = next(mnist_dataset)
plt.imshow(data['image'].asnumpy().squeeze(), cmap=plt.cm.gray)
plt.title(data['label'].asnumpy(), fontsize=20)
plt.show()
```

输出：

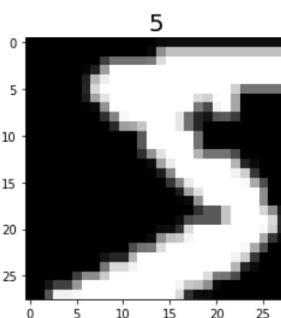


图2-3 数据增强后的效果

## 2.2.3 网络构建

MindSpore 将构建网络层的接口封装在 nn 模块中，我们将通过调用来构建不同类型的神经网络层。

### 步骤 1 全连接层

全连接层：mindspore.nn.Dense

- in\_channels：输入通道
- out\_channels：输出通道
- weight\_init：权重初始化，Default 'normal'.

代码：

```
import mindspore as ms
import mindspore.nn as nn
from mindspore import Tensor
import numpy as np

# 构造输入张量
input_a = Tensor(np.array([[1, 1, 1], [2, 2, 2]]), ms.float32)
print(input_a)

# 构造全连接网络，输入通道为 3，输出通道为 3
net = nn.Dense(in_channels=3, out_channels=3, weight_init=1)
output = net(input_a)
print(output)
```

输出：

```
[[1. 1. 1.]
 [2. 2. 2.]
 [3. 3. 3.]
 [6. 6. 6.]]
```

## 步骤 2 卷积层

代码：

```
conv2d = nn.Conv2d(1, 6, 5, has_bias=False, weight_init='normal', pad_mode='valid')
input_x = Tensor(np.ones([1, 1, 32, 32]), ms.float32)

print(conv2d(input_x).shape)
```

输出：

```
(1, 6, 28, 28)
```

## 步骤 3 ReLU 层

代码：

```
relu = nn.ReLU()
input_x = Tensor(np.array([-1, 2, -3, 2, -1]), ms.float16)
output = relu(input_x)

print(output)
```

输出：

```
[0. 2. 0. 2. 0.]
```

## 步骤 4 池化层

代码：

```
max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
input_x = Tensor(np.ones([1, 6, 28, 28]), ms.float32)

print(max_pool2d(input_x).shape)
```

输出：

```
(1, 6, 14, 14)
```

## 步骤5 Flatten 层

代码：

```
flatten = nn.Flatten()
input_x = Tensor(np.ones([1, 16, 5, 5]), ms.float32)
output = flatten(input_x)

print(output.shape)
```

输出：

```
(1, 400)
```

## 步骤6 定义模型类并查看参数

MindSpore 的 Cell 类是构建所有网络的基类，也是网络的基本单元。当用户需要神经网络时，需要继承 Cell 类，并重写\_\_init\_\_方法和 construct 方法。

代码：

```
class LeNet5(nn.Cell):
    """
    Lenet 网络结构
    """
    def __init__(self, num_class=10, num_channel=1):
        super(LeNet5, self).__init__()
        # 定义所需要的运算
        self.conv1 = nn.Conv2d(num_channel, 6, 5, pad_mode='valid')
        self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
        self.fc1 = nn.Dense(16 * 4 * 4, 120)
        self.fc2 = nn.Dense(120, 84)
        self.fc3 = nn.Dense(84, num_class)
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()

    def construct(self, x):
        # 使用定义好的运算构建前向网络
        x = self.conv1(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
```

```
x = self.conv2(x)
x = self.relu(x)
x = self.max_pool2d(x)
x = self.flatten(x)
x = self.fc1(x)
x = self.relu(x)
x = self.fc2(x)
x = self.relu(x)
x = self.fc3(x)
return x

#实例化模型，利用 parameters_and_names 方法查看模型的参数
modelle = LeNet5()
for m in modelle.parameters_and_names():
    print(m)
```

输出：

```
('conv1.weight', Parameter (name=conv1.weight, shape=(6, 1, 5, 5), dtype=Float32, requires_grad=True))
('conv2.weight', Parameter (name=conv2.weight, shape=(16, 6, 5, 5), dtype=Float32, requires_grad=True))
('fc1.weight', Parameter (name=fc1.weight, shape=(120, 400), dtype=Float32, requires_grad=True))
('fc1.bias', Parameter (name=fc1.bias, shape=(120,), dtype=Float32, requires_grad=True))
('fc2.weight', Parameter (name=fc2.weight, shape=(84, 120), dtype=Float32, requires_grad=True))
('fc2.bias', Parameter (name=fc2.bias, shape=(84,), dtype=Float32, requires_grad=True))
('fc3.weight', Parameter (name=fc3.weight, shape=(10, 84), dtype=Float32, requires_grad=True))
('fc3.bias', Parameter (name=fc3.bias, shape=(10,), dtype=Float32, requires_grad=True))
```

## 2.2.4 模型训练与评估

### 步骤 1 损失函数

损失函数用来评价模型的预测值和真实值不一样的程度，在这里，使用绝对误差损失函数 L1Loss。mindspore.nn.loss 也提供了许多其他常用的损失函数，如 SoftmaxCrossEntropyWithLogits、MSELoss、SmoothL1Loss 等。

我们给定输出值和目标值，计算损失值，使用方法如下所示：

代码：

```
import numpy as np
import mindspore.nn as nn
from mindspore import Tensor
import mindspore.dataset as ds
import mindspore as ms

loss = nn.L1Loss()
output_data = Tensor(np.array([[1, 2, 3], [2, 3, 4]]).astype(np.float32))
target_data = Tensor(np.array([[0, 2, 5], [3, 1, 1]]).astype(np.float32))
print(loss(output_data, target_data))
```

输出：

1.5

## 步骤 2 优化器

深度学习优化算法大概常用的有 SGD、Adam、Ftrl、lazyadam、Momentum、RMSprop、Lars、Proximal\_ada\_grad 和 lamb 这几种。

动量优化器：mindspore.nn.Momentum

代码

```
optim = nn.Momentum(params=model.trainable_params(), learning_rate=0.1, momentum=0.9, weight_decay=0.0)
```

## 步骤 3 模型编译

`mindspore.Model(network, loss_fn, optimizer, metrics)`

代码：

```
from mindspore import Model

# 定义神经网络
net = LeNet5()
# 定义损失函数
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
# 定义优化器
optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
# 模型编译
model = Model(network = net, loss_fn=loss, optimizer=optim, metrics={'accuracy'})
```

## 步骤 4 模型训练

代码：

```
import mindspore.dataset.transforms.c_transforms as C
import mindspore.dataset.vision.c_transforms as CV
from mindspore.train.callback import LossMonitor

DATA_DIR = './MNIST/train'
mnist_dataset = ds.MnistDataset(DATA_DIR)

resize_op = CV.Resize((28,28))
rescale_op = CV.Rescale(1/255,0)
hwc2chw_op = CV.HWC2CHW()

mnist_dataset = mnist_dataset.map(input_columns="image", operations=[rescale_op,resize_op, hwc2chw_op])
mnist_dataset = mnist_dataset.map(input_columns="label", operations=C.TypeCast(ms.int32))
mnist_dataset = mnist_dataset.batch(32)
loss_cb = LossMonitor(per_print_times=1000)
```

```
# dataset 是传入参数代表训练集，epoch 是训练集迭代轮数
model.train(epoch=1, train_dataset=mnist_dataset,callbacks=[loss_cb])
```

## 步骤 5 模型评估

代码：

```
# 测试集
DATA_DIR = './MNIST/test'
dataset = ds.MnistDataset(DATA_DIR)

resize_op = CV.Resize((28,28))
rescale_op = CV.Rescale(1/255,0)
hwc2chw_op = CV.HWC2CHW()

dataset = dataset.map(input_columns="image", operations=[rescale_op,resize_op, hwc2chw_op])
dataset = dataset.map(input_columns="label", operations=C.TypeCast(ms.int32))
dataset = dataset.batch(32)
model.eval(valid_dataset=dataset)
```

## 2.2.5 模型保存与加载

在上述网络模型训练完成后，可以通过两种形式对模型进行保存：

1. 简单的对网络模型进行保存，可以在训练前后进行保存。这种方式的优点是接口简单易用，但是只保留执行命令时候的网络模型状态；

代码：

```
import mindspore as ms

# 定义的网络模型为 net，一般在训练前或者训练后使用
ms.save_checkpoint(net, "./MyNet.ckpt") # net 为训练网络，"./MyNet.ckpt"为网络模型的保存路径。
```

输出：

```
epoch: 1 step: 1000, loss is 2.283555746078491
```

2. 在网络模型训练中进行保存，MindSpore 在网络模型训练的过程中，自动保存训练时候设定好的 epoch 数和 step 数的参数，也就是把模型训练过程中产生的中间权重参数也保存下来，方便进行网络微调和停止训练。

代码：

```
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig

# 设置 epoch_num 数量
epoch_num = 5

# 设置模型保存参数
```

```
config_ck = CheckpointConfig(save_checkpoint_steps=1875, keep_checkpoint_max=10)

# 应用模型保存参数
ckptpoint = ModelCheckpoint(prefix="lenet", directory="./lenet", config=config_ck)
model.train(epoch_num, mnist_dataset, callbacks=[ckptpoint])
```

## 2.2.6 自动微分

在训练神经网络时，最常用的算法是反向传播，在该算法中，根据损失函数对于给定参数的梯度来调整参数（模型权重）。

MindSpore 计算一阶导数方法 `mindspore.ops.GradOperation` (`get_all=False`, `get_by_list=False`, `sens_param=False`)，其中 `get_all` 为 `False` 时，只会对第一个输入求导，为 `True` 时，会对所有输入求导；`get_by_list` 为 `False` 时，不会对权重求导，为 `True` 时，会对权重求导；`sens_param` 对网络的输出值做缩放以改变最终梯度。

下面用 `MatMul` 算子的求导做深入分析。

### 步骤 1 对输入求一阶导数

如果需要对输入进行求导，首先需要定义一个需要求导的网络，以一个由 `MatMul` 算子构成的网络  $f(x,y)=z*x*y$  为例。

代码：

```
import numpy as np
import mindspore.nn as nn
import mindspore.ops as ops
from mindspore import Tensor
from mindspore import ParameterTuple, Parameter
from mindspore import dtype as mstype

class Net(nn.Cell):
    def __init__(self):
        super(Net, self).__init__()
        self.matmul = ops.MatMul()
        self.z = Parameter(Tensor(np.array([1.0], np.float32)), name='z')

    def construct(self, x, y):
        x = x * self.z
        out = self.matmul(x, y)
        return out

class GradNetWrtX(nn.Cell):
    def __init__(self, net):
        super(GradNetWrtX, self).__init__()
        self.net = net
        self.grad_op = ops.GradOperation()
```

```
def construct(self, x, y):
    gradient_function = self.grad_op(self.net)
    return gradient_function(x, y)
x = Tensor([[0.8, 0.6, 0.2], [1.8, 1.3, 1.1]], dtype=mstype.float32)
y = Tensor([[0.11, 3.3, 1.1], [1.1, 0.2, 1.4], [1.1, 2.2, 0.3]], dtype=mstype.float32)
output = GradNetWrtX(Net())(x, y)
print(output)
```

输出：

```
[[4.5099998 2.7      3.6000001]
 [4.5099998 2.7      3.6000001]]
```

## 步骤 2 对权重求一阶导数

若需要对权重的求导，将 ops.GradientOperation 中的 get\_by\_list 设置为 True。若需要对某些权重不进行求导，则在定义求导网络时，对相应的权重中 requires\_grad 设置为 False。

代码：

```
class GradNetWrtX(nn.Cell):
    def __init__(self, net):
        super(GradNetWrtX, self).__init__()
        self.net = net
        self.params = ParameterTuple(net.trainable_params())
        self.grad_op = ops.GradientOperation(get_by_list=True)

    def construct(self, x, y):
        gradient_function = self.grad_op(self.net, self.params)
        return gradient_function(x, y)
output = GradNetWrtX(Net())(x, y)
print(output)
```

输出：

```
(Tensor(shape=[1], dtype=Float32, value= [ 2.15359993e+01]),)
```

## 2.2.7 思考题

创建两个 Tensor，t1 和 t2，通过以下代码 t1 能否正常创建？比较两个 Tensor 的输出是否一样，如果不一样有什么区别？

```
t1 = Tensor(np.array([[1.2, 2.2], [3.2, 4.2]]), dtype=int32)
t2 = Tensor(np.array([[1.2, 2.2], [3.2, 4.2]]), dtype=float32)
```

## 2.3 MNIST 手写体字符实验

### 步骤 1 实验准备

在动手进行实践之前，确保你已经正确安装了 MindSpore。如果没有，可以通过 MindSpore 官网安装页面：<https://www.mindspore.cn/install/>，将 MindSpore 安装在你的电脑当中。

同时希望你拥有 Python 编程基础和概率、矩阵等基础数学知识。

推荐环境：

版本：MindSpore 2.2 及以上

编程语言：Python 3.7 及以上

### 步骤 2 数据准备

我们示例中用到的 MNIST 数据集是由 10 类 28\*28 的灰度图片组成，训练数据集包含 60000 张图片，测试数据集包含 10000 张图片。

目录结构如下：

```
└─ MNIST
    │
    │   test
    │   │   t10k-images.idx3-ubyte
    │   │   t10k-labels.idx1-ubyte
    │   │
    │   └─ train
    │       │   train-images.idx3-ubyte
    │       │   train-labels.idx1-ubyte
```

### 步骤 3 导入 Python 库&模块并配置运行信息

在使用前，导入需要的 Python 库。

代码：

```
#导入相关依赖库
import os
from matplotlib import pyplot as plt
import numpy as np

import mindspore as ms
import mindspore.context as context
import mindspore.dataset as ds
import mindspore.dataset.transforms as C
```

```
import mindspore.dataset.vision as CV
from mindspore.nn.metrics import Accuracy

from mindspore import nn
from mindspore.train import Model
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor

context.set_context(mode=context.GRAPH_MODE, device_target='CPU')
```

本实验中我们的运行模式是图模式。根据实际情况配置硬件信息，譬如代码运行在 Ascend AI 处理器上，则 device\_target 选择 Ascend，代码运行在 CPU、GPU 同理。详细参数说明，请参见 context.set\_context 接口说明。

#### 步骤 4 数据读取

获取数据（需要使用 pip 安装 download 工具包：pip install download）。

代码：

```
# Download data from open datasets
from download import download

url = "https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/" \
      "notebook/datasets/MNIST_Data.zip"
path = download(url, "./", kind="zip", replace=True)
```

利用 MindSpore 的数据读取功能读取 MNIST 数据集，并查看训练集和测试集数据量、样本信息。

代码：

```
DATA_DIR_TRAIN = "MNIST_Data/train" # 训练集信息
DATA_DIR_TEST = "MNIST_Data/test" # 测试集信息

#读取数据
ds_train = ds.MnistDataset(DATA_DIR_TRAIN)
ds_test = ds.MnistDataset(DATA_DIR_TEST)

#显示数据集的相关特性
print('训练数据集数量: ',ds_train.get_dataset_size())
print('测试数据集数量: ',ds_test.get_dataset_size())
image=ds_train.create_dict_iterator().__next__()
print('图像长/宽/通道数: ',image['image'].shape)
print('一张图像的标签样式: ',image['label']) #一共 10 类，用 0-9 的数字表达类别。
```

输出：

```
训练数据集数量: 60000
测试数据集数量: 10000
图像长/宽/通道数: (28, 28, 1)
```

一张图像的标签样式： 0

## 步骤 5 数据处理

数据集对于训练非常重要，好的数据集可以有效提高训练精度和效率。在加载数据集前，我们通常会对数据集进行一些处理。

### 定义数据集及数据操作

我们定义一个函数 `create_dataset` 来创建数据集。在这个函数中，我们定义好需要进行的数据增强和处理操作：

- 读取数据集。
- 定义进行数据增强和处理所需要的一些参数。
- 根据参数，生成对应的数据增强操作。
- 使用 `map` 映射函数，将数据操作应用到数据集。
- 对生成的数据集进行处理。

代码：

```
def datapipe(path, batch_size):
    image_transforms = [
        vision.Rescale(1.0 / 255.0, 0),
        vision.Normalize(mean=(0.1307,), std=(0.3081,)),
        vision.HWC2CHW()
    ]
    label_transform = transforms.TypeCast(ms.int32)

    dataset = ds.MnistDataset(path)
    dataset = dataset.map(image_transforms, 'image')
    dataset = dataset.map(label_transform, 'label')
    dataset = dataset.batch(batch_size)
    dataset = dataset.shuffle(buffer_size=32)
    return dataset

DS = datapipe(DATA_DIR_TEST, 32)
```

## 步骤 6 样本可视化

读取前 10 个样本，然后进行样本可视化，以此来确定样本是不是真实数据集。

代码：

```
#显示前 10 张图片以及对应标签,检查图片是否是正确的数据集
data = DS.create_dict_iterator().__next__()
images = data['image'].asnumpy()
labels = data['label'].asnumpy()
```

```
plt.figure(figsize=(15,5))
for i in range(1,11):
    plt.subplot(2, 5, i)
    plt.imshow(np.squeeze(images[i]))
    plt.title('Number: %s' % labels[i])
    plt.xticks([])
plt.show()
```

输出：

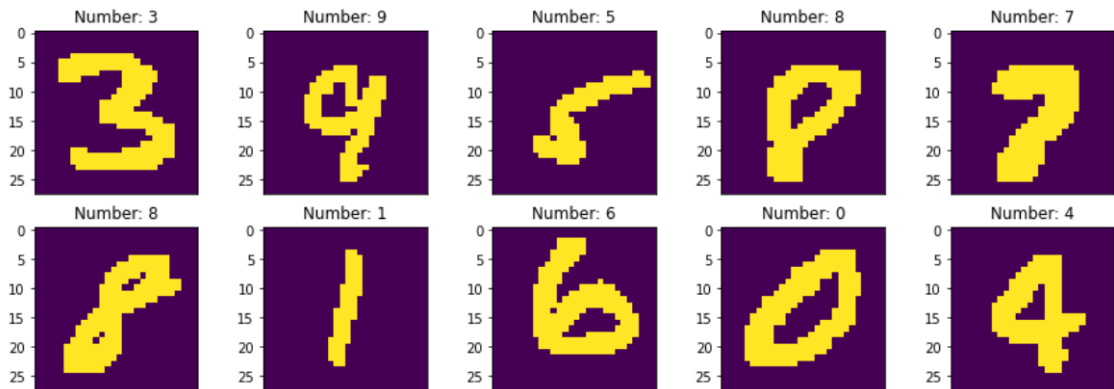


图2-4 样本可视化

## 步骤 7 定义网络

我们通过定义一个简单的全连接网络来完成图像识别，网络只有 3 层：

第一层全连接层，形状为 784\*512；

第二层全连接层，形状为 512\*128；

最后一层输出层，形状为 128\*10。

使用 MindSpore 定义神经网络需要继承 mindspore.nn.Cell。Cell 是所有神经网络（Conv2d 等）的基类。

神经网络的各层需要预先在\_\_init\_\_方法中定义，然后通过定义 construct 方法来完成神经网络的前向构造。定义网络各层如下：

代码：

```
#创建模型。模型包括 3 个全连接层，最后输出层使用 softmax 进行多分类，共分成（0-9）10 类
class Network(nn.Cell):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.dense_relu_sequential = nn.SequentialCell(
            nn.Dense(28*28, 512),
            nn.ReLU(),
            nn.Dense(512, 128),
            nn.ReLU(),
            nn.Dense(128, 10),
            nn.ReLU(),
            nn.Softmax(10))
```

```

        nn.Dense(512, 10)
    )

    def construct(self, x):
        x = self.flatten(x)
        logits = self.dense_relu_sequential(x)
        return logits

net = Network()

```

## 步骤 8 定义损失函数及优化器

**损失函数：**又叫目标函数，用于衡量预测值与实际值差异的程度。深度学习通过不停地迭代来缩小损失值。定义一个好的损失函数，可以有效提高模型的性能。

**优化器：**用于最小化损失函数，从而在训练过程中改进模型。

定义了损失函数后，可以得到损失函数关于权重的梯度。梯度用于指示优化器优化权重的方向，以提高模型性能。MindSpore 支持的损失函数有 SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss 等。这里使用 SoftmaxCrossEntropyWithLogits 损失函数。

MindSpore 提供了 callback 机制，可以在训练过程中执行自定义逻辑，这里使用框架提供的 ModelCheckpoint 为例。ModelCheckpoint 可以保存网络模型和参数，以便进行后续的 fine-tuning（微调）操作。

代码：

```

#创建网络，损失函数，评估指标 优化器，设定相关超参数
lr = 0.001
num_epoch = 10
momentum = 0.9
loss = nn.loss.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
metrics={"Accuracy": Accuracy()}
opt = nn.Adam(net.trainable_params(), lr)

```

## 步骤 9 开始训练

训练过程指的是将训练集数据传入网络对网络中的参数进行训练优化的过程。在 MindSpore 框架中 Model.train 方法用于完成这一过程。

代码：

```

#编译模型
model = Model(net, loss_fn=loss_fn, optimizer=optimizer, metrics={"Accuracy": Accuracy()})
config_ck = CheckpointConfig(save_checkpoint_steps=1875, keep_checkpoint_max=10)
ckpt_cb = ModelCheckpoint(prefix="checkpoint_net", directory = "./ckpt", config=config_ck)

#生成数据集
test_dataset = datapipe(DATA_DIR_TEST, batch_size=32)
train_dataset = datapipe(DATA_DIR_TRAIN, batch_size=32)

```

```
#训练模型
loss_cb = LossMonitor(per_print_times=1875)
time_cb = TimeMonitor(data_size=train_dataset.get_dataset_size())
print("===== Starting Training =====")
model.train(num_epoch, train_dataset, callbacks=[ckptpoint_cb, loss_cb, time_cb ], dataset_sink_mode=False)
```

训练过程中会打印 loss 值，类似下图。loss 值会波动，但总体来说 loss 值会逐步减小，精度逐步提高。每个人运行的 loss 值有一定随机性，不一定完全相同。训练过程中 loss 打印示例如下：

```
===== Starting Training =====
epoch: 1 step: 1875, loss is 0.06333521
epoch time: 18669.680 ms, per step time: 9.957 ms
epoch: 2 step: 1875, loss is 0.07061358
epoch time: 21463.662 ms, per step time: 11.447 ms
epoch: 3 step: 1875, loss is 0.043515638
epoch time: 25836.919 ms, per step time: 13.780 ms
epoch: 4 step: 1875, loss is 0.03468642
epoch time: 25553.150 ms, per step time: 13.628 ms
epoch: 5 step: 1875, loss is 0.03934026
epoch time: 27364.246 ms, per step time: 14.594 ms
epoch: 6 step: 1875, loss is 0.0023852987
epoch time: 31432.281 ms, per step time: 16.764 ms
epoch: 7 step: 1875, loss is 0.010915326
epoch time: 33697.183 ms, per step time: 17.972 ms
epoch: 8 step: 1875, loss is 0.011417691
epoch time: 29594.438 ms, per step time: 15.784 ms
epoch: 9 step: 1875, loss is 0.00044568744
epoch time: 28676.948 ms, per step time: 15.294 ms
epoch: 10 step: 1875, loss is 0.071476705
epoch time: 34999.863 ms, per step time: 18.667 ms
```

## 步骤 10 模型评估

本步骤利用原测试集去进行模型的评估。

代码：

```
#使用测试集评估模型，打印总体准确率
metrics=model.eval(ds_eval)
print(metrics)
```

输出：

```
{'Accuracy': 0.9812}
```

## 2.4 ResNet50 图像分类实验

本实验实现的是深度学习领域的经典案例——ResNet50 图像分类实验。整体流程如下：

- 处理需要的数据集，这里使用了花卉图像数据集。
- 定义一个网络，这里自己搭建 ResNet50 模型结构。
- 定义损失函数和优化器。
- 加载数据集并进行训练，训练完成后，利用测试集进行评估。

### 步骤 1 实验准备

在动手进行实践之前，确保你已经正确安装了 MindSpore。如果没有，可以通过 MindSpore 官网安装页面：<https://www.mindspore.cn/install/>，将 MindSpore 安装在你的电脑当中。

同时希望你拥有 Python 编码基础和概率、矩阵等基础数学知识。

推荐环境：

版本：MindSpore 2.2 及以上

编程语言：Python 3.7 及以上

### 步骤 2 数据准备

我们示例中用到的图像花卉数据集，该数据集是开源数据集，总共包括 5 种花的类型：分别是 daisy（雏菊，633 张），dandelion（蒲公英，898 张），roses（玫瑰，641 张），sunflowers（向日葵，699 张），tulips（郁金香，799 张），保存在 5 个文件夹当中，总共 3670 张，大小大概在 230M 左右。为了在模型部署上线之后进行测试，数据集在这里分成了 flower\_photos\_train 和 flower\_photos\_test 两部分。

目录结构如下（数据集中 flowers 文件夹下）：

flower\_photos\_train

- ├─ daisy
- ├─ dandelion
- ├─ roses
- ├─ sunflowers
- ├─ tulips
- └─ LICENSE.txt

flower\_photos\_test

- ├─ daisy
- └─ dandelion

---

- |— roses
- |— sunflowers
- |— tulips
- |— LICENSE.txt

新建文件夹 flowers，并下载数据文件：

```
https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower_photos_train.zip  
https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower_photos_test.zip
```

### 步骤3 导入 Python 库&模块并配置运行信息

在使用前，导入需要的 Python 库。

详细的 MindSpore 的模块说明，可以在 MindSpore API 页面中搜索查询。

可以通过 context.set\_context 来配置运行需要的信息，譬如运行模式、后端信息、硬件等信息。

导入 context 模块，配置运行需要的信息。

代码：

```
from easydict import EasyDict as edict  
# 字典访问，用来存储超参数  
import os  
# os 模块主要用于处理文件和目录  
import numpy as np  
# 科学计算库  
import matplotlib.pyplot as plt  
# 绘图库  
  
import mindspore  
# MindSpore 库  
import mindspore.dataset as ds  
# 数据集处理模块  
from mindspore.dataset.vision import c_transforms as vision  
# 图像增强模块  
from mindspore import context  
# 环境设置模块  
import mindspore.nn as nn  
# 神经网络模块  
from mindspore.train import Model  
# 模型编译  
from mindspore.nn.optim.momentum import Momentum  
# 动量优化器
```

```
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor
# 模型保存设置
from mindspore import Tensor
# 张量
from mindspore.train.serialization import export
# 模型导出
from mindspore.train.loss_scale_manager import FixedLossScaleManager
# 损失值平滑处理
from mindspore.train.serialization import load_checkpoint, load_param_into_net
# 模型加载
import mindspore.ops as ops
# 常见算子操作

# 设置 MindSpore 的执行模式和设备
context.set_context(mode=context.GRAPH_MODE, device_target="CPU")
```

#### 步骤 4 定义参数变量

edict 中存放的是模型训练和测试中所需要的各种参数配置。

代码：

```
cfg = edict({
    'data_path': 'flowers/flower_photos_train', #训练数据集路径
    'test_path': 'flowers/flower_photos_train', #测试数据集路径
    'data_size': 3616,
    'HEIGHT': 224, # 图片高度
    'WIDTH': 224, # 图片宽度
    '_R_MEAN': 123.68, # CIFAR10 的均值
    '_G_MEAN': 116.78,
    '_B_MEAN': 103.94,
    '_R_STD': 1, # 自定义的标准差
    '_G_STD': 1,
    '_B_STD': 1,
    '_RESIZE_SIDE_MIN': 256, # 图像增强 resize 最小值
    '_RESIZE_SIDE_MAX': 512,

    'batch_size': 32, # 批次大小
    'num_class': 5, # 分类类别
    'epoch_size': 5, # 训练次数
    'loss_scale_num': 1024,

    'prefix': 'resnet-ai', # 模型保存的名称
    'directory': './model_resnet', # 模型保存的路径
```

```
'save_checkpoint_steps': 10, # 每隔 10 步保存 ckpt
})
```

## 步骤 5 数据的读取和处理

数据集对于训练非常重要，好的数据集可以有效提高训练精度和效率。在加载数据集前，我们通常会对数据集进行一些处理。

### 定义数据集及数据操作

我们定义一个函数 `create_dataset` 来创建数据集。在这个函数中，我们定义好需要进行的数据增强和处理操作：

- 读取数据集。
- 定义进行数据增强和处理所需要的一些参数。
- 根据参数，生成对应的数据增强操作。
- 使用 `map` 映射函数，将数据操作应用到数据集。
- 对生成的数据集进行处理。
- 对处理好的数据进行样例展示。

代码：

```
# 数据处理
def read_data(path,config,usage="train"):
    # 从目录中读取图像的源数据集。
    dataset = ds.ImageFolderDataset(path,
                                    class_indexing={'daisy':0,'dandelion':1,'roses':2,'sunflowers':3,'tulips':4})

    # define map operations
    # 图像解码算子
    decode_op = vision.Decode()
    # 图像正则化算子
    normalize_op = vision.Normalize(mean=[cfg._R_MEAN, cfg._G_MEAN, cfg._B_MEAN], std=[cfg._R_STD,
cfg._G_STD, cfg._B_STD])
    # 图像调整大小算子
    resize_op = vision.Resize(cfg._RESIZE_SIDE_MIN)
    # 图像裁剪算子
    center_crop_op = vision.CenterCrop((cfg.HEIGHT, cfg.WIDTH))
    # 图像随机水平翻转算子
    horizontal_flip_op = vision.RandomHorizontalFlip()
    # 图像通道数转换算子
    channelswap_op = vision.HWC2CHW()
    # 图像随机裁剪解码编码调整大小算子
```

```

random_crop_decode_resize_op = vision.RandomCropDecodeResize((cfg.HEIGHT, cfg.WIDTH), (0.5, 1.0), (1.0, 1.0), max_attempts=100)

# 只对训练集做的预处理操作
if usage == 'train':
    dataset = dataset.map(input_columns="image", operations=random_crop_decode_resize_op)
    dataset = dataset.map(input_columns="image", operations=horizontal_flip_op)
# 只对测试集做的预处理操作
else:
    dataset = dataset.map(input_columns="image", operations=decode_op)
    dataset = dataset.map(input_columns="image", operations=resize_op)
    dataset = dataset.map(input_columns="image", operations=center_crop_op)

# 对全部数据集做的预处理操作
dataset = dataset.map(input_columns="image", operations=normalize_op)
dataset = dataset.map(input_columns="image", operations=channelswap_op)

# 对训练集做的批次处理
if usage == 'train':
    dataset = dataset.shuffle(buffer_size=10000) # 10000 as in imageNet train script
    dataset = dataset.batch(cfg.batch_size, drop_remainder=True)
# 对测试集做的批次处理
else:
    dataset = dataset.batch(1, drop_remainder=True)

# 数据增强
dataset = dataset.repeat(1)

dataset.map_model = 4

return dataset

# 查看训练集和测试集的数量
de_train = read_data(cfg.data_path, cfg, usage="train")
de_test = read_data(cfg.test_path, cfg, usage="test")
print('训练数据集数量: ', de_train.get_dataset_size()*cfg.batch_size) # get_dataset_size()获取批处理的大小。
print('测试数据集数量: ', de_test.get_dataset_size())

# 查看训练集的样图
data_next = de_train.create_dict_iterator(output_numpy=True).__next__()
print('通道数/图像长/宽: ', data_next['image'][0,...].shape)
print('一张图像的标签样式: ', data_next['label'][0]) # 一共 5 类, 用 0-4 的数字表达类别。

plt.figure()
plt.imshow(data_next['image'][0,0,...])

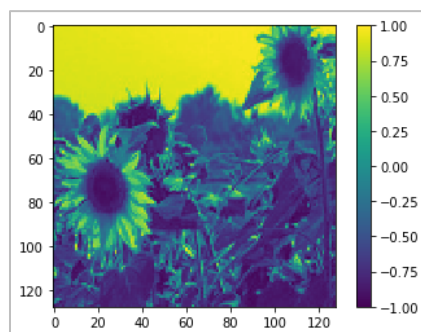
```

```
plt.colorbar()
plt.grid(False)
plt.show()
```

输出：

```
训练数据集数量： 3616
测试数据集数量： 32
通道数/图像长/宽： (3, 224, 224)
一张图像的标签样式： 3
```

数据处理展示：



## 步骤 6 模型构建训练

### ● 定义模型

残差块 (Residual Block)

将前面若干层的输出跳过中间层作为后几层的输入部分，也就是说后面特征层会有前几层的部分线性贡献。此种设计是为了克服随着网络层数加深而产生的学习效率变低和准确率无法有效提升的问题。

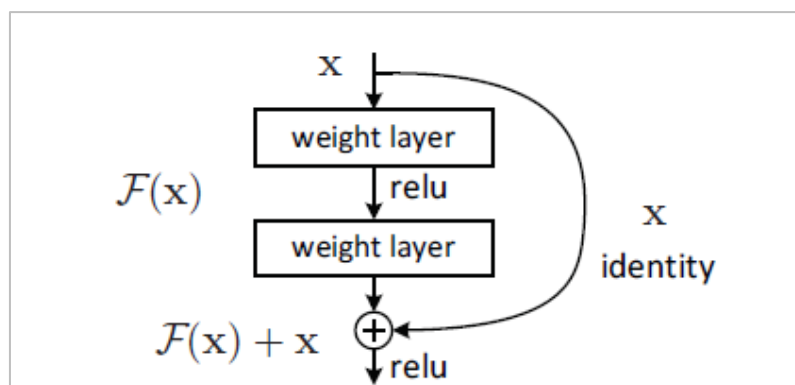


图2-5 残差块

如果维度相同：

$$y = F(x, W_i) + x$$

$$F = W_2 \sigma(W, x)$$

如果维度不同，则采用：

$$y = F(x, W_i) + W_s x$$

瓶颈（BottleNeck）模块：

瓶颈（BottleNeck）模块，思路和 Inception 模块一样，通过 1x1 conv 来巧妙地缩减或扩张 feature map 维度从而使 3x3 conv 的 filters 数目不受外界即上一层输入的影响，自然它的输出也不会影响到下一层 module。

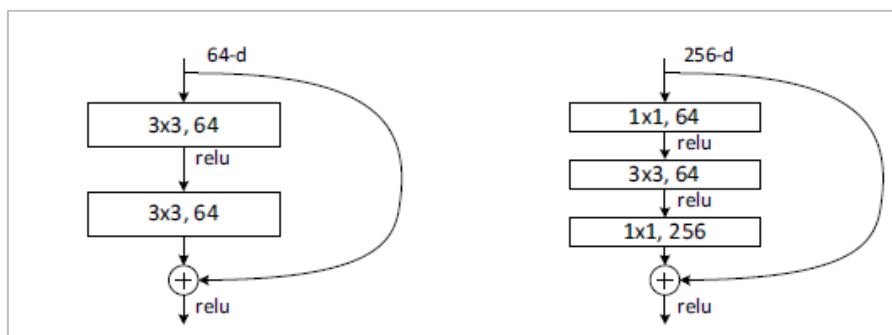


图2-6 BottleNeck

ResNet50 模型构建

ResNet50 有两个基本的块，分别名为 Conv Block 和 Identity Block，其中 Conv Block 输入和输出的维度是不一样的，所以不能连续串联，它的作用是改变网络的维度；Identity Block 输入维度和输出维度相同，可以串联，用于加深网络。

代码：

```

"""ResNet."""

# 定义权重初始化函数
def _weight_variable(shape, factor=0.01):
    init_value = np.random.randn(*shape).astype(np.float32) * factor
    return Tensor(init_value)

# 定义 3x3 卷积函数
def _conv3x3(in_channel, out_channel, stride=1):
    weight_shape = (out_channel, in_channel, 3, 3)
    weight = _weight_variable(weight_shape)
    return nn.Conv2d(in_channel, out_channel,
                     kernel_size=3, stride=stride, padding=0, pad_mode='same', weight_init=weight)

# 定义 1x1 卷积层函数
def _conv1x1(in_channel, out_channel, stride=1):
    weight_shape = (out_channel, in_channel, 1, 1)
    weight = _weight_variable(weight_shape)
    return nn.Conv2d(in_channel, out_channel,

```

```

        kernel_size=1, stride=stride, padding=0, pad_mode='same', weight_init=weight)

# 定义 7x7 卷积层函数
def _conv7x7(in_channel, out_channel, stride=1):
    weight_shape = (out_channel, in_channel, 7, 7)
    weight = _weight_variable(weight_shape)
    return nn.Conv2d(in_channel, out_channel,
                     kernel_size=7, stride=stride, padding=0, pad_mode='same', weight_init=weight)

# 定义 Batch Norm 层函数
def _bn(channel):
    return nn.BatchNorm2d(channel, eps=1e-4, momentum=0.9,
                          gamma_init=1, beta_init=0, moving_mean_init=0, moving_var_init=1)

# 定义最后一层的 Batch Norm 函数
def _bn_last(channel):
    return nn.BatchNorm2d(channel, eps=1e-4, momentum=0.9,
                          gamma_init=0, beta_init=0, moving_mean_init=0, moving_var_init=1)

# 定义全连接层函数
def _fc(in_channel, out_channel):
    weight_shape = (out_channel, in_channel)
    weight = _weight_variable(weight_shape)
    return nn.Dense(in_channel, out_channel, has_bias=True, weight_init=weight, bias_init=0)

# 构建残差模块
class ResidualBlock(nn.Cell):
    """
    ResNet V1 residual block definition.

    Args:
        in_channel (int): Input channel.
        out_channel (int): Output channel.
        stride (int): Stride size for the first convolutional layer. Default: 1.

    Returns:
        Tensor, output tensor.

    Examples:
        >>> ResidualBlock(3, 256, stride=2)
    """
    expansion = 4 # conv2_x--conv5_x 中，前两层的卷积核的个数是第三层（也就是输出通道）的 4 分之
    一。

    def __init__(self, in_channel, out_channel, stride=1):
        super(ResidualBlock, self).__init__()

```

```
# 前两层的卷积核个数等于输出通道的四分之一
channel = out_channel // self.expansion

# 第一层卷积
self.conv1 = _conv1x1(in_channel, channel, stride=1)
self.bn1 = _bn(channel)

# 第二层卷积
self.conv2 = _conv3x3(channel, channel, stride=stride)
self.bn2 = _bn(channel)

# 第三层卷积，其中卷积核个数等于输出通道
self.conv3 = _conv1x1(channel, out_channel, stride=1)
self.bn3 = _bn_last(out_channel)

# Relu 激活层
self.relu = nn.ReLU()

self.down_sample = False

# 当步长不为 1、或输出通道不等于输入通道时，进行图像下采样，用来调整通道数
if stride != 1 or in_channel != out_channel:
    self.down_sample = True
    self.down_sample_layer = None
# 用 1X1 卷积调整通道数
if self.down_sample:
    self.down_sample_layer = nn.SequentialCell([_conv1x1(in_channel, out_channel, stride), # 1X1 卷积
                                                _bn(out_channel)]) # Batch Norm

# 加法算子
self.add = ops.Add()

# 构建残差块
def construct(self, x):
    # 输入
    identity = x

    # 第一层卷积 1X1
    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

    # 第二层卷积 3X3
    out = self.conv2(out)
    out = self.bn2(out)
```

```

        out = self.relu(out)

        # 第三层卷积 1X1
        out = self.conv3(out)
        out = self.bn3(out)

        # 改变网络的维度
        if self.down_sample:
            identity = self.down_sample_layer(identity)

        # 加上残差
        out = self.add(out, identity)

        # Relu 激活
        out = self.relu(out)

        return out

# 构建残差网络
class ResNet(nn.Cell):
    """
    ResNet architecture.

    Args:
        block (Cell): Block for network.
        layer_nums (list): Numbers of block in different layers.
        in_channels (list): Input channel in each layer.
        out_channels (list): Output channel in each layer.
        strides (list): Stride size in each layer.
        num_classes (int): The number of classes that the training images are belonging to.

    Returns:
        Tensor, output tensor.

    Examples:
        >>> ResNet(ResidualBlock,
        >>>         [3, 4, 6, 3],
        >>>         [64, 256, 512, 1024],
        >>>         [256, 512, 1024, 2048],
        >>>         [1, 2, 2, 2],
        >>>         10)
    """

    # 输入参数为：残差块，残差块重复数，输入通道，输出通道，步长，图像类别数
    def __init__(self, block, layer_nums, in_channels, out_channels, strides, num_classes):
        super(ResNet, self).__init__()
        if not len(layer_nums) == len(in_channels) == len(out_channels) == 4:
            raise ValueError("the length of layer_num, in_channels, out_channels list must be 4!")

```

```
# 第一层卷积，卷积核 7x7，输入通道 3，输出通道 64，步长 2
self.conv1 = _conv7x7(3, 64, stride=2)
self.bn1 = _bn(64)
self.relu = ops.ReLU()

# 3x3 池化层，步长 2
self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, pad_mode="same")

# conv2_x 残差块
self.layer1 = self._make_layer(block,
                                layer_nums[0],
                                in_channel=in_channels[0],
                                out_channel=out_channels[0],
                                stride=strides[0])

# conv3_x 残差块
self.layer2 = self._make_layer(block,
                                layer_nums[1],
                                in_channel=in_channels[1],
                                out_channel=out_channels[1],
                                stride=strides[1])

# conv4_x 残差块
self.layer3 = self._make_layer(block,
                                layer_nums[2],
                                in_channel=in_channels[2],
                                out_channel=out_channels[2],
                                stride=strides[2])

# conv5_x 残差块
self.layer4 = self._make_layer(block,
                                layer_nums[3],
                                in_channel=in_channels[3],
                                out_channel=out_channels[3],
                                stride=strides[3])

# 均值算子
self.mean = ops.ReduceMean(keep_dims=True)

# Flatten 层
self.flatten = nn.Flatten()

# 输出层
self.end_point = _fc(out_channels[3], num_classes)

# 输入参数为：残差块，残差块重复数，输入通道，输出通道，步长
def _make_layer(self, block, layer_num, in_channel, out_channel, stride):
    """
    Make stage network of ResNet.
```

Args:

block (Cell): Resnet block.  
layer\_num (int): Layer number.  
in\_channel (int): Input channel.  
out\_channel (int): Output channel.  
stride (int): Stride size for the first convolutional layer.

Returns:

SequentialCell, the output layer.

Examples:

```
>>> _make_layer(ResidualBlock, 3, 128, 256, 2)
"""
# 搭建 convn_x 的残差块

layers = []

resnet_block = block(in_channel, out_channel, stride=stride)
layers.append(resnet_block)

for _ in range(1, layer_num):
    resnet_block = block(out_channel, out_channel, stride=1)
    layers.append(resnet_block)

return nn.SequentialCell(layers)

# 构建 ResNet 网络
def construct(self, x):
    x = self.conv1(x) # 第一层卷积 7X7, 步长为 2
    x = self.bn1(x)   # 第一层的 Batch Norm
    x = self.relu(x)  # Rule 激活层
    c1 = self.maxpool(x) # 最大池化 3X3, 步长为 2

    c2 = self.layer1(c1) # conv2_x 残差块
    c3 = self.layer2(c2) # conv3_x 残差块
    c4 = self.layer3(c3) # conv4_x 残差块
    c5 = self.layer4(c4) # conv5_x 残差块

    out = self.mean(c5, (2, 3)) # 平均池化层
    out = self.flatten(out) # Flatten 层
    out = self.end_point(out) # 输出层

    return out
```

```
# 构建 ResNet50 网络
def resnet50(class_num=5):
    """
    Get ResNet50 neural network.

    Args:
        class_num (int): Class number.

    Returns:
        Cell, cell instance of ResNet50 neural network.

    Examples:
        >>> net = resnet50(10)
    """
    return ResNet(ResidualBlock, # 残差块
                  [3, 4, 6, 3], # 残差块数量
                  [64, 256, 512, 1024], # 输入通道
                  [256, 512, 1024, 2048], # 输出通道
                  [1, 2, 2, 2], # 步长
                  class_num) # 输出类别数
```

### ● 开始训练

完成数据预处理、网络定义、损失函数和优化器定义之后，开始模型训练。模型训练包含 2 层迭代，数据集的多轮迭代 epoch 和每一轮迭代按分组从数据集中抽取数据，输入网络计算得到损失函数，然后通过优化器计算和更新训练参数的梯度。

#### 1. 下载预训练模型

新建目录 model\_resnet，通过

[https://download.mindspore.cn/models/r1.7/resnet50\\_ascend\\_v170\\_imagenet2012\\_official\\_cv\\_top1acc76.97\\_top5acc93.44.ckpt](https://download.mindspore.cn/models/r1.7/resnet50_ascend_v170_imagenet2012_official_cv_top1acc76.97_top5acc93.44.ckpt) 下载已经在 ImageNet 数据集上已经训练好的预训练模型文件，并存放在指定目录 model\_resnet 下。

#### 2. 加载预训练模型

通过 load\_checkpoint() 接口读取预训练模型文件获取字典格式的参数文件。

#### 3. 修改预训练模型参数

修改预训练模型参数的最后一层连接参数（预训练模型是在 ImageNet 数据集上训练的 1001 分类任务，而我们当前实验任务是实现 flowers 数据集的 5 分类任务）。因此我们需要修改最后一层全连接层的参数。

代码：

```
# 构建 ResNet50 网络，输出类别数为 5，对应 5 种花的类别
net=resnet50(class_num=cfg.num_class)
```

```
#读取预训练模型参数
param_dict =
load_checkpoint("model_resnet/resnet50_ascend_v170_imagenet2012_official_cv_top1acc76.97_top5acc93.44.ckpt
")

#展示读取的模型参数
print(param_dict)

#通过 mindspore.Parameter () 修改 end_point.weight 和 end_point.bias 对应的 shape
param_dict["end_point.weight"] = mindspore.Parameter(Tensor(param_dict["end_point.weight"][:5, :],
mindspore.float32), name="variable")
param_dict["end_point.bias"] = mindspore.Parameter(Tensor(param_dict["end_point.bias"][:5, :], mindspore.float32),
name="variable")

# 设置 Softmax 交叉熵损失函数
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

# 设置学习率
train_step_size = de_train.get_dataset_size()
lr = nn.cosine_decay_lr(min_lr=0.0001, max_lr=0.001, total_step=train_step_size *
cfg.epoch_size, step_per_epoch=train_step_size, decay_epoch=cfg.epoch_size)

# 设置动量优化器
opt = Momentum(net.trainable_params(), lr, momentum=0.9, weight_decay=1e-4, loss_scale=cfg.loss_scale_num)

# 损失值平滑, 解决训练过程中梯度过小的问题
loss_scale = FixedLossScaleManager(cfg.loss_scale_num, False)

# 模型编译, 输入网络结构, 损失函数, 优化器, 损失值平滑, 以及模型评估标准
model = Model(net, loss_fn=loss, optimizer=opt, loss_scale_manager=loss_scale, metrics={'acc'})

# 损失值监控
loss_cb = LossMonitor(per_print_times=train_step_size)

# 模型保存参数, 设置每隔多少步保存一次模型, 最多保存几个模型
ckpt_config = CheckpointConfig(save_checkpoint_steps=cfg.save_checkpoint_steps, keep_checkpoint_max=1)

# 模型保存, 设置模型保存的名称, 路径, 以及保存参数
ckptpoint_cb = ModelCheckpoint(prefix=cfg.prefix, directory=cfg.directory, config=ckpt_config)

print("===== Starting Training =====")
# 训练模型, 设置训练次数, 训练集, 回调函数, 是否采用数据下沉模式 (只可应用于 Ascend 和 GPU, 可加
快训练速度)
model.train(cfg.epoch_size, de_train, callbacks=[loss_cb, ckptpoint_cb], dataset_sink_mode=True)
# 训练时长大约 15-20 分钟
```

```
# 使用测试集进行模型评估，输出测试集的准确率
metric = model.eval(de_test)
print(metric)
```

输出：

```
===== Starting Training =====
epoch: 1 step: 113, loss is 0.23505911231040955
epoch: 2 step: 113, loss is 0.11479882150888443
epoch: 3 step: 113, loss is 0.13273288309574127
epoch: 4 step: 113, loss is 0.42304447293281555
epoch: 5 step: 113, loss is 0.14625898003578186
{'acc': 0.9587912087912088}
```

## 步骤 7 模型预测

通过训练好的权重文件，调用 `model.predict()` 对测试数据进行测试，输出预测结果和真实结果。

代码：

```
# 模型预测，从测试集中取 10 个样本进行测试，输出预测结果和真实结果
class_names = {0:'daisy',1:'dandelion',2:'roses',3:'sunflowers',4:'tulips'}
for i in range(10):
    test_ = de_test.create_dict_iterator().__next__()
    test = Tensor(test_['image'], mindspore.float32)
    # 模型预测
    predictions = model.predict(test)
    predictions = predictions.asnumpy()
    true_label = test_['label'].asnumpy()
    # 显示预测结果
    p_np = predictions[0, :]
    pre_label = np.argmax(p_np)
    print('第' + str(i) + '个 sample 预测结果: ', class_names[pre_label], ' 真实结果: ',
          class_names[true_label[0]])
```

输出：

```
第 0 个 sample 预测结果: sunflowers 真实结果: sunflowers
第 1 个 sample 预测结果: daisy 真实结果: daisy
第 2 个 sample 预测结果: roses 真实结果: roses
第 3 个 sample 预测结果: roses 真实结果: roses
第 4 个 sample 预测结果: roses 真实结果: roses
第 5 个 sample 预测结果: dandelion 真实结果: dandelion
第 6 个 sample 预测结果: dandelion 真实结果: dandelion
```

|                    |           |       |           |
|--------------------|-----------|-------|-----------|
| 第 7 个 sample 预测结果: | roses     | 真实结果: | roses     |
| 第 8 个 sample 预测结果: | tulips    | 真实结果: | tulips    |
| 第 9 个 sample 预测结果: | dandelion | 真实结果: | dandelion |

## 步骤 8 思考题

本实验中用到了 ResNet50 网络，该网络共有多少层（只计算卷积与全连接层）？并指出本实验中对模型训练定义了多少个 Epoch？网络的输出类别数是多少？

## 2.5 TextCNN 情感分析实验

本实验实现的是深度学习领域的经典案例——TextCNN 情感分析实验。整体流程如下：

- 下载需要的数据集，这里使用了 rt-polarity 数据集，定义数据预处理函数。
- 生成数据用于训练和验证。
- 定义 TextCNN 模型结构搭建、训练、评估、加载离线模型、在线推理函数。
- 定义训练需要的各种参数优化器、损失函数、保存检查点、时间监视等。
- 加载数据集并进行训练，训练完成后，利用测试集进行评估。

### 步骤 1 实验准备

在动手进行实践之前，确保你已经正确安装了 MindSpore。如果没有，可以通过 MindSpore 官网安装页面：<https://www.mindspore.cn/install/>，将 MindSpore 安装在你的电脑当中。

同时希望你拥有 Python 编码基础和概率、矩阵等基础数学知识。

推荐环境：

版本：MindSpore 2.2 及以上

编程语言：Python 3.7 及以上

### 步骤 2 数据准备

实例中使用的 rt-polarity，是电影英文评论数据，rt-polarity.pos 里面存储的是 5000 电影英文好评，rt-polarity.neg 里面存的是 5000 条英文版差评（数据文件在数据集文件夹中的 TextCNN 中）。

数据目录结构如下：

```
└─data
    ├── rt-polarity.pos
    └── rt-polarity.neg
```

### 步骤 3 导入 Python 库&模块并配置运行信息

在使用前，导入需要的 Python 库。

目前使用到 os 库，为方便理解，其他需要的库，我们在具体使用到时再说明。详细的 MindSpore 的模块说明，可以在 MindSpore API 页面中搜索查询。可以通过 context.set\_context 来配置运行需要的信息，譬如运行模式、后端信息、硬件等信息。

导入 context 模块，配置运行需要的信息。

代码：

```
import math
import numpy as np
import pandas as pd
import os
import math
import random
import codecs
from pathlib import Path

import mindspore
import mindspore.dataset as ds
import mindspore.nn as nn
from mindspore import Tensor
from mindspore import context
from mindspore.train.model import Model
from mindspore.nn.metrics import Accuracy
from mindspore.train.serialization import load_checkpoint, load_param_into_net
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor
from mindspore.ops import operations as ops

from easydict import EasyDict as edict

cfg = edict({
    'name': 'movie review',
    'pre_trained': False,
    'num_classes': 2,
    'batch_size': 64,
    'epoch_size': 4,
    'weight_decay': 3e-5,
    'data_path': './data/',
    'device_target': 'CPU',
    'device_id': 0,
    'keep_checkpoint_max': 1,
    'checkpoint_path': './ckpt/train_textcnn-4_149.ckpt',
    'word_len': 51,
    'vec_length': 40
})

context.set_context(mode=context.GRAPH_MODE, device_target=cfg.device_target, device_id=cfg.device_id)
```

本实验中我们的运行模式是图模式。根据实际情况配置硬件信息，譬如代码运行在 Ascend AI 处理器上，则 device\_target 选择 Ascend，代码运行在 CPU、GPU 同理。详细参数说明，请参

见 context.set\_context 接口说明 ([https://www.mindspore.cn/docs/zh-CN/r1.7/api\\_python/mindspore.context.html?highlight=context](https://www.mindspore.cn/docs/zh-CN/r1.7/api_python/mindspore.context.html?highlight=context)) 。

## 步骤 4 数据读取和处理

数据预览代码：

```
with open("./data/rt-polarity.neg", 'r', encoding='utf-8') as f:
    print("Negative reivews:")
    for i in range(5):
        print("{}:{}".format(i,f.readline()))
with open("./data/rt-polarity.pos", 'r', encoding='utf-8') as f:
    print("Positive reivews:")
    for i in range(5):
        print("{}:{}".format(i,f.readline()))
```

数据预览输出：

```
Negative reivews:
[0]:simplistic , silly and tedious .

[1]:it's so laddish and juvenile , only teenage boys could possibly find it funny .

[2]:exploitative and largely devoid of the depth or sophistication that would make watching such a graphic treatment of the crimes bearable .

[3]:[garbus] discards the potential for pathological study , exhuming instead , the skewed melodrama of the circumstantial situation .

[4]:a visually flashy but narratively opaque and emotionally vapid exercise in style and mystification .

Positive reivews:
[0]:the rock is destined to be the 21st century's new " conan " and that he's going to make a splash even greater than arnold schwarzenegger , jean-claud van damme or stev en segal .

[1]:the gorgeously elaborate continuation of " the lord of the rings " trilogy is so huge that a column of words cannot adequately describe co-writer/director peter jackso n's expanded vision of j . r . r . tolkien's middle-earth .

[2]:effective but too-tepid biopic

[3]:if you sometimes like to go to the movies to have fun , wasabi is a good place to start .

[4]:emerges as something rare , an issue movie that's so honest and keenly observed that it doesn't feel like one .
```

定义数据处理函数代码：

```
#定义数据生成类
class Generator():
    def __init__(self, input_list):
        self.input_list=input_list
    def __getitem__(self,item):
        return (np.array(self.input_list[item][0],dtype=np.int32),
                np.array(self.input_list[item][1],dtype=np.int32))
    def __len__(self):
        return len(self.input_list)

class MovieReview:
    """
    影评数据集
    """
    def __init__(self, root_dir, maxlen, split):
        """
        input:
            root_dir: 影评数据目录
```

```
maxlen: 设置句子最大长度
split: 设置数据集中训练/评估的比例
'''
self.path = root_dir
self.feelMap = {
    'neg':0,
    'pos':1
}
self.files = []

self.doConvert = False

mypath = Path(self.path)
if not mypath.exists() or not mypath.is_dir():
    print("please check the root_dir!")
    raise ValueError

# 在数据目录中找到文件
for root,_filename in os.walk(self.path):
    for each in filename:
        self.files.append(os.path.join(root,each))
    break

# 确认是否为两个文件.neg 与.pos
if len(self.files) != 2:
    print("There are {} files in the root_dir".format(len(self.files)))
    raise ValueError

# 读取数据
self.word_num = 0
self.maxlen = 0
self.minlen = float("inf")
self.maxlen = float("-inf")
self.Pos = []
self.Neg = []
for filename in self.files:
    self.read_data(filename)

self.text2vec(maxlen=maxlen)
self.split_dataset(split=split)

def read_data(self, filePath):
    with open(filePath,'r') as f:
        for sentence in f.readlines():
```

```
sentence = sentence.replace('\n','')\
                    .replace('"','')\
                    .replace('\\','')\
                    .replace(' ','')\
                    .replace('[','')\
                    .replace(']','')\
                    .replace('(','')\
                    .replace(')','')\
                    .replace(':', '')\
                    .replace('--','')\
                    .replace('-', ' ')\
                    .replace('\\"','')\
                    .replace('0','')\
                    .replace('1','')\
                    .replace('2','')\
                    .replace('3','')\
                    .replace('4','')\
                    .replace('5','')\
                    .replace('6','')\
                    .replace('7','')\
                    .replace('8','')\
                    .replace('9','')\
                    .replace('`','')\
                    .replace('=' , '')\
                    .replace('$','')\
                    .replace('/', '')\
                    .replace('*', '')\
                    .replace(';','')\
                    .replace('<b>','')\
                    .replace('%','')

sentence = sentence.split(' ')
sentence = list(filter(lambda x: x, sentence))
if sentence:
    self.word_num += len(sentence)
    self.maxlen = self.maxlen if self.maxlen >= len(sentence) else len(sentence)
    self.minlen = self.minlen if self.minlen <= len(sentence) else len(sentence)
    if 'pos' in filePath:
        self.Pos.append([sentence,self.feelMap['pos']])
    else:
        self.Neg.append([sentence,self.feelMap['neg']])

def text2vec(self, maxlen):
    """
```

# 将句子转化为向量



```

"""
# Vocab = {word : index}
self.Vocab = dict()

# self.Vocab['None']
for SentenceLabel in self.Pos+self.Neg:
    vector = [0]*maxlen
    for index, word in enumerate(SentenceLabel[0]):
        if index >= maxlen:
            break
        if word not in self.Vocab.keys():
            self.Vocab[word] = len(self.Vocab)
            vector[index] = len(self.Vocab) - 1
        else:
            vector[index] = self.Vocab[word]
    SentenceLabel[0] = vector
self.doConvert = True

def split_dataset(self, split):

    trunk_pos_size = math.ceil((1-split)*len(self.Pos))
    trunk_neg_size = math.ceil((1-split)*len(self.Neg))
    trunk_num = int(1/(1-split))
    pos_temp=list()
    neg_temp=list()
    for index in range(trunk_num):
        pos_temp.append(self.Pos[index*trunk_pos_size:(index+1)*trunk_pos_size])
        neg_temp.append(self.Neg[index*trunk_neg_size:(index+1)*trunk_neg_size])
    self.test = pos_temp.pop(2)+neg_temp.pop(2)
    self.train = [i for item in pos_temp+neg_temp for i in item]

    random.shuffle(self.train)

def get_dict_len(self):
    """
    获得数据集中文字组成的词典长度
    """
    if self.doConvert:
        return len(self.Vocab)
    else:
        print("Haven't finished Text2Vec")
        return -1

def create_train_dataset(self, epoch_size, batch_size):
    dataset = ds.GeneratorDataset(
        source=Generator(input_list=self.train),
        column_names=["data", "label"],

```

```

        shuffle=False
    )

    dataset=dataset.batch(batch_size=batch_size,drop_remainder=True)
    dataset=dataset.repeat(epoch_size)
    return dataset

def create_test_dataset(self, batch_size):
    dataset = ds.GeneratorDataset(
        source=Generator(input_list=self.test),
        column_names=["data", "label"],
        shuffle=False
    )

    dataset=dataset.batch(batch_size=batch_size,drop_remainder=True)
    return dataset

```

通过自定义 `read_data` 函数加载原始数据集后使用 `text2vec` 函数对数据进行文本进行向量化处理。使用 `split_dataset` 将数据分割为训练数据和测试数据，然后 `create_test_dataset` 和 `create_train_dataset` 调用 `mindspore.dataset.GeneratorDataset` 生成数据来进行训练和验证。

config 配置代码:

```
instance = MovieReview(root_dir=cfg.data_path, maxlen=cfg.word_len, split=0.9)
dataset = instance.create_train_dataset(batch_size=cfg.batch_size, epoch_size=cfg.epoch_size)
batch_num = dataset.get_dataset_size()
```

显示数据处理结果代码:

```
vocab_size=instance.get_dict_len()
print("vocab_size:{0}".format(vocab_size))
item =dataset.create_dict_iterator()
for i,data in enumerate(item):
    if i<1:
        print(data)
        print(data['data'][1])
    else:
        break
```

输出数据处理结果:

[illegible]

### 步骤5 配置训练参数

学习率设置代码：

```
learning_rate = []
warm_up = [1e-3 / math.floor(cfg.epoch_size / 5) * (i + 1) for _ in range(batch_num)
            for i in range(math.floor(cfg.epoch_size / 5))]
shrink = [1e-3 / (16 * (i + 1)) for _ in range(batch_num)
          for i in range(math.floor(cfg.epoch_size * 3 / 5))]
normal_run = [1e-3 for _ in range(batch_num) for i in
              range(cfg.epoch_size - math.floor(cfg.epoch_size / 5)
                    - math.floor(cfg.epoch_size * 2 / 5))]
learning_rate = learning_rate + warm_up + normal_run + shrink
```

通过 epoch 实现训练中的动态学习率。

## 步骤 6 TextCNN 模型定义

模型类定义了模型结构搭建、训练、评估、加载离线模型、在线推理函数。

代码：

```
def _weight_variable(shape, factor=0.01):
    init_value = np.random.randn(*shape).astype(np.float32) * factor
    return Tensor(init_value)

def make_conv_layer(kernel_size):
    weight_shape = (96, 1, *kernel_size)
    weight = _weight_variable(weight_shape)
    return nn.Conv2d(in_channels=1, out_channels=96, kernel_size=kernel_size, padding=1,
                    pad_mode="pad", weight_init=weight, has_bias=True)

class TextCNN(nn.Cell):
    def __init__(self, vocab_len, word_len, num_classes, vec_length):
        super(TextCNN, self).__init__()
        self.vec_length = vec_length
        self.word_len = word_len
        self.num_classes = num_classes

        self.unsqueeze = ops.ExpandDims()
        self.embedding = nn.Embedding(vocab_len, self.vec_length, embedding_table='normal')

        self.slice = ops.Slice()
        self.layer1 = self.make_layer(kernel_height=3)
        self.layer2 = self.make_layer(kernel_height=4)
        self.layer3 = self.make_layer(kernel_height=5)

        self.concat = ops.Concat(1)

        self.fc = nn.Dense(96*3, self.num_classes)
        self.drop = nn.Dropout(keep_prob=0.5)
        self.print = ops.Print()
        self.reduce_mean = ops.ReduceMax(keep_dims=False)
```

```
def make_layer(self, kernel_height):
    return nn.SequentialCell(
        [
            make_conv_layer((kernel_height,self.vec_length)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(self.word_len-kernel_height+1,1)),
        ]
    )

def construct(self,x):
    x = self.unsqueeze(x, 1)
    x = self.embedding(x)
    x1 = self.layer1(x)
    x2 = self.layer2(x)
    x3 = self.layer3(x)

    x1 = self.reduce_mean(x1, (2, 3))
    x2 = self.reduce_mean(x2, (2, 3))
    x3 = self.reduce_mean(x3, (2, 3))

    x = self.concat((x1, x2, x3))
    x = self.drop(x)
    x = self.fc(x)
    return x

net = TextCNN(vocab_len=instance.get_dict_len(), word_len=cfg.word_len,
              num_classes=cfg.num_classes, vec_length=cfg.vec_length)
print(net)
```

查看神经网络结构:

```
TextCNN(  
    (embedding): Embedding(vocab_size=18848, embedding_size=40, use_one_hot=False, embedding_table=Parameter (name=embedding.embedding_table), dtype=Float32, padding_idx=None)  
)  
  
{layer1}: SequentialCell(  
    (0): Conv2d(input_channels=1, output_channels=96, kernel_size=(3, 40), stride=(1, 1), pad_mode=pad, padding=1, dilation=(1, 1), group=1, has_bias=True, weight_init=[[[[  
1.47466036e-02 -8.85174982e-03 6.20904262e-04 ... 5.97969582e-03  
6.87394897e-03 -4.35171695e-03  
[ 1.25721507e-02 1.11018270e-02 3.73373111e-03 ... -1.32954121e-02  
1.67019237e-02 6.70977589e-03  
[ 5.16406354e-03 6.65343972e-03 -9.12646390e-03 ... 4.12355090e-04  
6.96751568e-03 2.65645944e-02]]]])  
[[[ 2.20983545e-03 9.51934140e-03 5.41285099e-03 ... -2.79155909e-03  
-9.01016127e-03 1.36778364e-02]  
[-9.36647598e-03 -1.73871801e-03 1.08188950e-02 ... 1.51296274e-03  
7.57558912e-04 5.44300769e-03]  
[ 1.13182161e-02 4.26522718e-04 1.49796065e-03 ... -3.42557859e-03  
-1.79516233e-03 -4.79884632e-03]]]])  
[[[-4.98654880e-03 -2.08284725e-02 2.18074489e-02 ... -7.44788814e-03  
-4.96819383e-03 1.28578511e-03]  
[ 1.82666897e-03 1.37834558e-02 -4.40165540e-03 ... -8.17192066e-03  
9.86121874e-03 1.46957850e-02]  
[ 1.46543132e-02 7.15820771e-03 5.12730749e-03 ... -1.95071772e-02  
-1.01376360e-03 6.32047653e-03]]]])
```

### 步骤7 定义训练的相关参数

**损失函数：**又叫目标函数，用于衡量预测值与实际值差异的程度。深度学习通过不停地迭代来缩小损失值。定义一个好的损失函数，可以有效提高模型的性能。

**优化器**：用于最小化损失函数，从而在训练过程中改进模型。

定义了损失函数后，可以得到损失函数关于权重的梯度。梯度用于指示优化器优化权重的方向，以提高模型性能。MindSpore 支持的损失函数有 SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss 等。这里使用 SoftmaxCrossEntropyWithLogits 损失函数。

MindSpore 提供了 Callback 机制，可以在训练过程中执行自定义逻辑，这里以框架提供的 ModelCheckpoint 为例。ModelCheckpoint 可以保存网络模型和参数，以便进行后续的 fine-tuning（微调）操作。

代码：

```
# 优化器、损失函数、保存检查点、时间监视器等设置
opt = nn.Adam(filter(lambda x: x.requires_grad, net.get_parameters()),
               learning_rate=learning_rate, weight_decay=cfg.weight_decay)
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
model = Model(net, loss_fn=loss, optimizer=opt, metrics={'acc': Accuracy()})
config_ck = CheckpointConfig(save_checkpoint_steps=int(cfg.epoch_size*batch_num/2),
                              keep_checkpoint_max=cfg.keep_checkpoint_max)
time_cb = TimeMonitor(data_size=batch_num)
ckpt_save_dir = "./ckpt"
ckptpoint_cb = ModelCheckpoint(prefix="train_textcnn", directory=ckpt_save_dir, config=config_ck)
loss_cb = LossMonitor()
```

## 步骤 8 启动训练

训练过程指的是将训练集数据传入网络对网络中的参数进行训练优化的过程。在 MindSpore 框架中 Model.train 方法用于完成这一过程。

代码：

```
model.train(cfg.epoch_size, dataset, callbacks=[time_cb, ckptpoint_cb, loss_cb])
print("train success")
```

输出：

```
epoch: 1 step: 596, loss is 0.023292765
epoch time: 36818.071 ms, per step time: 61.775 ms
epoch: 2 step: 596, loss is 0.0016317479
epoch time: 4320.621 ms, per step time: 7.249 ms
epoch: 3 step: 596, loss is 0.00025401893
epoch time: 4271.667 ms, per step time: 7.167 ms
epoch: 4 step: 596, loss is 0.0001774891
epoch time: 4332.598 ms, per step time: 7.269 ms
train success
```

## 步骤 9 测试评估

本步骤使用取单条评论文本数据，进行测试，输出情感类别及其概率。

代码：

```
def preprocess(sentence):
    sentence = sentence.lower().strip()
    sentence = sentence.replace('\n', '')\
        .replace('"', '')\
        .replace('\', '')\
        .replace('.', '')\
        .replace(',', '')\
        .replace('[', '')\
        .replace(']', '')\
        .replace('(', '')\
        .replace(')', '')\
        .replace(':', '')\
        .replace('--', '')\
        .replace('-', ' ')\
        .replace('\\', '')\
        .replace('0', '')\
        .replace('1', '')\
        .replace('2', '')\
        .replace('3', '')\
        .replace('4', '')\
        .replace('5', '')\
        .replace('6', '')\
        .replace('7', '')\
        .replace('8', '')\
        .replace('9', '')\
        .replace('~', '')\
        .replace('=', '')\
        .replace('$', '')\
        .replace('/', '')\
        .replace('*', '')\
        .replace('; ', '')\
        .replace('<b>', '')\
        .replace('%', '')\
        .replace("  ", " ")

    sentence = sentence.split(' ')
    maxlen = cfg.word_len
    vector = [0]*maxlen
    for index, word in enumerate(sentence):
        if index >= maxlen:
            break
        if word not in instance.Vocab.keys():
            print(word, "单词未出现在字典中")
        else:
            vector[index] = instance.Vocab[word]
    sentence = vector

    return sentence
```

```
def inference(review_en):  
    review_en = preprocess(review_en)  
    input_en = Tensor(np.array([review_en]).astype(np.int32))  
    output = net(input_en)  
    if np.argmax(np.array(output[0])) == 1:  
        print("Positive comments")  
    else:  
        print("Negative comments")
```

代码：

```
review_en = "the movie is so boring"  
inference(review_en)
```

输出：

Negative comments

## 步骤 10 思考题

如果想通过 GPU 跑以上实验，需要在哪个地方修改相关配置为 GPU？

# 3 PyTorch

---

## 3.1 实验介绍

### 3.1.1 关于本实验

本实验主要是介绍 PyTorch 的张量数据结构，通过对张量的一系列操作介绍，可以使学员对 PyTorch 的基本语法有所了解。在了解基础操作以后，通过 PyTorch 构建 LeNet、ResNet 网络结构实现图像分类应用。

### 3.1.2 实验目的

- 掌握 PyTorch 张量的创建方法。
- 掌握 PyTorch 张量的属性与方法。
- 掌握 PyTorch 网络搭建。

### 3.1.3 实验环境

本实验所需的框架为 PyTorch，建议版本为 2.0 及以上版本。实验可以在个人 PC 上完成也可以登录华为云购买 ModelArts 服务完成。

## 3.2 PyTorch 基础

### 3.2.1 创建 tensor

不同维度的张量分别表示不同的数据，0 维张量表示标量，1 维张量表示向量，2 维张量表示矩阵，3 维张量可以表示彩色图像的 RGB 三通道等等。

#### 步骤 1 随机初始化一个张量

代码：

```
import torch
x = torch.rand(4, 3)
print(x)
```

输出：

```
tensor([[0.2052, 0.2865, 0.9629],
        [0.3342, 0.8409, 0.0419],
        [0.8176, 0.9898, 0.8704],
        [0.0259, 0.0501, 0.3116]])
```

## 步骤 2 使用不同类型的数据创建 Tensor

代码：

```
import numpy as np
print(torch.Tensor(np.array([[1, 2], [3, 4]]))) # 使用数组创建
print(torch.Tensor([True,False])) # 使用 bool 类型创建
print(torch.Tensor((1,2,3,4,5))) # 使用元组创建
print(torch.Tensor([i for i in range(10)])) # 使用列表创建
```

输出：

```
tensor([[1., 2.],
        [3., 4.]])
tensor([1., 0.])
tensor([1., 2., 3., 4., 5.])
tensor([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

## 步骤 3 构造不同类型的 Tensor

代码：

```
print(torch.FloatTensor(2,3).type()) #构建一个 2*3 Float 类型的张量
print(torch.DoubleTensor(2,3).type()) #构建一个 2*3 Double 类型的张量
print(torch.HalfTensor (2,3).type()) #构建一个 2*3 HalfTenso 类型的张量
print(torch.ByteTensor(2,3).type()) #构建一个 2*3 Byte 类型的张量
print(torch.CharTensor(2,3).type()) #构建一个 2*3 Char 类型的张量
print(torch.ShortTensor(2,3).type()) #构建一个 2*3 Short 类型的张量
print(torch.IntTensor(2,3).type()) #构建一个 2*3 Int 类型的张量
print(torch.LongTensor(2,3).type()) #构建一个 2*3 Long 类型的张量
```

输出：

```
torch.FloatTensor
torch.DoubleTensor
torch.HalfTensor
torch.ByteTensor
torch.CharTensor
torch.ShortTensor
torch.IntTensor
torch.LongTensor
```

## 步骤 4 创建一些特殊张量

代码：

```
import torch
x = torch.zeros(4, 3, dtype=torch.long)
print(x) # 全零张量
x = torch.ones(4, 3, dtype=torch.long)
print(x) # 全 1 张量
x = x.new_ones(4, 3, dtype=torch.double)
print(x)
x = torch.randn_like(x, dtype=torch.float)
print(x) #
```

输出：

```
tensor([[[0, 0, 0],
         [0, 0, 0],
         [0, 0, 0],
         [0, 0, 0]])
tensor([[[1, 1, 1],
         [1, 1, 1],
         [1, 1, 1],
         [1, 1, 1]])
tensor([[[1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.]], dtype=torch.float64)
tensor([[-0.8838,  0.8029, -1.9473],
        [-0.7147, -1.0430, -0.3002],
        [ 1.2838,  1.9056, -1.6879],
        [-0.8628,  1.6469,  0.6585]])
```

## 3.2.2 Tensor 的属性

张量的属性包括形状 (shape) 和数据类型 (dtype) 。

- 形状：Tensor 的 shape，是一个 tuple。
- 数据类型：Tensor 的 dtype，是 MindSpore 的一个数据类型。

代码：

```
x = torch.Tensor(np.array([[1, 2], [3, 4]]))
print(x.shape) # 获取形状
print(x.size())
print(x.dim()) # 维数
print(x.device) # 所在硬件
print(x.dtype) # 数据类型
```

输出：

```
torch.Size([2, 2])
torch.Size([2, 2])
2
cpu
torch.float32
```

### 3.2.3 Tensor 的方法

代码：

```
x = torch.Tensor(np.array([[1, 2], [3, 4]]))

print(x.reshape(4,1)) # reshape,改变形状
x = x.unsqueeze(2) # 增加一个维度
print(x.shape)
x = x.squeeze(2) # 压缩维度
print(x.shape)

print(x.flatten()) # 展开为一维
x.to(device) # 移动到指定设备 device
```

输出：

```
tensor([[1.,
         2.,
         3.,
         4.]])
torch.Size([2, 2, 1])
torch.Size([2, 2])
tensor([1., 2., 3., 4.]])
```

### 3.2.4 加载数据集

#### 步骤 1 加载 MNIST 数据集

加载数据集之前，可以先从以下链接 <http://yann.lecun.com/exdb/mnist/>，下载 MNIST 数据集，并将训练文件和测试文件放到 MNIST 文件夹下面。如果没有下载，pytorch 会自动下载，但是要保证网络连接。

代码：

```
import matplotlib.pyplot as plt
from torchvision import datasets, transforms

# 从 mnist dataset 读取图片
```

```
train_dataset = datasets.MNIST(root='./MNIST',train=True,download=True,transform=transforms.ToTensor())
# 查看图像, 设置图像大小
plt.figure(figsize=(8,8))
i = 1

# 打印 3 张训练集的子图
for dic in train_dataset.train_data[:3]:
    plt.subplot(3,3,i)
    plt.imshow(dic)
    plt.axis('off')
    i += 1
plt.show()
```

输出:

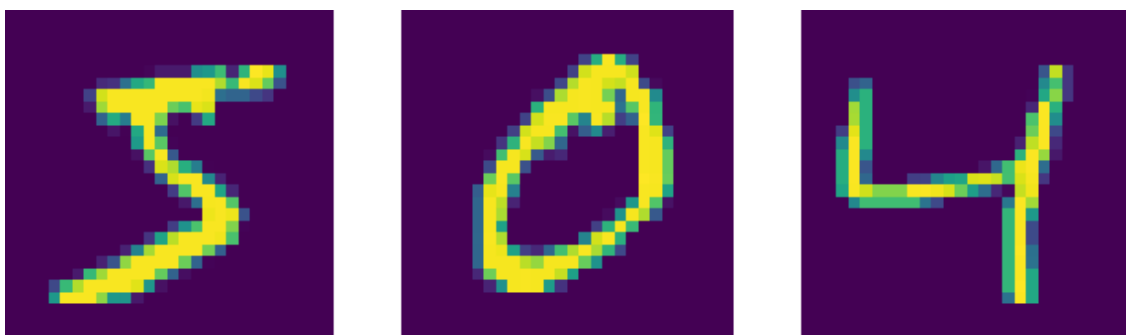


图3-1 MNIST 数据集样本

## 步骤 2 使用 ImageFolder 加载数据集

此处加载的数据为 ResNet 实验中所使用的花卉数据集。

```
train = torchvision.datasets.ImageFolder("./flower_photos_train/") # 加载数据

# 查看数据集属性
print("数据集类别")
print(train.classes)
print(train.class_to_idx)
print("数据集样本")
print(train.samples[0])
print("图片数据格式")
print(train.extensions[0])
```

输出:

```
数据集类别
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
{'daisy': 0, 'dandelion': 1, 'roses': 2, 'sunflowers': 3, 'tulips': 4}
数据集样本
('./flower_photos_train/daisy\\10140303196_b88d3d6cec.jpg', 0)
```

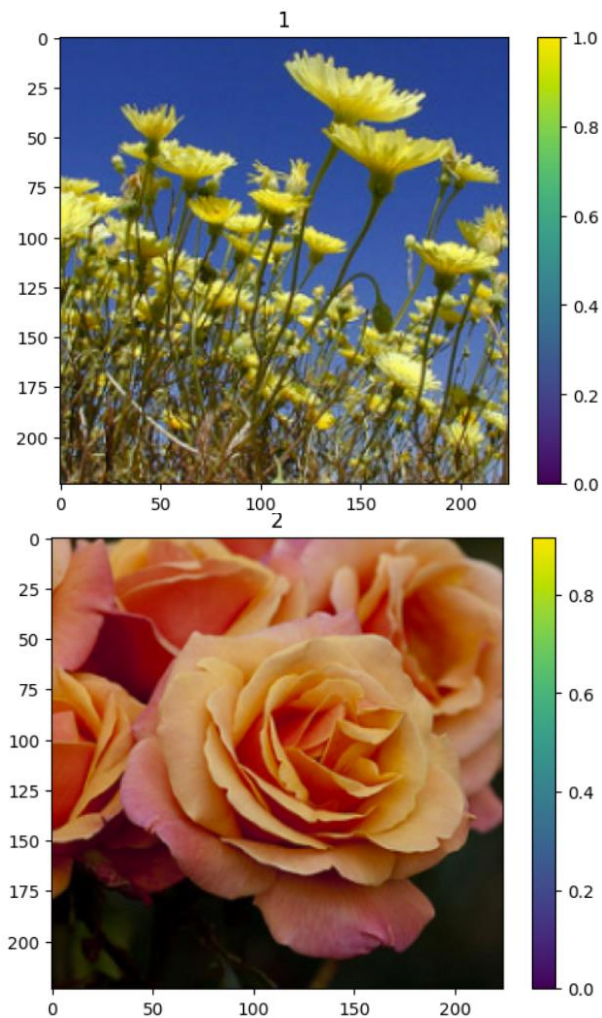
### 图片数据格式

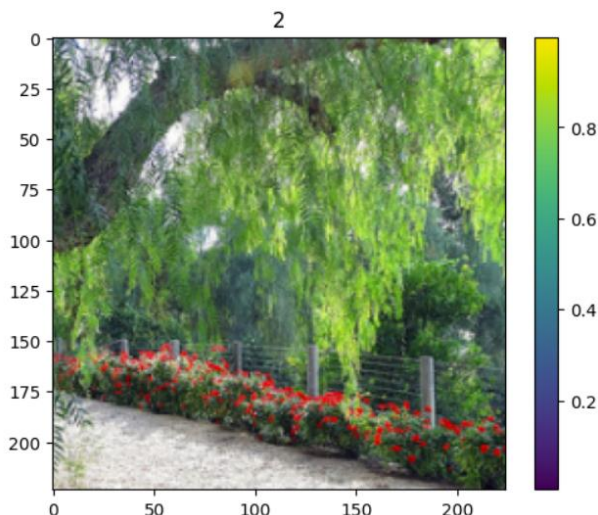
.jpg

对数据做可视化展示:

```
# 展示加载的图片数据
for i in range(5):
    plt.figure()
    random.seed = 2
    num = random.randint(0,3000)
    plt.imshow(train[num][0].permute(1,2,0)) # train[num] 返回一个 tuple 包含数据和标签对应的类别数字
    plt.title(train[num][1])
    plt.colorbar()
    plt.grid(False)
    plt.show()
```

输出:





### 步骤3 自定义数据集

在 PyTorch 中，自定义数据集类通常需要继承 `torch.utils.data.Dataset` 类，并实现 `__len__` 和 `__getitem__` 方法。这两个方法分别用于返回数据集中的样本数量和对给定索引的样本进行加载和预处理。

```
import os
import torch
from torch.utils.data import Dataset, DataLoader
from PIL import Image
from torchvision import transforms

# 伪代码
class CustomDataset(Dataset):
    def __init__(self, image_dir, label_file, transform=None):
        """
        image_dir (string): 存储图像的目录路径。
        label_file (string): 包含标签的文件路径，每一行对应一个标签。
        transform (callable, optional): 可选的图像转换操作，用于数据增强或预处理。默认为 None。
        """
        self.image_dir = image_dir # 图像目录路径
        self.labels = self._load_labels(label_file) # 加载标签
        self.image_filenames = sorted(os.listdir(image_dir)) # 获取图像文件名列表，并排序
        self.transform = transform # 图像转换操作

    def _load_labels(self, label_file):
        """
        从文件中加载标签
        Args:
            label_file (string): 包含标签的文件路径。
        Returns:
```

```
list: 标签列表, 每个元素是一个整数。
"""
with open(label_file, 'r') as f:
    labels = [int(line.strip()) for line in f] # 读取文件, 将每行转换为整数, 并存储在列表中
return labels

def __len__(self):
    """
    返回数据集中的样本数量
    Returns:样本数量。
    """
    return len(self.image_filenames) # 返回图像文件名列表的长度

def __getitem__(self, idx):
    """
    根据索引获取数据集中的样本
    Args:
        idx (int): 样本索引。
    Returns:
        tuple: 包含图像和标签的元组。
    """
    img_name = os.path.join(self.image_dir, self.image_filenames[idx]) # 构建图像文件的完整路径
    image = Image.open(img_name).convert('RGB') # 打开图像文件, 并转换为 RGB 模式
    label = self.labels[idx] # 获取对应的标签

    if self.transform:
        image = self.transform(image) # 如果提供了转换操作, 则对图像进行转换

    # 返回图像和标签的元组
    return image, label

if __name__ == "__main__":
    # 定义图像转换操作
    transform = transforms.Compose([
        # 数据处理操作
        pass
    ])

    # 图像目录路径和标签文件路径
    image_dir = 'path/to/images' # 替换为你的图像目录路径
    label_file = 'path/to/labels.txt' # 替换为你的标签文件路径

    # 创建数据集实例
```

```
dataset = CustomDataset(image_dir=image_dir, label_file=label_file, transform=transform)

# 创建数据加载器
dataloader = DataLoader(dataset, batch_size=32, shuffle=True, num_workers=4) # 批量大小 32, 打乱顺序, 使用 4 个工作进程
```

## 3.3 LeNet

使用 PyTorch 实现 LeNet 网络。

### 步骤 1 导入所需的工具包

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import os
import matplotlib.pyplot as plt
```

### 步骤 2 定义数据方式并在加载数据时使用

此处加载的数据集为 CIFAR10，共包含 10 个类别的 RGB 彩色图片，分别是飞机 (airplane)、汽车 (automobile)、鸟类 (bird)、猫 (cat)、鹿 (deer)、狗 (dog)、蛙类 (frog)、马 (horse)、船 (ship) 和卡车 (truck)。每个类别有 6000 张图像，数据集中一共有 60000 张图像，其中 50000 张用于训练，10000 张用于测试。每张图片的尺寸为 32x32 像素。

```
# 数据预处理
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# 下载并加载 CIFAR-10 数据集
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True, num_workers=1)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False, num_workers=1)
```

### 步骤 3 数据展示

```
# 展示图片
dataiter = iter(trainloader)
images, labels = next(dataiter)

# 创建网格来展示图片
imshow = torchvision.utils.make_grid(images)
```

```
# 转换为 numpy 数组并去掉归一化
imshow = imshow.numpy().transpose((1, 2, 0))

# 显示图片
plt.imshow(imshow)
plt.show()
```

#### 步骤 4 定义网络结构类并实例化

```
# 定义 LeNet 网络结构
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__() # 调用父类 nn.Module 的构造函数
        # 第一个卷积层, 输入通道数为 3, 输出通道数为 6, 卷积核大小为 5x5
        self.conv1 = nn.Conv2d(3, 6, kernel_size=5)
        # 第二个卷积层, 输入通道数为 6, 输出通道数为 16, 卷积核大小为 5x5
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        # 第一个全连接层, 输入特征数量为 16*5*5, 输出特征数量为 120
        self.fc1 = nn.Linear(16*5*5, 120)
        # 第二个全连接层, 输入特征数量为 120, 输出特征数量为 84
        self.fc2 = nn.Linear(120, 84)
        # 第三个全连接层 (输出层), 输入特征数量为 84, 输出特征数量为 10
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = torch.tanh(self.conv1(x))
        # 对 x 进行 2x2 的最大池化操作
        x = torch.max_pool2d(x, 2)
        # x 通过第二个卷积层, 然后应用 tanh 激活函数
        x = torch.tanh(self.conv2(x))
        # 对 x 进行 2x2 的最大池化操作
        x = torch.max_pool2d(x, 2)
        # 将 x 展平为一维向量, 准备输入到全连接层, 大小为(batch_size, 16*5*5)
        x = x.view(-1, 16*5*5)
        x = torch.tanh(self.fc1(x))
        # x 通过第二个全连接层, 然后应用 tanh 激活函数
        x = torch.tanh(self.fc2(x))
        # x 通过第三个全连接层 (输出层), 通常在这里不使用激活函数, 因为输出值可能需要用于分类等任务
        x = self.fc3(x)
        return x

# 实例化 LeNet 网络
```

```
net = LeNet()  
net.parameters # 查看模型参数
```

输出：

```
<bound method Module.parameters of LeNet(  
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))  
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))  
  (fc1): Linear(in_features=400, out_features=120, bias=True)  
  (fc2): Linear(in_features=120, out_features=84, bias=True)  
  (fc3): Linear(in_features=84, out_features=10, bias=True)  
)>
```

## 步骤 5 定义损失函数及优化器

```
criterion = nn.CrossEntropyLoss()  
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

## 步骤 6 定义训练过程

```
# 判断当前是否有训练好的参数可以加载  
if os.path.isfile("cifar10.pth"):  
    net.load_state_dict(torch.load("cifar10.pth"))  
  
# 用于存储损失值  
loss_history = []  
  
# 训练网络  
for epoch in range(50): # 50 个 epoch  
    running_loss = 0.0  
    for i, data in enumerate(trainloader, 0):  
        inputs, labels = data  
  
        optimizer.zero_grad()  
  
        outputs = net(inputs)  
        loss = criterion(outputs, labels)  
        loss.backward()  
        optimizer.step()  
  
        running_loss += loss.item()  
        if i % 200 == 199:  
            print(f'Epoch: {epoch + 1}, Batch: {i + 1} loss: {running_loss / 200:.3f}')  
            running_loss = 0.0  
  
    # 记录损失值  
    loss_history.append(loss.item())  
  
print('Finished Training')
```

## 步骤 7 可视化 loss 下降过程

```
# 绘制损失曲线
plt.plot(loss_history)
plt.xlabel('Batch')
plt.ylabel('Loss')
plt.title('Loss over Training Batches')
plt.show()
```

## 步骤 8 使用测试集对模型进行测试

```
# 测试网络在测试集上的表现
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct / total:.2f}%')
```

输出：

```
Accuracy of the network on the 10000 test images: 75.97%
```

## 3.4 ResNet50 图像分类

本实验实现的是深度学习领域的经典案例——ResNet50 图像分类实验。整体流程如下：

- 处理需要的数据集，这里使用了花卉图像数据集。
- 定义一个网络，这里自己搭建 ResNet50 模型结构。
- 定义损失函数和优化器。
- 加载数据集并进行训练，训练完成后，利用测试集进行评估。

### 步骤 1 实验准备

在动手进行实践之前，确保你已经正确安装了 PyTorch。

同时希望你拥有 Python 编码基础和概率、矩阵等基础数学知识。

推荐环境：

版本：PyTorch 2.5

编程语言：Python 3 ( $\geq 3.7$ )

## 步骤 2 数据准备

我们示例中用到的图像花卉数据集，该数据集是开源数据集，总共包括 5 种花的类型：分别是 daisy（雏菊，633 张），dandelion（蒲公英，898 张），roses（玫瑰，641 张），sunflowers（向日葵，699 张），tulips（郁金香，799 张），保存在 5 个文件夹当中，总共 3670 张，大小大概在 230M 左右。为了在模型部署上线之后进行测试，数据集在这里分成了 flower\_photos\_train 和 flower\_photos\_test 两部分。

目录结构如下（数据集中 flowers 文件夹下）

flower\_photos\_train

- |— daisy
- |— dandelion
- |— roses
- |— sunflowers
- |— tulips
- |— LICENSE.txt

flower\_photos\_test

- |— daisy
- |— dandelion
- |— roses
- |— sunflowers
- |— tulips
- |— LICENSE.txt

新建文件夹 flowers，并下载数据文件：

```
https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower_photos_train.zip  
https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower_photos_test.zip
```

## 步骤 3 导入 Python 库&模块并配置运行信息

在使用前，导入需要的 Python 库。

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

```
import torch.optim as optim
import matplotlib.pyplot as plt
from easydict import EasyDict as edict
import random
```

#### 步骤 4 定义常量字典

```
cfg = edict({
    'data_path': './flower_photos_train',    #训练数据集路径
    'test_path': 'flowers/flower_photos_train',    #测试数据集路径
    'data_size': 3616,
    'HEIGHT': 224, # 图片高度
    'WIDTH': 224, # 图片宽度
    '_R_MEAN': 123.68, # CIFAR10 的均值
    '_G_MEAN': 116.78,
    '_B_MEAN': 103.94,
    '_R_STD': 1, # 自定义的标准差
    '_G_STD': 1,
    '_B_STD': 1,
    '_RESIZE_SIDE_MIN': 256, # 图像增强 resize 最小值
    '_RESIZE_SIDE_MAX': 512,

    'batch_size': 1, # 批次大小
    'num_class': 5,    # 分类类别
    'epoch_size': 5, # 训练次数
    'num_workers': 1,
    'device': "cuda", # cpu、xpu、npu 等, 如果选用 npu 请确定是否具备 Ascend 训练资源和环境

    'prefix': 'renet50.pth', # 模型保存的名称
})
```

#### 步骤 5 数据读取和处理

定义数据处理方式:

```
# 使用 transforms.Compose 组合多个处理方式
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),

    transforms.ToTensor(),
    transforms.Normalize(mean=[cfg._R_MEAN, cfg._G_MEAN, cfg._B_MEAN], std=[cfg._R_STD, cfg._G_STD,
    cfg._B_STD]),
    transforms.Resize(cfg._RESIZE_SIDE_MIN),
    transforms.CenterCrop((cfg.HEIGHT, cfg.WIDTH)),
```

```
)  
  
transform_test = transforms.Compose([  
    transforms.ToTensor(),  
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),  
])
```

加载数据:

```
train = torchvision.datasets.ImageFolder(cfg. data_path, transform=transform_train)  
trainloader = torch.utils.data.DataLoader(train, batch_size=cfg.batch_size, shuffle=True,  
num_workers=cfg.num_workers)
```

## 步骤 6 定义网络结构

```
import torch.nn as nn  
  
# 基本块定义, 主要用于 ResNet-18 和 ResNet-34  
class BasicBlock(nn.Module):  
    expansion = 1 # 输出通道数的扩展倍数  
  
    def __init__(self, in_channels, out_channels, stride=1):  
        super(BasicBlock, self).__init__()  
  
        # 残差函数部分  
        self.residual_function = nn.Sequential(  
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False), # 第一个卷积层  
            nn.BatchNorm2d(out_channels), # 第一个批归一化层  
            nn.ReLU(inplace=True), # 激活函数  
            nn.Conv2d(out_channels, out_channels * BasicBlock.expansion, kernel_size=3, padding=1, bias=False),  
            # 第二个卷积层  
            nn.BatchNorm2d(out_channels * BasicBlock.expansion) # 第二个批归一化层  
        )  
  
        # 捷径 (shortcut) 部分, 用于调整维度和步长  
        self.shortcut = nn.Sequential()  
  
        # 如果维度不匹配, 使用 1x1 卷积来调整捷径的维度  
        if stride != 1 or in_channels != BasicBlock.expansion * out_channels:  
            self.shortcut = nn.Sequential(  
                nn.Conv2d(in_channels, out_channels * BasicBlock.expansion, kernel_size=1, stride=stride,  
bias=False),  
                nn.BatchNorm2d(out_channels * BasicBlock.expansion)  
            )  
  
    def forward(self, x):
```

```

# 残差连接和激活函数
return nn.ReLU(inplace=True)(self.residual_function(x) + self.shortcut(x))

# 瓶颈块定义，主要用于 ResNet-50, ResNet-101 和 ResNet-152
class Bottleneck(nn.Module):
    expansion = 4 # 输出通道数的扩展倍数

    def __init__(self, in_channels, out_channels, stride=1):
        super(Bottleneck, self).__init__()

        # 残差函数部分
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False), # 第一个 1x1 卷积层，用于降维
            nn.BatchNorm2d(out_channels), # 第一个批归一化层
            nn.ReLU(inplace=True), # 激活函数
            nn.Conv2d(out_channels, out_channels, stride=stride, kernel_size=3, padding=1, bias=False), # 3x3
卷积层
            nn.BatchNorm2d(out_channels), # 第二个批归一化层
            nn.ReLU(inplace=True), # 激活函数
            nn.Conv2d(out_channels, out_channels * Bottleneck.expansion, kernel_size=1, bias=False), # 第二
个 1x1 卷积层，用于升维
            nn.BatchNorm2d(out_channels * Bottleneck.expansion) # 第三个批归一化层
        )

        # shortcut 部分，用于调整维度和步长
        self.shortcut = nn.Sequential()

        # 如果维度不匹配，使用 1x1 卷积来调整捷径的维度
        if stride != 1 or in_channels != out_channels * Bottleneck.expansion:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels * Bottleneck.expansion, stride=stride, kernel_size=1,
bias=False),
                nn.BatchNorm2d(out_channels * Bottleneck.expansion)
            )

    def forward(self, x):
        # 残差连接和激活函数
        return nn.ReLU(inplace=True)(self.residual_function(x) + self.shortcut(x))

# ResNet 网络定义
class ResNet(nn.Module):

    def __init__(self, block, num_block, num_classes=100):
        super(ResNet, self).__init__()

```

```
self.in_channels = 64 # 初始通道数

# 第一个卷积层
self.conv1 = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True)
)

# 不同的层数配置，对应于不同的 ResNet 版本
self.conv2_x = self._make_layer(block, 64, num_block[0], 1)
self.conv3_x = self._make_layer(block, 128, num_block[1], 2)
self.conv4_x = self._make_layer(block, 256, num_block[2], 2)
self.conv5_x = self._make_layer(block, 512, num_block[3], 2)

# 全局平均池化层
self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))

# 全连接层，输出类别数根据需求设置
self.fc = nn.Linear(512 * block.expansion, num_classes)

def _make_layer(self, block, out_channels, num_blocks, stride):
    # 生成一个层的序列
    strides = [stride] + [1] * (num_blocks - 1) # 每个 block 的步长，除了第一个 block 外其余都为 1
    layers = []
    for stride in strides:
        layers.append(block(self.in_channels, out_channels, stride)) # 添加 block
        self.in_channels = out_channels * block.expansion # 更新通道数

    return nn.Sequential(*layers) # 返回层的序列

def forward(self, x):
    # 前向传播
    output = self.conv1(x)
    output = self.conv2_x(output)
    output = self.conv3_x(output)
    output = self.conv4_x(output)
    output = self.conv5_x(output)
    output = self.avg_pool(output) # 全局平均池化
    output = output.view(output.size(0), -1) # 展平为一维向量
    output = self.fc(output) # 全连接层

    return output
```

通过上面定义好的类，定义不同参数（层数）下的 ResNet 结构：

```
# 定义不同版本的 ResNet
def resnet18():
    """ 返回 ResNet-18 模型 """
    return ResNet(BasicBlock, [2, 2, 2, 2])

def resnet34():
    """ 返回 ResNet-34 模型 """
    return ResNet(BasicBlock, [3, 4, 6, 3])

def resnet50():
    """ 返回 ResNet-50 模型 """
    return ResNet(BottleNeck, [3, 4, 6, 3], num_classes=cfg.num_class) # 修改类别数量

def resnet101():
    """ 返回 ResNet-101 模型 """
    return ResNet(BottleNeck, [3, 4, 23, 3])

def resnet152():
    """ 返回 ResNet-152 模型 """
    return ResNet(BottleNeck, [3, 8, 36, 3])
```

后续实验使用 ResNet50。

## 步骤 7 定义损失函数和优化器，并初始化网络

```
import os

net = resnet50().to(cfg.device)
if os.path.isfile(cfg.prefix):
    net.load_state_dict(torch.load(cfg.prefix))

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9, weight_decay=0.0001)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor = 0.1, patience=5)
```

## 步骤 8 定义训练过程

```
for epoch in range(cfg.epoch_size):
    losses = []
    running_loss = 0
    for i, inp in enumerate(trainloader):
        inputs, labels = inp
        inputs, labels = inputs.to(cfg.device), labels.to(cfg.device)
```

```
optimizer.zero_grad()

outputs = net(inputs)
loss = criterion(outputs, labels)
losses.append(loss.item())

loss.backward()
optimizer.step()

running_loss += loss.item()

if i%200 == 0 and i > 0:
    print(f'Loss [{epoch+1}, {i}](epoch, minibatch): ', running_loss / 100)
    running_loss = 0.0

avg_loss = sum(losses)/len(losses)
scheduler.step(avg_loss)

print('Training Done')
```

### 步骤 9 模型保存

```
torch.save(net.state_dict(),cfg. prefix)
```

### 步骤 10 测试

```
test = torchvision.datasets.ImageFolder(cfg. test_path, transform=transform_test)
testloader = torch.utils.data.DataLoader(test, batch_size=cfg.batch_size, shuffle=True,
num_workers=cfg.num_workers)

correct = 0
total = 0

with torch.no_grad():
    for data in testloader:
        images, labels = data
        images, labels = images.to(cfg.device), labels.to(cfg.device)
        outputs = net(images)

        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy: ', 100*(correct/total), '%')
```

## 3.5 思考

- 为什么在 LeNet、ResNet 的训练过程中会出现 loss 不降反升的情况？

- LeNet 在 CIFAR10 的训练过程中，继续增加训练 epoch 次数是否会进一步提高准确率？为什么？
  - 如何在不改变网络结构的情况下进一步提升网络的准确率？
-

华为认证人工智能系列教程

HCIA-AI

模型部署与应用

实验指导手册

版本：4.0



华为技术有限公司

版权所有 © 华为技术有限公司 2025。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

#### 商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

#### 注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<https://e.huawei.com>

---

## 华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT技术架构与应用认证、云服务与平台认证两类认证。

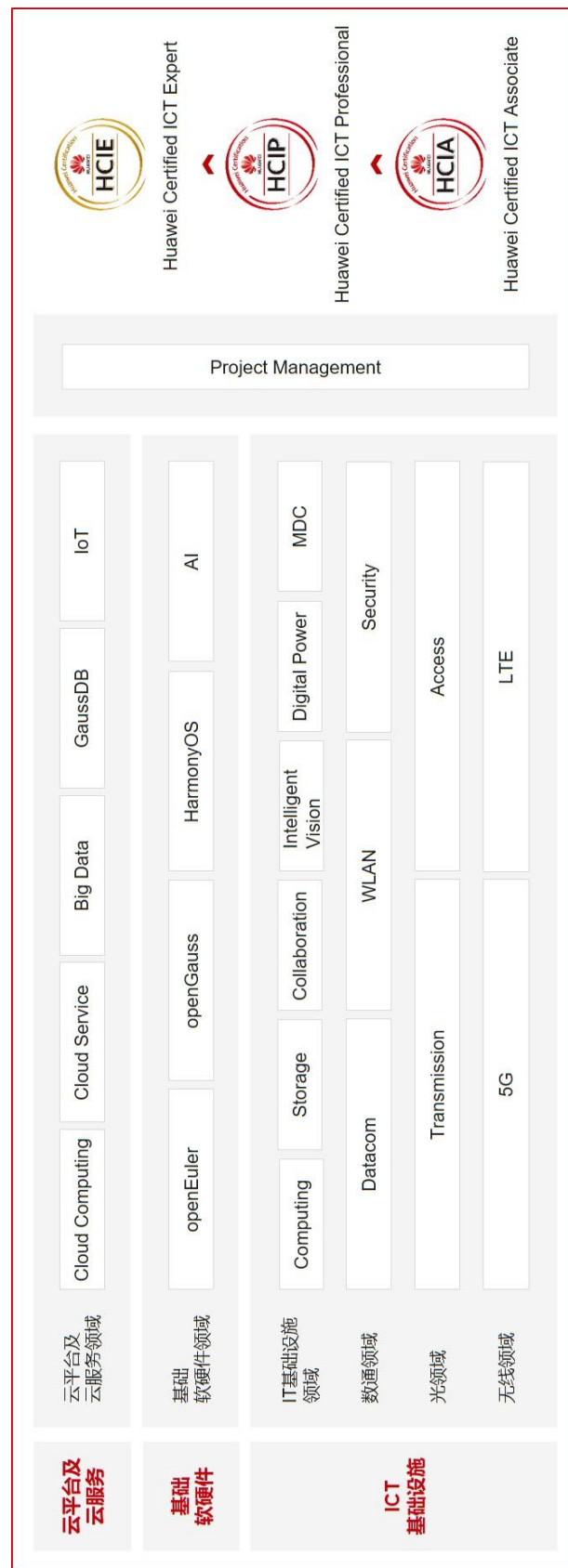
根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIA-AI V4.0定位于培养和认证具备使用机器学习、深度学习和大模型等算法进行AI产品和解决方案设计、开发和创新能力的工程师。

通过HCIA-AI V4.0认证，将证明您了解人工智能发展历史和华为AI解决方案，掌握传统机器学习、深度学习和大模型的相关算法；具备利用MindSpore开发框架和PyTorch开发框架进行搭建、训练、部署神经网络的能力；能够胜任人工智能领域开发工程师等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在人工智能世界的潮头浪尖！



# 前言

## 简介

本书为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-AI 考试的学员或者希望了解 AI 基础知识、LLM（Large Language Model，大语言模型）部署、API 调用等技术的读者。

## 内容描述

本实验指导书共包含 2 个实验，从 LLM 命令行部署开始，逐一介绍了命令行对话、LLM Web 部署、查询部署服务状态、调用 API 对话等操作。

- 实验一为 LLM 命令行部署实验，通过配置基础参数，即可通过命令行与 LLM 对话。
- 实验二为 LLM Web 部署实验，在服务部署成功后，可通过简单的代码通过调用 API 的方式与 LLM 对话。

## 读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本的 AI 知识背景，具备 Python 编程能力，了解计算机知识。

## 实验环境说明

设备介绍

为了满足 HCIA-AI 实验需要，建议每套实验环境采用以下配置：

设备名称、型号与版本的对应关系如下：

| 设备名称 | 配置要求               | 软件版本                                                       |
|------|--------------------|------------------------------------------------------------|
| PC机  | CPU为X86架构，支持AVX指令集 | 操作系统：Windows10及以上或<br>Ubuntu22.04及以上<br>Python版本：Python3.x |

# 目录

---

|                     |           |
|---------------------|-----------|
| <b>前 言</b>          | <b>3</b>  |
| 简介                  | 3         |
| 内容描述                | 3         |
| 读者知识背景              | 3         |
| 实验环境说明              | 3         |
| <b>1 LLM 命令行部署</b>  | <b>5</b>  |
| 1.1 实验介绍            | 5         |
| 1.1.1 关于本实验         | 5         |
| 1.1.2 实验目的          | 5         |
| 1.2 实验任务配置          | 5         |
| 1.2.1 配置思路          | 5         |
| 1.2.2 配置步骤          | 5         |
| <b>2 LLM Web 部署</b> | <b>11</b> |
| 2.1 实验介绍            | 11        |
| 2.1.1 关于本实验         | 11        |
| 2.1.2 实验目的          | 11        |
| 2.2 实验任务配置          | 11        |
| 2.2.1 配置思路          | 11        |
| 2.2.2 配置步骤          | 11        |
| 2.3 思考题             | 15        |

---

# 1 LLM 命令行部署

---

## 1.1 实验介绍

### 1.1.1 关于本实验

本实验使用 llama.cpp 进行模型部署（项目地址为 <https://github.com/ggerganov/llama.cpp>），模型文件为量化后的 DeepSeek-R1-Distill-Qwen-1.5B 模型（项目镜像地址为 <https://hf-mirror.com/bartowski/DeepSeek-R1-Distill-Qwen-1.5B-GGUF>）；llama.cpp 使用 C++ 实现，无需任何依赖，极大降低了 LLM 的部署门槛，适合初学者体验，而量化后的 DeepSeek-R1-Distill-Qwen-1.5B 进行推理时资源占用较小，可以在笔记本上使用 CPU 进行推理的情况下取得不错的速度，降低了硬件要求，同时这个模型也是 DeepSeek-R1 系列模型中最小的推理型模型，具备思考能力；通过简单的操作，即可在命令行中与 LLM 进行对话。

### 1.1.2 实验目的

- 掌握如何使用 llama.cpp 部署 LLM
- 了解部署 LLM 时常用的参数含义

## 1.2 实验任务配置

### 1.2.1 配置思路

- 下载 llama.cpp 可执行文件
- 下载量化后的 DeepSeek-R1-Distill-Qwen-1.5B 模型文件
- 配置不同的运行参数与 LLM 进行对话

### 1.2.2 配置步骤

**Note:** Windows 系统和 Ubuntu 系统在代码文件下载和模型文件下载的步骤有所区别，需要根据自己的操作系统执行对应的操作，部署部分命令一致，没有做区分。

## 1.2.2.1 下载 llama.cpp 可执行文件

### 1.2.2.1.1 Windows 用户

步骤 1 点击以下链接下载压缩包

[https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V4.0/chapter5/llama\\_cpp.zip](https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V4.0/chapter5/llama_cpp.zip)

步骤 2 对压缩包进行解压，进入文件夹，如下图所示

| <input type="checkbox"/> 名称                                           | 类型   | 大小       |
|-----------------------------------------------------------------------|------|----------|
| <input checked="" type="checkbox"/> llama-baby-llama.exe              | 应用程序 | 36 KB    |
| <input checked="" type="checkbox"/> llama-batched.exe                 | 应用程序 | 530 KB   |
| <input checked="" type="checkbox"/> llama-convert-llama2c-to-ggml.exe | 应用程序 | 67 KB    |
| <input checked="" type="checkbox"/> llama-cvector-generator.exe       | 应用程序 | 562 KB   |
| <input checked="" type="checkbox"/> llama-export-lora.exe             | 应用程序 | 544 KB   |
| <input checked="" type="checkbox"/> llama-gbnf-validator.exe          | 应用程序 | 41 KB    |
| <input checked="" type="checkbox"/> llama-server.exe                  | 应用程序 | 1,239 KB |
| <input checked="" type="checkbox"/> llama-gritlm.exe                  | 应用程序 | 538 KB   |
| <input checked="" type="checkbox"/> llama-gguf-split.exe              | 应用程序 | 45 KB    |
| <input checked="" type="checkbox"/> llama-imatrix.exe                 | 应用程序 | 565 KB   |
| <input checked="" type="checkbox"/> llama-infill.exe                  | 应用程序 | 579 KB   |
| <input checked="" type="checkbox"/> llama-llava-cli.exe               | 应用程序 | 706 KB   |
| <input checked="" type="checkbox"/> llama-lookup-create.exe           | 应用程序 | 543 KB   |

### 1.2.2.1.2 Ubuntu 用户

步骤 1 打开一个终端窗口，然后输入以下命令下载并解压文件

```
wget https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V4.0/chapter5/llama_cpp_ubuntu.zip
unzip llama_cpp_ubuntu.zip
```

步骤 2 输入以下命令进入到文件夹并查看文件

```
cd build/bin && ll
```

```
[root@Atlas800T2 bin]# ll
total 159700
-rw-r--r--. 1 root root 1078 Sep 24 17:18 LICENSE
-rwxr-xr-x. 1 root root 3176480 Sep 24 17:17 llama-baby-llama
-rwxr-xr-x. 1 root root 3676328 Sep 24 17:17 llama-batched
-rwxr-xr-x. 1 root root 3676416 Sep 24 17:17 llama-batched-bench
-rwxr-xr-x. 1 root root 3257576 Sep 24 17:18 llama-bench
-rwxr-xr-x. 1 root root 935088 Sep 24 17:17 llama-bench-matmult
-rwxr-xr-x. 1 root root 3717080 Sep 24 17:17 llama-cli
-rwxr-xr-x. 1 root root 3127896 Sep 24 17:17 llama-convert-llama2c-to-ggml
-rwxr-xr-x. 1 root root 3699232 Sep 24 17:17 llama-cvector-generator
-rwxr-xr-x. 1 root root 3680752 Sep 24 17:17 llama-embedding
-rwxr-xr-x. 1 root root 3676576 Sep 24 17:17 llama-eval-callback
-rwxr-xr-x. 1 root root 3704368 Sep 24 17:17 llama-export-lora
-rwxr-xr-x. 1 root root 2782008 Sep 24 17:17 llama-gbnf-validator
-rwxr-xr-x. 1 root root 939744 Sep 24 17:16 llama-gguf
-rwxr-xr-x. 1 root root 1007888 Sep 24 17:16 llama-gguf-hash
-rwxr-xr-x. 1 root root 2787264 Sep 24 17:17 llama-gguf-split
-rwxr-xr-x. 1 root root 3681008 Sep 24 17:17 llama-gritlm
-rwxr-xr-x. 1 root root 3714704 Sep 24 17:17 llama-imatrix
-rwxr-xr-x. 1 root root 3699600 Sep 24 17:17 llama-infill
-rwxr-xr-x. 1 root root 3951344 Sep 24 17:17 llama-llava-cli
-rwxr-xr-x. 1 root root 3681032 Sep 24 17:17 llama-lookahead
-rwxr-xr-x. 1 root root 3701688 Sep 24 17:17 llama-lookup
-rwxr-xr-x. 1 root root 3697232 Sep 24 17:17 llama-lookup-create
-rwxr-xr-x. 1 root root 982432 Sep 24 17:17 llama-lookup-merge
-rwxr-xr-x. 1 root root 3702016 Sep 24 17:17 llama-lookup-stats
-rwxr-xr-x. 1 root root 3946696 Sep 24 17:17 llama-minicpmv-cli
-rwxr-xr-x. 1 root root 3685296 Sep 24 17:17 llama-parallel
-rwxr-xr-x. 1 root root 3676272 Sep 24 17:17 llama-passkey
-rwxr-xr-x. 1 root root 3769448 Sep 24 17:18 llama-perplexity
-rwxr-xr-x. 1 root root 934568 Sep 24 17:18 llama-q8dot
-rwxr-xr-x. 1 root root 3130824 Sep 24 17:18 llama-quantize
-rwxr-xr-x. 1 root root 2819624 Sep 24 17:17 llama-quantize-stats
-rwxr-xr-x. 1 root root 3694488 Sep 24 17:18 llama-retrieval
-rwxr-xr-x. 1 root root 3676496 Sep 24 17:18 llama-save-load-state
```

### 1.2.2.2 下载 DeepSeek-R1-Distill-Qwen-1.5B 模型文件

#### 1.2.2.2.1 Windows 用户

步骤 1 点击以下链接下载模型文件

步骤 2 [https://hf-mirror.com/bartowski/DeepSeek-R1-Distill-Qwen-1.5B-GGUF/resolve/main/DeepSeek-R1-Distill-Qwen-1.5B-Q6\\_K\\_L.gguf](https://hf-mirror.com/bartowski/DeepSeek-R1-Distill-Qwen-1.5B-GGUF/resolve/main/DeepSeek-R1-Distill-Qwen-1.5B-Q6_K_L.gguf) 模型文件下载完成后移动到 llama\_cpp 文件夹，如下图所示

| 名称                                       | 类型      | 大小           |
|------------------------------------------|---------|--------------|
| DeepSeek-R1-Distill-Qwen-1.5B-Q6_K_L.... | GGUF 文件 | 1,540,254 KB |
| linenoise.cpp.txt                        | 文本文档    | 2 KB         |
| llama.cpp.txt                            | 文本文档    | 2 KB         |
| ggml-base.dll                            | 应用程序扩展  | 478 KB       |
| ggml-rpc.dll                             | 应用程序扩展  | 59 KB        |
| ggml-cpu.dll                             | 应用程序扩展  | 414 KB       |
| ggml-vulkan.dll                          | 应用程序扩展  | 17,082 KB    |
| ggml.dll                                 | 应用程序扩展  | 64 KB        |
| llama.dll                                | 应用程序扩展  | 1,195 KB     |
| llama-gguf.exe                           | 应用程序    | 29 KB        |
| llama-simple.exe                         | 应用程序    | 22 KB        |

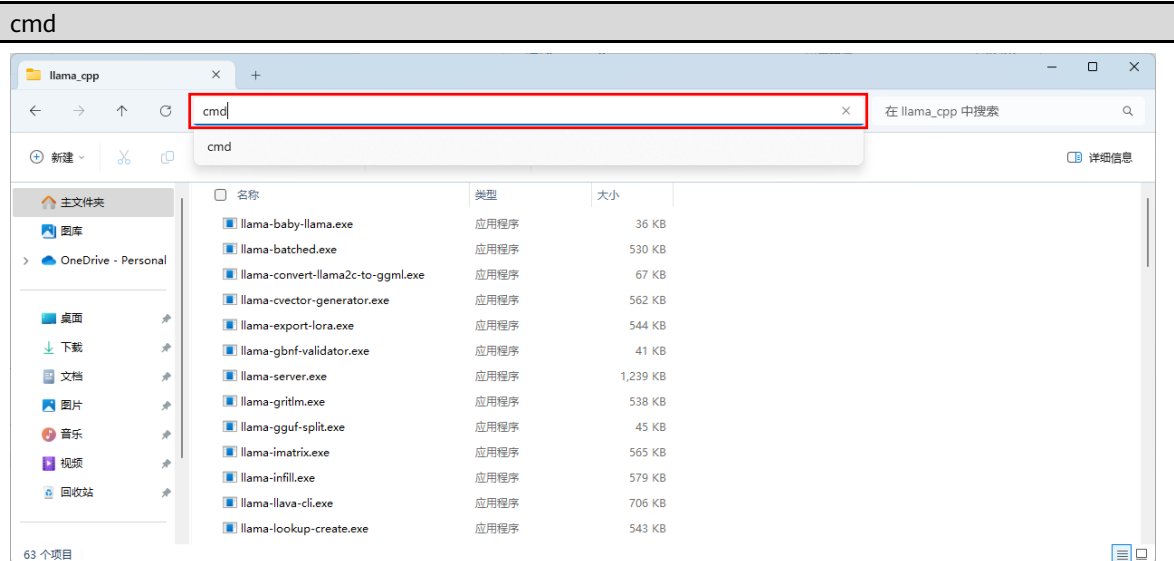
#### 1.2.2.2.2 Ubuntu 用户

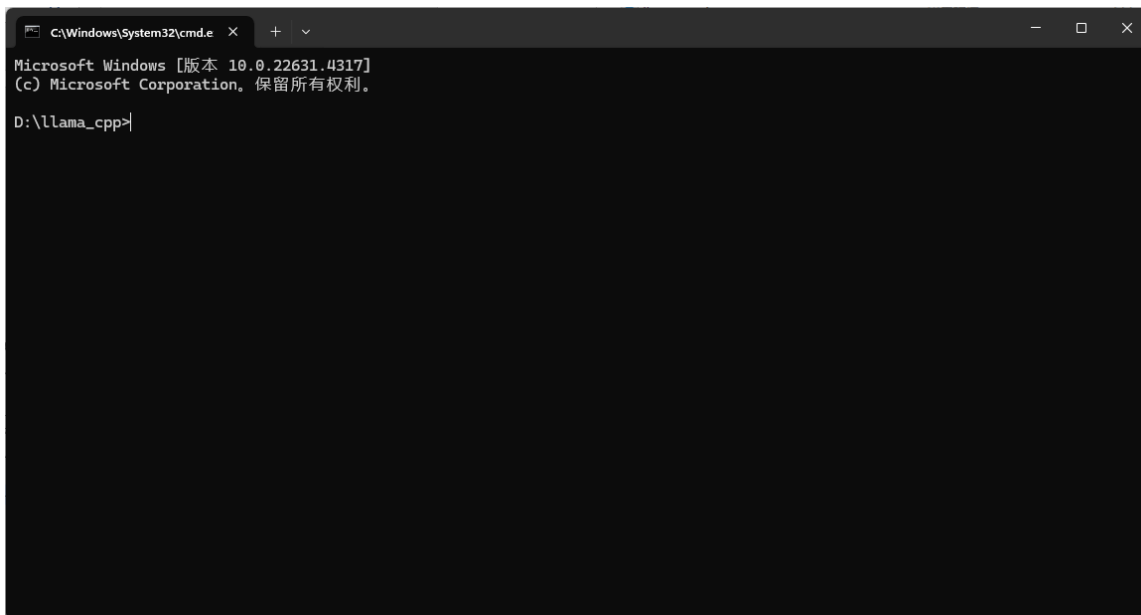
步骤 1 接着上一步输入以下命令下载模型文件，将模型文件下载到代码文件夹中

```
wget https://hf-mirror.com/bartowski/DeepSeek-R1-Distill-Qwen-1.5B-GGUF/resolve/main/DeepSeek-R1-Distill-Qwen-1.5B-Q6_K_L.gguf
```

#### 1.2.2.3 命令行部署

步骤 1 单击文件夹上方的导航栏，然后输入以下命令并回车，Ubuntu 用户可以直接到下一步





步骤 2 输入以下命令对模型执行推理

```
#对话模式
llama-cli.exe -m DeepSeek-R1-Distill-Qwen-1.5B-Q6_K_L.gguf -temp 0.6
```

部分参数说明：

**-m**：模型路径。

**-cnv**：对话模式（不打印特殊 token 和前缀、后缀，使用默认聊天模板，本实验提供的软件包默认开启对话模式，添加参数后反而会降低模型回答质量，因此命令行未添加该参数）。

**--chat-template**：聊天模板，如果模板与模型不匹配，则会出现模型回答乱码、不停止等问题，更多介绍可参考：[https://github.com/ggerganov/llama.cpp/wiki/Templates-supported-by-llama\\_chat\\_apply\\_template](https://github.com/ggerganov/llama.cpp/wiki/Templates-supported-by-llama_chat_apply_template)

**--color**：使用不同的颜色区分用户输入与模型输出。

**--temp**：调整输出文本的随机性，默认 0.8，本实验设置为 0.6 是因为 DeepSeek 官方推荐为 0.6。

**--repeat-penalty**：控制 token 序列在生成文本中的重复度，默认是 1.0，禁用。

**--top-k**：从概率最高的 K 个候选项中选择下一个 token，默认是 40。

更多参数介绍可参考以下链接：

<https://github.com/ggerganov/llama.cpp/tree/master/examples/main>

步骤 3 与模型进行对话，比如输入“你是谁？”，即可得到结果

步骤 4 不同配置参数的效果如下，其中<think></think>中间的文本为模型思考过程

```
llama-cli.exe --color -m DeepSeek-R1-Distill-Qwen-1.5B-Q6_K_L.gguf
```

```
C:\Windows\System32\cmd.e x + v
mirostat = 0, mirostat_lr = 0.100, mirostat_ent = 5.000
sampler chain: logits -> logit-bias -> penalties -> dry -> top-k -> typical -> top-p -> min-p -> xtc -> temp-ext -> dist
generate: n_ctx = 4096, n_batch = 2048, n_predict = -1, n_keep = 1

== Running in interactive mode. ==
- Press Ctrl+C to interject at any time.
- Press Return to return control to the AI.
- To return control without starting a new line, end your input with '/'.
- If you want to submit another line, end your input with '\'.
- Using default system message. To change it, set a different value via -p PROMPT or -f FILE argument.

You are a helpful assistant

> 你是谁?
<think>
您好! 我是由中国的深度求索 (DeepSeek) 公司开发的智能助手DeepSeek-R1。如您有任何任何问题, 我会尽我所能为您提供帮助。
</think>

您好! 我是由中国的深度求索 (DeepSeek) 公司开发的智能助手DeepSeek-R1。如您有任何任何问题, 我会尽我所能为您提供帮助。

> 1+1等于几?
<think>
</think>

1+1等于2, 这是最基础的数学事实。在不同的文化中, 数学的基本原理是一致的, 所以1+1=2是普遍成立的。

>
```

# 2 LLM Web 部署

---

## 2.1 实验介绍

### 2.1.1 关于本实验

本实验与实验 1 不同的地方为本实验使用 llama.cpp 部署 Web 服务，然后通过代码调用 API 来进行对话，llama.cpp 可以提供基础的 API 服务，同时也提供了 OpenAI-compatible 的 API 服务，使用者可以自行选择调用方式，llama.cpp 代码与模型文件与实验 1 保持一致，多了使用 Python 调用 API 的代码。

### 2.1.2 实验目的

- 掌握如何编写 Python 代码调用 API
- 掌握如何对 Web 的返回值进行结构化输出
- 掌握如何使用 llama.cpp 部署 Web 服务

## 2.2 实验任务配置

### 2.2.1 配置思路

- 使用 llama.cpp 部署 Web 服务
- 使用 Python 自带库调用 API
- 使用 openai 库调用 OpenAI-compatible 的 API

### 2.2.2 配置步骤

步骤 1 llama.cpp 代码和模型文件与实验 1 一致，打开命令行窗口后输入以下命令部署 Web 服务

```
llama-server.exe -m DeepSeek-R1-Distill-Qwen-1.5B-Q6_K_L.gguf -c 2048
```

部分参数说明：

-m：模型路径。

---

-c: 提示文本的长度，长度越长，模型可以处理的内容就越长（默认：4096,0=从模型中加载）。

更多参数介绍可参考以下链接：

<https://github.com/ggerganov/llama.cpp/tree/master/examples/server>

**Note:** 如果出现乱码，可在同一个窗口中终止程序，运行实验 1 中的命令，然后终止后运行本实验的命令。

```

srv      init: initializing slots, n_slots = 1
slot     init: id 0 | task -1 | new slot n_ctx_slot = 2048
main: model loaded
main: chat template, chat_template: {% if not add_generation_prompt is defined %}{% set add_generation_prompt =
false %}{% endif %}{% set ns = namespace(is_first=false, is_tool=false, is_output_first=true, system_prompt='') %}
{% for message in messages %}{%- if message['role'] == 'system' %}{% set ns.system_prompt = message['content']
%}{%- endif %}{%- endfor %}{% bos_token %}{% ns.system_prompt %}{%- for message in messages %}{%- if message['role']
== 'user' %}{%- set ns.is_tool = false -%}{% if '<| User |>' + message['content'] %}{%- endif %}{%- if message['role']
== 'assistant' and message['content'] is none %}{%- set ns.is_tool = false -%}{%- for tool in message['tool
_calls'] %}{%- if not ns.is_first %}{% '<| Assistant |>' + tool_calls_begin |><| tool_call_begin |>' + tool['type']
+ '<| tool_sep |>' + tool['function']['name'] + '\n' + '```json' + '\n' + tool['function']['arguments'] + '\n'
+ '```' + '<| tool_call_end |>' %}{%- set ns.is_first = true -%}{%- else %}{% '\n' + '<| tool_call_begin |>' + too
l['type'] + '<| tool_sep |>' + tool['function']['name'] + '\n' + '```json' + '\n' + tool['function']['arguments']
+ '\n' + '```' + '<| tool_call_end |>' %}{% '<| tool_calls_end |>' + end_of_sentence |>' %}{%- endif %}{%- endfor
%}{%- endif %}{%- if message['role'] == 'assistant' and message['content'] is not none %}{%- if ns.is_tool %}{%
'<| tool_outputs_end |>' + message['content'] + '<| end_of_sentence |>' %}{%- set ns.is_tool = false -%}{%- else %
%}{% set content = message['content'] %}{% if '</think>' in content %}{% set content = content.split('</think>')[
-1] %}{%- endif %}{% '<| Assistant |>' + content + '<| end_of_sentence |>' %}{%- endif %}{%- endif %}{%- if message
['role'] == 'tool' %}{%- set ns.is_tool = true -%}{%- if ns.is_output_first %}{% '<| tool_outputs_begin |>' + tool
_output_begin |>' + message['content'] + '<| tool_output_end |>' %}{%- set ns.is_output_first = false %}{%- else
%}{% '\n' + '<| tool_output_begin |>' + message['content'] + '<| tool_output_end |>' %}{%- endif %}{%- endif %}{%- endf
or -%}{%- if ns.is_tool %}{% '<| tool_outputs_end |>' %}{%- endif %}{%- if add_generation_prompt and not ns.is_tool %
%}{% '<| Assistant |>' %}{%- endif %}{% example_format: 'You are a helpful assistant

<| User |>Hello<| Assistant |>Hi there<| end_of_sentence |><| User |>How are you?<| Assistant |>'
main: server is listening on http://127.0.0.1:8080 - starting the main loop
srv update_slots: all slots are idle

```

步骤 2 打开电脑上任意 Python 编辑器，如 jupyter notebook，可点击以下链接下载包含代码的 jupyter notebook 文件，也可根据后面的步骤自己编写代码运行

<https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V4.0/chapter5/Web%20API%E8%B0%83%E7%94%A8.ipynb>

步骤 3 输入以下代码查看 Web 服务状态，状态码 200 表示正常，503 表示异常，红色部分为 Web 服务请求地址，如果在前面步骤有修改 Web 服务部署地址，这里需要同步修改

```

import requests
import json

def get_completion():
    response = requests.get(url='http://localhost:8080/health')
    return response

print(get_completion())

```

```
import requests
import json

def get_completion():
    response = requests.get(url='http://localhost:8080/health')
    return response

print(get_completion())

<Response [200]>
```

- 步骤 4 输入以下代码使用 Python 自带的 requests 库调用原始 API，输入提示时需要按照模板区分用户与模型的输入，其余参数配置可以参考以下链接：  
<https://github.com/ggerganov/llama.cpp/tree/master/examples/server#post-completion-given-a-prompt-it-returns-the-predicted-completion>

```
import requests
import json

def get_completion(prompt):
    headers = {'Content-Type': 'application/json'}
    data = {"prompt": prompt}
    response = requests.post(url='http://localhost:8080/completions', headers=headers,
data=json.dumps(data))
    return response.json()['content']

print(get_completion('< | User | >你好< | Assistant | >'))
```

#### 使用Python自带库调用原始API

```
[1]: import requests
import json

def get_completion(prompt):
    headers = {'Content-Type': 'application/json'}
    data = {"prompt": prompt}
    response = requests.post(url='http://localhost:8080/completions', headers=headers, data=json.dumps(data))
    return response.json()['content']

print(get_completion('< | User | >你好< | Assistant | >'))

<think>

</think>

你好！很高兴见到你，有什么我可以帮忙的吗？
```

- 步骤 5 输入以下代码使用 Python 自带的 requests 库调用 OpenAI-compatible API，这种方式使用起来会方便很多，，最后的输出结果是一个列表，可以根据需要，使用正则表达式等方式提取自己想要的内容。

```
import requests
import json

def get_completion(prompt):
    headers = {'Content-Type': 'application/json', 'Authorization': 'Bearer no-key'}
    data = {
        "model": "gpt-3.5-turbo",
```

```
"messages": [
{
    "role": "user",
    "content": prompt
}
]
}

response = requests.post(url='http://localhost:8080/v1/chat/completions', headers=headers,
data=json.dumps(data))
return response.json()['choices']

print(get_completion('你好'))
```

```
import requests
import json
def get_completion(prompt):
    headers = {'Content-Type': 'application/json', 'Authorization': 'Bearer no-key'}
    data = {
        "model": "gpt-3.5-turbo",
        "messages": [
            {
                "role": "user",
                "content": prompt
            }
        ]
    }
    response = requests.post(url='http://localhost:8080/v1/chat/completions', headers=headers, data=json.dumps(data))
    return response.json()['choices']

print(get_completion('你好'))

[{'finish_reason': 'stop', 'index': 0, 'message': {'content': '<think>\n\n</think>\n\n你好！有什么可以帮助你吗？', 'tool_calls': None, 'role': 'assistant'}}
```

步骤 6 输入以下代码使用 openai 库调用 OpenAI-compatible API，如果提示无法导入 openai，需输入以下命令安装对应的库，这种方式可以很方便的取得我们想要的输出，配置运行参数等

```
#安装依赖
pip install openai
```

```
import openai

client = openai.OpenAI(
    base_url="http://localhost:8080/v1", # 本地部署服务请求地址
    api_key = "sk-no-key-required"
)

completion = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[

        {"role": "user", "content": "你是谁？"}
    ]
)
```

```
print(completion)
print(completion.choices[0].message)
print(completion.choices[0].message.content)
```

```
import openai

client = openai.OpenAI(
    base_url="http://localhost:8080/v1", # 本地部署服务请求地址
    api_key = "sk-no-key-required"
)

completion = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "user", "content": "你是谁? "}
    ]
)

print(completion)
print('-----')
print(completion.choices[0].message)
print('-----')
print(completion.choices[0].message.content)
```

ChatCompletion(id='chatcmpl-q1O3foCjVR2pWTPqR1DIsA6D9KwNzgZP', choices=[Choice(finish\_reason='stop', index=0, logprobs=None, message=ChatCompletionMessage(content='<think>\n\n</think>\n\n您好！我是由中国的深度求索（DeepSeek）公司开发的智能助手DeepSeek-R1。如您有任何任何问题，我会尽我所能为您提供帮助。', refusal=None, role='assistant', audio=None, function\_call=None, tool\_calls=None))], created=1739347953, model='gpt-3.5-turbo', object='chat.completion', service\_tier=None, system\_fingerprint='b4689-90e4dba4', usage=CompletionUsage(completion\_tokens=40, prompt\_tokens=6, total\_tokens=46, completion\_tokens\_details=None, prompt\_tokens\_details=None), timings={'prompt\_n': 5, 'prompt\_ms': 97.121, 'prompt\_per\_token\_ms': 19.4242, 'prompt\_per\_second': 51.48217172393201, 'predicted\_n': 40, 'predicted\_ms': 1855.562, 'predicted\_per\_token\_ms': 46.38905, 'predicted\_per\_second': 21.556811359577317})

ChatCompletionMessage(content='<think>\n\n</think>\n\n您好！我是由中国的深度求索（DeepSeek）公司开发的智能助手DeepSeek-R1。如您有任何任何问题，我会尽我所能为您提供帮助。', refusal=None, role='assistant', audio=None, function\_call=None, tool\_calls=None)

<think>

</think>

您好！我是由中国的深度求索（DeepSeek）公司开发的智能助手DeepSeek-R1。如您有任何任何问题，我会尽我所能为您提供帮助。

## 2.3 思考题

我们在部署 LLM 时，哪些参数对模型输出的影响最大？

华为认证人工智能系列教程

# HCIA-AI人工智能

## 实验环境搭建指南

版本：4.0



华为技术有限公司

**版权所有 © 华为技术有限公司 2025。 保留一切权利。**

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## **商标声明**



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## **注意**

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

**华为技术有限公司**

地址：            深圳市龙岗区坂田华为总部办公楼            邮编： 518129

网址：            <https://e.huawei.com>

---

## 华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施认证、基础软硬件认证、云平台及云服务认证三类认证。

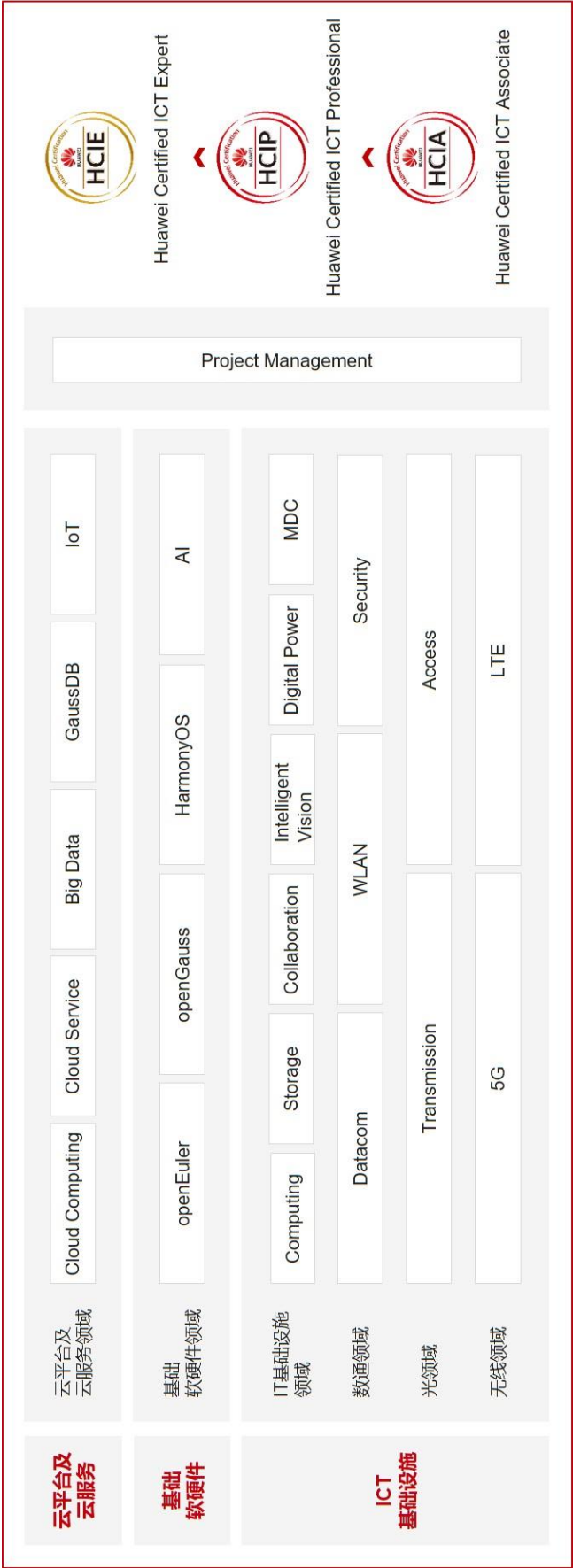
根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIA-AI V4.0 定位于培养和认证具备AI基础知识、掌握机器学习、深度学习和大模型等基础算法，可使用AI开发框架进行AI产品和应用开发的工程师。

通过HCIA-AI V4.0 认证，将证明您了解人工智能发展历史、人工智能前沿应用场景，掌握传统机器学习、深度学习和大模型的相关基础算法；具备利用AI开发框架进行搭建、训练、部署神经网络的能力；能够胜任人工智能领域产品开发和技术支持等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在人工智能世界的潮头浪尖！



## 简介

本书为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-AI 考试的学员或者希望了解 AI 基础知识及 MindSpore、PyTorch 编程基础的读者。以下三个部分请学员根据自身电脑的环境进行配置。

## 内容描述

本实验指导书共包含 3 个部分。

- 第一部分为 Windows 实验环境搭建。
- 第二部分为 Ubuntu 实验环境搭建。
- 第三部分为华为云使用指导手册。

## 读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本环境搭建指南的读者应首先具备以下基本条件：

- 具有基本的 Python 知识背景，熟悉基本 Python 编程知识。
-



# 目录

---

|                         |           |
|-------------------------|-----------|
| <b>前 言</b>              | <b>3</b>  |
| 简介                      | 3         |
| 内容描述                    | 3         |
| 读者知识背景                  | 3         |
| <b>1 Windows 实验环境配置</b> | <b>3</b>  |
| 1.1 Anaconda 安装         | 3         |
| 1.1.1 Anaconda 简介       | 3         |
| 1.1.2 安装步骤              | 3         |
| 1.2 更换国内源               | 6         |
| 1.2.1 Conda 命令更换国内源     | 6         |
| 1.2.2 pip 换源            | 7         |
| 1.3 Anaconda 虚拟环境介绍     | 7         |
| 1.3.1 虚拟环境介绍            | 7         |
| 1.3.2 利用命令提示符创建虚拟环境     | 8         |
| 1.3.3 激活虚拟环境            | 9         |
| 1.3.4 查看虚拟环境            | 9         |
| 1.3.5 删除虚拟环境            | 11        |
| 1.4 安装 MindSpore        | 11        |
| 1.4.1 MindSpore 版本安装    | 11        |
| <b>2 Ubuntu 实验环境配置</b>  | <b>17</b> |
| 2.1 Anaconda 安装         | 17        |
| 2.1.1 安装步骤              | 17        |
| 2.2 更换国内源               | 20        |
| 2.2.1 Conda 命令更换国内源     | 20        |
| 2.2.2 pip 换源            | 21        |
| 2.3 安装 MindSpore        | 21        |
| 2.3.1 MindSpore 版本安装    | 21        |

---



|                                             |           |
|---------------------------------------------|-----------|
| <b>3 华为云使用指导手册 .....</b>                    | <b>23</b> |
| 3.1 简介 .....                                | 23        |
| 3.2 华为云账号申请与实名认证 .....                      | 23        |
| 3.2.1 注册步骤 .....                            | 23        |
| 3.3 Access Key 和 Secret Access Key 的获取..... | 26        |
| 3.3.1 简介 .....                              | 26        |
| 3.3.2 生成 AK 和 SK.....                       | 26        |
| 3.4 华为云常用产品入口 .....                         | 27        |
| 3.4.1 ModelArts .....                       | 27        |

# 1 Windows 实验环境配置

---

## 1.1 Anaconda 安装

### 1.1.1 Anaconda 简介

Anaconda 是一个用于科学计算的 Python 发行版，支持 Linux，MacOS，Windows，提供了包管理与环境管理的功能，可以方便地解决多版本 Python 并存、切换以及各种第三方包安装问题。Anaconda 利用命令行工具 conda 来进行 package 和 environment 的管理（也可以使用 pip），并且已经包含了 Python 和相关的配套工具。Anaconda 可适用于企业级大数据分析的 python 工具。安装 Anaconda 发行版将自动安装超过 250 个包，安装完 Anaconda 发行版后也可以通过 PyPI 安装额外的开源包。Anaconda 不仅可以做数据分析，甚至可以用在大数据和人工智能领域。

Anaconda 有如下特性将极大方便开发者的使用：

1. Anaconda 发行版安装自带 Python，安装完了 Anaconda 就不需要再安装 Python 了。
2. 开发时，有时需要不同的框架进行支撑，只需要在 Anaconda 上添加虚拟环境，就可以在不同环境中完成开发，且不需要顾及兼容性问题，可以为特殊项目配置相应的环境，方便管理。

### 1.1.2 安装步骤

#### 步骤 1 下载 Anaconda

登陆 Anaconda 官网下载安装包，<https://repo.anaconda.com/archive/>，本例中选择 Windows 安装包，Windows 64 位用户推荐下载安装包：Anaconda3-20xx.xx-x-Windows-x86\_64.exe。（本例中以安装 Anaconda 过程为例，学员也可以使用 Miniconda，Miniconda 安装过程与本例类似。Miniconda 是 Anaconda 的轻量级替代，默认只包含 python 和 conda，可以通过 pip 或 conda 来安装所需的包。）

#### 步骤 2 安装 Anaconda

---

双击下载好的 Anaconda3-xxxx.xx-Windows-x86\_64.exe 文件，出现如下界面，点击 Next。

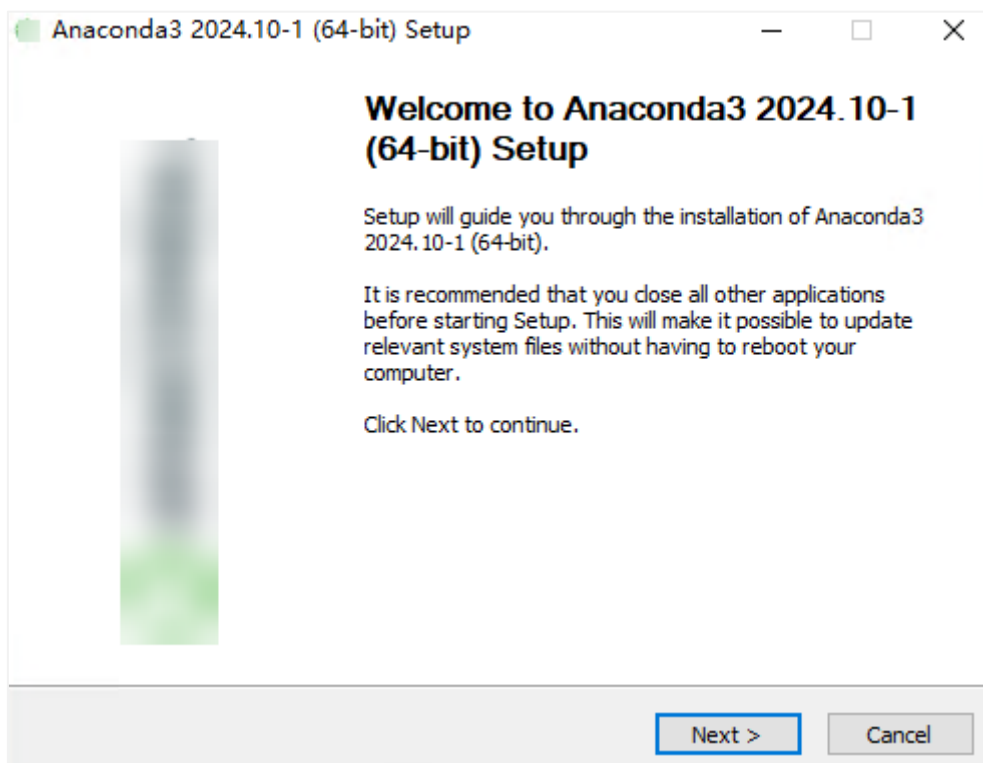
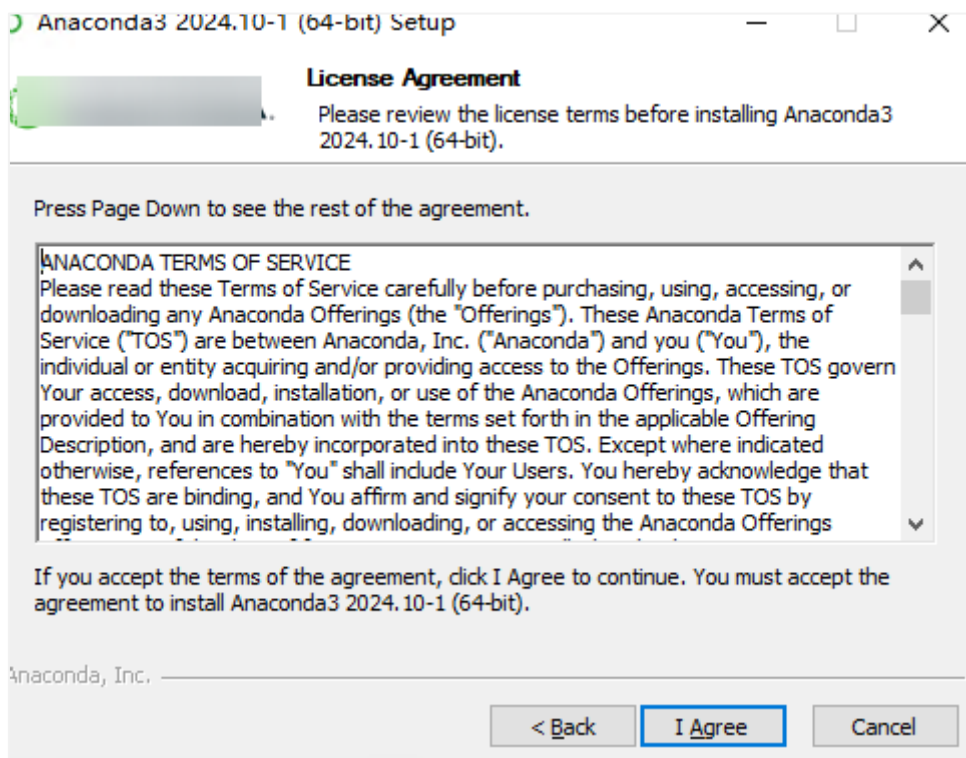


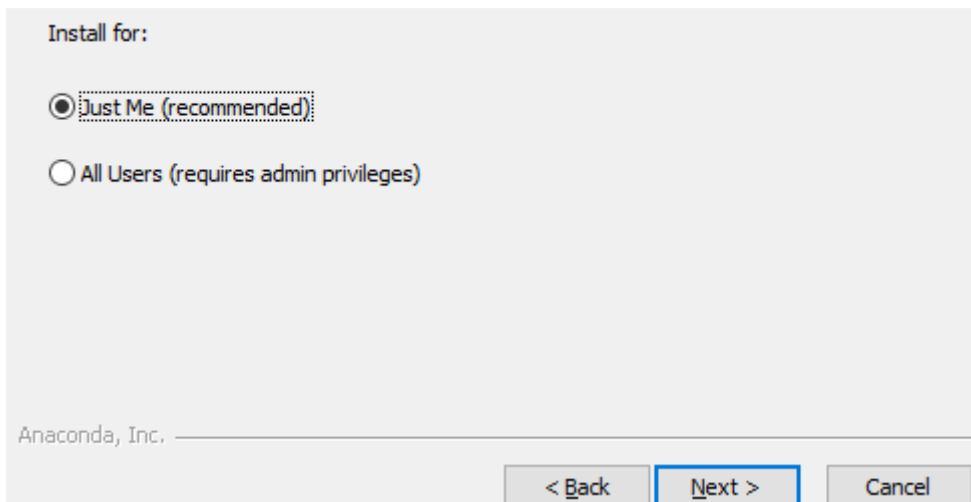
图1-1 Anaconda3 安装程序

点击 I Agree。



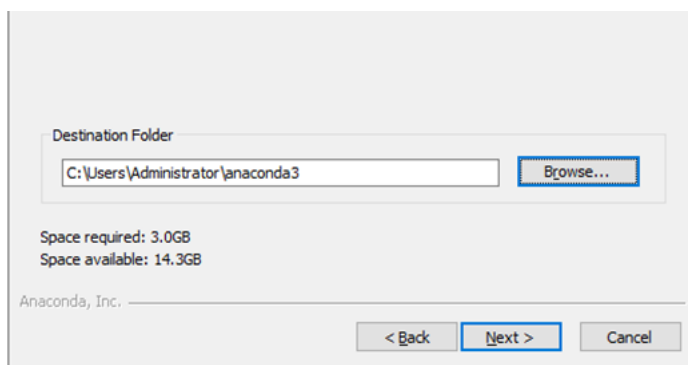
**图1-2 Anaconda3 安装许可**

Install for: Just me 还是 All Users，这里直接 Just Me，继续点击 Next 。


**图1-3 选择安装用户**

### 步骤 3 设置安装路径

选择软件安装路径，继续 Next。


**图1-4 选择软件安装路径**

### 步骤 4 配置环境变量

**两个都勾选**，第一个是加入环境变量，第二个是默认使用 Python 3.9，可以减少我们后续配置的工作，点击 Install 开始安装。（注意：如果本机器上已经装有 Python 的其他版本，建议先卸载，然后安装 Anaconda。如未删除，容易引发环境冲突）。

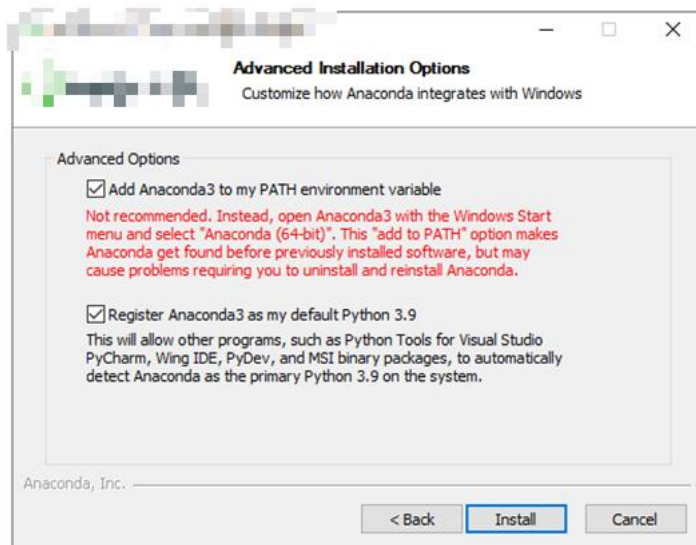


图1-5 选择将 Anaconda3 加入环境变量

等待完成安装，点击 finish 即可。

## 1.2 更换国内源

### 1.2.1 Conda 命令更换国内源

通常情况下，使用 conda 命令或者 pip 命令都是从国外的服务器上索引并下载需要的包，这在网速不佳的情况下会消耗大量的时间。所以这里建议更换国内的源，能节省大量下载时间。

在 cmd 命令提示符界面输入以下命令生成 .condarc 文件。

```
conda config --set show_channel_urls yes
```

管理员: C:\windows\system32\cmd.exe

```
Microsoft Windows [版本 10.0.20348.2655]
(c) Microsoft Corporation。保留所有权利。

C:\Users\Administrator>
C:\Users\Administrator>conda config --set show_channel_urls yes
```

图1-6 生成 .condarc 文件

在安装用户目录下可复制 .condarc 存一份备份文件，并以记事本方式打开 .condarc 文件，并用以下配置进行替换。

```
channels:
  - defaults
show_channel_urls: true
default_channels:
```

```
- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/main
- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/r
- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/msys2
custom_channels:
conda-forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
pytorch-lts: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
MindSpore: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
```

保存.condarc 文件后，在 cmd 命令行中执行以下命令清除索引缓存。

```
conda clean -i
```

## 1.2.2 pip 换源

上一小节我们介绍了如何给 conda 换源，这里我们介绍如何给 pip 换源。pip 源有两种方式可以更换，一种是临时更换，仅限于当前的这一条命令，另一种为默认更换，更换后所有命令默认以更换后的源来进行资源下载。

### 1.2.2.1 临时更换 pip 源

在每一条安装包命令后面加-l 镜像源地址，即可更换，如下所示即在当前环境下安装清华大学开源镜像站上可适配本机架构的 MindSpore 最新版：

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple MindSpore
```

### 1.2.2.2 更换默认 pip 源

pip 版本大于 10.0.0 可通过在 cmd 命令行中执行以下命令配置 pip 工具默认索引与下载源（需注意该配置步骤仅针对当前虚拟环境，在下面步骤中新建虚拟环境后需重新执行）。

```
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

## 1.3 Anaconda 虚拟环境介绍

### 1.3.1 虚拟环境介绍

Anaconda 的虚拟环境是它如此流行的主要原因，它可以在本机上虚拟出多个相互无关的 Python 环境供开发者使用，当不同 Python 模块间存在依赖混乱的情况，或者存在不同开发框架应用时，利用虚拟环境对不同环境进行隔离是非常不错的选择。

注意：虚拟环境中的各个模块是相互独立的，如果你在 A 环境安装了 MindSpore，同时，你想在 B 环境使用 MindSpore，那么你需要在 B 环境完成同样的安装，这也同样适用于 Spyder 和 Jupyter Notebook。

## 1.3.2 利用命令提示符创建虚拟环境

步骤 1 任务栏搜索框中搜索 cmd，打开命令提示符：

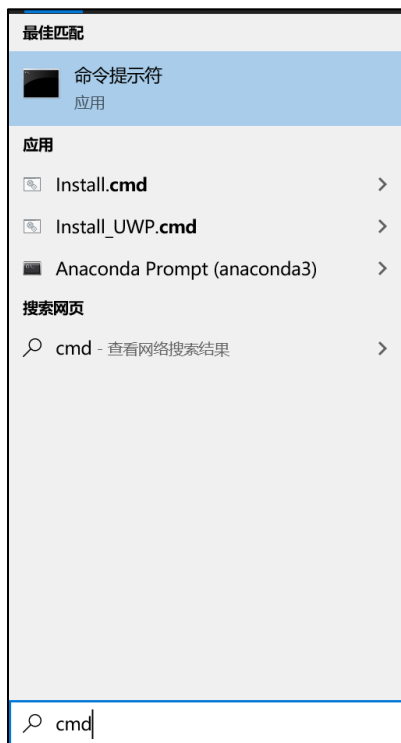


图1-7 打开命令提示符

步骤 2 输入命令 `conda create -n 环境名称 python==x.x`，其中环境名称自拟，Python 版本需要自己确定：

```
C:\Users\Administrator>conda create -n MS python=3.7
Channels:
 - https://repo.anaconda.com/pkgs/main
 - https://repo.anaconda.com/pkgs/r
 - https://repo.anaconda.com/pkgs/msys2
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: C:\Users\Administrator\anaconda3\envs\MS

added / updated specs:
 - python=3.7
```

**图1-8 创建 MS 虚拟环境**

步骤 3 出现选择时，选 y

等待安装完成。

```
Proceed ([y]/n)? y

Downloading and Extracting Packages:
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate MS
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

**图1-9 MS 虚拟环境创建完成**

### 1.3.3 激活虚拟环境

在命令提示符中输入命令 `activate 环境名称`，激活对应的虚拟环境，命令行前方括号内名称改变，则证明启动并进入该环境：

```
C:\Users\Administrator>conda activate MS
(MS) C:\Users\Administrator>_
```

**图1-10 激活 MS 虚拟环境**

此时就可以使用 pip 或 conda 命令开始安装各模块。

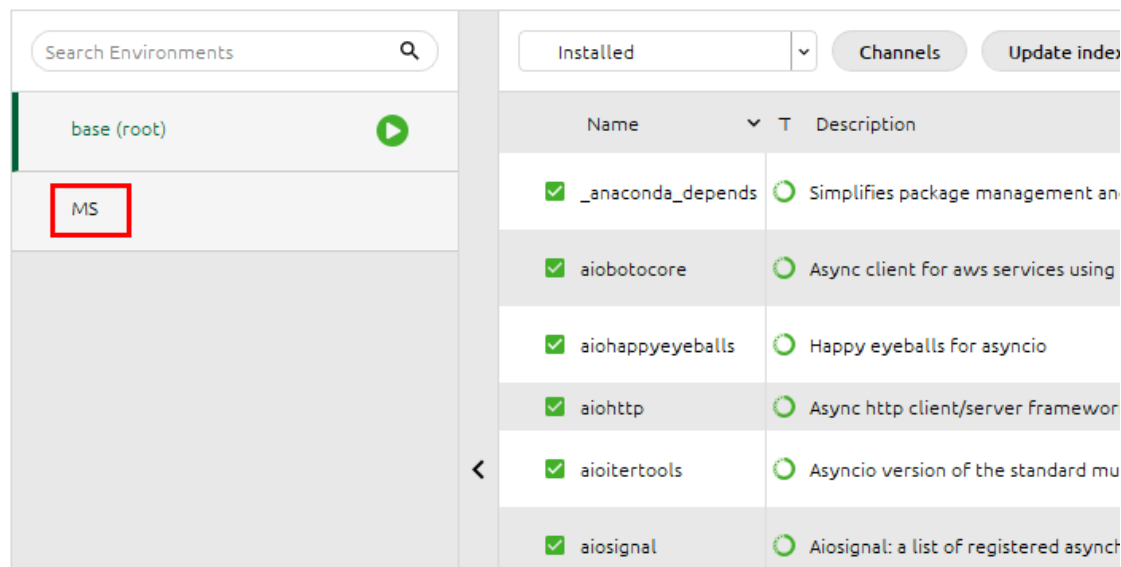
退出环境：

```
conda deactivate
```

### 1.3.4 查看虚拟环境

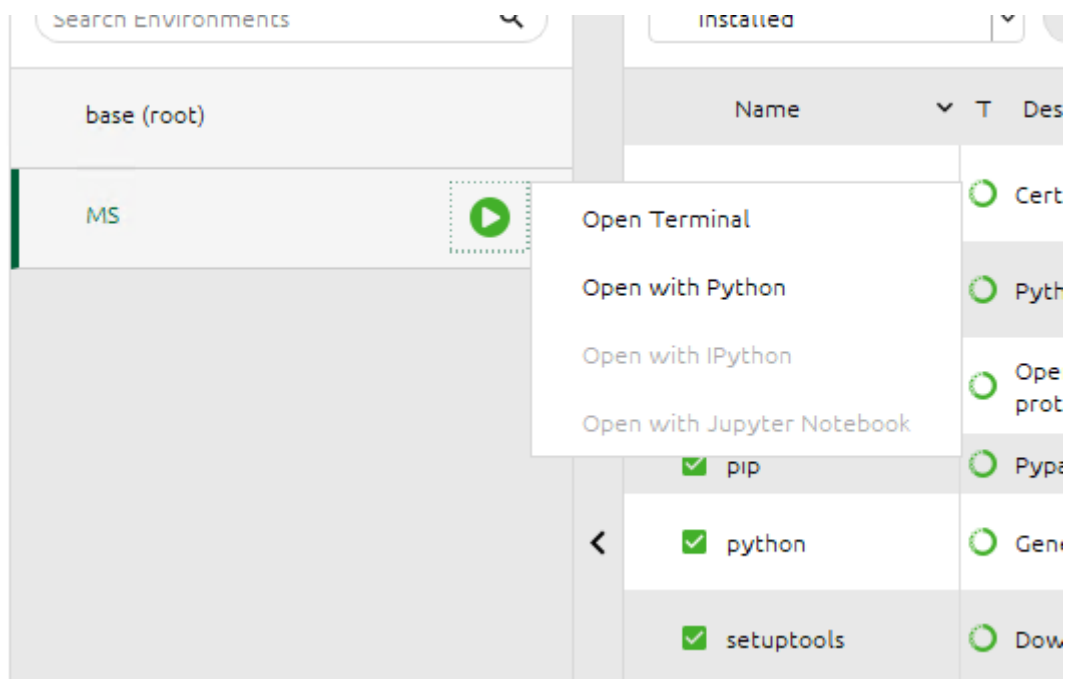
可以通过 Anaconda Navigator 查看已经创建的虚拟环境：

## IDA.NAVIGATOR



**图1-11 Anaconda Navigator 上已创建的虚拟环境**

也可以通过环境名称右边的小三角直接进入虚拟环境对应的命令提示符：



**图1-12 打开对应虚拟环境的 Terminal**

此时，不需要再启动虚拟环境，就可以将模块安装到当前环境下。

### 1.3.5 删除虚拟环境

在命令提示符中输入命令 `conda remove -n 环境名称 --all`，并输入 `y`，即可删除环境。如图所示：

```
C:\Users\Administrator>conda remove -n MS --all

Remove all packages in environment C:\Users\Administrator\anaconda3\envs\MS:

## Package Plan ##

  environment location: C:\Users\Administrator\anaconda3\envs\MS

The following packages will be REMOVED:

ca-certificates-2024.12.31-haa95532_0
certifi-2022.12.7-py37haa95532_0
openssl-1.1.1w-h2bbff1b_0
pip-22.3.1-py37haa95532_0
python-3.7.16-h6244533_0
setuptools-65.6.3-py37haa95532_0
sqlite-3.45.3-h2bbff1b_0
vc-14.40-haa95532_2
vs2015_runtime-14.42.34433-h9531ae6_2
wheel-0.38.4-py37haa95532_0
wincertstore-0.2-py37haa95532_2

Proceed ([y]/n)?
```

图1-13 删除虚拟环境

## 1.4 安装工具库

### 1.4.1 MindSpore 安装

步骤 1 创建虚拟环境（如果已完成本环境搭建指南 1.3 节，可跳过该步骤）

虚拟环境的创建是为了保证各环境间不冲突，其关系类似于虚拟机和系统的关系。输入以下命令创建虚拟环境：

```
conda create -n MS python=3.9.
```

红色斜体部分是环境名称，可以自定义。python=3.9 为指定新创建的虚拟环境 python 版本，如果不指定 python 小版本即安装当前大版本下最高的 python 小版本。如果想指定 python 小版本可输入如 `python=3.7.10`。

安装过程中如果遇到 y/n 选择，请输入 y。

```
C:\Users\Administrator>conda create -n MS python=3.9
Channels:
 - https://repo.anaconda.com/pkgs/main
 - https://repo.anaconda.com/pkgs/r
 - https://repo.anaconda.com/pkgs/msys2
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Administrator\anaconda3\envs\MS

added / updated specs:
 - python=3.9

The following packages will be downloaded:

package                                     build                                     7.8 MB https://repo.anaconda.com/pkgs/main
-----
openssl-3.0.15                             h827c3e9_0                             2.4 MB https://repo.anaconda.com/pkgs/main
pip-24.2                                   py39haa95532_0                         19.6 MB https://repo.anaconda.com/pkgs/main
python-3.9.21                             h8205438_1                             1.6 MB https://repo.anaconda.com/pkgs/main
setuptools-75.1.0                         py39haa95532_0                         115 KB https://repo.anaconda.com/pkgs/main
tzdata-2024b                             h04d1e81_0                             137 KB https://repo.anaconda.com/pkgs/main
wheel-0.44.0                               py39haa95532_0
-----
Total:                                     31.7 MB

The following NEW packages will be INSTALLED:

ca-certificates pkgs/main/win-64::ca-certificates-2024.12.31-haa95532_0
openssl         pkgs/main/win-64::openssl-3.0.15-h827c3e9_0
pip             pkgs/main/win-64::pip-24.2-py39haa95532_0
python         pkgs/main/win-64::python-3.9.21-h8205438_1
setuptools     pkgs/main/win-64::setuptools-75.1.0-py39haa95532_0
sqlite         pkgs/main/win-64::sqlite-3.45.3-h2bfff1b_0
tzdata        pkgs/main/noarch::tzdata-2024b-h04d1e81_0
vc            pkgs/main/win-64::vc-14.40-haa95532_2
vs2015_runtime pkgs/main/win-64::vs2015_runtime-14.42.34433-h9531ae6_2
wheel         pkgs/main/win-64::wheel-0.44.0-py39haa95532_0

Proceed ([y]/n)? _
```

图1-14 创建 MS 虚拟环境

步骤 2 激活虚拟环境（如果已完成本环境搭建指南 1.3 节，可跳过该步骤）

安装完成后，需要激活虚拟环境，输入以下命令激活虚拟环境：conda activate **MS17**。红色斜体部分是步骤 1 中创建的虚拟环境名称。激活后可以在命令行前看到（**虚拟环境名**）。

```
C:\Users\Administrator>conda activate MS
(MS) C:\Users\Administrator>_
```

图1-15 激活虚拟环境

步骤 3 安装 MindSpore

接上一步骤完成后，在浏览器中输入地址：<https://www.mindspore.cn/install>。

获取安装命令。

## 一、获取安装命令

|      |                                                                                                                                                                                                                                                                                            |              |                        |               |               |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|------------------------|---------------|---------------|
| 版本   | <b>2.4.10</b>                                                                                                                                                                                                                                                                              | 2.4.1        | master (Nightly build) |               |               |
| 硬件平台 | <b>Ascend</b>                                                                                                                                                                                                                                                                              | CPU          | GPU CUDA 10.1          | GPU CUDA 11.1 | GPU CUDA 11.6 |
| 操作系统 | <b>Linux-aarch64</b>                                                                                                                                                                                                                                                                       | Linux-x86_64 | Windows-x64            | MacOS-aarch64 | MacOS-x86_64  |
| 编程语言 | <b>Python 3.9</b>                                                                                                                                                                                                                                                                          | Python 3.10  | Python 3.11            |               |               |
| 安装方式 | <b>Pip</b>                                                                                                                                                                                                                                                                                 | Conda        | Source                 | Docker        |               |
| 安装命令 | <pre>pip install https://ms-release.obs.cn-north-4.myhuaweicloud.com/2.4.10/MindSpore/unified/aarch64/mindspore-2.4.10-cp39-cp39-linux_aarch64.whl --trusted-host ms-release.obs.cn-north-4.myhuaweicloud.com -i https://pypi.tuna.tsinghua.edu.cn/simple # 注意参考下方安装指南，添加运行所需的环境变量配置</pre> |              |                        |               |               |

图1-16 安装命令

根据自己的环境及需求选取。如：选取 2.4.10、CPU、Windows-X64、Python 3.9、pip 后，在 cmd 命令行中输入安装命令。

```
pip install https://ms-release.obs.cn-north-4.myhuaweicloud.com/2.4.10/MindSpore/cpu/x86_64/mindspore-2.4.10-cp39-cp39-win_amd64.whl --trusted-host ms-release.obs.cn-north-4.myhuaweicloud.com -i https://pypi.tuna.tsinghua.edu.cn/simple
```

显示 Successfully 则表示安装成功。

```
(MS) C:\Users\Administrator>pip install https://ms-release.obs.cn-north-4.myhuaweicloud.com/2.4.10/MindSpore/cpu/x86_64/mindspore-2.4.10-cp39-cp39-win_amd64.whl --trusted-host ms-release.obs.cn-north-4.myhuaweicloud.com -i https://pypi.tuna.tsinghua.edu.cn/simple
Collecting mindspore==2.4.10
  Downloading https://ms-release.obs.cn-north-4.myhuaweicloud.com/2.4.10/MindSpore/cpu/x86_64/mindspore-2.4.10-cp39-cp39-win_amd64.whl (100.5 MB)
----- 100.5/100.5 MB 77.3 MB/s eta 0:00:00
Collecting numpy<2.0.0,>=1.20.0 (from mindspore==2.4.10)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/b5/42/054082bd8220bbf6f297f982f0a8f5479fcb55c8b511d928df07b965369/numpy-1.26.4-cp39-cp39-win_amd64.whl (15.8 MB)
----- 15.8/15.8 MB 165.5 MB/s eta 0:00:00
Collecting protobuf>=3.13.0 (from mindspore==2.4.10)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/ac/e2/48d46adc86369ff092eaece3e537f76b3baaab45ca3dde257838cde831d2/protobuf-5.29.3-cp39-cp39-win_amd64.whl (434 kB)
Collecting asttokens>=2.0.4 (from mindspore==2.4.10)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/25/8a/c46dce25341b5bce5472c718902eb3d38600a903b14fa6aececf3f21a46f/asttokens-3.0.0-py3-none-any.whl (26 kB)
Collecting pillow>=6.2.0 (from mindspore==2.4.10)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/21/a6/f51d47675940b5c63b08ff0575b3518428b4acb891f88526fa4ee1edab6f/pillow-11.1.0-cp39-cp39-win_amd64.whl (2.6 MB)
----- 2.6/2.6 MB 18.9 MB/s eta 0:00:00
Collecting scipy>=1.5.4 (from mindspore==2.4.10)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/3e/77/dab54fe647a08ee4253963bcd8f9cf17509c8ca64d6335141422fe2e2114/scipy-1.13.1-cp39-cp39-win_amd64.whl (46.2 MB)
----- 46.2/46.2 MB 95.0 MB/s eta 0:00:00
Collecting packaging>=20.0 (from mindspore==2.4.10)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/88/ef/eb23f262cca3c0c4eb7ab1933c3b1f03d021f2c48f54763065b6f0e321be/packaging-24.2-py3-none-any.whl (65 kB)
Collecting psutil>=5.6.1 (from mindspore==2.4.10)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/7b/d7/7831438e6c3ebbf6a01a927127a6cb42ad3ab844247f3c5b96bea25d73d/psutil-6.1.1-cp37-abi3-win_amd64.whl (254 kB)
Collecting astunparse>=1.6.3 (from mindspore==2.4.10)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/2b/03/13dde6512ad7b4557eb792fbc0c653af6076b81e5941d36ec61f7ce6028/astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Collecting safetensors>=0.4.0 (from mindspore==2.4.10)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/86/ca/aa489392ec6fb59223ffce825461ef811a3affd417121a2088be7a5758b/safetensors-0.5.2-cp38-abi3-win_amd64.whl (303 kB)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\administrator\anaconda3\envs\ms\lib\site-packages (from astunparse==1.6.3->mindspore==2.4.10) (0.44.0)
Collecting six<2.0,>=1.6.1 (from astunparse==1.6.3->mindspore==2.4.10)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/b7/ce/149a00dd41f10bc29e5921b496af8b574d8413afcd5e30dfa0ed46c2cc5e/six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, safetensors, psutil, protobuf, pillow, packaging, numpy, asttokens, scipy, astunparse, mindspore
Successfully installed asttokens-3.0.0 astunparse-1.6.3 mindspore-2.4.10 numpy-1.26.4 packaging-24.2 pillow-11.1.0 protobuf-5.29.3 psutil-6.1.1 safetensors-0.5.2 scipy-1.13.1 six-1.17.0
```

**图1-17 成功通过 pip 安装 MindSpore 界面****步骤 4 安装成功测试**

接上一步骤完成后，在 cmd 命令行中输入以下命令验证 MindSpore 是否安装成功。

```
python -c "import mindspore;mindspore.run_check()"
```

如果输出以下内容则表示 MindSpore 安装成功。

MindSpore version: 版本号

The result of multiplication calculation is correct, MindSpore has been installed successfully!

## 1.4.2 PyTorch 安装

新建虚拟环境 PyTorch 并激活（创建过程参考上面步骤）。

### 步骤 1 安装 PyTorch

进入 PyTorch 官网获取安装命令：<https://pytorch.org/get-started/locally/>。

**NOTE:** Latest PyTorch requires Python 3.9 or later.

|                   |                                           |           |           |                   |          |
|-------------------|-------------------------------------------|-----------|-----------|-------------------|----------|
| PyTorch Build     | Stable (2.5.1)                            |           |           | Preview (Nightly) |          |
| Your OS           | Linux                                     |           | Mac       |                   | Windows  |
| Package           | Conda                                     |           | Pip       |                   | LibTorch |
| Language          | Python                                    |           |           | C++ / Java        |          |
| Compute Platform  | CUDA 11.8                                 | CUDA 12.1 | CUDA 12.4 | ROCm 6.2          | CPU      |
| Run this Command: | pip3 install torch torchvision torchaudio |           |           |                   |          |

图1-18 PyTorch 安装命令

根据自己的需求选取对应的配置。

在虚拟环境中安装：

```
(PyTorch) C:\Users\Administrator>pip3 install torch torchvision torchaudio
Collecting torch
  Downloading torch-2.5.1-cp39-cp39-win_amd64.whl.metadata (28 kB)
Collecting torchvision
  Downloading torchvision-0.20.1-cp39-cp39-win_amd64.whl.metadata (6.2 kB)
Collecting torchaudio
  Downloading torchaudio-2.5.1-cp39-cp39-win_amd64.whl.metadata (6.5 kB)
Collecting filelock (from torch)
  Downloading filelock-3.16.1-py3-none-any.whl.metadata (2.9 kB)
Collecting typing_extensions>=4.8.0 (from torch)
  Downloading typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Collecting networkx (from torch)
  Downloading networkx-3.2.1-py3-none-any.whl.metadata (5.2 kB)
Collecting jinja2 (from torch)
  Downloading jinja2-3.1.5-py3-none-any.whl.metadata (2.6 kB)
Collecting fsspec (from torch)
  Downloading fsspec-2024.12.0-py3-none-any.whl.metadata (11 kB)
Collecting sympy<=1.13.1 (from torch)
  Downloading sympy-1.13.1-py3-none-any.whl.metadata (12 kB)
Collecting mpmath<1.4.0>=1.1.0 (from sympy==1.13.1->torch)
  Downloading mpmath-1.3.0-py3-none-any.whl.metadata (3.6 kB)
Collecting numpy (from torchvision)
  Downloading numpy-2.0.2-cp39-cp39-win_amd64.whl.metadata (59 kB)
Collecting pillow!=8.3.*,>=5.3.0 (from torchvision)
  Downloading pillow-11.1.0-cp39-cp39-win_amd64.whl.metadata (9.3 kB)
Collecting MarkupSafe>=2.0 (from jinja2->torch)
  Downloading MarkupSafe-3.0.2-cp39-cp39-win_amd64.whl.metadata (4.1 kB)
Downloading torch-2.5.1-cp39-cp39-win_amd64.whl (203.0 MB)
----- 203.0/203.0 MB 71.3 MB/s eta 0:00:00
Downloading sympy-1.13.1-py3-none-any.whl (6.2 MB)
----- 6.2/6.2 MB 191.2 MB/s eta 0:00:00
Downloading torchvision-0.20.1-cp39-cp39-win_amd64.whl (1.6 MB)
----- 1.6/1.6 MB ? eta 0:00:00
Downloading torchaudio-2.5.1-cp39-cp39-win_amd64.whl (2.4 MB)
----- 2.4/2.4 MB 135.7 MB/s eta 0:00:00
Downloading pillow-11.1.0-cp39-cp39-win_amd64.whl (2.6 MB)
----- 2.6/2.6 MB 76.2 MB/s eta 0:00:00
Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Downloading filelock-3.16.1-py3-none-any.whl (16 kB)
Downloading fsspec-2024.12.0-py3-none-any.whl (183 kB)
Downloading jinja2-3.1.5-py3-none-any.whl (134 kB)
Downloading networkx-3.2.1-py3-none-any.whl (1.6 MB)
----- 1.6/1.6 MB 6.3 MB/s eta 0:00:00
Downloading numpy-2.0.2-cp39-cp39-win_amd64.whl (15.9 MB)
----- 15.9/15.9 MB 14.3 MB/s eta 0:00:00
Downloading MarkupSafe-3.0.2-cp39-cp39-win_amd64.whl (15 kB)
Downloading mpmath-1.3.0-py3-none-any.whl (536 kB)
----- 536.2/536.2 kB 17.1 MB/s eta 0:00:00
Installing collected packages: mpmath, typing_extensions, sympy, pillow, numpy, networkx, MarkupSafe, fsspec, filelock, jinja2, torch, torchvision, torch
audio
Successfully installed MarkupSafe-3.0.2 filelock-3.16.1 fsspec-2024.12.0 jinja2-3.1.5 mpmath-1.3.0 networkx-3.2.1 numpy-2.0.2 pillow-11.1.0 sympy-1.13.1
torch-2.5.1 torchaudio-2.5.1 torchvision-0.20.1 typing_extensions-4.12.2
```

图1-19 PyTorch 安装

出现 successfully 即安装完成。

## 步骤 2 安装成功测试

接上一步骤完成后，在 cmd 命令行中（PyTorch 虚拟环境中）输入以下命令验证是否安装成功。

```
python -c "import torch;print(torch.__version__)"
```

如果输出版本号则安装成功。

```
2.5.1+cpu
```

## 1.4.3 其他工具库安装

激活对应的虚拟环境后使用 pip 或者 conda 进行安装（如安装实验中需要的 easydict）：

```
pip install easydict
```

# 2 Ubuntu 实验环境配置

## 2.1 Anaconda 安装

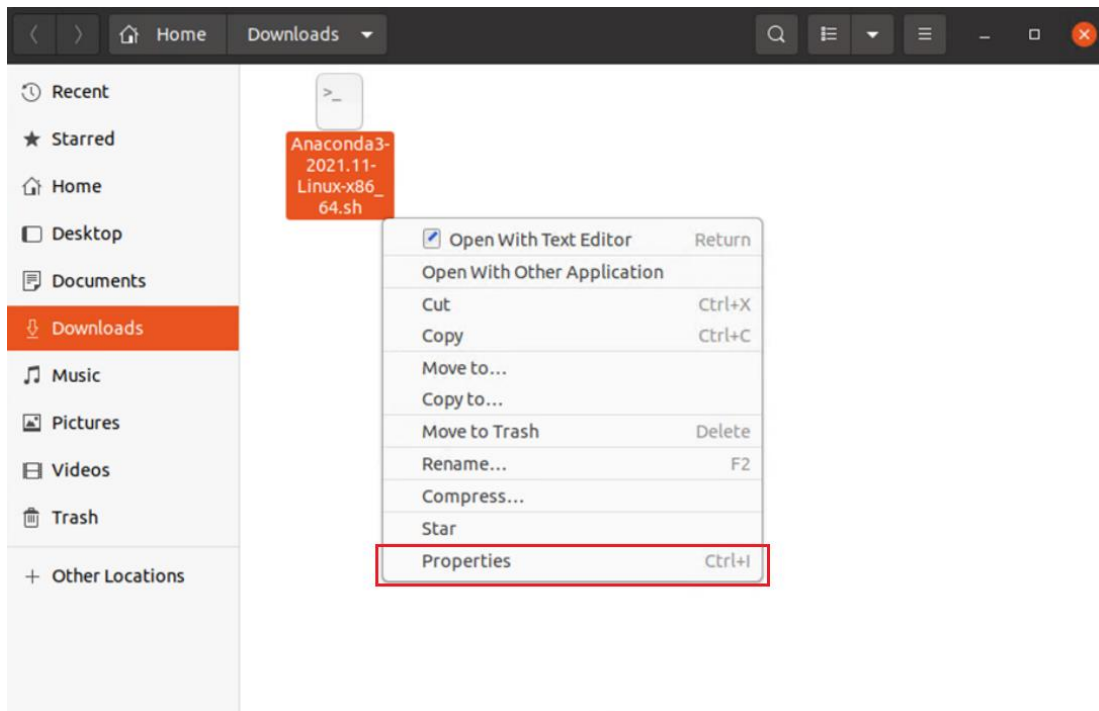
### 2.1.1 安装步骤

#### 步骤 1 下载 Anaconda

登陆 Anaconda 官网下载安装包，<https://repo.anaconda.com/archive/>，本例中选择 Linux 安装包，Linux 64 位用户推荐下载安装包：Anaconda3-20xx.xx-Linux-x86\_64.exe。（本例中以安装 Anaconda 过程为例，学员也可以使用 Miniconda，Miniconda 安装过程与本例类似。Miniconda 是 Anaconda 的轻量级替代，默认只包含 python 和 conda，可以通过 pip 或 conda 来安装所需的包。）

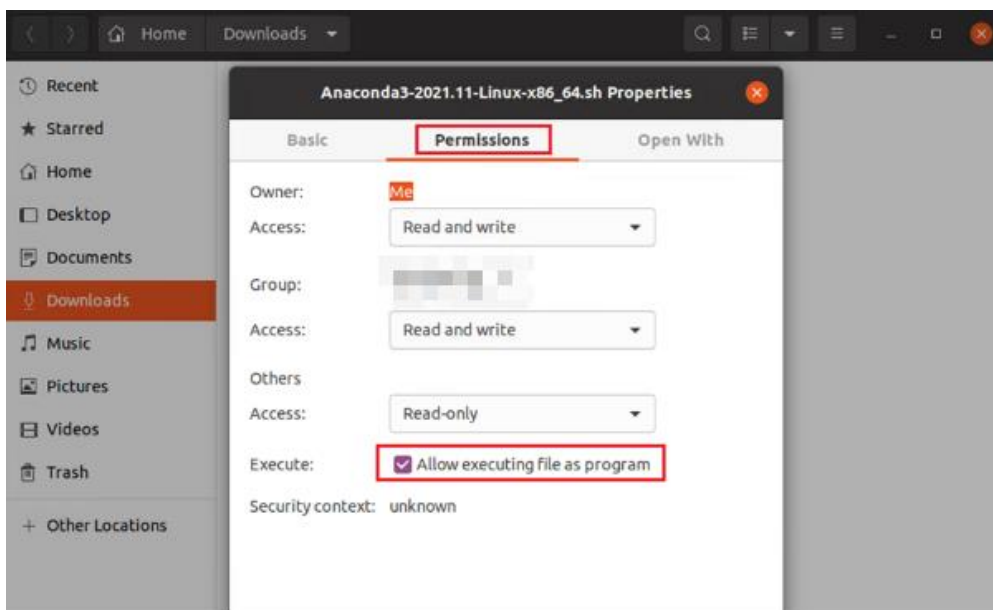
#### 步骤 2 安装 Anaconda

找到下载好的 Anaconda3-xxxx.xx-Linux-x86\_64.sh 文件，然后右击文件，点击 Properties。



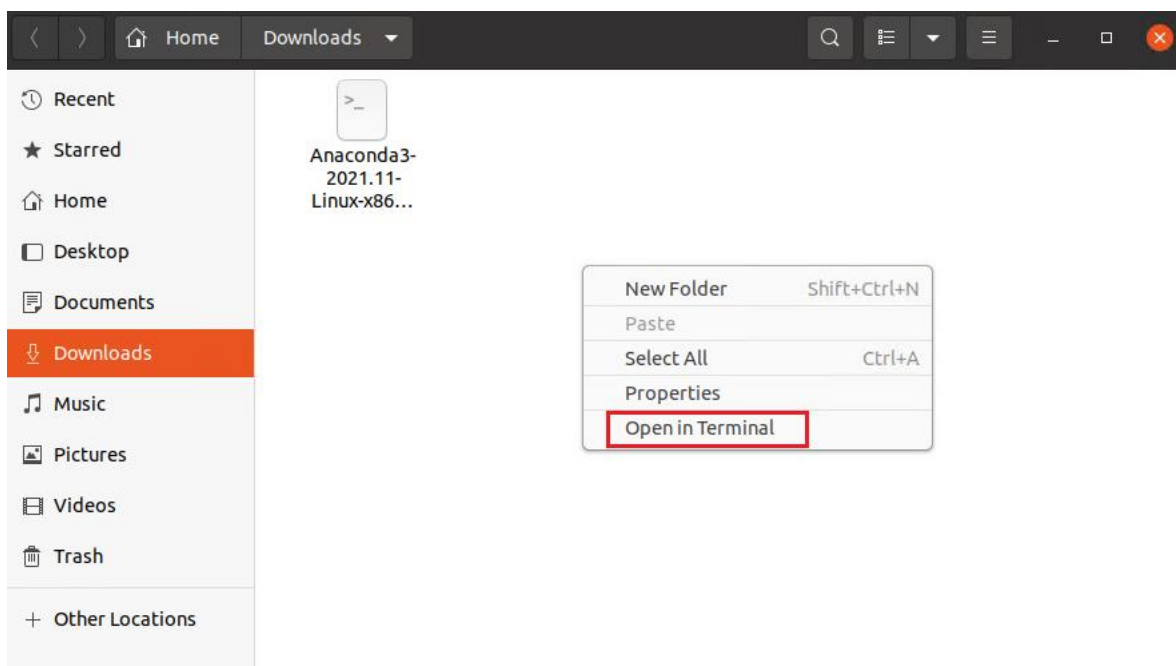
**图2-1 右击打开文件属性**

选择 Permissions 选项卡并在 “Execute” 处进行勾选。



**图2-2 勾选允许可执行文件**

在文件所在位置右击空白处，新建一个终端。



**图2-3 在当前目录打开 Terminal**

于 Terminal 中输入以下命令进行安装。

```
./Anaconda3-2021.11-Linux-x86_64..sh
```

安装过程需要按“ENTER”键阅读安装许可，阅读完毕后需接受或拒绝，默认为 no，需手动输入 yes。接受许可后将确认 Anaconda3 安装路径，学员可自行配置 Linux 安装路径地址，按“ENTER”键即默认将 Anaconda3 装至用户目录下。

```
Please answer 'yes' or 'no':
>>> yes

Anaconda3 will now be installed into this location:
/home/username/anaconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

[/home/username/anaconda3] >>>
```

图2-4 Anaconda3 安装路径

安装结束需要输入 yes 初始化 Anaconda3。

```
zope.interface      pkgs/main/linux-64::zope.interface-5.4.0-py39h7f8727e_0
zstd                pkgs/main/linux-64::zstd-1.4.9-haebb681_0

Preparing transaction: done
Executing transaction: -

  Installed package of scikit-learn can be accelerated using scikit-learn-intelex.
  More details are available here: https://intel.github.io/scikit-learn-intele
x

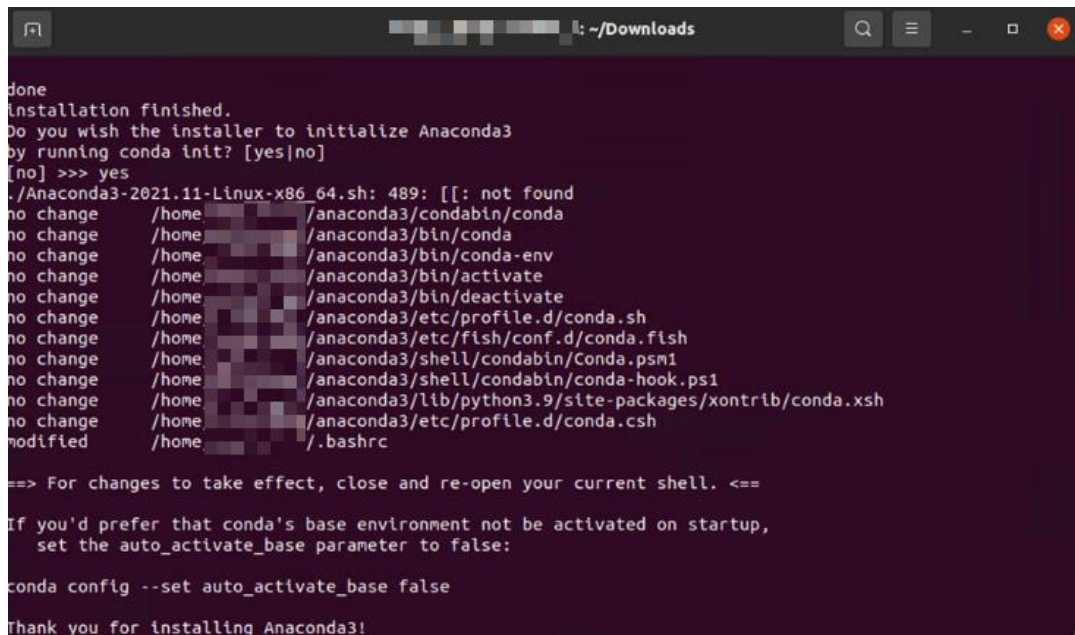
  For example:

    $ conda install scikit-learn-intele
    $ python -m sklearn my_application.py

done
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes/no]
[no] >>>
```

图2-5 初始化 Anaconda3

完成上述操作后，Anaconda3 安装成功。退出当前 Terminal 并新建 Terminal 窗口以重新初始化命令行环境变量。



```
done
Installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> yes
./Anaconda3-2021.11-Linux-x86_64.sh: 489: [[: not found
no change /home/anaconda3/condabin/conda
no change /home/anaconda3/bin/conda
no change /home/anaconda3/bin/conda-env
no change /home/anaconda3/bin/activate
no change /home/anaconda3/bin/deactivate
no change /home/anaconda3/etc/profile.d/conda.sh
no change /home/anaconda3/etc/fish/conf.d/conda.fish
no change /home/anaconda3/shell/condabin/Conda.psm1
no change /home/anaconda3/shell/condabin/conda-hook.ps1
no change /home/anaconda3/lib/python3.9/site-packages/xontrib/conda.xsh
no change /home/anaconda3/etc/profile.d/conda.csh
modified /home/.bashrc

==> For changes to take effect, close and re-open your current shell. <==

If you'd prefer that conda's base environment not be activated on startup,
set the auto_activate_base parameter to false:

conda config --set auto_activate_base false

Thank you for installing Anaconda3!
```

图2-6 Anaconda3 安装完成界面

## 2.2 更换国内源

### 2.2.1 Conda 命令更换国内源

通常情况下，使用 conda 命令或者 pip 命令都是从国外的服务器上索引并下载我们需要的包，这在网速不佳的情况下会消耗大量的时间。所以这里我们建议更换国内的源，能节省大量下载时间。

新建的 Terminal 界面输入以下命令生成 condaarc 文件。

```
conda config --set show_channel_urls yes
```

在安装用户目录下可复制 condaarc 存一份备份文件，并通过 Vim 工具打开 condaarc 文件，并用以下配置进行替换。

```
channels:
  - defaults
show_channel_urls: true
default_channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2
custom_channels:
  conda-forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch-lts: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
```

```
MindSpore: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
```

保存.condarc 文件后，在 cmd 命令行中执行以下命令清除索引缓存。

```
conda clean -i
```

## 2.2.2 pip 换源

上一小节我们介绍了如何给 conda 换源，这里我们介绍如何给 pip 换源。pip 源有两种方式可以更换，一种是临时更换，仅限于当前的这一条命令，另一种为默认更换，更换后所有命令默认以更换后的源来进行资源下载。

### 2.2.2.1 临时更换 pip 源

在每一条安装包命令后面加 -i 镜像源地址，即可更换，如下所示即在当前环境下安装清华大学开源镜像站上可适配本机架构的 MindSpore 最新版：

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple MindSpore
```

### 2.2.2.2 更换默认 pip 源

pip 版本大于 10.0.0 可通过在 cmd 命令行中执行以下命令配置 pip 工具默认索引与下载源（需注意该配置步骤仅针对当前虚拟环境，在下面步骤中新建虚拟环境后需重新执行）。

```
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

## 2.3 安装 MindSpore

### 2.3.1 MindSpore 版本安装

步骤 1 创建虚拟环境（以下步骤可参考 1.3 中创建过程）

创建虚拟环境 MS。

步骤 2 激活虚拟环境

激活虚拟环境（MS）。

步骤 3 安装 MindSpore

接上一步骤完成后，在 Terminal 中输入

```
pip install https://ms-release.obs.cn-north-4.myhuaweicloud.com/2.4.10/MindSpore/unified/x86_64/mindspore-2.4.10-cp39-cp39-linux_x86_64.whl --trusted-host ms-release.obs.cn-north-4.myhuaweicloud.com -i https://pypi.tuna.tsinghua.edu.cn/simple
```

显示 Successfully 则表示安装成功。

#### 步骤 4 安装成功测试

接上一步骤完成后，在 cmd 命令行中输入以下命令验证 MindSpore 是否安装成功。

```
python -c "import mindspore;mindspore.run_check()"
```

如果输出以下内容则表示 MindSpore 安装成功。

```
MindSpore version: 版本号
```

```
The result of multiplication calculation is correct, MindSpore has been installed successfully!
```

### 2.3.2 PyTorch

#### 步骤 1 创建虚拟环境（以下步骤可参考 1.3 中创建过程）

创建虚拟环境 PyTorch。

#### 步骤 2 激活虚拟环境

激活虚拟环境（PyTorch）。

#### 步骤 3 安装 PyTorch

接上一步骤完成后，在 Terminal 中输入

```
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu
```

显示 Successfully 则表示安装成功。

#### 步骤 4 安装成功测试

接上一步骤完成后，在 cmd 命令行中（PyTorch 虚拟环境中）输入以下命令验证是否安装成功。

```
python -c "import torch;print(torch.__version__)"
```

如果输出版本号则安装成功。

```
2.5.1+cpu
```

# 3 华为云使用指导手册

---

## 3.1 简介

华为云提供了非常丰富的云资源以及云服务，可以满足开发者的需要。本手册主要聚焦于如何申请华为云上的相应资源，主要包括了：账号申请与实名认证，AK 和 SK 的获取，华为云主要服务介绍。

通过学习本手册，你可以：

- 理解人工智能开发环境的具体内容。
- 申请人工智能开发环境。
- 基于搭建的环境进行功能测试。

## 3.2 华为云账号申请与实名认证

1. 用于在华为云官网开通相应服务，确保课程实验部分顺利进行。
2. 用于体验华为云提供的各种便捷的云服务，包括深度学习，机器学习，图像识别，语音识别，自然语言处理等。

### 3.2.1 注册步骤

#### 步骤 1 进入华为云官网并注册账号

进入华为云官网：<http://www.huaweicloud.com/>，并点击首页右上角“注册”。

---



图3-1 用户注册界面

点击“开通”。并点击“实名认证”进入实名认证界面。



图3-2 账号开通完成界面

## 步骤 2 实名认证

点击“个人认证”进入个人认证方式界面。

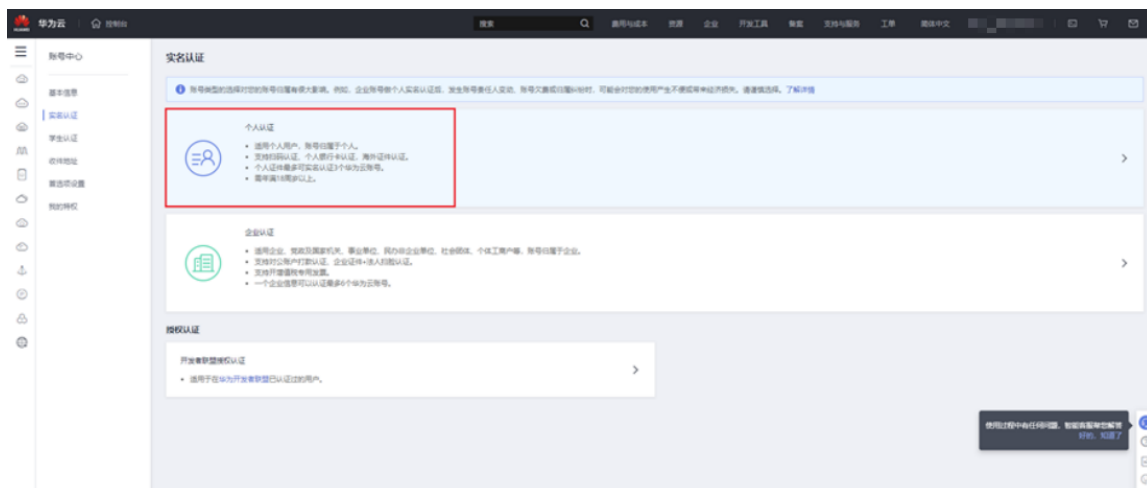


图3-3 实名认证界面

推荐使用“扫码认证”即时完成实名认证。



图3-4 个人实名认证界面

本例中以扫码认证方式为例。通过进行相关个人信息认证后即可完成实名认证。



图3-5 扫码进行实名认证

## 3.3 Access Key 和 Secret Access Key 的获取

### 3.3.1 简介

AK 和 SK 是华为云中各项服务所需要的“身份代码”，当第一次使用华为云的某些服务时，都会遇到需要输入 AK 和 SK 密码的情况。

### 3.3.2 生成 AK 和 SK

#### 步骤 1 进入我的凭证

登录好已注册的华为云账号后，可以在页面右上角个人用户名处自动下拉菜单找到“我的凭证”。

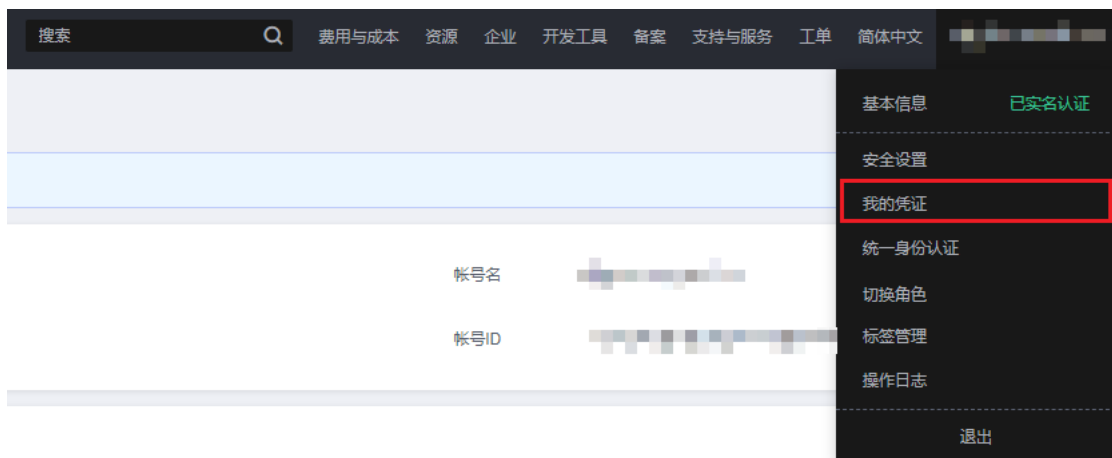


图3-6 “我的凭证” 界面入口

#### 步骤 2 新增访问秘钥

进入“我的凭证”界面后可以在左侧点击“访问密钥”处进入密钥管理界面。密钥管理界面处点击“新增访问密钥”即可添加一个访问密钥，每个访问密钥仅能下载一次。



图3-7 新增密钥

注意：新增访问密钥后会自动下载一个表格文件，该文件存有华为云的访问密钥，注意保存。后期如果不想使用 AK/SK 认证方式也可以通过此界面进行停用或删除。

## 3.4 华为云常用产品入口

### 3.4.1 ModelArts

ModelArts 是面向开发者的一站式 AI 开发平台，为机器学习与深度学习提供海量数据预处理及半自动化标注、大规模分布式 Training、自动化模型生成，及端-边-云模型按需部署能力，帮助用户快速创建和部署模型，管理全周期 AI workflow。

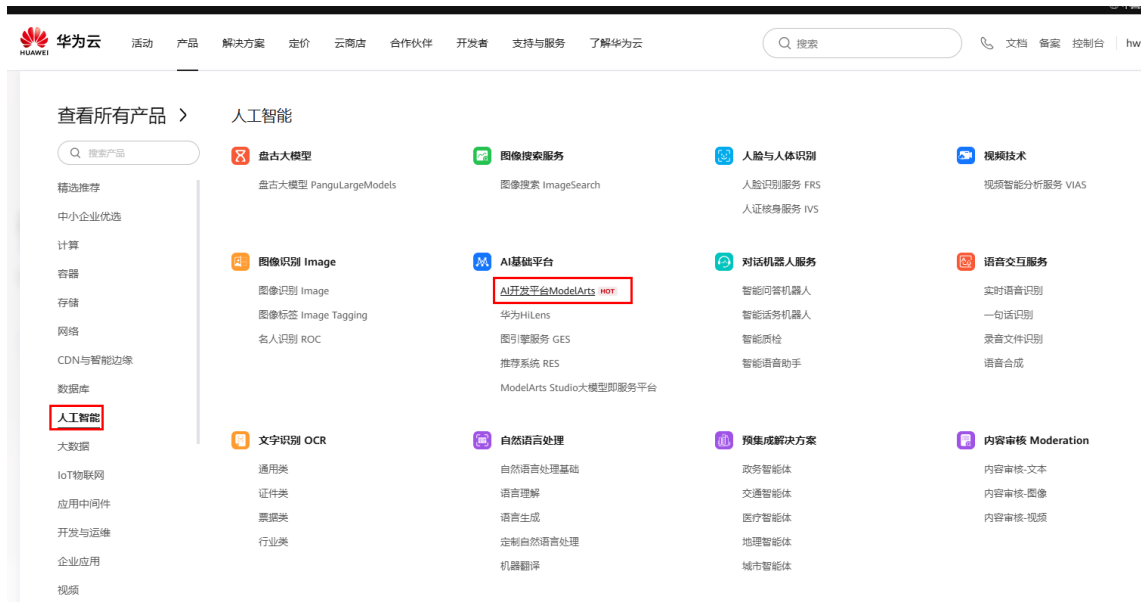


图3-8 ModelArts 入口

进入 ModelArts 后你可以在左侧“开发环境” - “Notebook” 选择创建基于 Ascend 910 或 GPU 的开发环境。

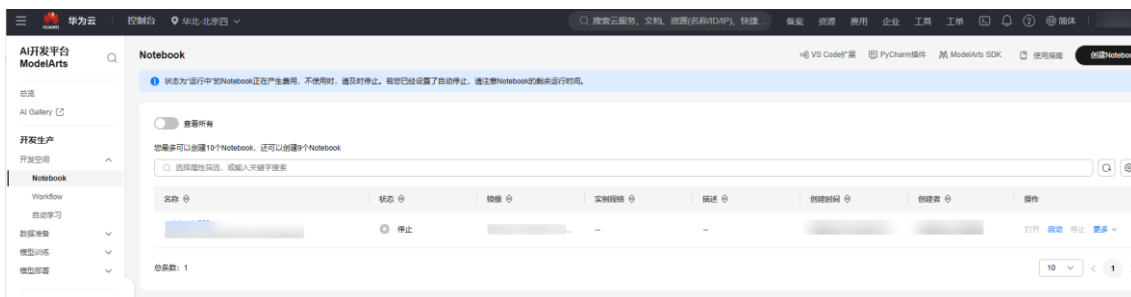


图3-9 创建开发环境

## 3.4.2 华为云资源购买

步骤 1 登录华为云，找到弹性云服务器产品；

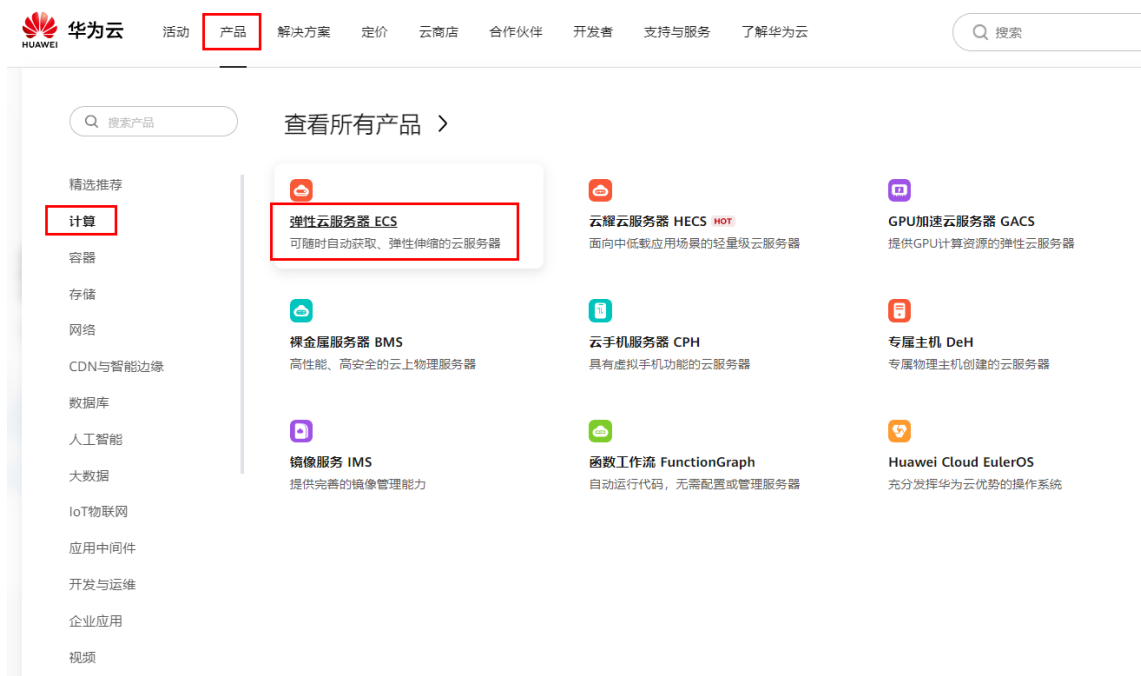


图3-10 弹性云服务器

步骤 2 点击购买，进入产品购买界面；



图3-11 购买界面

步骤 3 选择**按需计费**，x86 计算，内存优化型（实验对内存容量要求较高）；



图3-12 配置选择

步骤 4 选择内存为 32GiB 的实例，系统选择 EulerOS，然后点击下一步；

全选

m7

am7

m6

m3ne

收起帮助 ^

可应对大型内存数据集和高网络场景。适用于内存要求高，数据量大并且数据访问量，同时要求快速的数据交换和处理。

| 实例类型                                     | 规格名称         | vCPUs   | 内存     | CPU                   |
|------------------------------------------|--------------|---------|--------|-----------------------|
| <input type="radio"/> 内存优化型m7            | m7.large.8   | 2vCPUs  | 16GiB  | Intel Ice Lake 3.0GHz |
| <input checked="" type="radio"/> 内存优化型m7 | m7.xlarge.8  | 4vCPUs  | 32GiB  | Intel Ice Lake 3.0GHz |
| <input type="radio"/> 内存优化型m7            | m7.2xlarge.8 | 8vCPUs  | 64GiB  | Intel Ice Lake 3.0GHz |
| <input type="radio"/> 内存优化型m7            | m7.3xlarge.8 | 12vCPUs | 96GiB  | Intel Ice Lake 3.0GHz |
| <input type="radio"/> 内存优化型m7            | m7.4xlarge.8 | 16vCPUs | 128GiB | Intel Ice Lake 3.0GHz |
| <input type="radio"/> 内存优化型m7            | m7.6xlarge.8 | 24vCPUs | 192GiB | Intel Ice Lake 3.0GHz |
| <input type="radio"/> 内存优化型m7            | m7.8xlarge.8 | 32vCPUs | 256GiB | Intel Ice Lake 3.0GHz |

当前规格

内存优化型 | m7.xlarge.8 | 4vCPUs | 32GiB

您当前选择规格，仅支持使用SCSI磁盘模式挂载磁盘，不支持使用VBD磁盘模式挂载磁盘。磁盘标识为wwn号。

公共镜像

私有镜像

共享镜像

市场镜像

?

Huawei Cloud EulerOS

Huawei Cloud EulerOS 2.0 标准版 64位(40GiB)

C

☒ 免费试用一个月主机安全基础防护 ?
 ☐ 购买高阶防护 ?

图3-13 系统选择

步骤 5 网络设置界面选择现在购买弹性公网 IP，**按流量计费**，带宽选择 300Mbit/s，实验流量基本为入流量，不计费，其余默认即可，最后完成实例购买；

弹性公网IP

☒ 现在购买
 ☐ 使用已有
 ☐ 暂不购买 ?

线路

全动态BGP

静态BGP ?

不低于99.95%可用性保障

公网带宽

按带宽计费

按流量计费

加入共享带宽

流量较大或较稳定的场景

流量小或流量波动较大场景

多业务流量错峰分布场景 ?

指定带宽上限，按实际使用的出公网流量计费，与使用时间无关。

带宽大小

5

10

20

50

100

自定义

300

带宽范围: 1-300 Mbit/s

免费开启DDoS基础防护

释放行为

☐ 随实例释放
 

对于设置了随实例释放的弹性公网IP，将在删除云服务器同时执行删除。

1

+

台

配置费用 **¥1.4878/小时** + 弹性公网IP流量费用 **¥0.80/GB** ?

图3-14 EIP

步骤6 实例购买成功后可通过 CloudShell 登录，如果电脑安装了 Xshell 等软件也可以使用；



图3-15 登录

步骤7 输入购买实例时设置的账号密码；

区域：华北-北京四

刷新

云服务器：ecs-d992

端口：22

用户名：root

认证方式：密码认证

密码：

会话名称：

☒ 打开远程主机文件树

注意：

- 为确保连接的安全性，系统将对超过 **20分钟** 没有活跃的会话进行自动断开。
- 请确认安全组中来源为CloudShell代理IP的远程端口（SSH默认端口为22）已经允许。
- 当远程登录后操作卡顿时，建议查看一下机器的CPU、内存情况，请定义[云监控](#)在主机异常时通过短信等多种方式通知。
- 华为云CloudShell不会保存您的密码，请妥善保管以防丢失。

连接 取消

图3-16 登录信息