智能芯片原理与应用

实验指导书

实验一：深度学习编程框架使用

华为智能基座课程

广东工业大学

计算机学院

2025 年 3 月

1. 实验介绍

1.1 实验介绍

1.1.1 关于本实验

实验前理论课应该完成深度学习编程框架的讲解，学生已了解、基于 Tensorflow 或其他深度学习编程框架的基本原理。本实验介绍了 Window/Linux 系统通过 Miniconda 安装 Tensorflow、搭建 python 虚拟环境。

1.1.2 实验目标

1. 熟悉 Tensorflow 基本使用，包括 tensor、placeholder、variable、计算图搭建（动态图、静态图）的使用，了解 Tensorflow 命令式、声明式编程，并完成相关代码。
2. 熟悉使用 Keras 前端，以及未使用 Keras 前端，利用 Tensorflow 搭建 MLP 和 CNN 基本模型的方法，并完成相关代码。熟悉执行训练和测试过程。

1.1.3 软件版本介绍

类别	版本	获取方式	说明
Windows	**Windows11、10**	**/**	**需要是 64 位系统，CPU 支持 AVX2 指令集**
Ubuntu	**Ubuntu 22.04 / 18.04.4**	**https://ubuntu.com/download/desktop**	**需要是 64 位系统，CPU 支持 AVX2 指令集**
PyCharm	**2020.1.4 Community Edition**	**https://www.jetbrains.com/**	**/ vscode 二选一**
VScode	**1.77**	**https://code.visualstudio.com/download**	**/ pycharm 二选一**

Miniconda	**Python3.x**	**官方下载地址：** **https://docs.conda.io/en/latest/miniconda.html** **清华镜像源地址：** **https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/**	**Miniconda 可在线安装不同的 Python 版本，无需刻意下载特定版本，但需要下载 64 位，Python3.x 版本**

其中 Pycharm 和 VScode 任选其一。

1.2 软件介绍

1.2.1 Tensorflow 深度学习编程框架

Tensorflow 是是一个由 Google 开发的开源深度学习框架，其被设计成跨平台的。目前 Tensorflow 支持大多数深度学习模型的算子。它使用数据流图来进行数值计算。TensorFlow 的主要优势是它的灵活性和可扩展性，它可以在各种硬件和操作系统上运行，并且可以支持分布式计算。

TensorFlow 使用 Python 作为主要的编程语言，并提供了一个称为 Keras 的高级 API，可以简化模型的构建和训练过程。它还包括了许多预训练模型和数据集，使得使用者可以更加轻松地进行模型的快速实现和验证。TensorFlow 的计算图机制允许用户在执行计算之前定义计算图，从而使得计算可以高度优化并可以在不同的设备上运行。

此外，TensorFlow 提供了一种名为 TensorBoard 的工具，可以用于可视化模型的训练过程和结果，这对于深度学习模型的调试和优化非常有帮助。

1.2.2 Miniconda 介绍

Miniconda 是一个轻量级的 Anaconda 版本，它包括了 Python 解释器和 conda 包管理器，但只包含少量的基本包和库。它的主要目的是为了让用户能够更加灵活地配置自己的 Python 环境，同时避免 Anaconda 安装过多的软件包和库，造成磁盘空间和性能的浪费。

Conda 是一个跨平台的包管理器和环境管理器，它可以用于安装、升级和删除软件包，并且可以创建和管理多个独立的 Python 环境。Conda 可以在 Linux、Windows 和 MacOS 等操作系统上运行，并且可以管理不同版本的 Python 和各种依赖库。Conda 还可以用于创建和共享自定义软件包，方便其他用户进行安装和使用。

2 环境搭建

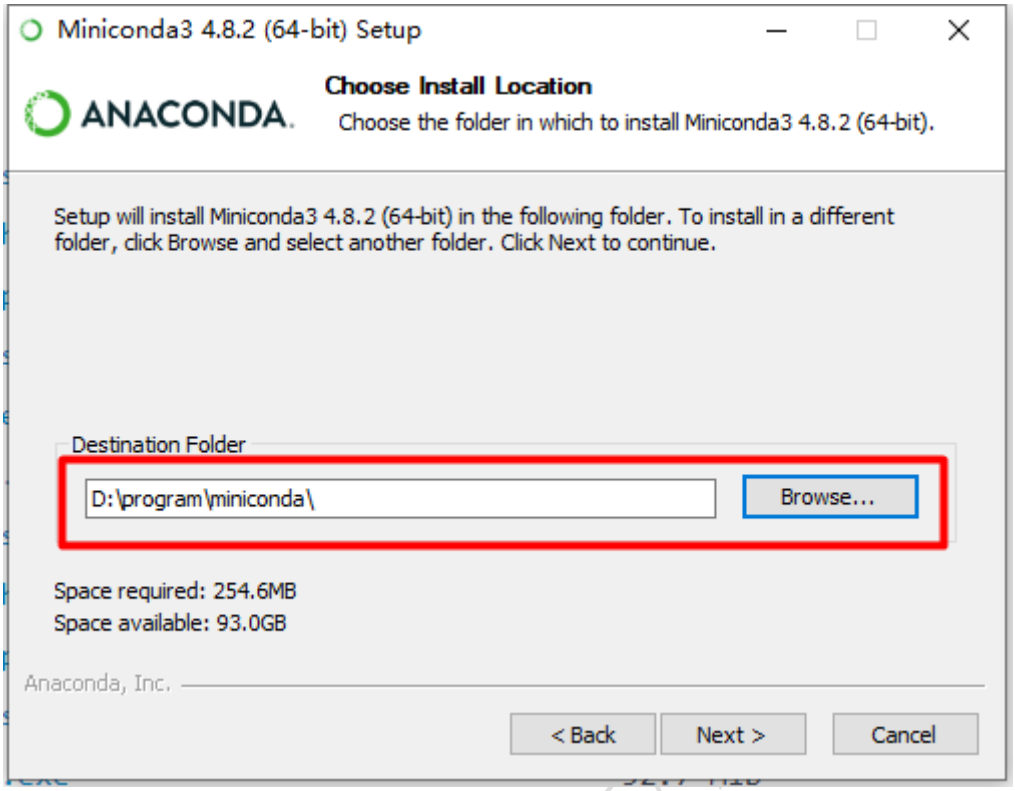
2.1 以本地 Windows 环境搭建为例（有 Ubuntu 系统使用经验的推荐在 Ubuntu 系统中安装）

2.1.1 Miniconda 安装

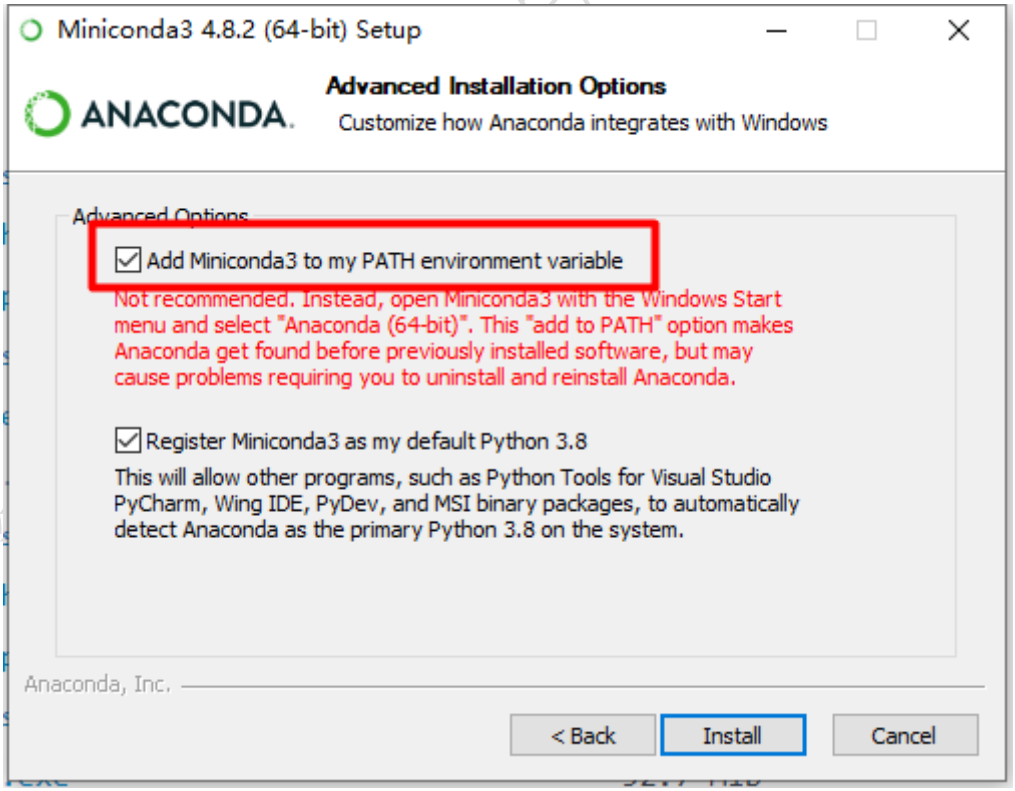
从 1.1.2 提供的链接下载 Miniconda 的 Windows 版本对应的 64 位安装包，由于官方源下载速度慢，实验所用安装包为清华源下载，带有 x86_64 的为 64 位安装包。

Miniconda3-py38_4.8.2-Windows-x86_64.exe	52.7 MiB	2020-03-12 00:09

双击安装包进行安装，点击 next，然后选择安装位置。



环境变量打勾，这样可以直接在命令行中启动 Miniconda。

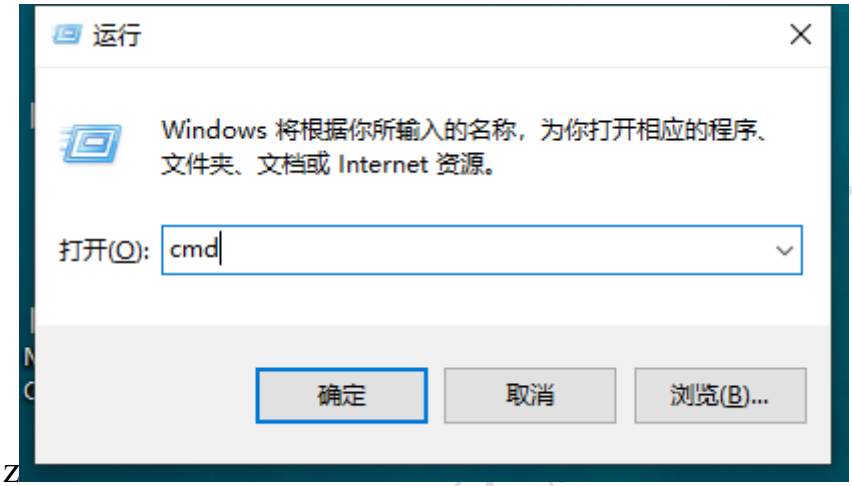


等待安装成功，然后点击 Finish。

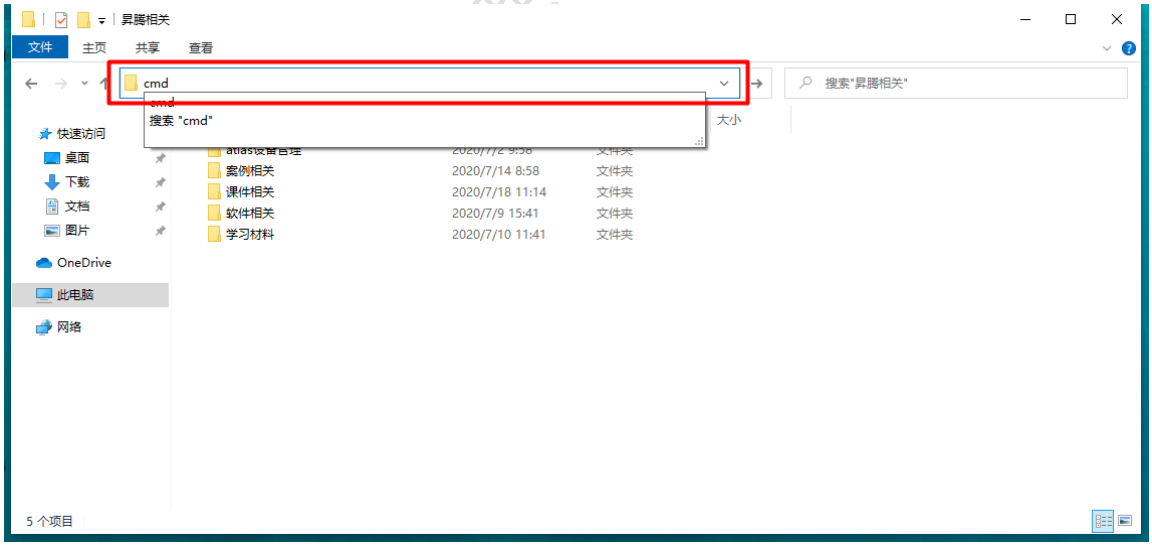
2.1.2 创建虚拟环境

在 Window 中有多种方式开启命令行窗口，这里介绍两种，按下 win+R 键，然后输入 cmd 点击确定，或者任意打开一个文件夹，在上方地址栏输入 cmd，然后按回车键。

运行打开命令行界面如下图所示：



地址栏打开命令行界面如下图所示：



打开命令行窗口之后，输入以下命令创建虚拟环境，Python 版本为 3.7.5，创建过程需要输入 y 确认。

```
>> conda create -n tf python==3.7.5
```

虚拟环境创建成功后输入对应名称即可进入对应虚拟环境

```
>> activate tf
```

2.1.3 Pip 换源

Python 可以通过 pip 和 conda 两种方式来安装包，但是两者所安装的包并不完全兼容，在实际使用过程中建议只选择一种方式来安装包，本实验使用的是 pip，但是由于 pip 的官方源在国外，直连速度较慢，因此需要换为国内的镜像源。

方法一：更换清华源

打开命令行，激活虚拟环境，利用如下命令进行设置。

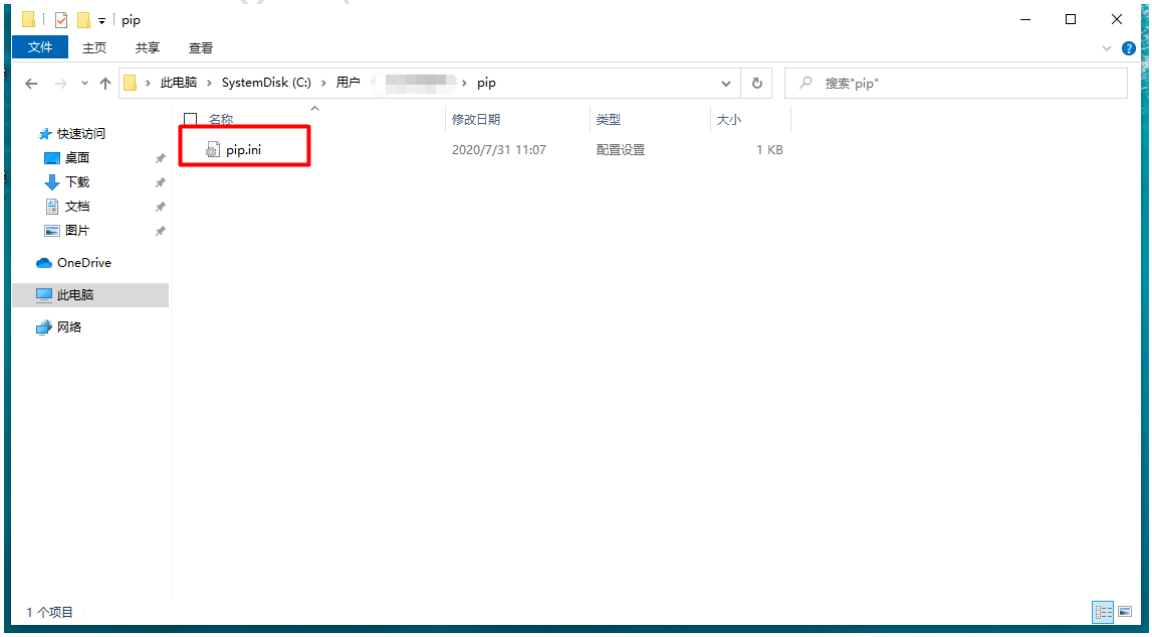
```
python -m pip install --upgrade pip
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

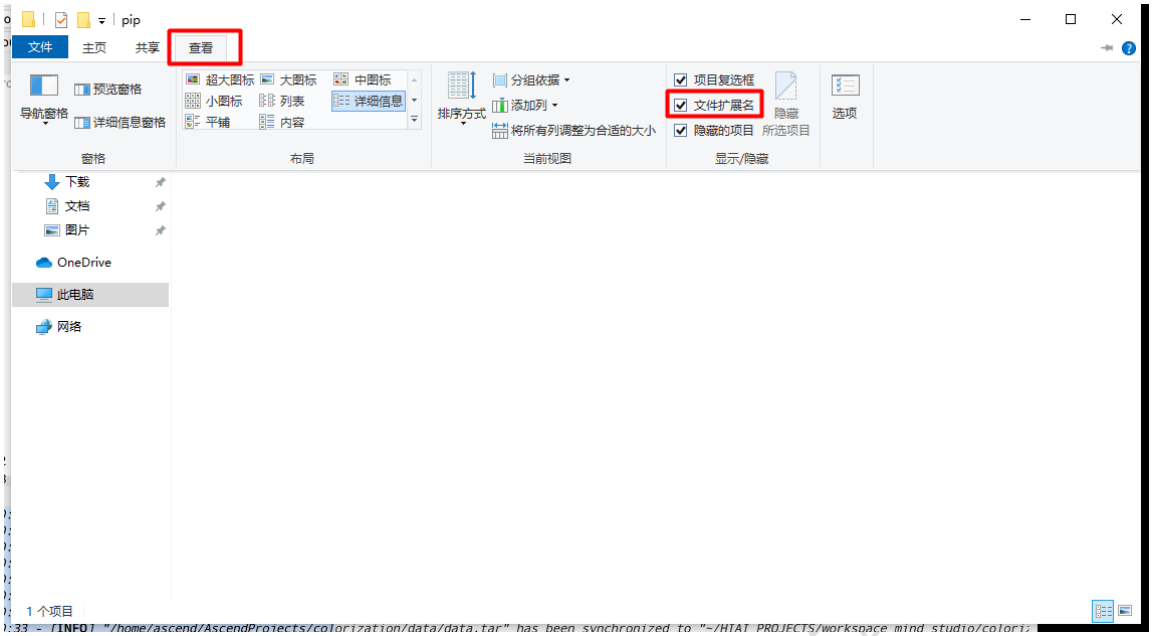
执行成功后显示如下结果：

Writing to C:\Users\<UserName>\AppData\Roaming\pip\pip.ini

方法二：手动设置清华源

打开此电脑在 C:\Users\<UserName>\AppData\Roaming\下，新建一个 pip 文件夹。新建一个文本文件，然后改名 pip.ini，该文件就是 pip 的配置文件，如果改完之后图标没变化，说明没有显示文件扩展名，点击查看，随后勾选显示文件扩展名。





打开 pip.ini 文件，将以下内容粘贴进去并保存。

[global]

index-url = https://pypi.tuna.tsinghua.edu.cn/simple

2.1.4 安装 Tensorflow

首先在终端中，输入以下命令激活 tensorflow 安装虚拟环境。

(Ubuntu/Linux)

>> conda activate tf

(Windows)

>> activate tf

成功激活环境后，利用 pip 命令安装最新版本的 tensorflow 库

>> pip install tensorflow

►此命令默认安装 tf 的最新版本，若要指定版本：

>> pip install tensorflow==2.12.0

安装完毕后，可在命令行验证安装是否成功

>> python

>>> import tensorflow as tf

>>> tf.__version__

安装正常一般会显示

```
>>> import tensorflow as tf
>>> tf.__version__
'2.12.0'
```

►注：若同学有 GPU 设备，可查阅 tensorflow-gpu 库的安装方式，但要注意 CUDA develop toolkit 库的兼容版本。

►注：TensorFlow 2.0 版本默认打开动态图模式，静态图模式的实验，需要安装 1.15 版本。（Tensorflow 1.0 的最后一个版本）

>> pip install tensorflow==1.15.0

3 Tensorflow 的基本使用

3.1 张量的定义与使用

首先激活 python 虚拟环境，确保成功激活，命令行窗口会显示‘(tf)’字样。之后，执行 python 命令，进入 python 命令行界面。

引入 tensorflow 库

```
>>>import tensorflow as tf
```

若成功引入，通常无任何显示。

#之后，创建一个张量

```
>>>tensor=tf.constant(value=1.0, shape=[2,2])
```

这里的 1.0 就是张量的值，[2,2]表示张量的维度。另外，根据张量的维度，可以将 TensorFlow 的张量分为：

- 标量（0 维张量）：使用 tf.constant()函数创建。
- 向量（1 维张量）：使用列表或数组来创建。
- 矩阵（2 维张量）：使用两个列表或嵌套列表来创建。
- 张量（多维张量）：依据需要使用列表来创建。

在实际应用中，对于 1 维、2 维、多维张量的使用，可以结合 numpy array 来创建。

1. 1 维张量

1 维张量通常称作向量，可以使用 numpy array 创建。例如，创建一个形状为(3,)的向量：

```
>>>import numpy as np
>>>vector = np.array([1, 2, 3])
>>>print(vector.shape)
```

输出为(3,)，即一个形状为(3,)的向量。

在 TensorFlow 中，可以使用 tf.constant()函数创建一个 1 维张量（手工指定）：

```
>>>tensor1D = tf.constant([1, 2, 3])
>>>print(tensor1D.shape)
```

输出为(3,)，即一个形状为(3,)的张量。

或者将 vector 直接赋值给 tensor1D

```
>>>tensor1D=tf.constant(vector)
>>>print(tensor1D.shape)
```

2. 2 维张量

2 维张量通常称作矩阵，可以使用 numpy array 创建。例如，创建一个形状为(2, 3)的矩阵：

```
>>>matrix = np.array([[1, 2, 3], [4, 5, 6]])
>>>print(matrix.shape)
```

输出为(2, 3)，即一个形状为(2, 3)的矩阵。类似地，在 TensorFlow 中，可以使用 tf.constant()函数创建一个 2 维张量：

```
>>>tensor2D = tf.constant([[1, 2, 3], [4, 5, 6]])
>>>print(tensor2D.shape)
```

输出为(2, 3)，即一个形状为(2, 3)的张量。

同理，亦可通过直接赋值给 tensor 的方式生成

```
>>>tensor2D = tf.constant(matrix)
>>>print(tensor2D.shape)
```

3. 多维张量

多维张量是指 3 维或 3 维以上的张量。在 numpy 中，可以使用 array()函数创建多维数组。例如，创建一个形状为(2, 3, 4)的 3 维张量：

```
>>>tensor3D = np.array([[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11,
12]], [[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]]])
>>>print(tensor3D.shape)
```

输出为(2, 3, 4)，即一个形状为(2, 3, 4)的 3 维张量。在 TensorFlow 中：

```
>>>tensor3D = tf.constant([[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10,
11, 12]], [[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23,
24]]])
>>>print(tensor3D.shape)
```

输出为(2, 3, 4)，即一个形状为(2, 3, 4)的 3 维张量。

张量(Tensor)的概念对于理解 Tensorflow 中的数据处理和构成是非常重要的。通常，对于计算机视觉任务来说，接触最多的是图像类型数据，通常会出现四维张量，即为(N，H，W，C)维度的 tensor。利用上述张量。

查阅 Tensorflow 的 Document，测试使用下在 tensorflow 中，tensor 的不同维度的 tensor 的加、减、乘、除操作，例如矩阵乘法、向量内积、矩阵乘以向量、向量乘以矩阵、标量乘以矩阵等，亦可尝试使用 tf.negative()函数用于取反、tf.abs()函数用于取绝对值、tf.sqrt()函数用于求平方根、tf.exp()函数用于求指数、tf.log()函数用于求对数等。

3.2 Tensorflow 的静态图 vs.动态图

A. 静态图（需要安装 Tensorflow==1.15.0）

在 tf_basic.py 文件中，定义了一个简单的 a*b 计算图，尝试通过改变 constant 数据，执行 Tensor 的相关代数运算，并利用 Session 的 run 方法得到执行结果。（注意下列代码，可直接利用 python tf_basic.py 运行，也可以在 python 命令行环境，一行一行输入执行。建议在 python 命令行中一行一行输入，可以和动态图命令式编程形成对比。）

```
# -*- coding: utf-8 -*-
import tensorflow as tf
# 手动关闭 Eager 动态图模式:
tf.compat.v1.disable_eager_execution()
##################################################
# 尝试改变这里的 a,b,c Tensor 的大小和类型
a = tf.constant([[1.0, 2.0]])
b = tf.constant([[3.0], [4.0]])
c = tf.constant([[5.0], [6.0]])
# 尝试改变这里的计算图定义方法，可利用上面的代数运算
y1 = tf.matmul(a,b)   # 这里是 a*b 内积
y2 = tf.matmul(a,c)   # 这里是 a*c 内积

# 通过 with 方法执行 run 函数,可以定义不同的计算图
# 运行到不同的节点
with tf.Session() as sess:
    print(sess.run([y1,y2]))
```

由此可知，不经过 session 的 run 方法，无法得到运行结果。

B. 动态图（需要安装 Tensorflow==2.12.0）

Tensorflow 2.0 后已经默认打开动态图机制，可通过以下方式进行测试，体验命令式编程所写即所得的结果。即不用等到使用 run 方

法才能得到结果。

```
>>>import tensorflow as tf
>>>print(tf.__version__)
>>>tf.compat.v1.enable_eager_execution()
>>>print(tf.executing_eagerly())
>>>A = [[2.]]
>>>B = [[3.]]
>>>m = tf.matmul(A, B)
>>>print(m)
>>>n = tf.add(A, B)
>>>print(n)
```

3.3 占位符 Placeholder 使用（Tensorflow==1.15）

在静态图模式下，通常通过定义 Placeholder 占位符的方式导入输入变量的值。Placeholder 在申请时仅声明输入数据的大小及数据类型，并没有指定数据，在执行 feed_dict 时，真正赋值。尝试测试下面的例子，熟悉 Placeholder 的使用。

```
>>>import tensorflow as tf
>>>X = tf.placeholder(tf.float32, name="X")
>>>Y = tf.placeholder(tf.float32, name="Y")
>>>z = tf.add(X, Y)
>>>with tf.Session() as session:
>>>    result = session.run(z, feed_dict={X: 1.0, Y: 1.0})
>>>    print(result)
```

3.4 Variable 使用

Variable 可以理解为神经网络中的可微分参数。不同于 constant、placeholder 类型，需要通过调用全局初始化函数，进行参数初始化，并通过定义计算图、session run 函数才能运行得到结果。尝试测试下面的例子，熟悉 Variable 的使用。

```
# -*- coding: utf-8 -*-
import tensorflow as tf

# 定义输入占位符
X = tf.placeholder(tf.float32, shape=(None, 10), name="X")
```

```
# 定义权重,类比一层全连接层,单层感知器
W = tf.Variable(tf.random_normal([10, 1]), name="W")

# 定义偏置项
b = tf.Variable(tf.zeros([1]), name="b")

# 定义 Op 计算节点,仅为线性单元,未给激活函数
z = tf.matmul(X, W) + b

# 创建会话
with tf.Session() as sess:
    # 对 W 和 b 进行全局变量初始化
    sess.run(tf.global_variables_initializer())
    # 模拟一批输入数据
    x_input = [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]]
    # 计算 z 节点
    result = sess.run(z, feed_dict={X: x_input})
    print(result)
```

尝试变换占位符、W 权重的数量、b 的数量，测试其他代码的运行结果。

4 使用 Tensorflow 编程实现基本的多层感知器 MLP 实现 MNIST 分类

4.1 MINIST 数据集

MNIST（Modified National Institute of Standards and Technology）是一个计算机视觉常用的手写数字数据集。它包含了 60,000 张 28×28 像素的训练图像和 10,000 张测试图像，这些图像涵盖了 0-9 的数字，以及相应的标签。

MNIST 数据集最早由美国国家标准与技术研究院（NIST）创建，用于识别邮政编码中的邮政编号。但随着计算机视觉和机器学习的发展，MNIST 数据集已经成为了一个重要的基准测试集，广泛应用于图像识别、机器学习算法的训练和评估等领域。在 MNIST 数据集上

训练分类模型，旨在让模型能够正确地识别数字图像。因为 MNIST 数据集过于简单，所以，通常我们会使用它作为机器学习领域的入门级别数据集。

4.2 Keras 简单模式

由于 Tensorflow2.0 后默认集成了 Keras 库，在实际应用中，可通过调用 Keras 作为前端，调用 Tensorflow 的后端 Op 的类，实现模型结构的快速实现与部署。尝试完成下列代码，可命名为 tf_keras_mlp.py 文件，执行 MLP 模型的快速训练。

```
# 引入依赖库
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist

# 准备 MNIST 数据集，由于在 tensorflow 的 tutorial 中已经集成了 MNIST 数据
库，
# 因此可以直接调用加载，其中 x 表示图像数据，y 表示标签数据
# train 表示训练数据，test 表示测试数据，切记训练中不要加入测试数据
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 将训练的图像 28x28 像素的数据，拉成 784 的向量，并归一化到 0~1 之间，即除以
255
x_train = x_train.reshape(x_train.shape[0],
784).astype('float32') / 255
x_test = x_test.reshape(x_test.shape[0], 784).astype('float32') /
255

# 调用 keras 的 util 库将类别标签转换为 one-hot 编码（独热编码）
# 其中 10 表示 0~9，10 个手写字符的类。
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

# 创建
model = keras.Sequential([
#######################################################
# 在这里写代码
# 实现一个输入 784 -> 512 -> 256 -> 10 的 MLP 网络
# 可以发散思维试试其他网络结构
# 需要查询：keras.layers.Dense()的使用，及激活函数设定方法
```

```
##########################################################
])

# 指定训练用 loss function，优化器，以及评估的指标类型
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# 利用 keras model 的 fit 函数训练整个模型
model.fit(x_train, y_train, epochs=5, batch_size=128,
validation_data=(x_test, y_test))
```

4.3 Tenorflow 的一般模式

由于 Tensorflow2.0 后默认开启动态图模式，可通过关闭 eager 模式切换回原来的版本进行编码，本部分意在熟悉没有 Keras 库时，直接调用 Tensorflow 库，如果进行模型构建和训练。

阅读并完成下列代码，给出执行结果：

```
# 首先，我们需要导入必要的库：
import numpy as np
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import os
# 关闭 Eager 模式：
tf.compat.v1.disable_eager_execution()

####################准备数据#########################
# 然后，我们读入 MNIST 数据集：
mnist = input_data.read_data_sets('./data/', one_hot=True)
# read_data_sets() 函数时已经默认进行了归一化和 one-hot 编码操作

train_x, train_y = mnist.train.images, mnist.train.labels
test_x, test_y = mnist.test.images, mnist.test.labels
# 定义模型的输入和输出：

input_data = tf.placeholder(tf.float32, [None, 784])
output_data = tf.placeholder(tf.float32, [None, 10])

# 其中，input_data 表示模型的输入，是一个形状为(None, 784)的张量，None 表示可以接受任意长度的输入，784 表示每个样本有 784 个特征；output_data 表示模型的输出，是一个形状为(None, 10)的张量，10 表示共有 10 个分类。
```

```
####################准备模型参数#########################
# 然后，我们定义模型的参数：
w1 = tf.Variable(tf.truncated_normal([784, ?], stddev=0.1))
b1 = tf.Variable(tf.zeros([?]))
w2 = tf.Variable(tf.truncated_normal([?, ?], stddev=0.1))
b2 = tf.Variable(tf.zeros([?]))
w3 = tf.Variable(tf.truncated_normal([?, ?], stddev=0.1))
b3 = tf.Variable(tf.zeros([?]))


####################准备前向计算图#########################
layer1 = tf.nn.relu(tf.matmul(input_data, w1) + b1)
# 定义 Layer2，和输出层并完成下列代码
# layer2 = ?
# output = ?

# 其中，layer1 表示第一层的输出，使用了 ReLU 激活函数；
# layer2 表示第二层的输出，使用了 ReLU 激活函数；
# output 表示输出层的输出，没有使用激活函数。

####################定义损失函数和优化器####################

loss =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=
output_data, logits=output))
# 其中，tf.nn.softmax_cross_entropy_with_logits_v2()函数用于计算交叉熵
损失函数
# tf.train.AdamOptimizer()函数用于创建 Adam 优化器，初始 lr 设置为 0.001
learning_rate= 0.001
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
# 定义训练 op
train_op = optimizer.minimize(loss)

####################定义 accuaracy op####################
correct_pred = tf.equal(tf.argmax(output, 1),
tf.argmax(output_data, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

####################设置超参数和模型存储路径####################
batch_size = 128
num_epochs = 5
num_batches = int(train_x.shape[0] / batch_size)
ckpt_dir = './model'
```

```
if not os.path.exists(ckpt_dir):
    os.makedirs(ckpt_dir)

####################最后，我们定义会话并进行训练####################
with tf.Session() as sess:
    # 全局初始化参数
    sess.run(tf.global_variables_initializer())

    for epoch in range(num_epochs):
        for batch in range(num_batches):
            # 取一个 batch 的数据完成代码
            batch_x, batch_y = ?
            # 运行优化器并计算损失和准确率，完成下列代码，要 run 到哪个 op?
            _, batch_loss, batch_acc = sess.run([?, ?, ?],
feed_dict={input_data: ?, output_data: ?})
        # 每 2 个 epoch 保存一次模型
        if (epoch + 1) % 2 == 0:
            saver = tf.train.Saver()
            saver.save(sess, "{}/checkpoint-{}" .format(ckpt_dir,
str(epoch+1)))
        # 计算测试集上的准确率和损失
        test_loss, test_acc = sess.run([loss, accuracy],
feed_dict={input_data: test_x, output_data: test_y})
        # tf.get_default_graph().finalize()

            # 输出信息
        # print("Epoch:", epoch+1, " Loss:", batch_loss, "
Accuracy:", batch_acc, "Test Loss:", test_loss, " Test
Accuracy:", test_acc)
        print("Epoch: {}/{}, Train Loss: {:.4f}, Train Acc. :
{:.4f}, Test loss: {:.4f}, Test Acc. : {:.4f}".format(
            epoch+1, num_epochs, batch_loss, batch_acc, test_loss,
test_acc
        ))
```

5 使用 Tensorflow 编程实现基本的 CNN 模型实现 MNIST 分类

通过查询：keras.layers.Conv2D()，keras.layers.MaxPooling2D(),

keras.layers.Flatten(), keras.layers.Dense()的使用，及激活函数设定方法，阅读并完成下列代码，记录实验过程。

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Reshape the input data to be 4D [batch, height, width,
channels]
x_train = x_train.reshape(x_train.shape[0], 28, 28,
1).astype('float32') / 255
x_test = x_test.reshape(x_test.shape[0], 28, 28,
1).astype('float32') / 255

# Convert the labels to one-hot encoded vectors
y_train = keras.utils.to_categorical(y_train, 10)
y_test  = keras.utils.to_categorical(y_test, 10)

# Create the CNN model
#########################################################
# 完成 channels 数为 64-pooling-128-pooling-flatten-fc128-fc10
# 其中 kernel size 3,3, pool size 2,2
#########################################################
model = keras.Sequential([
    keras.layers.Conv2D(?, kernel_size=(?, ?), activation='relu',
input_shape=(28, 28, 1)),
    keras.layers.MaxPooling2D(pool_size=(?, ?)), # 28 x 28 -> 14 x
14, 32
    keras.layers.Conv2D(?, kernel_size=(?, ?), activation='relu',
input_shape=(14, 14, ?)),
    keras.layers.MaxPooling2D(pool_size=(?, ?)), # 14 x 14 -> 7 x
7, 64
    keras.layers.Flatten(),
    keras.layers.Dense(?, activation='relu'),
    keras.layers.Dense(?, activation='softmax')
])
# 输出网络结构信息
model.summary()
# Compile the model with categorical crossentropy loss and Adam
optimizer
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the model on the training data
model.fit(x_train, y_train, epochs=5, batch_size=128,
validation_data=(x_test, y_test))
```

思考问题（注意：思考题需要回答，并填写到实验报告中）：

1. 比较 Tensorflow 的静、动态图相应的代码写法，结合课程上课内容，总结并论述静态和动态图的优点和缺点。
2. 根据代码实现过程，结合 Keras 和直接调用 Tensorflow 库的代码书写难度，总结并简述使用 Tensorflow 训练模型的一般步骤（分几步），可结合课程中的训练和测试过程总结。

（**Optional**）拓展训练（**可选作，做完可填写到实验报告中**）：

1. 结合 MLP 和 CNN 的使用过程，更换数据集，例如 CIFAR10、Fashion-MNIST 等数据集，查阅相关资料进行模型训练，并给出实验代码和训练结果。

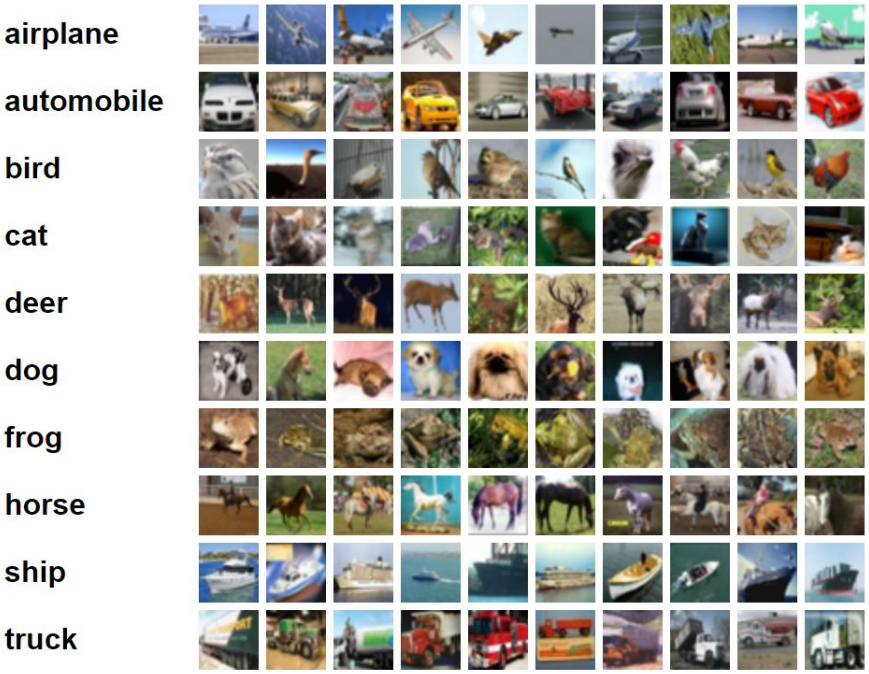


图 1 CIFAR-10 数据集[1]

[1] https://www.cs.toronto.edu/~kriz/cifar.html

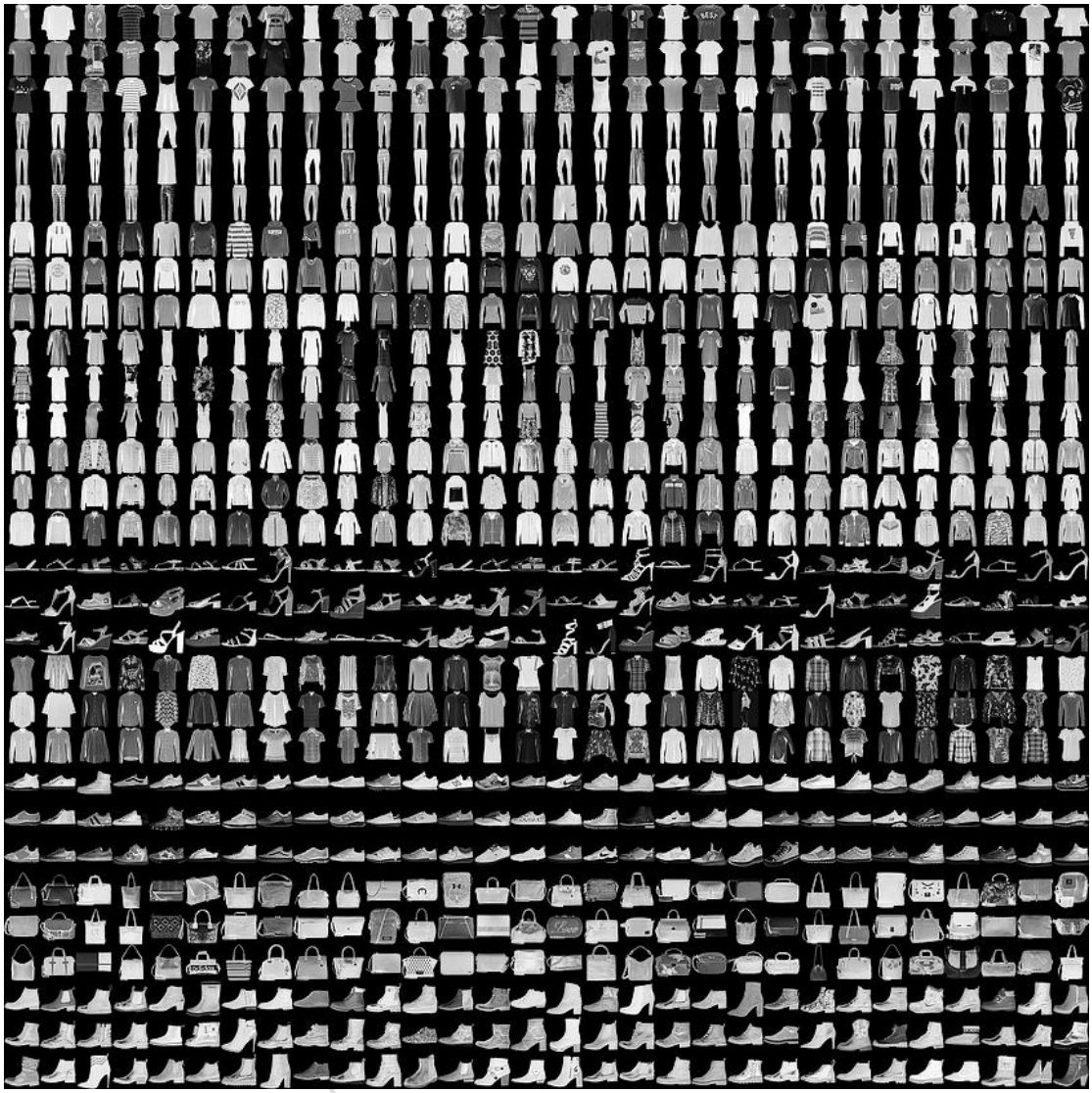


图 2 Fashion-MNIST 数据集[2]

[2] https://github.com/zalandoresearch/fashion-mnist