深度学习实验手册

一、基础理论

实验一：自定义感知机

```
# 00_percetron.py
# 实现感知机

```

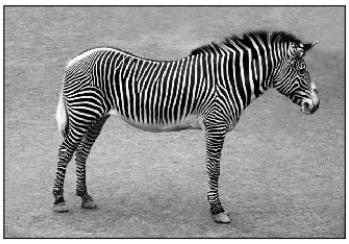
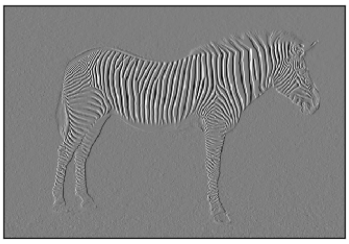
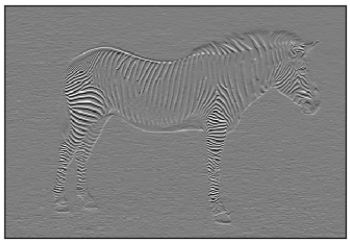
```python
# 实现逻辑和
def AND(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.7
    tmp = x1 * w1 + x2 * w2
    if tmp <= theta:
        return 0
    else:
        return 1

print(AND(1, 1))
print(AND(1, 0))


# 实现逻辑或
def OR(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.2
    tmp = x1 * w1 + x2 * w2
    if tmp <= theta:
        return 0
    else:
        return 1

print(OR(0, 1))
print(OR(0, 0))

# 实现异或
def XOR(x1, x2):
    s1 = not AND(x1, x2) # 与非门
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y

print(XOR(1, 0))
print(XOR(0, 1))
print(XOR(1, 1))
print(XOR(0, 0))
```

实验二：验证图像卷积运算效果

```python
from scipy import signal
from scipy import misc
import matplotlib.pyplot as plt
import numpy as np
import scipy.ndimage as sn

im = misc.imread("data/zebra.png", flatten=True)
# face = sn.imread("data/zebra.png", flatten=True)
```

```
flt = np.array([[-1, 0, 1],
                [-2, 0, 2],
                [-1, 0, 1]])

flt2 = np.array([[1, 2, 1],
                 [0, 0, 0],
                 [-1, -2, -1]])

# 把图像的face数组和设计好的卷积和作二维卷积运算,设计边界处理方式为symm
conv_img1 = signal.convolve2d(im, flt,
                              boundary='symm',
                              mode='same').astype("int32")

conv_img2 = signal.convolve2d(im, flt2,
                              boundary='symm',
                              mode='same').astype("int32")

plt.figure("Conv2D")
plt.subplot(131)
plt.imshow(im, cmap='gray')  # 显示原始的图
plt.xticks([])
plt.yticks([])

plt.subplot(132)
plt.xticks([])
plt.yticks([])
plt.imshow(conv_img1, cmap='gray')  # 卷积后的图

plt.subplot(133)
plt.xticks([])
plt.yticks([])
plt.imshow(conv_img2, cmap='gray')  # 卷积后的图

plt.show()
```

执行结果：

二、Tensorflow

实验一：查看Tensorflow版本

```
from __future__ import absolute_import, division, print_function,
unicode_literals

# 导入TensorFlow和tf.keras
import tensorflow as tf
from tensorflow import keras

# 导入辅助库
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

实验二：Helloworld程序

```
# tf的helloworld程序
import tensorflow as tf

hello = tf.constant('Hello, world!')  # 定义一个常量
sess = tf.Session()  # 创建一个session
print(sess.run(hello))  # 计算
sess.close()
```

实验三：张量相加

```
# 常量加法运算示例
import tensorflow as tf
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # 调整警告级别

a = tf.constant(5.0)  # 定义常量a
b = tf.constant(1.0)  # 定义常量a
c = tf.add(a, b)
print("c:", c)

graph = tf.get_default_graph()  # 获取缺省图
print(graph)

with tf.Session() as sess:
    print(sess.run(c))  # 执行计算
```

实验四：查看图对象

```python
# 常量加法运算示例
import tensorflow as tf
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # 调整警告级别

a = tf.constant(5.0)  # 定义常量a
b = tf.constant(1.0)  # 定义常量a
c = tf.add(a, b)
print("c:", c)

graph = tf.get_default_graph()  # 获取缺省图
print(graph)

with tf.Session() as sess:
    print(sess.run(c))  # 执行计算
    print(a.graph)  # 通过tensor获取graph对象
    print(c.graph)  # 通过op获取graph对象
    print(sess.graph)  # 通过session获取graph对象
```

实验五：指定执行某个图

```python
# 创建多个图，指定图运行
import tensorflow as tf
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # 调整警告级别

a = tf.constant(5.0)  # 定义常量a
b = tf.constant(1.0)  # 定义常量a
c = tf.add(a, b)

graph = tf.get_default_graph()  # 获取缺省图
print(graph)

graph2 = tf.Graph()
print(graph2)
with graph2.as_default(): #设置为默认图
    d = tf.constant(11.0)

with tf.Session(graph=graph2) as sess:
    print(sess.run(d))  # 执行计算
    # print(sess.run(c))  # 报错
```

实验六：查看张量属性

```python
# 创建多个图，指定图运行

```

```python
import tensorflow as tf
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # 调整警告级别

# a = tf.constant(5.0)  # 定义常量a
# a = tf.constant([1,2,3])
a = tf.constant([[1,2,3],[4,5,6]])

with tf.Session() as sess:
    print(sess.run(a))  # 执行计算
    print("name:", a.name)
    print("dtype:", a.dtype)
    print("shape:", a.shape)
    print("op:", a.op)
    print("graph:", a.graph)
```

实验七：生成张量

```python
# 创建张量操作
import tensorflow as tf

# 生成值全为0的张量
tensor_zeros = tf.zeros(shape=[2, 3], dtype="float32")
# 生成值全为1的张量
tensor_ones = tf.ones(shape=[2, 3], dtype="float32")
# 创建正态分布张量
tensor_nd = tf.random_normal(shape=[10],
                             mean=1.7,
                             stddev=0.2,
                             dtype="float32")
# 生成和输入张量形状一样的张量，值全为1
tensor_zeros_like = tf.zeros_like(tensor_ones)

with tf.Session() as sess:
    print(tensor_zeros.eval())  # eval表示在session中计算该张量
    print(tensor_ones.eval())
    print(tensor_nd.eval())
    print(tensor_zeros_like.eval())
```

实验八：张量类型转换

```python
# 张量类型转换
import tensorflow as tf

tensor_ones = tf.ones(shape=[2, 3], dtype="int32")
tensor_float = tf.constant([1.1, 2.2, 3.3])

with tf.Session() as sess:
    print(tf.cast(tensor_ones, tf.float32).eval())
    # print(tf.cast(tensor_float, tf.string).eval()) #不支持浮点数到字符串直接转换
```

实验九：占位符使用

```python
# 占位符示例
import tensorflow as tf

# 不确定数据，先使用占位符占个位置
plhd = tf.placeholder(tf.float32, [2, 3])  # 2行3列的tensor
plhd2 = tf.placeholder(tf.float32, [None, 3])  # N行3列的tensor

with tf.Session() as sess:
    d = [[1, 2, 3],
         [4, 5, 6]]
    print(sess.run(plhd, feed_dict={plhd: d}))
    print("shape:", plhd.shape)
    print("name:", plhd.name)
    print("graph:", plhd.graph)
    print("op:", plhd.op)
    print(sess.run(plhd2, feed_dict={plhd2: d}))
```

实验十：改变张量形状

```python
# 改变张量形状示例(重点)
import tensorflow as tf

pld = tf.placeholder(tf.float32, [None, 3])
print(pld)

pld.set_shape([4, 3])
print(pld)
# pld.set_shape([3, 3]) #报错，静态形状一旦固定就不能再设置静态形状

# 动态形状可以创建一个新的张量，改变时候一定要注意元素的数量要匹配
new_pld = tf.reshape(pld, [3, 4])
print(new_pld)
# new_pld = tf.reshape(pld, [2, 4]) # 报错，元素的数量不匹配
```

```
15
16 with tf.Session() as sess:
17     pass
```

实验十一：数学计算

```
1  # 数学计算示例
2  import tensorflow as tf
3
4  x = tf.constant([[1, 2], [3, 4]], dtype=tf.float32)
5  y = tf.constant([[4, 3], [3, 2]], dtype=tf.float32)
6
7  x_add_y = tf.add(x, y)  # 张量相加
8  x_mul_y = tf.matmul(x, y)  # 张量相乘
9  log_x = tf.log(x)  # log(x)
10
11 # reduce_sum: 此函数计算一个张量的各个维度上元素的总和
12 x_sum_1 = tf.reduce_sum(x, axis=[1]) #0-列方向 1-行方向
13
14 # segment_sum: 沿张量的片段计算总和
15 # 函数返回的是一个Tensor,它与data有相同的类型,与data具有相同的形状
16 # 但大小为 k(段的数目)的维度0除外
17 data = tf.constant([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=tf.float32)
18 segment_ids = tf.constant([0, 0, 0, 1, 1, 2, 2, 2, 2, 2],
   dtype=tf.int32)
19 x_seg_sum = tf.segment_sum(data, segment_ids)  # [6, 9, 40]
20
21 with tf.Session() as sess:
22     print(x_add_y.eval())
23     print(x_mul_y.eval())
24     print(x_mul_y.eval())
25     print(log_x.eval())
26     print(x_sum_1.eval())
27     print(x_seg_sum.eval())
```

实验十二：变量使用示例

```
1  # 变量OP示例
2  import tensorflow as tf
3  # 创建普通张量
4  a = tf.constant([1, 2, 3, 4, 5])
5  # 创建变量
6  var = tf.Variable(tf.random_normal([2, 3], mean=0.0, stddev=1.0),
7                    name="variable")
8
```

```
# 变量必须显式初始化，这里定义的是初始化操作，并没有运行
init_op = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init_op)
    print(sess.run([a, var]))
```

实验十三：可视化

第一步：编写代码

```
# 变量OP示例
import tensorflow as tf

''' 变量OP
1. 变量OP能够持久化保存，普通张量则不可
2. 当定义一个变量OP时，在会话中进行初始化
3. name参数：在tensorboard使用的时候显示名字，可以让相同的OP进行区分
'''

# 创建普通张量
a = tf.constant([1, 2, 3, 4, 5])
# 创建变量
var = tf.Variable(tf.random_normal([2, 3], mean=0.0, stddev=1.0),
                  name="variable")

b = tf.constant(3.0, name="a")
c = tf.constant(4.0, name="b")
d = tf.add(b, c, name="add")

# 变量必须显式初始化，这里定义的是初始化操作，并没有运行
init_op = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init_op)
    # 将程序图结构写入事件文件
    fw = tf.summary.FileWriter("../summary/", graph=sess.graph)
    print(sess.run([a, var]))
```

第二步：启动tensorborad

```
tensorboard  --logdir="PycharmProjects/tensorflow_study/summary/"
```

第三步：访问tensorborad主页

```
http://127.0.0.1:6006
```

实验十四：实现线性回归

```python
# 线性回归示例
import tensorflow as tf

# 第一步：创建数据
x = tf.random_normal([100, 1], mean=1.75, stddev=0.5, name="x_data")
y_true = tf.matmul(x, [[2.0]]) + 5.0  # 矩阵相乘必须是二维的

# 第二步：建立线性回归模型
# 建立模型时，随机建立权重、偏置 y = wx + b
# 权重需要不断更新，所以必须是变量类型. trainable指定该变量是否能随梯度下降一起变化
weight = tf.Variable(tf.random_normal([1, 1], name="w"),
                     trainable=True)  # 训练过程中值是否允许变化
bias = tf.Variable(0.0, name="b", trainable=True)  # 偏置
y_predict = tf.matmul(x, weight) + bias  # 计算 wx + b

# # 第三步：求损失函数，误差(均方差)
loss = tf.reduce_mean(tf.square(y_true - y_predict))

# # 第四步：使用梯度下降法优化损失
# 学习率是比价敏感的参数，过小会导致收敛慢，过大可能导致梯度爆炸
train_op = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

#### 收集损失值
tf.summary.scalar("losses", loss)
merged = tf.summary.merge_all() #将所有的摘要信息保存到磁盘

init_op = tf.global_variables_initializer()
with tf.Session() as sess:  # 通过Session运行op
    sess.run(init_op)
    # 打印初始权重、偏移值
    print("weight:", weight.eval(), " bias:", bias.eval())

    #### 指定事件文件
    fw = tf.summary.FileWriter("../summary/", graph=sess.graph)

    for i in range(500):  # 循环执行训练
        sess.run(train_op)  # 执行训练
        summary = sess.run(merged) #### 运行合并摘要op
        fw.add_summary(summary, i) #### 写入文件
        print(i, ":", i, "weight:", weight.eval(), " bias:",
bias.eval())
```

实验十五：模型保存与加载

```python
# 模型保存示例
import tensorflow as tf
import os

# 第一步：创建数据
x = tf.random_normal([100, 1], mean=1.75, stddev=0.5, name="x_data")
y_true = tf.matmul(x, [[2.0]]) + 5.0  # 矩阵相乘必须是二维的

# 第二步：建立线性回归模型
# 建立模型时，随机建立权重、偏置 y = wx + b
# 权重需要不断更新，所以必须是变量类型. trainable指定该变量是否能随梯度下降一起变化
weight = tf.Variable(tf.random_normal([1, 1], name="w"),
                     trainable=True)  # 训练过程中值是否允许变化
bias = tf.Variable(0.0, name="b", trainable=True)  # 偏置
y_predict = tf.matmul(x, weight) + bias  # 计算 wx + b

# # 第三步：求损失函数，误差(均方差)
loss = tf.reduce_mean(tf.square(y_true - y_predict))

# # 第四步：使用梯度下降法优化损失
# 学习率是比价敏感的参数，过小会导致收敛慢，过大可能导致梯度爆炸
train_op = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

# 收集损失值
tf.summary.scalar("losses", loss)
merged = tf.summary.merge_all() #将所有的摘要信息保存到磁盘

init_op = tf.global_variables_initializer()

saver = tf.train.Saver() #实例化Saver
with tf.Session() as sess:  # 通过Session运行op
    sess.run(init_op)
    print("weight:", weight.eval(), " bias:", bias.eval())     # 打印初始权重、偏移值
    fw = tf.summary.FileWriter("../summary/", graph=sess.graph) # 指定事件文件
    # 训练之前，加载之前训练的模型，覆盖之前的参数
    if os.path.exists("../model/linear_model/checkpoint"):
        saver.restore(sess, "../model/linear_model/")

    for i in range(500):  # 循环执行训练
        sess.run(train_op)  # 执行训练
        summary = sess.run(merged) # 运行合并后的tensor
        fw.add_summary(summary, i)
        print(i, ":", i, "weight:", weight.eval(), " bias:", bias.eval())

```

```
    saver.save(sess, "../model/linear_model/")
```

实验十五：CSV文件读取

```
# csv文件读取示例
import tensorflow as tf
import os
def csv_read(filelist):
    # 2. 构建文件队列
    file_queue = tf.train.string_input_producer(filelist)
    # 3. 构建csv reader，读取队列内容（一行）
    reader = tf.TextLineReader()
    k, v = reader.read(file_queue)
    # 4. 对每行内容进行解码
    ## record_defaults：指定每一个样本每一列的类型，指定默认值
    records = [["None"], ["None"]]
    example, label = tf.decode_csv(v, record_defaults=records)  # 每行两个值
    # 5. 批处理
    # batch_size: 跟队列大小无关，只决定本批次取多少数据
    example_bat, label_bat = tf.train.batch([example, label],
                                            batch_size=9,
                                            num_threads=1,
                                            capacity=9)
    return example_bat, label_bat


if __name__ == "__main__":
    # 1. 找到文件，构造一个列表
    dir_name = "./test_data/"
    file_names = os.listdir(dir_name)
    file_list = []
    for f in file_names:
        file_list.append(os.path.join(dir_name, f))  # 拼接目录和文件名

    example, label = csv_read(file_list)
    # 开启session运行结果
    with tf.Session() as sess:
        coord = tf.train.Coordinator() # 定义线程协调器
        # 开启读取文件线程
        # 调用 tf.train.start_queue_runners 之后，才会真正把tensor推入内存序列中
        # 供计算单元调用，否则会由于内存序列为空，数据流图会处于一直等待状态
        # 返回一组线程
        threads = tf.train.start_queue_runners(sess, coord=coord)
        print(sess.run([example, label])) # 打印读取的内容
        # 回收线程
```

```
        coord.request_stop()
        coord.join(threads)
```

实验十六：图片文件读取

```
# 图片文件读取示例
import tensorflow as tf
import os

def img_read(filelist):
    # 1. 构建文件队列
    file_queue = tf.train.string_input_producer(filelist)
    # 2. 构建reader读取文件内容，默认读取一张图片
    reader = tf.WholeFileReader()
    k, v = reader.read(file_queue)

    # 3. 对每行内容进行解码
    img = tf.image.decode_jpeg(v)  # 每行两个值

    # 4. 批处理，图片需要处理成统一大小
    img_resized = tf.image.resize(img, [200, 200])  # 200*200
    img_resized.set_shape([200, 200, 3])  # 固定样本形状，批处理时对数据形状有要求
    img_bat = tf.train.batch([img_resized],
                             batch_size=10,
                             num_threads=1)
    return img_bat


if __name__ == "__main__":
    # 1. 找到文件，构造一个列表
    dir_name = "../data/test_img/"
    file_names = os.listdir(dir_name)
    file_list = []
    for f in file_names:
        file_list.append(os.path.join(dir_name, f))  # 拼接目录和文件名
    imgs = img_read(file_list)
    # 开启session运行结果
    with tf.Session() as sess:
        coord = tf.train.Coordinator()  # 定义线程协调器
        # 开启读取文件线程
        # 调用 tf.train.start_queue_runners 之后，才会真正把tensor推入内存序列中
        # 供计算单元调用，否则会由于内存序列为空，数据流图会处于一直等待状态
        # 返回一组线程
        threads = tf.train.start_queue_runners(sess, coord=coord)
        # print(sess.run([imgs]))  # 打印读取的内容
```

```
        imgs = imgs.eval()

        # 回收线程
        coord.request_stop()
        coord.join(threads)

## 显示图片
print(imgs.shape)
import matplotlib.pyplot as plt

plt.figure("Img Show", facecolor="lightgray")

for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(imgs[i].astype("int32"))

plt.tight_layout()
plt.show()
```

实验十七：实现手写体识别

```
# 手写体识别
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import pylab

# 读入数据集(如果没有则在线下载)，并转换成独热编码
# 如果不能下载，则到http://yann.lecun.com/exdb/mnist/进行手工下载，下载后拷贝到当前MNIST_data目录下
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

x = tf.placeholder(tf.float32, [None, 784])  # 占位符，输入
y = tf.placeholder(tf.float32, [None, 10])  # 占位符，输出

W = tf.Variable(tf.random_normal([784, 10]))  # 权重
b = tf.Variable(tf.zeros([10]))  # 偏置值

# 构建模型
pred_y = tf.nn.softmax(tf.matmul(x, W) + b)  # softmax分类
print("pred_y.shape:", pred_y.shape)
# 损失函数
cross_entropy = -tf.reduce_sum(y * tf.log(pred_y),
                               reduction_indices=1)  # 求交叉熵
cost = tf.reduce_mean(cross_entropy)  # 求损失函数平均值

```

```
# 参数设置
lr = 0.01
# 梯度下降优化器
optimizer = tf.train.GradientDescentOptimizer(lr).minimize(cost)

training_epochs = 200
batch_size = 100
saver = tf.train.Saver()
model_path = "../model/mnist/mnist_model.ckpt"  # 模型路径

# 启动session
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    # 循环开始训练
    for epoch in range(training_epochs):
        avg_cost = 0.0
        total_batch = int(mnist.train.num_examples / batch_size)  # 计算总批次

        # 遍历全数据集
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)  # 读取一个批次样本
            params = {x: batch_xs, y: batch_ys}  # 训练参数

            o, c = sess.run([optimizer, cost], feed_dict=params)  # 执行训练

            avg_cost += (c / total_batch)  # 求平均损失值

        print("epoch: %d, cost=%.9f" % (epoch + 1, avg_cost))

    print("Finished!")

    # 模型评估
    correct_pred = tf.equal(tf.argmax(pred_y, 1), tf.argmax(y, 1))
    # 计算准确率
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
    print("accuracy:", accuracy.eval({x: mnist.test.images,
                                      y: mnist.test.labels}))
    # 将模型保存到文件
    save_path = saver.save(sess, model_path)
    print("Model saved:", save_path)

# 测试模型
with tf.Session() as sess:
```

```
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, model_path)  # 加载模型

    batch_xs, batch_ys = mnist.test.next_batch(2)  # 读取2个测试样本
    output = tf.argmax(pred_y, 1)  # 预测结果值

    output_val, predv = sess.run([output, pred_y],  # 操作
                                 feed_dict={x: batch_xs})  # 参数

    print("预测结论:\n", output_val, "\n")
    print("实际结果:\n", batch_ys, "\n")
    print("预测概率:\n", predv, "\n")

    # 显示图片
    im = batch_xs[0]  # 第1个测试样本数据
    im = im.reshape(-1, 28)
    pylab.imshow(im)
    pylab.show()

    im = batch_xs[1]  # 第2个测试样本数据
    im = im.reshape(-1, 28)
    pylab.imshow(im)
    pylab.show()
```

实验十八：利用CNN实现服饰识别

```
# 在fashion_mnist数据集实现服饰识别
import tensorflow as tf
from tensorflow.contrib.learn.python.learn.datasets.mnist import
read_data_sets

class FashionMnist():
    out_featrues1 = 12  # 第一个组卷积池化层输出特征数量(等于第一个卷积层卷积核
数量)
    out_featrues2 = 24  # 第二个组卷积池化层输出特征数量(等于第二个卷积层卷积核
数量)
    con_neurons = 512 # 全连接层神经元数量

    def __init__(self, path):
        """
        构造方法
        :param path:指定数据集路径
        :return:
        """
        self.sess = tf.Session() # 会话
        self.data = read_data_sets(path, one_hot=True) # 读取样本文件对象

```

```python
    def init_weight_variable(self, shape):
        """
        初始化权重方法
        :param shape:指定初始化张量的形状
        :return:经过初始化后的张量
        """
        inital = tf.truncated_normal(shape, stddev=0.1) # 截尾正态分布
        return tf.Variable(inital)

    def init_bias_variable(self, shape):
        """
        初始化偏置
        :param shape:指定初始化张量的形状
        :return:经过初始化后的张量
        """
        inital = tf.constant(1.0, shape=shape)
        return tf.Variable(inital)

    def conv2d(self, x, w):
        """
        二维卷积方法
        :param x:原始数据
        :param w:卷积核
        :return:返回卷积后的结果
        """
        # input : 输入数据[batch, in_height, in_width, in_channels]
        # filter : 卷积窗口[filter_height, filter_width, in_channels, out_channels]
        # strides: 卷积核每次移动步数，对应着输入的维度方向
        # padding='SAME' :  输入和输出的张量形状相同
        return tf.nn.conv2d(x,  # 原始数据
                            w, # 卷积核
                            strides=[1, 1, 1, 1], # 各个维度上的步长值
                            padding="SAME") # 输入和输出矩阵大小相同

    def max_pool_2x2(self, x):
        """
        池化函数
        :param x:原始数据
        :return:池化后的数据
        """
        return tf.nn.max_pool(x,# 原始数据
                              ksize=[1, 2, 2, 1], # 池化区域大小
                              strides=[1, 2, 2, 1], # 各个维度上的步长值
                              padding="SAME")

```

```
64      def create_conv_pool_layer(self, input, input_features,
out_features):
65          """
66          卷积、激活、池化层
67          :param input:原始数据
68          :param input_features:输入特征数量
69          :param out_features:输出特征数量
70          :return:卷积、激活、池化层后的数据
71          """
72          filter = self.init_weight_variable([5, 5, input_features,
out_features])#卷积核
73          b_conv = self.init_bias_variable([out_features]) # 偏置，数量和
卷积输出大小一致
74
75          h_conv = tf.nn.relu(self.conv2d(input, filter) + b_conv)#卷积，
结果做relu激活
76          h_pool = self.max_pool_2x2(h_conv) #对激活操作输出做max池化
77          return h_pool
78
79      def create_fc_layer(self, h_pool_flat, input_featrues,
con_neurons):
80          """
81          创建全连接层
82          :param h_pool_flat:输入数据，经过拉伸后的一维张量
83          :param input_featrues:输入特征大小
84          :param con_neurons:神经元数量
85          :return:全连接
86          """
87          w_fc = self.init_weight_variable([input_featrues,
con_neurons])#输出数量等于神经元数量
88          b_fc = self.init_bias_variable([con_neurons]) #偏置数量等于输出数
量
89          h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, w_fc) + b_fc) #计算
wx+b并且做relu激活
90          return h_fc1
91
92      def build(self):
93          """
94          组建CNN
95          :return:
96          """
97          # 输入数据，N个28*28经过拉伸后的张量
98          self.x = tf.placeholder(tf.float32, shape=[None, 784])
99          x_image = tf.reshape(self.x, [-1, 28, 28, 1]) # 28*28单通道
100         self.y_ = tf.placeholder(tf.float32, shape=[None, 10]) # 标签，
对应10个类别
101         # 第一组卷积池化层
```

```
        h_pool1 = self.create_conv_pool_layer(x_image, 1, self.out_featrues1)
        # 第二组卷积池化层
        h_pool2 = self.create_conv_pool_layer(h_pool1, # 上一层输出作为输入
                                              self.out_featrues1, # 上一层输出特征数量作为输入特征数量
                                              self.out_featrues2)# 第二层输出特征数量
        # 全连接层
        h_pool2_flat_features = 7 * 7 * self.out_featrues2 # 计算特征点数量
        h_pool2_flat = tf.reshape(h_pool2, [-1, h_pool2_flat_features])#拉升成一维张量
        h_fc = self.create_fc_layer(h_pool2_flat, # 输入
                                    h_pool2_flat_features, # 输入特征数量
                                    self.con_neurons) # 输出特征数量
        # dropout层（通过随机丢弃一部分神经元的更新，防止过拟合）
        self.keep_prob = tf.placeholder("float") # 丢弃率
        h_fc1_drop = tf.nn.dropout(h_fc, self.keep_prob)
        # 输出层
        w_fc = self.init_weight_variable([self.con_neurons, 10])#512行10列，产生10个输出
        b_fc = self.init_bias_variable([10]) # 10个偏置
        y_conv = tf.matmul(h_fc1_drop, w_fc) + b_fc # 计算wx+b，预测结果

        # 评价
        correct_prediction = tf.equal(tf.argmax(y_conv, 1),#取出预测概率中最大的值的索引
                                      tf.argmax(self.y_, 1))#取出真实概率中最大的值的索引
        # 将上一步得到的bool类型数组转换为浮点型，并求准确率
        self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

        # 损失函数
        loss_func = tf.nn.softmax_cross_entropy_with_logits(labels=self.y_,#真实值

 logits=y_conv)#预测值
        cross_entropy = tf.reduce_mean(loss_func)
        # 优化器
        optimizer = tf.train.AdamOptimizer(0.001)
        self.train_step = optimizer.minimize(cross_entropy)

    def train(self):
        self.sess.run(tf.global_variables_initializer()) #初始化
```

```
        merged = tf.summary.merge_all() #摘要合并

        batch_size = 100
        print("beging training...")

        for i in range(10): # 迭代训练
            total_batch = int(self.data.train.num_examples / batch_size)#计算批次数量

            for j in range(total_batch):
                batch = self.data.train.next_batch(batch_size)#获取一个批次样本
                params = {self.x: batch[0], self.y_:batch[1],#输入、标签
                          self.keep_prob: 0.5} #丢弃率

                t, acc = self.sess.run([self.train_step, self.accuracy],#要执行的op
                                       params) # 喂入参数
                if j % 100 == 0:
                    print("epoch: %d, pass: %d, acc: %f"  % (i, j, acc))
    # 评价
    def eval(self, x, y, keep_prob):
        params = {self.x: x, self.y_: y, self.keep_prob: 1.0}
        test_acc = self.sess.run(self.accuracy, params)
        print('Test accuracy %f' % test_acc)
        return test_acc

    # 关闭会话
    def close(self):
        self.sess.close()

if __name__ == "__main__":
    mnist = FashionMnist('FASHION_MNIST_data/')
    mnist.build()
    mnist.train()

    print('\n----- Test -----')
    xs, ys = mnist.data.test.next_batch(100)
    mnist.eval(xs, ys, 0.5)
    mnist.close()
```

三、PaddlePaddle

实验一：Helloworld

```python
# helloworld示例
import paddle.fluid as fluid

# 创建两个类型为int64, 形状为1*1张量
x = fluid.layers.fill_constant(shape=[1], dtype="int64", value=5)
y = fluid.layers.fill_constant(shape=[1], dtype="int64", value=1)
z = x + y # z只是一个对象,没有run,所以没有值

# 创建执行器
place = fluid.CPUPlace() # 指定在CPU上执行
exe = fluid.Executor(place) # 创建执行器
result = exe.run(fluid.default_main_program(),
                 fetch_list=[z]) #返回哪个结果
print(result) # result为多维张量
```

实验二：张量相加

```python
import paddle.fluid as fluid
import numpy

# 创建x, y两个1行1列，类型为float32的变量(张量)
x = fluid.layers.data(name="x", shape=[1], dtype="float32")
y = fluid.layers.data(name="y", shape=[1], dtype="float32")

result = fluid.layers.elementwise_add(x, y)  # 两个张量按元素相加

place = fluid.CPUPlace()  # 指定在CPU上执行
exe = fluid.Executor(place)  # 创建执行器
exe.run(fluid.default_startup_program())  # 初始化网络

# a = numpy.array([int(input("x:"))]) #输入x, 并转换为数组
# b = numpy.array([int(input("y:"))]) #输入y, 并转换为数组

# a = numpy.array([1, 2, 3])  # 输入x, 并转换为数组
# b = numpy.array([4, 5, 6])  # 输入y, 并转换为数组

a = numpy.array([[1, 1, 1], [2, 2, 2]])  # 输入x, 并转换为数组
b = numpy.array([[3, 3, 3], [4, 4, 4]])  # 输入y, 并转换为数组

params = {"x": a, "y": b}
outs = exe.run(fluid.default_main_program(),  # 默认程序上执行
               feed=params,  # 喂入参数
               fetch_list=[result])  # 获取结果
for i in outs:
    print(i)
```

实验三：简单线性回归

```
# 简单线性回归
import paddle
import paddle.fluid as fluid
import numpy as np
import matplotlib.pyplot as plt

train_data = np.array([[0.5], [0.6], [0.8], [1.1], [1.4]]).astype('float32')
y_true = np.array([[5.0], [5.5], [6.0], [6.8], [6.8]]).astype('float32')

# 定义数据数据类型
x = fluid.layers.data(name="x", shape=[1], dtype="float32")
y = fluid.layers.data(name="y", shape=[1], dtype="float32")
# 通过全连接网络进行预测
y_preict = fluid.layers.fc(input=x, size=1, act=None)
# 添加损失函数
cost = fluid.layers.square_error_cost(input=y_preict, label=y)
avg_cost = fluid.layers.mean(cost)  # 求均方差
# 定义优化方法
optimizer = fluid.optimizer.SGD(learning_rate=0.01)
optimizer.minimize(avg_cost)  # 指定最小化均方差值

# 搭建网络
place = fluid.CPUPlace()  # 指定在CPU执行
exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())  # 初始化系统参数

# 开始训练，迭代100次
costs = []
iters = []
values = []
params = {"x": train_data, "y": y_true}
for i in range(200):
    outs = exe.run(feed=params, fetch_list=[y_preict.name, avg_cost.name])
    iters.append(i)  # 迭代次数
    costs.append(outs[1][0])  # 损失值

# 线性模型可视化
tmp = np.random.rand(10, 1)
tmp = tmp * 2
tmp.sort(axis=0)
x_test = np.array(tmp).astype("float32")
params = {"x": x_test, "y":x_test}
```

```
y_out = exe.run(feed=params, fetch_list=[y_preict.name])
y_test = y_out[0]

# 损失函数可视化
plt.figure("Trainging")
plt.title("Training Cost", fontsize=24)
plt.xlabel("Iter", fontsize=14)
plt.ylabel("Cost", fontsize=14)
plt.plot(iters, costs, color="red", label="Training Cost")
plt.grid()

# 线性模型可视化
plt.figure("Inference")
plt.title("Linear Regression", fontsize=24)
plt.plot(x_test, y_test, color="red", label="inference")
plt.scatter(train_data, y_true)

plt.legend()
plt.grid()
plt.show()
```

实验四：波士顿房价预测

```
# 多元回归示例：波士顿房价预测
''' 数据集介绍:
 1) 共506行，每行14列，前13列描述房屋特征信息，最后一列为价格中位数
 2) 考虑了犯罪率（CRIM）          宅用地占比（ZN）
    非商业用地所占尺寸（INDUS）   查尔斯河虚拟变量（CHAS）
    环保指数（NOX）              每栋住宅的房间数（RM）
    1940年以前建成的自建单位比例（AGE）    距离5个波士顿就业中心的加权距离（DIS）
    距离高速公路便利指数（RAD）            每一万元不动产税率（TAX）
    教师学生比（PTRATIO）                  黑人比例（B）
    房东属于中低收入比例（LSTAT）
'''
import paddle
import paddle.fluid as fluid
import numpy as np
import math
import os
import matplotlib.pyplot as plt

# step1: 数据准备
# paddle提供了uci_housing训练集、测试集，直接读取并返回数据
BUF_SIZE = 500
BATCH_SIZE = 20

# 训练数据集读取器
```

```
random_reader =
paddle.reader.shuffle(paddle.dataset.uci_housing.train(),
                                          buf_size=BUF_SIZE)  # 创建随机读取
器
train_reader = paddle.batch(random_reader, batch_size=BATCH_SIZE)  #
训练数据读取器

# 测试数据集读取器
random_tester =
paddle.reader.shuffle(paddle.dataset.uci_housing.test(),
                                          buf_size=BUF_SIZE)
test_reader = paddle.batch(random_tester, batch_size=BATCH_SIZE)

# 打印数据
# train_data = paddle.dataset.uci_housing.train()  # test()
# for sample_data in train_data():
#     print(sample_data)

# step2: 配置网络
# 定义输入、输出，类型均为张量
x = fluid.layers.data(name="x", shape=[13], dtype="float32")
y = fluid.layers.data(name="y", shape=[1], dtype="float32")
# 定义个简单的线性网络，连接输出层、输出层
y_predict = fluid.layers.fc(input=x,  # 输入数据
                            size=1,  # 输出值个数
                            act=None)  # 激活函数
# 定义损失函数，并将损失函数指定给优化器
cost = fluid.layers.square_error_cost(input=y_predict,  # 预测值，张量
                                      label=y)  # 期望值，张量
avg_cost = fluid.layers.mean(cost)  # 求损失值平均数
optimizer = fluid.optimizer.SGDOptimizer(learning_rate=0.001)  # 使用随
机梯度下降优化器
opts = optimizer.minimize(avg_cost)  # 优化器最小化损失值

# 创建新的program用于测试计算
test_program = fluid.default_main_program().clone(for_test=True)

# step3: 模型训练、模型评估
place = fluid.CPUPlace()
exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())

feeder = fluid.DataFeeder(place=place, feed_list=[x, y])

iter = 0
iters = []
train_costs = []
```

```
67
68  EPOCH_NUM = 120
69  model_save_dir = "model/fit_a_line.model"  # 模型保存路径
70  for pass_id in range(EPOCH_NUM):
71      train_cost = 0
72      i = 0
73      for data in train_reader():
74          i += 1
75          train_cost = exe.run(program=fluid.default_main_program(),
76                               feed=feeder.feed(data),
77                               fetch_list=[avg_cost])
78          if i % 20 == 0:  # 每20笔打印一次损失函数值
79              print("PassID: %d, Cost: %0.5f" % (pass_id, train_cost[0]
    [0]))
80          iter = iter + BATCH_SIZE  # 加上每批次笔数
81          iters.append(iter)  # 记录笔数
82          train_costs.append(train_cost[0][0])  # 记录损失值
83
84  # 保存模型
85  if not os.path.exists(model_save_dir):  # 如果存储模型的目录不存在，则创建
86      os.makedirs(model_save_dir)
87  fluid.io.save_inference_model(model_save_dir,  # 保存模型的路径
88                                ["x"],  # 预测需要喂入的数据
89                                [y_predict],  # 保存预测结果的变量
90                                exe)  # 模型
91  # 训练过程可视化
92  plt.figure("Training Cost", facecolor="gray")
93  plt.title("Training Cost", fontsize=24)
94  plt.xlabel("iter", fontsize=14)
95  plt.ylabel("cost", fontsize=14)
96  plt.plot(iters, train_costs, color="red", label="Training Cost")
97  plt.grid()
98  # plt.show()
99  plt.savefig("train.png")
100
101 # step4: 模型预测
102 infer_exe = fluid.Executor(place)  # 创建用于预测的Executor
103 infer_scope = fluid.core.Scope()  # 修改全局/默认作用域，运行时中的所有变量都
    将分配给新的scope
104 infer_result = [] #预测值列表
105 ground_truths = [] #真实值列表
106
107 # with fluid.scope_guard(infer_scope):
108 # 加载模型，返回三个值
109 # program: 预测程序(包含了数据、计算规则)
110 # feed_target_names: 需要喂入的变量
111 # fetch_targets: 预测结果保存的变量
```

```
[infer_program, feed_target_names, fetch_targets] = \
    fluid.io.load_inference_model(model_save_dir,  # 模型保存路径
                                  infer_exe)  # 要执行模型的Executor
# 获取测试数据
infer_reader = paddle.batch(paddle.dataset.uci_housing.test(),
                            batch_size=200)  # 测试数据读取器
test_data = next(infer_reader())  # 获取一条数据
test_x = np.array([data[0] for data in test_data]).astype("float32")
test_y = np.array([data[1] for data in test_data]).astype("float32")

x_name = feed_target_names[0]  # 模型中保存的输入参数名称
results = infer_exe.run(infer_program,  # 预测program
                        feed={x_name: np.array(test_x)},  # 喂入预测的值
                        fetch_list=fetch_targets)  # 预测结果
# 预测值
for idx, val in enumerate(results[0]):
    print("%d: %.2f" % (idx, val))
    infer_result.append(val)

# 真实值
for idx, val in enumerate(test_y):
    print("%d: %.2f" % (idx, val))
    ground_truths.append(val)

# 可视化
plt.figure('scatter', facecolor='lightgray')
plt.title("TestFigure", fontsize=24)
x = np.arange(1, 30)
y = x
plt.plot(x, y)
plt.xlabel("ground truth", fontsize=14)
plt.ylabel("infer result", fontsize=14)
plt.scatter(ground_truths, infer_result, color="green", label="Test")
plt.grid()
plt.savefig("predict.png")
plt.show()
```

实验五：增量模型训练

1）模型训练与保存

```
# 线性回归增量训练、模型保存、固化
import paddle
import paddle.fluid as fluid
import numpy as np
import matplotlib.pyplot as plt
import os
```

```
7
8  train_data = np.array([[0.5], [0.6], [0.8], [1.1],
   [1.4]]).astype('float32')
9  y_true = np.array([[5.0], [5.5], [6.0], [6.8],
   [6.8]]).astype('float32')
10
11 # 定义数据数据类型
12 x = fluid.layers.data(name="x", shape=[1], dtype="float32")
13 y = fluid.layers.data(name="y", shape=[1], dtype="float32")
14 # 通过全连接网络进行预测
15 y_predict = fluid.layers.fc(input=x, size=1, act=None)
16 # 添加损失函数
17 cost = fluid.layers.square_error_cost(input=y_predict, label=y)
18 avg_cost = fluid.layers.mean(cost)  # 求均方差
19 # 定义优化方法
20 optimizer = fluid.optimizer.SGD(learning_rate=0.01)
21 optimizer.minimize(avg_cost)  # 指定最小化均方差值
22
23 # 搭建网络
24 place = fluid.CPUPlace()  # 指定在CPU执行
25 exe = fluid.Executor(place)
26 exe.run(fluid.default_startup_program())  # 初始化系统参数
27
28 model_save_dir = "./model/lr_persis/"
29 if os.path.exists(model_save_dir):
30     fluid.io.load_persistables(exe, model_save_dir,
   fluid.default_main_program())
31     print("加载增量模型成功.")
32
33 # 开始迭代训练
34 costs = []
35 iters = []
36 values = []
37 params = {"x": train_data, "y": y_true}
38 for i in range(50):
39     outs = exe.run(feed=params, fetch_list=[y_predict.name,
   avg_cost.name])
40     iters.append(i)  # 迭代次数
41     costs.append(outs[1][0])  # 损失值
42     print("%d: %f" % (i, outs[1][0]))
43
44 # 损失函数可视化
45 plt.figure("Trainging")
46 plt.title("Training Cost", fontsize=24)
47 plt.xlabel("Iter", fontsize=14)
48 plt.ylabel("Cost", fontsize=14)
```

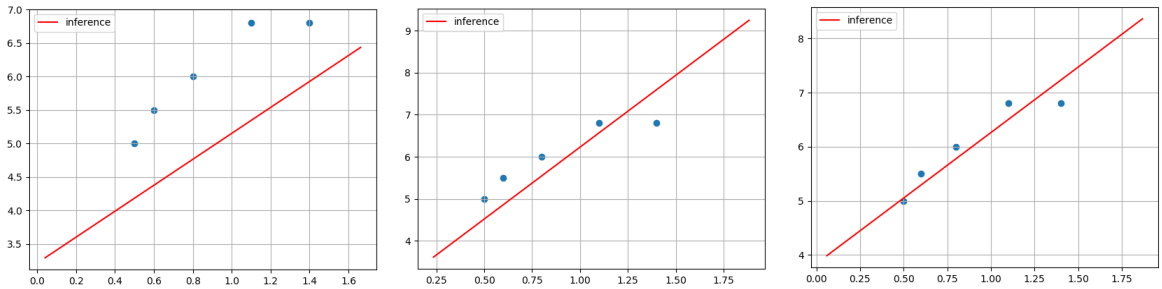
```
plt.plot(iters, costs, color="red", label="Training Cost")  # 绘制损失函数曲线
plt.grid()  # 绘制网格线
plt.savefig("train.png")  # 保存图片


plt.legend()
plt.grid()  # 绘制网格线
plt.savefig("infer.png")  # 保存图片
# plt.show()  # 显示图片
print("训练完成.")

# 保存增量模型
if not os.path.exists(model_save_dir):  # 如果存储模型的目录不存在，则创建
    os.makedirs(model_save_dir)
fluid.io.save_persistables(exe, model_save_dir,
fluid.default_main_program())

print("保存增量模型成功.")

# 保存最终模型
freeze_dir = "./model/lr_freeze/"
if not os.path.exists(freeze_dir):  # 如果存储模型的目录不存在，则创建
    os.makedirs(freeze_dir)
fluid.io.save_inference_model(freeze_dir,  # 保存模型的路径
                              ["x"],  # 预测需要喂入的数据
                              [y_predict],  # 保存预测结果的变量
                              exe)  # 模型

print("模型保存成功.")
```

2）模型加载与使用

```
# 增量模型加载
import paddle
import paddle.fluid as fluid
import numpy as np
import math
import os
import matplotlib.pyplot as plt

train_data = np.array([[0.5], [0.6], [0.8], [1.1],
[1.4]]).astype('float32')
y_true = np.array([[5.0], [5.5], [6.0], [6.8],
[6.8]]).astype('float32')
```

```
11
12  # 模型预测
13  infer_exe = fluid.Executor(fluid.CPUPlace())  # 创建用于预测的Executor
14  infer_result = [] #预测值列表
15
16  freeze_dir = "./model/lr_freeze/"
17  [infer_program, feed_target_names, fetch_targets] = \
18      fluid.io.load_inference_model(freeze_dir,  # 模型保存路径
19                                    infer_exe)  # 要执行模型的Executor
20
21
22  tmp = np.random.rand(10, 1)  # 生成10行1列的均匀随机数组
23  tmp = tmp * 2  # 范围放大到0~2之间
24  tmp.sort(axis=0)  # 排序
25  x_test = np.array(tmp).astype("float32")
26  x_name = feed_target_names[0]  # 模型中保存的输入参数名称
27
28  # 执行预测
29  y_out = infer_exe.run(infer_program,  # 预测program
30                        feed={x_name: x_test},  # 喂入预测的值
31                        fetch_list=fetch_targets)  # 预测结果
32  y_test = y_out[0]
33
34
35  # 线性模型可视化
36  plt.figure("Inference")
37  plt.title("Linear Regression", fontsize=24)
38  plt.plot(x_test, y_test, color="red", label="inference")  # 绘制模型线条
39  plt.scatter(train_data, y_true)  # 原始样本散点图
40
41  plt.legend()
42  plt.grid()  # 绘制网格线
43  plt.savefig("infer.png")  # 保存图片
44  plt.show()  # 显示图片
```

三次增量训练效果：



实验六：水果识别

1. 数据预处理部分：

```python
# 02_fruits.py
# 利用深层CNN实现水果分类
# 数据集：爬虫从百度图片搜索结果爬取
# 内容：包含1036张水果图片，共5个类别（苹果288张、香蕉275张、葡萄216张、橙子276张、梨251张）

############################## 预处理部分 ################################
import os

name_dict = {"apple":0, "banana":1, "grape":2, "orange":3, "pear":4}
data_root_path = "data/fruits/" # 数据样本所在目录
test_file_path = data_root_path + "test.txt" #测试文件路径
train_file_path = data_root_path + "train.txt" # 训练文件路径
name_data_list = {} # 记录每个类别有哪些图片  key:水果名称  value:图片路径构成的列表

# 将图片路径存入name_data_list字典中
def save_train_test_file(path, name):
    if name not in name_data_list: # 该类别水果不在字典中，则新建一个列表插入字典
        img_list = []
        img_list.append(path) # 将图片路径存入列表
        name_data_list[name] = img_list # 将图片列表插入字典
    else: # 该类别水果在字典中，直接添加到列表
        name_data_list[name].append(path)

# 遍历数据集下面每个子目录，将图片路径写入上面的字典
dirs = os.listdir(data_root_path) # 列出数据集目下所有的文件和子目录
for d in dirs:
    full_path = data_root_path + d  # 拼完整路径

    if os.path.isdir(full_path): # 是一个子目录
        imgs = os.listdir(full_path) # 列出子目录中所有的文件
        for img in imgs:
            save_train_test_file(full_path + "/" + img, #拼图片完整路径
                                 d) # 以子目录名称作为类别名称
    else: # 文件
        pass

# 将name_data_list字典中的内容写入文件
## 清空训练集和测试集文件
with open(test_file_path, "w") as f:
    pass

with open(train_file_path, "w") as f:
    pass

```

```
# 遍历字典，将字典中的内容写入训练集和测试集
for name, img_list in name_data_list.items():
    i = 0
    num = len(img_list) # 获取每个类别图片数量
    print("%s: %d张" % (name, num))
    # 写训练集和测试集
    for img in img_list:
        if i % 10 == 0: # 每10笔写一笔测试集
            with open(test_file_path, "a") as f: #以追加模式打开测试集文件
                line = "%s\t%d\n" % (img, name_dict[name]) # 拼一行
                f.write(line) # 写入文件
        else: # 训练集
            with open(train_file_path, "a") as f: #以追加模式打开测试集文件
                line = "%s\t%d\n" % (img, name_dict[name]) # 拼一行
                f.write(line) # 写入文件

        i += 1 # 计数器加1

print("数据预处理完成.")
```

2. 模型训练与评估

```
import paddle
import paddle.fluid as fluid
import numpy
import sys
import os
from multiprocessing import cpu_count
import time
import matplotlib.pyplot as plt

def train_mapper(sample):
    """
    根据传入的样本数据(一行文本)读取图片数据并返回
    :param sample: 元组，格式为(图片路径，类别)
    :return:返回图像数据、类别
    """
    img, label = sample # img为路基，label为类别
    if not os.path.exists(img):
        print(img, "图片不存在")

    # 读取图片内容
    img = paddle.dataset.image.load_image(img)
    # 对图片数据进行简单变换，设置成固定大小
    img = paddle.dataset.image.simple_transform(im=img, # 原始图像数据
                                                resize_size=100, # 图像要设置的大小
```

```
                                                crop_size=100, # 裁剪图像大小
                                                is_color=True, # 彩色图像
                                                is_train=True) # 随机裁剪
    # 归一化处理，将每个像素值转换到0~1
    img = img.astype("float32") / 255.0
    return img, label  # 返回图像、类别

# 从训练集中读取数据
def train_r(train_list, buffered_size=1024):
    def reader():
        with open(train_list, "r") as f:
            lines = [line.strip() for line in f] # 读取所有行，并去空格
            for line in lines:
                # 去掉一行数据的换行符，并按tab键拆分，存入两个变量
                img_path, lab = line.replace("\n","").split("\t")
                yield img_path, int(lab) # 返回图片路径、类别(整数)
    return paddle.reader.xmap_readers(train_mapper, # 将reader读取的数进一步处理
                                      reader, # reader读取到的数据传递给train_mapper
                                      cpu_count(), # 线程数量
                                      buffered_size) # 缓冲区大小

# 定义reader
BATCH_SIZE = 32  # 批次大小
trainer_reader = train_r(train_list=train_file_path) #原始reader
random_train_reader = paddle.reader.shuffle(reader=trainer_reader,
                                            buf_size=1300) # 包装成随机读取器
batch_train_reader = paddle.batch(random_train_reader,
                                  batch_size=BATCH_SIZE) # 批量读取器
# 变量
image = fluid.layers.data(name="image", shape=[3, 100, 100], dtype="float32")
label = fluid.layers.data(name="label", shape=[1], dtype="int64")

# 搭建CNN函数
# 结构：输入层 --> 卷积/激活/池化/dropout --> 卷积/激活/池化/dropout -->
#       卷积/激活/池化/dropout --> fc --> dropout --> fc(softmax)
def convolution_neural_network(image, type_size):
    """
    创建CNN
    :param image: 图像数据
    :param type_size: 输出类别数量
```

```
    :return: 分类概率
    """
    # 第一组 卷积/激活/池化/dropout
    conv_pool_1 = fluid.nets.simple_img_conv_pool(input=image, # 原始图像数据
                                                  filter_size=3, # 卷积核大小
                                                  num_filters=32, # 卷积核数量
                                                  pool_size=2, # 2*2区域池化
                                                  pool_stride=2, # 池化步长值
                                                  act="relu")#激活函数
    drop = fluid.layers.dropout(x=conv_pool_1, dropout_prob=0.5)

    # 第二组
    conv_pool_2 = fluid.nets.simple_img_conv_pool(input=drop, # 以上一个drop输出作为输入
                                                  filter_size=3, # 卷积核大小
                                                  num_filters=64, # 卷积核数量
                                                  pool_size=2, # 2*2区域池化
                                                  pool_stride=2, # 池化步长值
                                                  act="relu")#激活函数
    drop = fluid.layers.dropout(x=conv_pool_2, dropout_prob=0.5)

    # 第三组
    conv_pool_3 = fluid.nets.simple_img_conv_pool(input=drop, # 以上一个drop输出作为输入
                                                  filter_size=3, # 卷积核大小
                                                  num_filters=64, # 卷积核数量
                                                  pool_size=2, # 2*2区域池化
                                                  pool_stride=2, # 池化步长值
                                                  act="relu")#激活函数
    drop = fluid.layers.dropout(x=conv_pool_3, dropout_prob=0.5)

    # 全连接层
    fc = fluid.layers.fc(input=drop, size=512, act="relu")
    # dropout
```

```
    drop = fluid.layers.dropout(x=fc, dropout_prob=0.5)
    # 输出层(fc)
    predict = fluid.layers.fc(input=drop, # 输入
                              size=type_size, # 输出值的个数(5个类别)
                              act="softmax") # 输出层采用softmax作为激活函数
    return predict

# 调用函数，创建CNN
predict = convolution_neural_network(image=image, type_size=5)
# 损失函数:交叉熵
cost = fluid.layers.cross_entropy(input=predict, # 预测结果
                                  label=label) # 真实结果
avg_cost = fluid.layers.mean(cost)
# 计算准确率
accuracy = fluid.layers.accuracy(input=predict, # 预测结果
                                 label=label) # 真实结果
# 优化器
optimizer = fluid.optimizer.Adam(learning_rate=0.001)
optimizer.minimize(avg_cost) # 将损失函数值优化到最小

# 执行器
# place = fluid.CPUPlace()
place = fluid.CUDAPlace(0) # GPU训练
exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())
# feeder
feeder = fluid.DataFeeder(feed_list=[image, label],  # 指定要喂入数据
                          place=place)

model_save_dir = "model/fruits/" # 模型保存路径
costs = [] # 记录损失值
accs = [] # 记录准确度
times = 0
batches = [] # 迭代次数

# 开始训练
for pass_id in range(40):
    train_cost = 0 # 临时变量，记录每次训练的损失值
    for batch_id, data in enumerate(batch_train_reader()): # 循环读取样本，执行训练
        times += 1
        train_cost, train_acc = exe.run(program=fluid.default_main_program(),
                                        feed=feeder.feed(data), # 喂入参数
```

```
                                        fetch_list=[avg_cost,
accuracy])# 获取损失值、准确率
        if batch_id % 20 == 0:
            print("pass_id:%d, step:%d, cost:%f, acc:%f" %
                  (pass_id, batch_id, train_cost[0], train_acc[0]))
            accs.append(train_acc[0]) # 记录准确率
            costs.append(train_cost[0]) # 记录损失值
            batches.append(times) # 记录迭代次数

# 训练结束后，保存模型
if not os.path.exists(model_save_dir):
    os.makedirs(model_save_dir)
fluid.io.save_inference_model(dirname=model_save_dir,
                              feeded_var_names=["image"],
                              target_vars=[predict],
                              executor=exe)
print("训练保存模型完成!")

# 训练过程可视化
plt.title("training", fontsize=24)
plt.xlabel("iter", fontsize=20)
plt.ylabel("cost/acc", fontsize=20)
plt.plot(batches, costs, color='red', label="Training Cost")
plt.plot(batches, accs, color='green', label="Training Acc")
plt.legend()
plt.grid()
plt.show()
plt.savefig("train.png")
```

3. 预测

```
from PIL import Image

# 定义执行器
place = fluid.CPUPlace()
infer_exe = fluid.Executor(place)
model_save_dir = "model/fruits/" # 模型保存路径

# 加载数据
def load_img(path):
    img = paddle.dataset.image.load_and_transform(path, 100, 100,
False).astype("float32")
    img = img / 255.0
    return img

infer_imgs = [] # 存放要预测图像数据
test_img = "./data/grape_1.png" #待预测图片
```

```
infer_imgs.append(load_img(test_img)) #加载图片，并且将图片数据添加到待预测列表
infer_imgs = numpy.array(infer_imgs) # 转换成数组

# 加载模型
infer_program, feed_target_names, fetch_targets = \
    fluid.io.load_inference_model(model_save_dir, infer_exe)
# 执行预测
results = infer_exe.run(infer_program, # 执行预测program
                        feed={feed_target_names[0]: infer_imgs}, # 传入待预测图像数据
                        fetch_list=fetch_targets) #返回结果
print(results)

result = numpy.argmax(results[0]) # 取出预测结果中概率最大的元素索引值
for k, v in name_dict.items(): # 将类别由数字转换为名称
    if result == v:  # 如果预测结果等于v，打印出名称
        print("预测结果:", k) # 打印出名称

# 显示待预测的图片
img = Image.open(test_img)
plt.imshow(img)
plt.show()
```

实验七：中文文本分类

1. 数据预处

```
# 中文资讯分类示例
# 任务：根据样本，训练模型，将新的文本划分到正确的类别
'''
数据来源：从网站上爬取56821条中文新闻摘要
数据类容：包含10类(国际、文化、娱乐、体育、财经、汽车、教育、科技、房产、证券)
'''

############################## 数据预处理 ##############################
import os
from multiprocessing import cpu_count
import numpy as np
import paddle
import paddle.fluid as fluid

# 定义公共变量
data_root = "data/news_classify/" # 数据集所在目录
data_file = "news_classify_data.txt" # 原始样本文件名
test_file = "test_list.txt" # 测试集文件名称
train_file = "train_list.txt" # 训练集文件名称
```

```python
dict_file = "dict_txt.txt" # 编码后的字典文件

data_file_path = data_root + data_file # 样本文件完整路径
dict_file_path = data_root + dict_file # 字典文件完整路径
test_file_path = data_root + test_file # 测试集文件完整路径
train_file_path = data_root + train_file # 训练集文件完整路径

# 生成字典文件：把每个字编码成一个数字，并存入文件中
def create_dict():
    dict_set = set()  # 集合，去重
    with open(data_file_path, "r", encoding="utf-8") as f: # 打开原始样本文件
        lines = f.readlines() # 读取所有的行

    # 遍历每行
    for line in lines:
        title = line.split("_!_")[-1].replace("\n", "") #取出标题部分，并取出换行符
        for w in title: # 取出标题部分每个字
            dict_set.add(w) # 将每个字存入集合进行去重

    # 遍历集合，每个字分配一个编码
    dict_list = []
    i = 0 # 计数器
    for s in dict_set:
        dict_list.append([s, i]) # 将"文字,编码"键值对添加到列表中
        i += 1

    dict_txt = dict(dict_list) # 将列表转换为字典
    end_dict = {"<unk>": i} # 未知字符
    dict_txt.update(end_dict) # 将未知字符编码添加到字典中

    # 将字典保存到文件中
    with open(dict_file_path, "w", encoding="utf-8") as f:
        f.write(str(dict_txt))  # 将字典转换为字符串并存入文件

    print("生成字典完成.")

# 对一行标题进行编码
def line_encoding(title, dict_txt, label):
    new_line = ""  # 返回的结果
    for w in title:
        if w in dict_txt: # 如果字已经在字典中
            code = str(dict_txt[w])  # 取出对应的编码
        else:
            code = str(dict_txt["<unk>"]) # 取未知字符的编码
        new_line = new_line + code + "," # 将编码追加到新的字符串后
```

```
65
66      new_line = new_line[:-1] # 去掉最后一个逗号
67      new_line = new_line + "\t" + label + "\n" # 拼接成一行，标题和标签用\t
    分隔
68      return new_line
69
70
71  # 对原始样本进行编码，对每个标题的每个字使用字典中编码的整数进行替换
72  # 产生编码后的句子，并且存入测试集、训练集
73  def create_data_list():
74      # 清空测试集、训练集文件
75      with open(test_file_path, "w") as f:
76          pass
77      with open(train_file_path, "w") as f:
78          pass
79
80      # 打开原始样本文件，取出标题部分，对标题进行编码
81      with open(dict_file_path, "r", encoding="utf-8") as f_dict:
82          # 读取字典文件中的第一行(只有一行)，通过调用eval函数转换为字典对象
83          dict_txt = eval(f_dict.readlines()[0])
84
85      with open(data_file_path, "r", encoding="utf-8") as f_data:
86          lines = f_data.readlines()
87
88      # 取出标题并编码
89      i = 0
90      for line in lines:
91          words = line.replace("\n", "").split("_!_") # 拆分每行
92          label = words[1] # 分类
93          title = words[3] # 标题
94
95          new_line = line_encoding(title, dict_txt, label)  # 对标题进行编
    码
96          if i % 10 == 0: # 每10笔写一笔测试集文件
97              with open(test_file_path, "a", encoding="utf-8") as f:
98                  f.write(new_line)
99          else: # 写入训练集
100             with open(train_file_path, "a", encoding="utf-8") as f:
101                 f.write(new_line)
102         i += 1
103     print("生成测试集、训练集结束.")
104
105 create_dict()  # 生成字典
106 create_data_list() # 生成训练集、测试集
```

2. 模型训练与评估

```python
# 读取字典文件，并返回字典长度
def get_dict_len(dict_path):
    with open(dict_path, "r", encoding="utf-8") as f:
        line = eval(f.readlines()[0])  # 读取字典文件内容，并返回一个字典对象

    return len(line.keys())


# 定义data_mapper，将reader读取的数据进行二次处理
# 将传入的字符串转换为整型并返回
def data_mapper(sample):
    data, label = sample  # 将sample元组拆分到两个变量
    # 拆分句子，将每个编码转换为数字，并存入一个列表中
    val = [int(w) for w in data.split(",")]
    return val, int(label)  # 返回整数列表，标签(转换成整数)


# 定义reader
def train_reader(train_file_path):
    def reader():
        with open(train_file_path, "r") as f:
            lines = f.readlines()  # 读取所有的行
            np.random.shuffle(lines)  # 打乱所有样本

            for line in lines:
                data, label = line.split("\t")  # 拆分样本到两个变量中
                yield data, label

    return paddle.reader.xmap_readers(data_mapper,  # reader读取的数据进行下一步处理函数
                                      reader,  # 读取样本的reader
                                      cpu_count(),  # 线程数
                                      1024)  # 缓冲区大小


# 读取测试集reader
def test_reader(test_file_path):
    def reader():
        with open(test_file_path, "r") as f:
            lines = f.readlines()

            for line in lines:
                data, label = line.split("\t")
                yield data, label

    return paddle.reader.xmap_readers(data_mapper,
```

```
                                        reader,
                                        cpu_count(),
                                        1024)


# 定义网络
def CNN_net(data, dict_dim, class_dim=10, emb_dim=128, hid_dim=128,
hid_dim2=98):
    # embedding(词嵌入层)：生成词向量，得到一个新的粘稠的实向量
    # 以使用较少的维度，表达更丰富的信息
    emb = fluid.layers.embedding(input=data, size=[dict_dim, emb_dim])

    # 并列两个卷积、池化层
    conv1 = fluid.nets.sequence_conv_pool(input=emb,  # 输入，上一个词嵌
入层的输出作为输入
                                          num_filters=hid_dim,  # 卷积
核数量
                                          filter_size=3,  # 卷积核大小
                                          act="tanh",  # 激活函数
                                          pool_type="sqrt")  # 池化类型

    conv2 = fluid.nets.sequence_conv_pool(input=emb,  # 输入，上一个词嵌
入层的输出作为输入
                                          num_filters=hid_dim2,  # 卷积
核数量
                                          filter_size=4,  # 卷积核大小
                                          act="tanh",  # 激活函数
                                          pool_type="sqrt")  # 池化类型
    output = fluid.layers.fc(input=[conv1, conv2],  # 输入
                             size=class_dim,  # 输出类别数量
                             act="softmax")  # 激活函数
    return output

# 定义模型、训练、评估、保存
model_save_dir = "model/news_classify/"  # 模型保存路径

words = fluid.layers.data(name="words", shape=[1], dtype="int64",
                          lod_level=1) # 张量层级
label = fluid.layers.data(name="label", shape=[1], dtype="int64")

# 获取字典长度
dict_dim = get_dict_len(dict_file_path)
# 调用函数创建CNN
model = CNN_net(words, dict_dim)
# 定义损失函数
cost = fluid.layers.cross_entropy(input=model, # 预测结果
                                  label=label) # 真实结果
```

```
avg_cost = fluid.layers.mean(cost) # 求损失函数均值
# 准确率
acc = fluid.layers.accuracy(input=model, # 预测结果
                            label=label) # 真实结果
# 克隆program用于模型测试评估
# for_test如果为True，会少一些优化
test_program = fluid.default_main_program().clone(for_test=True)
# 定义优化器
optimizer = fluid.optimizer.AdagradOptimizer(learning_rate=0.001)
optimizer.minimize(avg_cost)

# 定义执行器
place = fluid.CPUPlace()
exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())

# 准备数据
tr_reader = train_reader(train_file_path)
batch_train_reader = paddle.batch(reader=tr_reader, batch_size=128)

ts_reader = test_reader(test_file_path)
batch_test_reader = paddle.batch(reader=ts_reader, batch_size=128)

feeder = fluid.DataFeeder(place=place, feed_list=[words, label]) #
feeder

# 开始训练
for pass_id in range(20):
    for batch_id, data in enumerate(batch_train_reader()):
        train_cost, train_acc =
exe.run(program=fluid.default_main_program(),
                                        feed=feeder.feed(data), # 喂入
数据
                                        fetch_list=[avg_cost, acc]) #
要获取的结果
        # 打印
        if batch_id % 100 == 0:
            print("pass_id:%d, batch_id:%d, cost:%f, acc:%f" %
                  (pass_id, batch_id, train_cost[0], train_acc[0]))

    # 每轮次训练完成后，进行模型评估
    test_costs_list = [] # 存放所有的损失值
    test_accs_list = [] # 存放准确率

    for batch_id, data in enumerate(batch_test_reader()):  # 读取一个批
次测试数据
```

```
        test_cost, test_acc = exe.run(program=test_program, # 执行test_program
                                      feed=feeder.feed(data), # 喂入测试数据
                                      fetch_list=[avg_cost, acc])  # 要获取的结果
        test_costs_list.append(test_cost[0]) # 记录损失值
        test_accs_list.append(test_acc[0]) # 记录准确率

    # 计算平均准确率和损失值
    avg_test_cost = sum(test_costs_list) / len(test_costs_list)
    avg_test_acc = sum(test_accs_list) / len(test_accs_list)

    print("pass_id:%d, test_cost:%f, test_acc:%f" %
          (pass_id, avg_test_cost, avg_test_acc))

# 保存模型
if not os.path.exists(model_save_dir):
    os.makedirs(model_save_dir)
fluid.io.save_inference_model(model_save_dir, # 模型保存路径
                              feeded_var_names=[words.name], # 使用模型时需传入的参数
                              target_vars=[model], # 预测结果
                              executor=exe) # 执行器
print("模型保存完成.")
```

3. 预测

```
model_save_dir = "model/news_classify/"

def get_data(sentence):
    # 读取字典中的内容
    with open(dict_file_path, "r", encoding="utf-8") as f:
        dict_txt = eval(f.readlines()[0])

    keys = dict_txt.keys()
    ret = []  # 编码结果
    for s in sentence:  # 遍历句子
        if not s in keys:  # 字不在字典中，取未知字符
            s = "<unk>"
        ret.append(int(dict_txt[s]))

    return ret

# 创建执行器
place = fluid.CPUPlace()
exe = fluid.Executor(place)
```

```
exe.run(fluid.default_startup_program())

print("加载模型")
infer_program, feeded_var_names, target_var = \
    fluid.io.load_inference_model(dirname=model_save_dir, executor=exe)

# 生成测试数据
texts = []
data1 = get_data("在获得诺贝尔文学奖7年之后，莫言15日晚间在山西汾阳贾家庄如是说")
data2 = get_data("综合'今日美国'、《世界日报》等当地媒体报道，芝加哥河滨警察局表示")
data3 = get_data("中国队无缘2020年世界杯")
data4 = get_data("中国人民银行今日发布通知，降低准备金率，预计释放4000亿流动性")
data5 = get_data("10月20日,第六届世界互联网大会正式开幕")
data6 = get_data("同一户型，为什么高层比低层要贵那么多？")
data7 = get_data("揭秘A股周涨5%资金动向：追捧2类股，抛售600亿香饽饽")
data8 = get_data("宋慧乔陷入感染危机，前夫宋仲基不戴口罩露面，身处国外神态轻松")
data9 = get_data("此盆栽花很好养，花美似牡丹，三季开花，南北都能养，很值得栽培")# 不属于任何一个类别

texts.append(data1)
texts.append(data2)
texts.append(data3)
texts.append(data4)
texts.append(data5)
texts.append(data6)
texts.append(data7)
texts.append(data8)
texts.append(data9)

# 获取每个句子词数量
base_shape = [[len(c) for c in texts]]
# 生成数据
tensor_words = fluid.create_lod_tensor(texts, base_shape, place)
# 执行预测
result = exe.run(program=infer_program,
                 feed={feeded_var_names[0]: tensor_words}, # 待预测的数据
                 fetch_list=target_var)

# print(result)

names = ["文化", "娱乐", "体育", "财经", "房产", "汽车", "教育", "科技", "国际", "证券"]

# 获取最大值的索引
for i in range(len(texts)):
    lab = np.argsort(result)[0][i][-1]  # 取出最大值的元素下标
```

```
    print("预测结果：%d，名称:%s，概率:%f" % (lab, names[lab], result[0][i][lab]))
```

实验八：中文情绪分析

1. 数据预处理与模型训练

```
# 中文情绪分析示例：数据预处理部分
''' 数据集介绍
中文酒店评论，7766笔数据，分为正面、负面评价
'''
import paddle
import paddle.dataset.imdb as imdb
import paddle.fluid as fluid
import numpy as np
import os
import random
from multiprocessing import cpu_count

# 数据预处理，将中文文字解析出来，并进行编码转换为数字，每一行文字存入数组
mydict = {}  # 存放出现的字及编码，格式： 好,1
code = 1
data_file = "data/hotel_discuss2.csv"  # 原始样本路径
dict_file = "data/hotel_dict.txt" # 字典文件路径
encoding_file = "data/hotel_encoding.txt" # 编码后的样本文件路径
puncts = " \n"  # 要剔除的标点符号列表

with open(data_file, "r", encoding="utf-8-sig") as f:
    for line in f.readlines():
        # print(line)
        trim_line = line.strip()
        for ch in trim_line:
            if ch in puncts:  # 符号不参与编码
                continue

            if ch in mydict:  # 已经在编码字典中
                continue
            elif len(ch) <= 0:
                continue
            else:  # 当前文字没在字典中
                mydict[ch] = code
                code += 1
    code += 1
    mydict["<unk>"] = code  # 未知字符

# 循环结束后，将字典存入字典文件
with open(dict_file, "w", encoding="utf-8-sig") as f:
```

```
    f.write(str(mydict))
    print("数据字典保存完成！")


# 将字典文件中的数据加载到mydict字典中
def load_dict():
    with open(dict_file, "r", encoding="utf-8-sig") as f:
        lines = f.readlines()
        new_dict = eval(lines[0])
    return new_dict

# 对评论数据进行编码
new_dict = load_dict()  # 调用函数加载
with open(data_file, "r", encoding="utf-8-sig") as f:
    with open(encoding_file, "w", encoding="utf-8-sig") as fw:
        for line in f.readlines():
            label = line[0]  # 标签
            remark = line[1:-1]  # 评论

            for ch in remark:
                if ch in puncts:  # 符号不参与编码
                    continue
                else:
                    fw.write(str(mydict[ch]))
                    fw.write(",")
            fw.write("\t" + str(label) + "\n")  # 写入tab分隔符、标签、换行符

print("数据预处理完成")

# 获取字典的长度
def get_dict_len(dict_path):
    with open(dict_path, 'r', encoding='utf-8-sig') as f:
        lines = f.readlines()
        new_dict = eval(lines[0])

    return len(new_dict.keys())

# 创建数据读取器train_reader和test_reader
# 返回评论列表和标签
def data_mapper(sample):
    dt, lbl = sample
    val = [int(word) for word in dt.split(",") if word.isdigit()]
    return val, int(lbl)

# 随机从训练数据集文件中取出一行数据
def train_reader(train_list_path):
```

```
    def reader():
        with open(train_list_path, "r", encoding='utf-8-sig') as f:
            lines = f.readlines()
            np.random.shuffle(lines)  # 打乱数据

            for line in lines:
                data, label = line.split("\t")
                yield data, label

    # 返回xmap_readers，能够使用多线程方式读取数据
    return paddle.reader.xmap_readers(data_mapper,  # 映射函数
                                      reader,  # 读取数据内容
                                      cpu_count(),  # 线程数量
                                      1024)  # 读取数据队列大小

# 定义LSTM网络
def lstm_net(ipt, input_dim):
    ipt = fluid.layers.reshape(ipt, [-1, 1],
                               inplace=True) # 是否替换，True则表示输入和返回是同一个对象
    # 词嵌入层
    emb = fluid.layers.embedding(input=ipt, size=[input_dim, 128], is_sparse=True)

    # 第一个全连接层
    fc1 = fluid.layers.fc(input=emb, size=128)

    # 第一分支：LSTM分支
    lstm1, _ = fluid.layers.dynamic_lstm(input=fc1, size=128)
    lstm2 = fluid.layers.sequence_pool(input=lstm1, pool_type="max")

    # 第二分支
    conv = fluid.layers.sequence_pool(input=fc1, pool_type="max")

    # 输出层：全连接
    out = fluid.layers.fc([conv, lstm2], size=2, act="softmax")

    return out

# 定义输入数据，lod_level不为0指定输入数据为序列数据
dict_len = get_dict_len(dict_file)  # 获取数据字典长度
rmk = fluid.layers.data(name="rmk", shape=[1], dtype="int64", lod_level=1)
label = fluid.layers.data(name="label", shape=[1], dtype="int64")
# 定义长短期记忆网络
model = lstm_net(rmk, dict_len)

```

```
# 定义损失函数，情绪判断实际是一个分类任务，使用交叉熵作为损失函数
cost = fluid.layers.cross_entropy(input=model, label=label)
avg_cost = fluid.layers.mean(cost)  # 求损失值平均数
# layers.accuracy接口，用来评估预测准确率
acc = fluid.layers.accuracy(input=model, label=label)

# 定义优化方法
# Adagrad(自适应学习率，前期放大梯度调节，后期缩小梯度调节)
optimizer = fluid.optimizer.AdagradOptimizer(learning_rate=0.001)
opt = optimizer.minimize(avg_cost)

# 定义网络
# place = fluid.CPUPlace()
place = fluid.CUDAPlace(0)
exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())  # 参数初始化

# 定义reader
reader = train_reader(encoding_file)
batch_train_reader = paddle.batch(reader, batch_size=128)

# 定义输入数据的维度，数据的顺序是一条句子数据对应一个标签
feeder = fluid.DataFeeder(place=place, feed_list=[rmk, label])

for pass_id in range(40):
    for batch_id, data in enumerate(batch_train_reader()):
        train_cost, train_acc =
exe.run(program=fluid.default_main_program(),
                                        feed=feeder.feed(data),
                                        fetch_list=[avg_cost, acc])

        if batch_id % 20 == 0:
            print("pass_id: %d, batch_id: %d, cost: %0.5f, acc:%.5f" %
                  (pass_id, batch_id, train_cost[0], train_acc))

print("模型训练完成......")

# 保存模型
model_save_dir = "model/chn_emotion_analyses.model"
if not os.path.exists(model_save_dir):
    print("create model path")
    os.makedirs(model_save_dir)

fluid.io.save_inference_model(model_save_dir,  # 保存路径
                              feeded_var_names=[rmk.name],
                              target_vars=[model],
                              executor=exe)  # Executor
```

```
177
178 print("模型保存完成, 保存路径: ", model_save_dir)
```

2. 预测

```
1  import paddle
2  import paddle.fluid as fluid
3  import numpy as np
4  import os
5  import random
6  from multiprocessing import cpu_count
7
8  data_file = "data/hotel_discuss2.csv"
9  dict_file = "data/hotel_dict.txt"
10 encoding_file = "data/hotel_encoding.txt"
11 model_save_dir = "model/chn_emotion_analyses.model"
12
13 def load_dict():
14     with open(dict_file, "r", encoding="utf-8-sig") as f:
15         lines = f.readlines()
16         new_dict = eval(lines[0])
17         return new_dict
18
19 # 根据字典对字符串进行编码
20 def encode_by_dict(remark, dict_encoded):
21     remark = remark.strip()
22     if len(remark) <= 0:
23         return []
24
25     ret = []
26     for ch in remark:
27         if ch in dict_encoded:
28             ret.append(dict_encoded[ch])
29         else:
30             ret.append(dict_encoded["<unk>"])
31
32     return ret
33
34
35 # 编码,预测
36 lods = []
37 new_dict = load_dict()
38 lods.append(encode_by_dict("总体来说房间非常干净,卫浴设施也相当不错,交通也比较便
   利", new_dict))
39 lods.append(encode_by_dict("酒店交通方便，环境也不错，正好是我们办事地点的旁边，
   感觉性价比还可以", new_dict))
```

```
lods.append(encode_by_dict("设施还可以，服务人员态度也好，交通还算便利", new_dict))
lods.append(encode_by_dict("酒店服务态度极差，设施很差", new_dict))
lods.append(encode_by_dict("我住过的最不好的酒店,以后决不住了", new_dict))
lods.append(encode_by_dict("说实在的我很失望，我想这家酒店以后无论如何我都不会再去了", new_dict))

# 获取每句话的单词数量
base_shape = [[len(c) for c in lods]]

# 生成预测数据
place = fluid.CPUPlace()
infer_exe = fluid.Executor(place)
infer_exe.run(fluid.default_startup_program())

tensor_words = fluid.create_lod_tensor(lods, base_shape, place)

infer_program, feed_target_names, fetch_targets = fluid.io.load_inference_model(dirname=model_save_dir, executor=infer_exe)
# tvar = np.array(fetch_targets, dtype="int64")
results = infer_exe.run(program=infer_program,
                  feed={feed_target_names[0]: tensor_words},
                  fetch_list=fetch_targets)

# 打印每句话的正负面预测概率
for i, r in enumerate(results[0]):
    print("负面: %0.5f, 正面: %0.5f" % (r[0], r[1]))
```

实验九：利用VGG实现图像分类

第一部分：预处理

```
import os

# 定义一组变量
name_dict = {"apple":0, "banana":1, "grape":2, "orange":3, "pear":4} # 水果名称和数组对应字典
data_root_path = "data/fruits/"  # 数据样本所在目录
test_file_path = data_root_path + "test.txt" # 测试集文件路径
train_file_path = data_root_path + "train.txt" # 训练集文件路径
name_data_list = {} # 记录每个类别的图片路径

# 将文件路径存入临时字典
def save_train_test_file(path, name):
    if name not in name_data_list:  # 当前水果没有在字典中，新增
        img_list = []
```

```
        img_list.append(path)  # 将图片添加到列表
        name_data_list[name] = img_list # 将“名称-图片列表”键值对插入字典
    else: # 当前水果已经在字典中，添加到相应的列表
        name_data_list[name].append(path)

# 遍历所有子目录，读取出所有图片文件，并插入字典、保存到测试集、训练集
dirs = os.listdir(data_root_path)
for d in dirs:
    full_path = data_root_path + d  # 目录名称 + 子目录名称

    if os.path.isdir(full_path): # 目录
        imgs = os.listdir(full_path) # 列出子目录中的文件
        for img in imgs:
            save_train_test_file(full_path + "/" + img, # 图片文件完整路径
                                 d) # 子目录名称（类别名称）
    else: # 文件
        pass

# 将字典中的内容保存文件中
with open(test_file_path, "w") as f: # 清空测试集文件
    pass
with open(train_file_path, "w") as f: # 清空训练集文件
    pass

# 遍历字典，将内容写入文件
for name, img_list in name_data_list.items():
    i = 0
    num = len(img_list)  # 每个类别图片数量
    print("%s: %d张图像" % (name, num))

    for img in img_list:  # 遍历每个列表，将图片路径存入文件
        if i % 10 == 0: # 每10张写一张到测试集
            with open(test_file_path, "a") as f:
                line = "%s\t%d\n" % (img, name_dict[name])
                f.write(line) # 写入文件
        else: # 其它写入训练集
            with open(train_file_path, "a") as f:
                line = "%s\t%d\n" % (img, name_dict[name])
                f.write(line) # 写入文件
        i += 1

print("数据预处理完成.")
```

第二部分：模型搭建与训练

```
import paddle
import paddle.fluid as fluid
```

```
import numpy as np
import sys
import os
from multiprocessing import cpu_count
import matplotlib.pyplot as plt

# 数据准备
## 定义reader
## train_mapper函数：对传入的图片路径进行读取，返回图像数据（多通道矩阵）、标签
def train_mapper(sample):
    img, label = sample  # 将sample中值赋给img, label

    if not os.path.exists(img):
        print(img, "图片文件不存在")

    # 读取图片文件
    img = paddle.dataset.image.load_image(img)  # 读取图片
    # 对图像进行简单变换：修剪、设置大小，输出(3, 100, 100)张量
    img = paddle.dataset.image.simple_transform(im=img, # 原图像数据
                                                resize_size=100, # 重设图像大小为100*100
                                                crop_size=100, # 裁剪成100*100
                                                is_color=True, # 彩色图像
                                                is_train=True) # 是否用于训练，影响裁剪策略
    # 对图像数据进行归一化处理，像素的值全部计算压缩到0~1之间
    img = img.astype("float32") / 255.0

    return img, label   # 返回图像数据、标签

# 读取训练集文件，将路径、标签作为参数调用train_mapper函数
def train_r(train_list, buffered_size=1024):
    def reader():
        with open(train_list, "r") as f: # 打开训练集
            lines = [line.strip() for line in f]  # 读取出所有样本行

            for line in lines:
                # 去掉每行的换行符，并根据tab字符进行拆分，得到两个字段
                img_path, lab = line.replace("\n", "").split("\t")
                yield img_path, int(lab)

    # xmap_readers高阶函数，作用是将reader产生的数据穿个train_mapper函数进行下一步处理
    return paddle.reader.xmap_readers(train_mapper, # 二次处理函数
                                      reader, # 原始reader
```

```
                                        cpu_count(), # 线程数(和cpu数量一
致)
                                        buffered_size) # 缓冲区大小

BATCH_SIZE = 32  # 批次大小

# 定义reader
train_reader = train_r(train_list=train_file_path)# train_file_path为训
练集文件路径
random_train_reader = paddle.reader.shuffle(reader=train_reader, # 原始
reader
                                            buf_size=1300) # 缓冲区大小
batch_train_reader = paddle.batch(random_train_reader,
                                  batch_size=BATCH_SIZE)  # 批量读取器
# 定义变量
image = fluid.layers.data(name="image", shape=[3, 100, 100],
dtype="float32")
label = fluid.layers.data(name="label", shape=[1], dtype="int64")

# 创建VGG模型
def vgg_bn_drop(image, type_size):
    def conv_block(ipt, num_filter, groups, dropouts):
        # 创建Convolution2d, BatchNorm, DropOut, Pool2d组
        return fluid.nets.img_conv_group(input=ipt, # 输入图像像,
[N,C,H,W]格式
                                         pool_stride=2, # 池化步长值
                                         pool_size=2, # 池化区域大小
                                         conv_num_filter=[num_filter]
* groups, #卷积核数量
                                         conv_filter_size=3, # 卷积核大
小
                                         conv_act="relu", # 激活函数
                                         conv_with_batchnorm=True,#是否
使用batch normal
                                         pool_type="max") # 池化类型
    conv1 = conv_block(image, 64, 2, [0.0, 0]) # 最后一个参数个数和组数量
相对应
    conv2 = conv_block(conv1, 128, 2, [0.0, 0])
    conv3 = conv_block(conv2, 256, 3, [0.0, 0.0, 0.0])
    conv4 = conv_block(conv3, 512, 3, [0.0, 0.0, 0.0])
    conv5 = conv_block(conv4, 512, 3, [0.0, 0.0, 0.0])

    drop = fluid.layers.dropout(x=conv5, dropout_prob=0.2) # 待调整
    fc1 = fluid.layers.fc(input=drop, size=512, act=None)

    bn = fluid.layers.batch_norm(input=fc1, act="relu") # batch normal
    drop2 = fluid.layers.dropout(x=bn, dropout_prob=0.0)
```

```
    fc2 = fluid.layers.fc(input=drop2, size=512, act=None)
    predict = fluid.layers.fc(input=fc2, size=type_size,
act="softmax")

    return predict

# 调用上面的函数创建VGG
predict = vgg_bn_drop(image=image, type_size=5) # type_size和水果类别一
致
# 损失函数
cost = fluid.layers.cross_entropy(input=predict, # 预测值
                                  label=label) # 真实值
avg_cost = fluid.layers.mean(cost)
# 计算准确率
accuracy = fluid.layers.accuracy(input=predict, # 预测值
                                 label=label)# 真实值
# 优化器
optimizer = fluid.optimizer.Adam(learning_rate=0.0001) # 自适应梯度下降优
化器
optimizer.minimize(avg_cost)

# 创建Executor
place = fluid.CUDAPlace(0) # GPU上执行
exe = fluid.Executor(place) # 执行器
exe.run(fluid.default_startup_program()) # 初始化

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

costs = []  # 记录损失值
accs = [] # 记录准确率
times = 0
batches = [] # 记录批次

for pass_id in range(20):
    train_cost = 0

    for batch_id, data in enumerate(batch_train_reader()):
        times += 1

        train_cost, train_acc =
exe.run(program=fluid.default_main_program(),
                                        feed=feeder.feed(data), #喂入一
个batch数据
                                        fetch_list=[avg_cost,
accuracy]) #获取结果
        if batch_id % 20 == 0:
```

```
            print("pass_id:%d, bat_id:%d, cost:%f, acc:%f" %
                  (pass_id, batch_id, train_cost[0], train_acc[0]))
            accs.append(train_acc[0]) # 记录准确率
            costs.append(train_cost[0]) # 记录损失值
            batches.append(times) # 记录训练批次数

# 保存模型
model_save_dir = "model/fruits/" # 模型保存路径
if not os.path.exists(model_save_dir): # 如果模型路径不存在则创建
    os.makedirs(model_save_dir)
fluid.io.save_inference_model(dirname=model_save_dir, # 保存路径
                              feeded_var_names=["image"], # 使用模型需传入的参数
                              target_vars=[predict], # 模型结果
                              executor=exe) # 模型
print("模型保存完成.")

# 训练过程可视化
plt.figure('training', facecolor='lightgray')
plt.title("training", fontsize=24)
plt.xlabel("iter", fontsize=20)
plt.ylabel("cost/acc", fontsize=20)
plt.plot(batches, costs, color='red', label="Training Cost")
plt.plot(batches, accs, color='green', label="Training Acc")
plt.legend()
plt.grid()
plt.show()
plt.savefig("train.png")
```

第三部分：测试

同实验五