深度学习方法与实践实验三

2020 年 3 月 15 日

年　级:	2020 级	学　　号:	2016012963
姓　名:	董佩杰	指导老师:	牛新

1　设计变量共享网络进行 MNIST 分类

1.1　实验要求

其将图片样本分为上下两半 X1,X2；分别送入 input1,input2。后续的两个路径的线性加权模块 share_base(X)=X*W 共享一个变量 name='w'

整个分类模型可描述为 softmax(share_base(X1)+share_base(X2)+b)

注意：b 是一个变量，share_base 是一层全连接，没有偏置

网络结构如下所示:

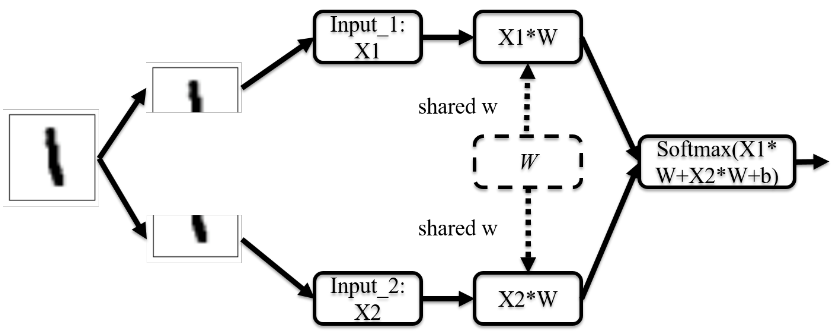


图 1: 网络结构图

1.2　具体实现

- 必须实现要求的共享网络结构，图像是上下分半，网络能够打印出类似下方给的网络结构图。
- 训练分类精度上 0.85 以上

1. 模型部分实现

```python
inputs = Input(shape=(392, ), name="D1_input")
outputs = Dense(num_classes, name="D1")(inputs)
share_base = Model(inputs=inputs, outputs=outputs, name="seq1")

x1 = Input(shape=(392, ), name="input_1")
x2 = Input(shape=(392, ), name="input_2")
s1 = share_base(x1)
s2 = share_base(x2)

b = K.zeros(shape=(10))
x = s1 + s2 + b
x = Activation('softmax', name='activation')(x)

siamese_net = Model(inputs=[x1, x2], outputs=x)
```

2. 使用 plot_model 绘制模型结构

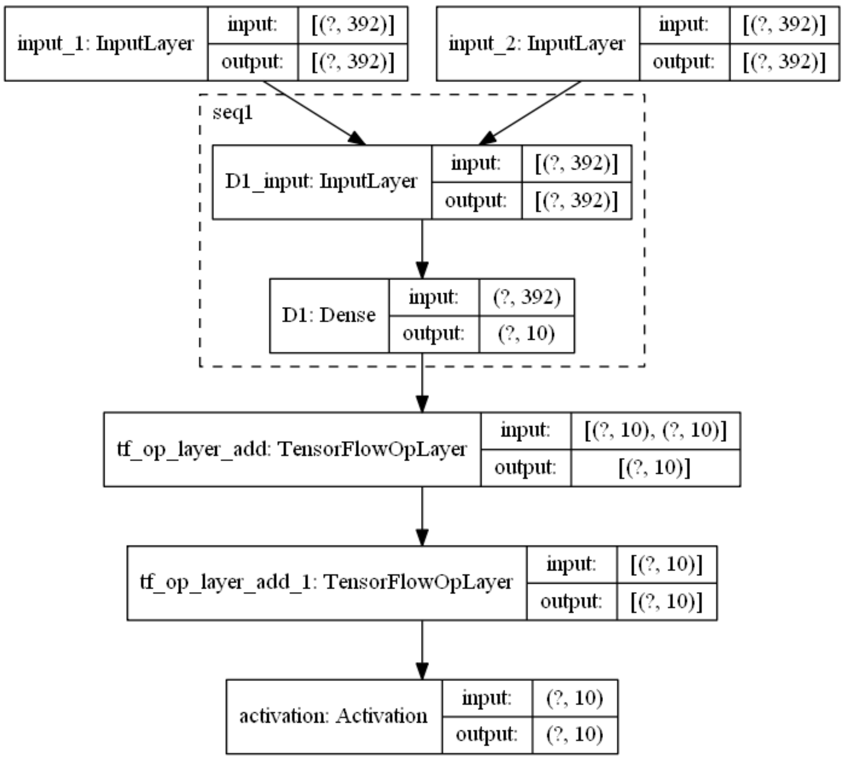


图 2: keras 绘制出模型结构

3. 对网络层中的变量进行可视化

D1 对应的权重形状应为 392x10, 所以可视化的时候分别可视化 10 个 14x28=392 大小的 weights。

```python
train_weights=siamese_net.get_layer('seq1').get_layer('D1').kernel.numpy()

print(train_weights.shape)

num = np.arange(0, 392, 1, dtype="float")
num = num.reshape((14, 28))
plt.figure(num='Weights', figsize=(10, 10))
# 创建一个名为Weights的窗口,并设置大小
for i in range(10):  #W.shape[1]
num = train_weights[:, i: i+1].reshape((14, -1))
```

```python
    plt.subplot(2, 5, i + 1)
    num = num * 255.
    plt.imshow(num, cmap=plt.get_cmap('hot'))
    plt.title('weight %d image.' % (i + 1))  # 第i + 1幅图片
plt.show()
```

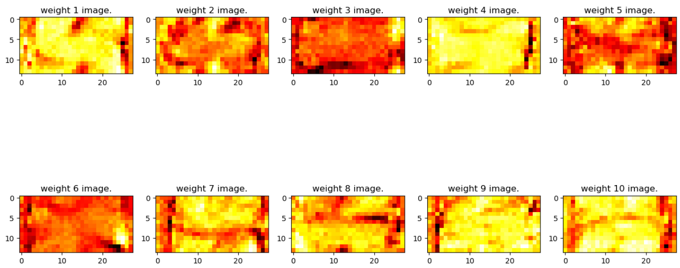
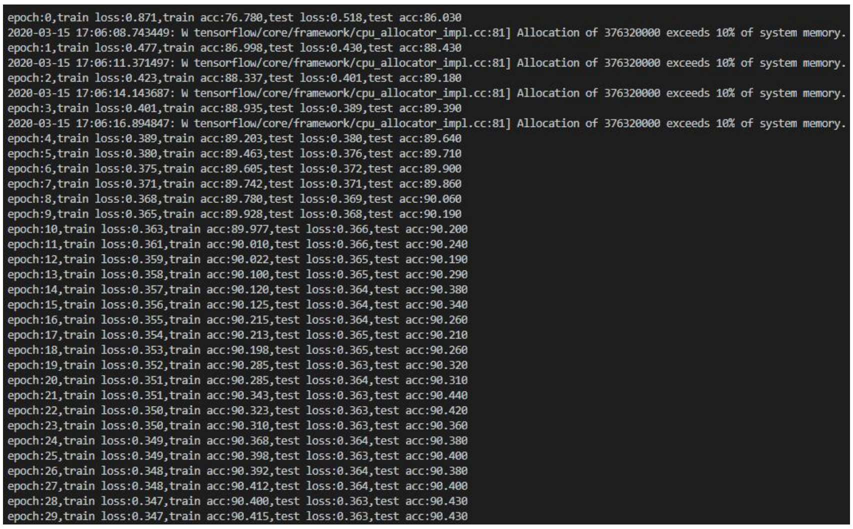


图 3: 可视化结果

4. 训练输出 log



```
epoch:0,train loss:0.871,train acc:76.780,test loss:0.518,test acc:86.030
2020-03-15 17:06:08.743449: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 376320000 exceeds 10% of system memory.
epoch:1,train loss:0.477,train acc:86.998,test loss:0.430,test acc:88.430
2020-03-15 17:06:11.371497: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 376320000 exceeds 10% of system memory.
epoch:2,train loss:0.423,train acc:88.337,test loss:0.401,test acc:89.180
2020-03-15 17:06:14.143687: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 376320000 exceeds 10% of system memory.
epoch:3,train loss:0.401,train acc:88.935,test loss:0.389,test acc:89.390
2020-03-15 17:06:16.894847: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 376320000 exceeds 10% of system memory.
epoch:4,train loss:0.389,train acc:89.203,test loss:0.380,test acc:89.640
epoch:5,train loss:0.380,train acc:89.463,test loss:0.376,test acc:89.710
epoch:6,train loss:0.375,train acc:89.605,test loss:0.372,test acc:89.900
epoch:7,train loss:0.371,train acc:89.742,test loss:0.371,test acc:89.860
epoch:8,train loss:0.368,train acc:89.780,test loss:0.369,test acc:90.060
epoch:9,train loss:0.365,train acc:89.928,test loss:0.368,test acc:90.190
epoch:10,train loss:0.363,train acc:89.977,test loss:0.366,test acc:90.200
epoch:11,train loss:0.361,train acc:90.010,test loss:0.366,test acc:90.240
epoch:12,train loss:0.359,train acc:90.022,test loss:0.365,test acc:90.190
epoch:13,train loss:0.358,train acc:90.100,test loss:0.365,test acc:90.290
epoch:14,train loss:0.357,train acc:90.120,test loss:0.364,test acc:90.380
epoch:15,train loss:0.356,train acc:90.125,test loss:0.364,test acc:90.340
epoch:16,train loss:0.355,train acc:90.215,test loss:0.364,test acc:90.260
epoch:17,train loss:0.354,train acc:90.213,test loss:0.365,test acc:90.210
epoch:18,train loss:0.353,train acc:90.198,test loss:0.365,test acc:90.260
epoch:19,train loss:0.352,train acc:90.285,test loss:0.363,test acc:90.320
epoch:20,train loss:0.351,train acc:90.285,test loss:0.364,test acc:90.310
epoch:21,train loss:0.351,train acc:90.343,test loss:0.363,test acc:90.440
epoch:22,train loss:0.350,train acc:90.323,test loss:0.363,test acc:90.420
epoch:23,train loss:0.350,train acc:90.310,test loss:0.363,test acc:90.360
epoch:24,train loss:0.349,train acc:90.368,test loss:0.364,test acc:90.380
epoch:25,train loss:0.349,train acc:90.398,test loss:0.363,test acc:90.400
epoch:26,train loss:0.348,train acc:90.392,test loss:0.364,test acc:90.380
epoch:27,train loss:0.348,train acc:90.412,test loss:0.364,test acc:90.400
epoch:28,train loss:0.347,train acc:90.400,test loss:0.363,test acc:90.430
epoch:29,train loss:0.347,train acc:90.415,test loss:0.363,test acc:90.430
```

图 4: 训练 log

5. 结果以及全部代码

模型可以在 10 个 epoch 以内训练集和测试集均能达到 90% 的准确率。

```python
import os
import time

import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Model, Sequential
from tensorflow.keras import backend as K
```

```
from tensorflow.keras.layers import (Activation, Conv2D, Dense, Flatten, Input,
concatenate)
from tensorflow.keras.utils import plot_model

num_classes = 10
total_epoch = 30
mnist = tf.keras.datasets.mnist

#1. prepare datasets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0

x_train = x_train.reshape(x_train.shape[0], 2, −1)
x_test = x_test.reshape(x_test.shape[0], 2, −1)

y_train = tf.one_hot(y_train, num_classes)
y_test = tf.one_hot(y_test, num_classes)

train_ds = tf.data.Dataset.from_tensor_slices(
(x_train, y_train)).shuffle(1000).batch(32)
test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)

#2. net_build
inputs = Input(shape=(392, ), name=”D1_input”)
outputs = Dense(num_classes, name=”D1”)(inputs)
share_base = Model(inputs=inputs, outputs=outputs, name=”seq1”)

x1 = Input(shape=(392, ), name=”input_1”)
x2 = Input(shape=(392, ), name=”input_2”)
s1 = share_base(x1)
s2 = share_base(x2)

b = K.zeros(shape=(10))
x = s1 + s2 + b
x = Activation('softmax', name='activation')(x)

siamese_net = Model(inputs=[x1, x2], outputs=x)

plot_model(siamese_net,to_file='./siamese_net.png',show_shapes=True,
expand_nested=True)

#3. train and test
loss_ce = tf.keras.losses.categorical_crossentropy
optimizer = tf.keras.optimizers.Adam(3e−4)

# metrics用于记录指标
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_acc = tf.keras.metrics.CategoricalAccuracy(name='train_acc')
test_loss = tf.keras.metrics.Mean(name='test_loss')
test_acc = tf.keras.metrics.CategoricalAccuracy(name='test_loss')

def train_step(images, labels):
        part1 = images[:, 0]
        part2 = images[:, 1]
        with tf.GradientTape() as tape:
                outputs = siamese_net([part1, part2])
                loss = loss_ce(labels, outputs)
        gradients = tape.gradient(loss, siamese_net.trainable_variables)
```

```python
        optimizer.apply_gradients(zip(gradients, siamese_net.trainable_variables))

        train_loss(loss)
        train_acc(labels, outputs)

def test_step(images, labels):
        part1 = images[:, 0]
        part2 = images[:, 1]
        outputs = siamese_net([part1, part2])
        loss = loss_ce(labels, outputs)

        test_loss(loss)
        test_acc(labels, outputs)


for epoch in range(total_epoch):
        train_acc.reset_states()
        train_loss.reset_states()
        test_acc.reset_states()
        test_loss.reset_states()
        for images, labels in train_ds:
                train_step(images, labels)

        for images, labels in test_ds:
                test_step(images, labels)

        print(
        "epoch:%d,train loss:%.3f,train acc:%.3f,test loss:%.3f,test acc:%.3f"
        % (epoch, train_loss.result(), train_acc.result() * 100,
        test_loss.result(), test_acc.result() * 100))

#4. draw weights of 10 classes
train_weights=siamese_net.get_layer('seq1').get_layer('D1').kernel.numpy()

print(train_weights.shape)

num = np.arange(0, 392, 1, dtype="float")
num = num.reshape((14, 28))
plt.figure(num='Weights', figsize=(10, 10))  # 创建一个名为Weights的窗口,并设置大小
for i in range(10):  #W.shape[1]
        num = train_weights[:, i: i+1].reshape((14, -1))
        plt.subplot(2, 5, i + 1)
        num = num * 255.
        plt.imshow(num, cmap=plt.get_cmap('hot'))
        plt.title('weight %d image.' % (i + 1))  # 第i + 1幅图片
plt.show()
print(np.min(num))
print(np.max(num))
```

2 连体网络 MINIST 识别

2.1 实验要求

构建如下图所示识别模型：该模型由两个相同的网络 G(x) 组成。两个网络共享相同的参数 W.

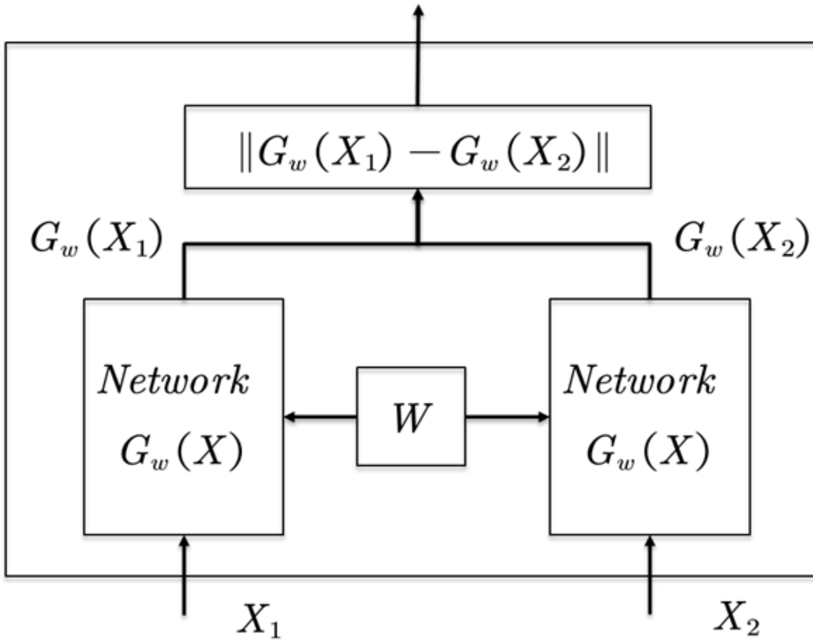


图 5: 网络结构

该模型实现如下的功能，输入两个 MINIST 图片，判断是不是同一个数字。

例如，输入负样本对：X1=6 的图片，X2=9 的图片输出：1; 输入正样本对：X1=3 的图片，X2=3 的图片输出：0

G(x) 是一个一般的全连接网络（两边的网络结构是一样的！共享参数 W、b 等），由结构可以自己设计。比如建议两层网络：hidden1：784(28x28)->500; hidden2: 500->10，使用 relu。也可以尝试其他节点数组合，和其他非线性变换函数。

强调：G(X）的功能定义为提取一张 mnist 图像的特征。

该模型的训练采用如下损失函数：

$$\begin{aligned} L(W,Y,X_1,X_2) &= (1-Y)L_G(E_W) + YL_1(E_W) \\ &= (1-Y)\frac{2}{Q}(E_W)^2 + (Y)2Qe^{-\frac{2.77}{Q}E_W} \end{aligned}$$

其中 Q 是一个常数，用于控制正负样本的平衡，类似于 focal loss。

E_W 是两个网络输出特征的 L_2 距离，$E_W = ||G_W(X_1) - G_W(X_2)||$

2.2 具体实现

1. 模型实现

构造一个孪生网络，其中获得的 embedding 是一个 10 维的向量。

```
class MyModel(Model):
        def __init__(self):
                super(MyModel, self).__init__()
                self.d1 = Dense(500, activation='relu')
                self.d2 = Dense(10, activation='softmax')

        def call(self, x):
                x = self.d1(x)
                embedding = self.d2(x)
```

```
                        return embedding
```

2. 数据加载

数据的 label 是严重不平衡的，所以数据处理部分使用均衡采样的方法来让正样本和负样本比例为 1:1。具体代码如下：

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

x_train = x_train.reshape(x_train.shape[0], -1)
x_test = x_test.reshape(x_test.shape[0], -1)

train_ds = tf.data.Dataset.from_tensor_slices(
(x_train, y_train)).shuffle(1000).batch(BATCH_SIZE)

test_ds = tf.data.Dataset.from_tensor_slices(
(x_test, y_test)).batch(BATCH_SIZE)


def balance_sample(train_ds, test_ds, train=True):
        train_ds = iter(train_ds)
        test_ds = iter(test_ds)
        if train:
                x1, y1 = next(train_ds)
                x2, y2 = next(train_ds)
        else:
                x1, y1 = next(test_ds)
                x2, y2 = next(test_ds)

        y1 = y1[..., np.newaxis]
        y2 = y2[..., np.newaxis]

        idx_same = np.where(y1 == y2) # 找到相同的下角标
        idx_rand = np.random.randint(BATCH_SIZE, size=len(idx_same))  # 随机取样
        index = np.union1d(idx_same, idx_rand).astype(np.int64)  # 所有需要取样的样本

        data_list = []
        label_list = []

        judge = np.array(y1 != y2)

        for ix in index:
                data_list.append([x1[ix], x2[ix]])
                label_list.append(judge[ix])

        return np.array(data_list), np.array(label_list)
```

3. Loss 部分具体实现

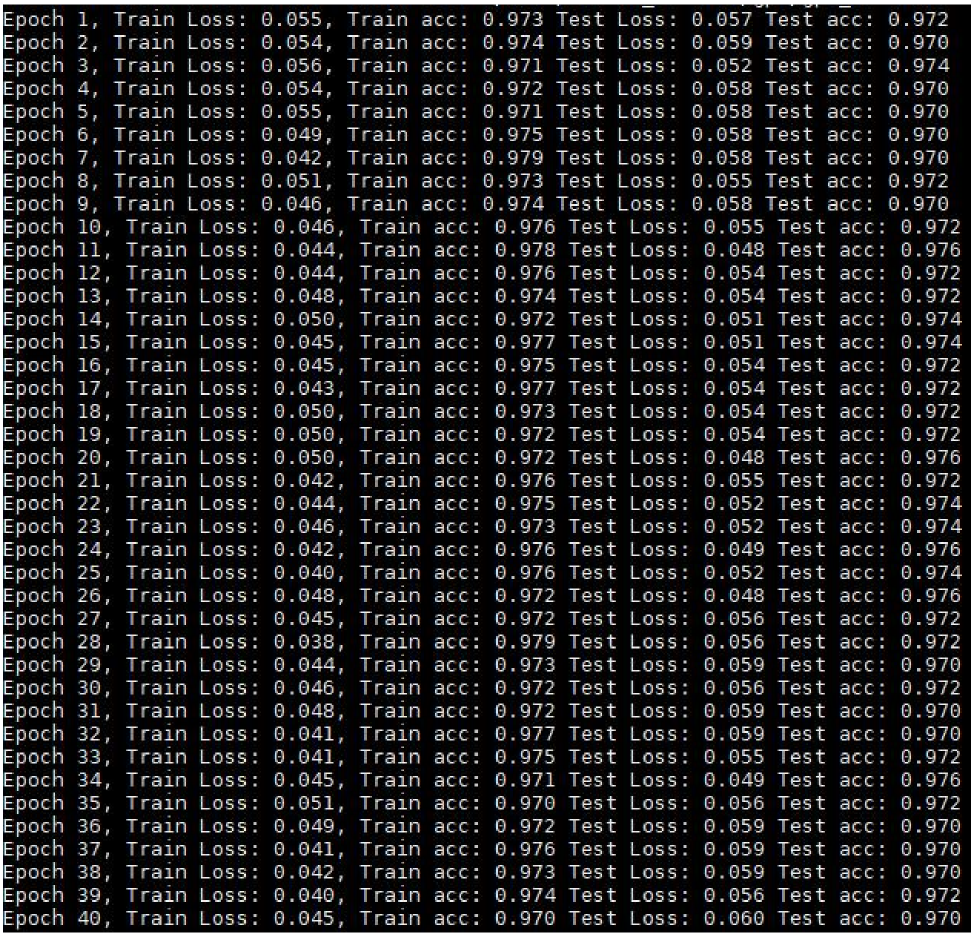
Loss 部分主要有两个方法，一个是计算两个 embedding 的距离，一个是实现要求中的 loss，其中需要说明的是这里的 Q 取 1，由于采样的时候采用的是均衡采样，正负样本比例为 1:1，所以不需要调整 Q。

```

def dist(output1, output2):
        E = K.sqrt(K.sum(K.square(output1 - output2), 1))  # dim=1
        return E

```

```python

def loss_object(Y, E, Q=1):
        pos_loss = Y * 2 * Q * K.exp((-2.77 * E) / Q)
        neg_loss = 2 * (1 - Y) * (E**2) / Q
        return pos_loss + neg_loss
```

4. 训练 log

通过调参 (主要是 batch size 和 learning rate), 模型结果很快能达到 97% 以上。



```
Epoch 1, Train Loss: 0.055, Train acc: 0.973 Test Loss: 0.057 Test acc: 0.972
Epoch 2, Train Loss: 0.054, Train acc: 0.974 Test Loss: 0.059 Test acc: 0.970
Epoch 3, Train Loss: 0.056, Train acc: 0.971 Test Loss: 0.052 Test acc: 0.974
Epoch 4, Train Loss: 0.054, Train acc: 0.972 Test Loss: 0.058 Test acc: 0.970
Epoch 5, Train Loss: 0.055, Train acc: 0.971 Test Loss: 0.058 Test acc: 0.970
Epoch 6, Train Loss: 0.049, Train acc: 0.975 Test Loss: 0.058 Test acc: 0.970
Epoch 7, Train Loss: 0.042, Train acc: 0.979 Test Loss: 0.058 Test acc: 0.970
Epoch 8, Train Loss: 0.051, Train acc: 0.973 Test Loss: 0.055 Test acc: 0.972
Epoch 9, Train Loss: 0.046, Train acc: 0.974 Test Loss: 0.058 Test acc: 0.970
Epoch 10, Train Loss: 0.046, Train acc: 0.976 Test Loss: 0.055 Test acc: 0.972
Epoch 11, Train Loss: 0.044, Train acc: 0.978 Test Loss: 0.048 Test acc: 0.976
Epoch 12, Train Loss: 0.044, Train acc: 0.976 Test Loss: 0.054 Test acc: 0.972
Epoch 13, Train Loss: 0.048, Train acc: 0.974 Test Loss: 0.054 Test acc: 0.972
Epoch 14, Train Loss: 0.050, Train acc: 0.972 Test Loss: 0.051 Test acc: 0.974
Epoch 15, Train Loss: 0.045, Train acc: 0.977 Test Loss: 0.051 Test acc: 0.974
Epoch 16, Train Loss: 0.045, Train acc: 0.975 Test Loss: 0.054 Test acc: 0.972
Epoch 17, Train Loss: 0.043, Train acc: 0.977 Test Loss: 0.054 Test acc: 0.972
Epoch 18, Train Loss: 0.050, Train acc: 0.973 Test Loss: 0.054 Test acc: 0.972
Epoch 19, Train Loss: 0.050, Train acc: 0.972 Test Loss: 0.054 Test acc: 0.972
Epoch 20, Train Loss: 0.050, Train acc: 0.972 Test Loss: 0.048 Test acc: 0.976
Epoch 21, Train Loss: 0.042, Train acc: 0.976 Test Loss: 0.055 Test acc: 0.972
Epoch 22, Train Loss: 0.044, Train acc: 0.975 Test Loss: 0.052 Test acc: 0.974
Epoch 23, Train Loss: 0.046, Train acc: 0.973 Test Loss: 0.052 Test acc: 0.974
Epoch 24, Train Loss: 0.042, Train acc: 0.976 Test Loss: 0.049 Test acc: 0.976
Epoch 25, Train Loss: 0.040, Train acc: 0.976 Test Loss: 0.052 Test acc: 0.974
Epoch 26, Train Loss: 0.048, Train acc: 0.972 Test Loss: 0.048 Test acc: 0.976
Epoch 27, Train Loss: 0.045, Train acc: 0.972 Test Loss: 0.056 Test acc: 0.972
Epoch 28, Train Loss: 0.038, Train acc: 0.979 Test Loss: 0.056 Test acc: 0.972
Epoch 29, Train Loss: 0.044, Train acc: 0.973 Test Loss: 0.059 Test acc: 0.970
Epoch 30, Train Loss: 0.046, Train acc: 0.972 Test Loss: 0.056 Test acc: 0.972
Epoch 31, Train Loss: 0.048, Train acc: 0.972 Test Loss: 0.059 Test acc: 0.970
Epoch 32, Train Loss: 0.041, Train acc: 0.977 Test Loss: 0.059 Test acc: 0.970
Epoch 33, Train Loss: 0.041, Train acc: 0.975 Test Loss: 0.055 Test acc: 0.972
Epoch 34, Train Loss: 0.045, Train acc: 0.971 Test Loss: 0.049 Test acc: 0.976
Epoch 35, Train Loss: 0.051, Train acc: 0.970 Test Loss: 0.056 Test acc: 0.972
Epoch 36, Train Loss: 0.049, Train acc: 0.972 Test Loss: 0.059 Test acc: 0.970
Epoch 37, Train Loss: 0.041, Train acc: 0.976 Test Loss: 0.059 Test acc: 0.970
Epoch 38, Train Loss: 0.042, Train acc: 0.973 Test Loss: 0.059 Test acc: 0.970
Epoch 39, Train Loss: 0.040, Train acc: 0.974 Test Loss: 0.056 Test acc: 0.972
Epoch 40, Train Loss: 0.045, Train acc: 0.970 Test Loss: 0.060 Test acc: 0.970
```

图 6: 训练 log

5. 全部代码

```python
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Model
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Conv2D, Dense, Flatten

tf.keras.backend.set_floatx('float64')
mnist = keras.datasets.mnist

#########################
EPOCHS = 100
BATCH_SIZE = 1000
LEN_IMAGE_SIZE = 784
```

```python

lr = 3e-4
e_w = 1.0
iters = 5
######################

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

x_train = x_train.reshape(x_train.shape[0], -1)
x_test = x_test.reshape(x_test.shape[0], -1)

train_ds = tf.data.Dataset.from_tensor_slices(
(x_train, y_train)).shuffle(1000).batch(BATCH_SIZE)

test_ds = tf.data.Dataset.from_tensor_slices(
(x_test, y_test)).batch(BATCH_SIZE)


def balance_sample(train_ds, test_ds, train=True):
        train_ds = iter(train_ds)
        test_ds = iter(test_ds)
        if train:
                x1, y1 = next(train_ds)
                x2, y2 = next(train_ds)
        else:
                x1, y1 = next(test_ds)
                x2, y2 = next(test_ds)

        y1 = y1[..., np.newaxis]
        y2 = y2[..., np.newaxis]

        idx_same = np.where(y1 == y2) # 找到相同的下角标
        idx_rand = np.random.randint(BATCH_SIZE, size=len(idx_same))  # 随机取样
        index = np.union1d(idx_same, idx_rand).astype(np.int64)  # 所有需要取样的样本

        data_list = []
        label_list = []

        judge = np.array(y1 != y2)

        for ix in index:
                data_list.append([x1[ix], x2[ix]])
                label_list.append(judge[ix])

        return np.array(data_list), np.array(label_list)


class MyModel(Model):
        def __init__(self):
                super(MyModel, self).__init__()
                self.d1 = Dense(500, activation='relu')
                self.d2 = Dense(10, activation='softmax')

        def call(self, x):
                x = self.d1(x)
                embedding = self.d2(x)
                return embedding

```

```
def dist(output1, output2):
    E = K.sqrt(K.sum(K.square(output1 - output2), 1))  # dim=1
    return E


def loss_object(Y, E, Q=1):
    pos_loss = Y * 2 * Q * K.exp((-2.77 * E) / Q)
    neg_loss = 2 * (1 - Y) * (E**2) / Q
    return pos_loss + neg_loss


model = MyModel()

optimizer = tf.keras.optimizers.Adam(lr)

train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.Accuracy(name='train_accuracy')

test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.Accuracy(name='test_accuracy')

def train_epoch(train_ds):
    for i in range(iters):
        with tf.GradientTape() as tape:
            data, label = balance_sample(train_ds, test_ds, train=True)
            output1 = model(data[:, 0])
            output2 = model(data[:, 1])
            E = dist(output1, output2)

            label = np.squeeze(label, 1)

            loss = loss_object(label, E)
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
        E = K.cast(E >= e_w, dtype='float64')
        train_loss(loss)
        train_accuracy(label, E)

def test_epoch(test_ds):
    for i in range(iters):
        data, label = balance_sample(train_ds, test_ds, train=False)
        output1 = model(data[:, 0])
        output2 = model(data[:, 1])
        E = dist(output1, output2)
        loss = loss_object(label, E)
        E = K.cast(E >= e_w, dtype='float64')
        test_loss(loss)
        test_accuracy(label, E)

for epoch in range(EPOCHS):
    train_loss.reset_states()
    train_accuracy.reset_states()
    test_loss.reset_states()
    test_accuracy.reset_states()

    train_epoch(train_ds)
    test_epoch(test_ds)
```

```
            print('Epoch {}, Train Loss: {:.3f}, Train acc: {:.3f}, Test Loss: {:.3f} Test acc: {:.3f}'.
        format(epoch + 1, train_loss.result(), train_accuracy.result(),
            test_loss.result(), test_accuracy.result()))
```