作业四　　连体网络 MNIST 优化

2020 年 3 月 29 日

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

1　实验要求

1. 构建平衡测试集：

（1）正例（同一数字对）、反例（不同数字对）样例比为 1：1。

（2）正例中，10 个数字类型各占 1/10。反例中，不同数字对的所有组合共 $C_{10}^2 = 45$ 种，要求比例也为相同，即反例中，45 种组合每个组合比例为 1/45。测试集正反例总数不少于 9000 个。

（3）写一个测试集打印脚本，打印出构建好的测试集中类型数量信息，例如下：

```
Positive (0,0): 450
Positive (1,1): 450
...
Positive (9,9): 450
Pos Total:4500
Negtive (0,1): 100
Negtive (0,2): 100
...
Negtive (8,9): 100
Neg Total:4500
Total: 9000
```

2. 训练好网络后，在如上定义的测试集上测试精度 Accuracy>0.9

提交内容需要有：代码，文档（运行截图，结果截图（包括 PR 曲线，测试集数量统计打印列表等））

注意：孪生网络的输出是距离特征距离 e_w，训练目的是用给定 loss 调整样本对的距离 e_w, 使得两个同样数字组成的样本对（正样本对）特征距离 e_w 近！两个不同数字组成的样本对（负样本对）的距离 e_w 远！。分类，是特征距离 e_w 孪生网络训练好之后的网络应用，用来判断输入是正样本对，还是负样本对。分类时，采用一个距离阈值 t 作为正、负样本对，$t < e_w$ 时，判定输入是正样本对，$t > e_w$ 时，判定为负样本对。t 的选取可以通过网络在 test 数据集上的 roc 或 pr 曲线获得，一般取曲线的拐点。

2 实验过程

2.1 数据集构建及均衡采样

数据集构建部分代码：

```
(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[..., np.newaxis]
x_test = x_test[..., np.newaxis]

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)
```

均衡采样：

```
def balanced_batch(batch_x, batch_y, num_cls=10):
  batch_x = np.array(batch_x)
  batch_y = np.array(batch_y)
  # batch_x MNIST样本 batch_y, MNIST标签 num_cls
  # （数字类型个数，10，为了让10个数字类型都充分采样正负样本对）
  batch_size = len(batch_y)

  pos_per_cls_e = round(batch_size/2/num_cls/2)  # bs最少40+
  pos_per_cls_e *= 2

  # 根据y进行排序
  index = np.array(batch_y).argsort()
  ys_1 = batch_y[index]

  num_class = []
  pos_samples = []
  neg_samples = set()

  cur_ind = 0

  for item in set(ys_1):
      num_class.append((ys_1 == item).sum())
      # 记录有多少个一样的
      num_pos = pos_per_cls_e

      while(num_pos > num_class[-1]):
          num_pos -= 2
      # 找一个恰好大于num_class个数的值
      # 作为选取的正样本的个数

      pos_samples.extend(np.random.choice(
          index[cur_ind:cur_ind+num_class[-1]],
          num_pos, replace=False).tolist())
      # 正样本

      neg_samples = neg_samples | (
          set(index[cur_ind:cur_ind+num_class[-1]])-set(list(pos_samples)))
      cur_ind += num_class[-1]

```

```
    neg_samples = list(neg_samples)

    x1_index = pos_samples[::2]
    x2_index = pos_samples[1:len(pos_samples)+1:2]

    x1_index.extend(neg_samples[::2])
    x2_index.extend(neg_samples[1:len(neg_samples)+1:2])

    p_index = np.random.permutation(len(x1_index))  # shuffle操作

    x1_index = np.array(x1_index)[p_index]
    x2_index = np.array(x2_index)[p_index]

    r_x1_batch = batch_x[x1_index]
    r_x2_batch = batch_x[x2_index]
    # 得到最终重排后的结果

    r_y_batch = np.array(batch_y[x1_index] !=
                         batch_y[x2_index], dtype=np.float32)

    return r_x1_batch, r_x2_batch, r_y_batch
```

2.2 孪生网络结构

这部分是基于 demo 部分改动，生成 10 维的 embedding。具体代码如下：

```
class TFLeNet(tf.keras.Model):
        def __init__(self, num_classes=10):
                super(TFLeNet, self).__init__(name='TFLeNet')
                self.num_classes = num_classes
                self.conv1 = tf.keras.layers.Conv2D(6, kernel_size=(
                        5, 5), strides=(1, 1), activation='relu', padding='same')
                self.pool1 = tf.keras.layers.MaxPooling2D(2, strides=(2, 2))
                self.conv2 = tf.keras.layers.Conv2D(16, kernel_size=(
                        5, 5), strides=(1, 1), activation='relu', padding='valid')
                self.pool2 = tf.keras.layers.MaxPooling2D(2, strides=(2, 2))
                self.flatten = tf.keras.layers.Flatten()
                self.dense1 = tf.keras.layers.Dense(120, activation='relu')
                self.dense2 = tf.keras.layers.Dense(84, activation='relu')
                self.dense3 = tf.keras.layers.Dense(num_classes, activation='softmax')

        def call(self, inputs):
                x = self.conv1(inputs)
                x = self.pool1(x)
                x = self.conv2(x)
                x = self.pool2(x)
                x = self.flatten(x)
                x = self.dense1(x)
                x = self.dense2(x)
                return self.dense3(x)
```

2.3 参数选取以及训练过程

```
model = TFLeNet()

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(images, labels):
  with tf.GradientTape() as tape:
    data1, data2, label = balanced_batch(images, labels)
    output1 = model(data1)
    output2 = model(data2)
    E = dist(output1, output2)
    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 plot_PR(precision, recall, area, thresholds):
  from matplotlib.ticker import FuncFormatter
  plt.style.use('ggplot')
  plt.plot(recall, precision, 'b.-', label="PR-curve", markersize=1)
  plt.plot(recall[:-1], thresholds, 'r.-', label='threshold', markersize=1)
  plt.xlabel("Recall")
  plt.ylabel("Precision")
  plt.xlim(xmin=0, xmax=1.02)
  plt.ylim(ymin=0, ymax=1.02)
  plt.legend()
  plt.grid(True)
  plt.gca().yaxis.set_major_formatter(FuncFormatter(to_percent))
  plt.gca().xaxis.set_major_formatter(FuncFormatter(to_percent))
  plt.savefig('PR-curve_auc_%.3f.png' % area, dpi=200)
  plt.close()
def test_epoch(images, labels):
  data1, data2, label = balanced_batch(images, labels)
  output1 = model(data1)
  output2 = model(data2)
  E = dist(output1, output2)

precision, recall, _thresholds = precision_recall_curve(label, E)

area = metrics.auc(recall, precision)

plot_PR(precision, recall, area, _thresholds)

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()
```

```
    for images, labels in train_ds:
      train_epoch(images, labels)

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

    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()))
```

参数主要是通过 PR 曲线进行选取的，训练结束后的 PR 曲线如下图：

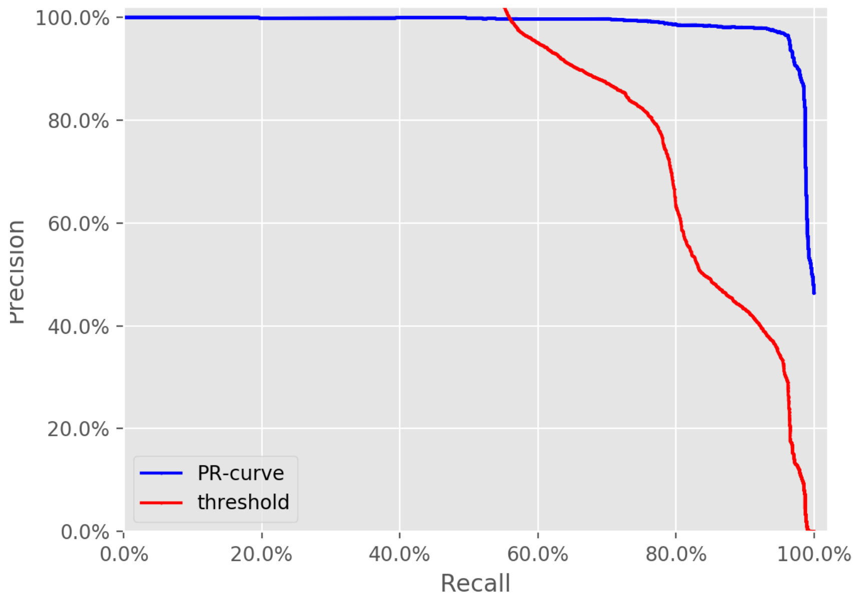

图 1: PR 曲线

红色曲线是 recall 对应的阈值，在 Precision=Recall 的时候对应的阈值大概在 0.4 左右，所以设置 $e_w = 0.4$

训练日志如下：