深度学习方法与实践第七次作业

模型评估与梯度优化

姓名：杨玉雷
学号：18023040

1.连体网络 MINIST 优化

问题描述：

与作业 3"深度学习方法与实践课程 3: 神经网络作业"选做题任务一致，即输入为两个 MNIST 图片，以及两者是否为相同数字的标签(0 为相同数字，1 为不同数字)，输出为网络给出两者是否为同一数字的预测结果。

网络结构可以自己设计。比如两层网络：hidden1：784(28x28)->500; hidden2: 500->10，使用 relu。也可以尝试 Lenet 网络或其他结构。

要求：

（1）构建平衡测试集：正例（同一数字对）、反例（不同数字对）样例比为 1：1。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^2_10=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。

（2）测试集正反例总数不少于 9000 个。(注意，如果要对平衡的测试集有良好的效果，训练的数据集，也应该是平衡的。即我们课上讲的，训练、测试的数据分布要一致。否则，训练的模型是不符合任务需求的。)

（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
```

训练好网络后（ACC>0.9），根据不同正反例分类阈值绘制 P-R 曲线并计算 AP 值。

提交：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））

2.实验过程

实验思路：

（1）获取训练数据和测试数据。使用 getTrainDatas()函数获取训练数据进行训练，每次训练的数据分成 450 的正样本对和 450 的负样本对；使用和 getTestTatas()函数获取测试数据。

（2）计算样本的真实标签。根据样本 y1 和 y2 计算真实标签 label，label 是类似[0,0,0,0,…,1,1,1]的，其中 label[i]==0 表示 x1[i]和 x2[i]为相同数字的图片，label[i]==1 表示 x1[i]和 x2[i]为不同数字的图片。

（3）提取特征。训练数据的网络采用 Lenet 网络结构；样本 x1 和 x2 的特征通过 Lenet 网络提取。

（4）计算 X1 和 X2 特征的欧氏距离。距离越小表示 x1 和 x2 特征越相似，即对应图片上的数字也更相同。

（5）计算预测标签。将距离调整到(0,1)之间,设置 threshold。由此可得到由 0 和 1 组成的预测标签。其中 0 表示样本的距离在(0, threshold)之间，1 表示样本的距离在（threshold，1）之间。

（6）计算准确率。根据样本真实标签和预测标签，计算相同 index 时数量 count 和总的样本数 sum，由 count/sum 即可得到准确率。

（7）计算 loss。根据作业三进阶作业里的 loss 公式计算 loss。

（8）采用 AdamOptimizer 优化器式 loss 降低。

（9）在迭代完一次后，网络更新参数。再重复步骤（1）-（8）直到达到迭代次数或者准确率达到 95%可提前结束训练；

（10）测试。在测试集上测试准确率并画出 pr 曲线、计算 AP 值。

实验代码：

（1）lenet.py

```
class Lenet:
    '''
    threshold:根据距离判断是否是同一数字的阈值
    '''
    def __init__(self, learning_rate, sigma, mu,threshold,Q):
        self.learning_rate = learning_rate
        self.Q = Q
        self.sigma = sigma
        self.mu = mu
        self.threshold=threshold
        # 当 在 创 建 的 时 候 运 行 画 图
        self._build_graph()

    # 涉及网络的所有画图 build graph 过程,常用一个 build graph 封起来
    def _build_graph(self, network_name='Lenet'):
        self._setup_placeholders_graph()
        self._compute_feature_ew()
        self._get_label()
```

```
        self._compute_loss_graph()
        self._compute_y_prediction()
        self._create_train_op_graph()
        self._compute_acc_graph()
        self.merged_summary = tf.summary.merge_all()

    # Lenet 网络的卷积层
    def _cnn_layer(self, scope_name, W_name, b_name, x, filter_shape, conv_strides, b_shape, padding_tag='VALID'):
        with tf.variable_scope(scope_name) as scope:
            conv_weights = tf.get_variable(name=W_name, shape=filter_shape,
                                           initializer=tf.truncated_normal_initializer(mean=self.mu, stddev=self.sigma))
            conv_biases = tf.get_variable(name=b_name, shape=b_shape, initializer=tf.constant_initializer(0.0))
            # 使用边长为 5，深度为 32 的过滤器，过滤器移动的步长为 1，且使用全 0 填充
            conv = tf.nn.conv2d(x, conv_weights, strides=conv_strides, padding=padding_tag)
            return tf.nn.relu(tf.nn.bias_add(conv, conv_biases))

    #Lenet 网络池化层
    def _pooling_layer(self, scope_name, relu, pool_ksize, pool_strides, padding_tag='VALID'):
        with tf.variable_scope(scope_name) as scope:
            return tf.nn.max_pool(relu, ksize=pool_ksize, strides=pool_strides, padding=padding_tag)

    # 将 pool2 拉直
    def _flatten(self, pool2):

        pool_shape = pool2.get_shape().as_list()
        length = pool_shape[1] * pool_shape[2] * pool_shape[3]
        return tf.reshape(pool2, [pool_shape[0], length])

    # 全连接层
    def _fully_connected_layer(self, scope_name, W_name, b_name, x, W_shape, b_shape):
        with tf.variable_scope(scope_name) as scope:
            fc_weights = tf.get_variable(W_name, W_shape,initializer=tf.truncated_normal_initializer(mean=self.mu, stddev=self.sigma))
            fc_biases = tf.get_variable(b_name, b_shape, initializer=tf.constant_initializer(0.0))
            return tf.nn.relu(tf.matmul(x, fc_weights) + fc_biases)
```

```
    def _setup_placeholders_graph(self):
        # self.x= tf.placeholder(tf.float32, (None,28,28,1),name='input_x')
        # self.y= tf.placeholder(tf.int32, (None))  # 在模型中的占位

        # x1 和 x2 中对应的前 4500 是正样本，即 y 相同；后 4500 是负样本，y 不同
        self.x1 = tf.placeholder("float", shape=[None,28,28,1], name='x1')
        self.x2 = tf.placeholder("float", shape=[None,28,28,1], name='x2')
        self.y1 = tf.placeholder("float", shape=[None, 10], name='y1')
        self.y2 = tf.placeholder("float", shape=[None, 10], name='y2')

    #获取图片特征
    def _get_feature(self, scope_name,x):
        with tf.variable_scope(scope_name,reuse=tf.AUTO_REUSE):
            # 第一个卷积层
            # Input = 32x32x1. Output = 28x28x6.
            # 卷积核：[filter_height, filter_width, in_channels, out_channe
ls]
            self.conv1_relu = self._cnn_layer('layer_1_conv', 'conv1_w', 'c
onv1_b',x, (5, 5, 1, 6), [1, 1, 1, 1],[6])

            # 第一个池化层
            # Input = 28x28x6. Output = 14x14x6.
            self.pool1 = self._pooling_layer('layer_2_pooling', self.conv1_
relu, [1, 2, 2, 1], [1, 2, 2, 1])

            # 第二个卷积层
            # Output = 10x10x16.

            self.conv2_relu = self._cnn_layer('layer_3_conv', 'conv2_w', 'c
onv2_b', self.pool1, (5, 5, 6, 16), [1, 1, 1, 1], [16])

            # 第二个池化层
            # Input = 10x10x16. Output = 5x5x16.
            self.pool2 = self._pooling_layer('layer_4_pooling', self.conv2_
relu, [1, 2, 2, 1], [1, 2, 2, 1])

            # Tensor to vector:输入维度由 Nx5x5x16 压平后变为 Nx400
            self.pool2 = flatten(self.pool2)

            # 第一个全连接层
            # Input = 256. Output = 120.
            self.fc1_relu = self._fully_connected_layer('layer_5_fc1', 'fc1
```

```
_w', 'fc1_b', self.pool2, (256, 120), [120])

            # 第二个全连接层
            # Input = 120. Output = 84.
            self.fc2_relu = self._fully_connected_layer('layer_6_fc2', 'fc2
_w', 'fc2_b', self.fc1_relu, (120, 84), [84])

            # 第三个全连接层
            # Input = 84. Output = 10.
            self.fc3_relu = self._fully_connected_layer('layer_7_fc3', 'fc3
_w', 'fc3_b', self.fc2_relu, (84, 10), [10])

            self.digits = self.fc3_relu  # 100*10
            return self.digits

    #计算欧式距离，最后加上一个 1e-6 防止梯度消失
    def _compute_feature_ew(self):
        f1 = self._get_feature('x1_getfeature', self.x1)
        f2 = self._get_feature('x2_getfeature', self.x2)
        self.ew= tf.sqrt(tf.reduce_sum(tf.square(f1 - f2), 1) + 1e-6)

    #计算预测标签值
    def _compute_y_prediction(self):
        with tf.variable_scope('predict_label'):
            #将距离转化在 0 到 1 之间，距离越小越相似
            self.predict_label = self.ew / tf.reduce_max(self.ew)
            ones=tf.ones_like(self.label)
            zeros=tf.zeros_like(self.label)
            #设置距离大于自定义的 threshold 时为 1，小于为 threshold 则为 0，即
距离为 0 的为相同数字的图片
            self.predict_label=tf.where(self.predict_label<self.threshold,x
=zeros,y=ones)

    #计算真实标签值
    def _get_label(self):
        #label:[F,F,F,F,F,F,...,T,T,T,T,T,T],然后转换成[0,0,0,0, ..., 1,1,
1]，即此处标签值为 0 则表示两图片相同，为 1 则不同
        self.label = tf.cast(tf.not_equal(tf.argmax(self.y1, axis=1), tf.ar
gmax(self.y2, axis=1)), dtype=tf.float32)

    #计算 loss
    def _compute_loss_graph(self):
        with tf.name_scope("loss_function"):
```

```
            #采用作业三中的损失函数：
            t1 = (1 - self.label) * (2 / self.Q) * self.ew * self.ew
            t2 = self.label * 2 * self.Q * tf.exp((-2.77)/self.Q * self.ew)
            loss = tf.add(t1, t2)
            self.loss = tf.reduce_mean(loss)
        tf.summary.scalar("loss", self.loss)

    #Adam 优化器
    def _create_train_op_graph(self):
        self.train_op = tf.train.AdamOptimizer(self.learning_rate).minimize
(self.loss)

    #计算准确率
    def _compute_acc_graph(self):
        with tf.name_scope("accuracy"):
            # calculate correct
            #预测 label 和真实 label 相同则预测正确
            temp=tf.subtract(self.predict_label,self.label)
            zeroNum=tf.cast(tf.count_nonzero(temp),dtype=float)
            sum=tf.cast(tf.size(self.label),dtype=float)
            self.accuracy=(sum-zeroNum)/sum
            tf.summary.scalar("accuracy", self.accuracy)
```

（2）train.py

```
mninst = input_data.read_data_sets('MNIST_data/', one_hot=True)

def createDataLib(batch_size):
    #构建样本库
    full = 0  # 样本饱和
    dataLib_x = [[] for i in range(10)]  # 样本库
    # print(np.array(dataLib_x).shape) #(10, 0)
    dataLib_y = [[] for i in range(10)]

    while full != 10:
        if signal=="train":
           temp_x, temp_y = mninst.train.next_batch(batch_size)
        elif signal=="test":
           temp_x, temp_y = mninst.test.next_batch(batch_size)
        for i in range(0, batch_size):  # 将每行数据分类
            classNo = np.argmax(temp_y[i])

            if len(dataLib_x[classNo]) == batch_size:  # 样本库中每个类别数
据存 batch_size 个
```

```
                continue

            if len(dataLib_x[classNo]) == batch_size - 1:
                dataLib_x[classNo].append(temp_x[i])
                dataLib_y[classNo].append(temp_y[i])
                full += 1
                continue

            dataLib_x[classNo].append(temp_x[i])
            dataLib_y[classNo].append(temp_y[i])
    return dataLib_x,dataLib_y

#获取训练数据集
def getTrainDatas(batch_size):
    input_x1 = []
    input_x2 = []
    input_y1 = []
    input_y2 = []

    dataLib_x,dataLib_y=createDataLib(batch_size,"train")
    # 取得正例
    for i in range(0, 10):
        for j in range(0, 450):#每个种类的正样本
            randomNumber1 = random.randint(0, batch_size - 1)
            input_x1.append(dataLib_x[i][randomNumber1]) #i 表示类别
            input_y1.append(dataLib_y[i][randomNumber1])
            randomNumber2 = random.randint(0, batch_size - 1)
            input_x2.append(dataLib_x[i][randomNumber2])
            input_y2.append(dataLib_y[i][randomNumber2])

    # 取得反例
    for i in range(0, 9):
        for j in range(i + 1, 10):
            for k in range(0, 100):#每个种类的负样本 append100 个
                randomNumber1 = random.randint(0, batch_size - 1)
                input_x1.append(dataLib_x[i][randomNumber1])
                input_y1.append(dataLib_y[i][randomNumber1])
                randomNumber2 = random.randint(0, batch_size - 1)
                input_x2.append(dataLib_x[j][randomNumber2])
                input_y2.append(dataLib_y[j][randomNumber2])

    input_x1 = np.array(input_x1).reshape((-1, 28, 28, 1))
    input_x2 = np.array(input_x2).reshape((-1, 28, 28, 1))
    input_y1 = np.array(input_y1).reshape((-1, 10))
```

```
    input_y2 = np.array(input_y2).reshape((-1, 10))

    # print(input_x1.shape)  #(9000, 28, 28, 1)
    return input_x1, input_x2, input_y1, input_y2


#获取测试数据集
def getTestTatas(batch_size):
    # 组装构造例子
        input_x1 = []
        input_x2 = []
        input_y1 = []
        input_y2 = []

        dataLib_x,dataLib_y=createDataLib(batch_size,"test")

        # 取得正例
        for i in range(0, 10):
            for j in range(0, 450):
                randomNumber1 = random.randint(0, batch_size - 1)
                input_x1.append(dataLib_x[i][randomNumber1])
                input_y1.append(dataLib_y[i][randomNumber1])
                randomNumber2 = random.randint(0, batch_size - 1)
                input_x2.append(dataLib_x[i][randomNumber2])
                input_y2.append(dataLib_y[i][randomNumber2])
            print('Positive (%d,%d):' % (i, i), '%d' % (450))
        print('Pos Total:%d' % (4500))

        # 取得反例
        for i in range(0, 9):
            for j in range(i + 1, 10):
                for k in range(0, 100):
                    randomNumber1 = random.randint(0, batch_size - 1) #[0,batch_size - 1]中的一个
                    input_x1.append(dataLib_x[i][randomNumber1])
                    input_y1.append(dataLib_y[i][randomNumber1])
                    randomNumber2 = random.randint(0, batch_size - 1)
                    input_x2.append(dataLib_x[j][randomNumber2])
                    input_y2.append(dataLib_y[j][randomNumber2])
                print('Positive (%d,%d):' % (i, j), '%d' % (100))

        print('Neg Total:%d' % (4500))

        print('Total:%d' % (9000))
```

```
        input_x1 = np.array(input_x1).reshape((-1, 28, 28, 1))
        input_x2 = np.array(input_x2).reshape((-1, 28, 28, 1))
        input_y1 = np.array(input_y1).reshape((-1, 10))
        input_y2 = np.array(input_y2).reshape((-1, 10))

        return input_x1, input_x2, input_y1, input_y2

def main():
    LOGDIR="tensorboard"
    '''
    0.001: learning_rate
    0.1: sigma
    0: mu
    0.2: threshold
    15: Q
    '''
    lenet= Lenet(0.001,0.1,0,0.2,15)
    merged_summary = tf.summary.merge_all()
    with tf.Session() as sess:
        writer = tf.summary.FileWriter(LOGDIR + "/")
        writer.add_graph(sess.graph)
        sess.run(tf.global_variables_initializer(),sess.run(tf.local_variab
les_initializer()))
        #开始训练
        print("Start Training:")
        for i in range(20000):
            input_x1, input_x2, input_y1, input_y2 = getTrainDatas(1024)
            if i % 50 == 0:
                s = sess.run(merged_summary, feed_dict={
                    lenet.x1: input_x1,
                    lenet.x2: input_x2,
                    lenet.y1: input_y1,
                    lenet.y2: input_y2
                })
                writer.add_summary(s, i)
            _, loss, label, acc,ew = sess.run([lenet.train_op,
                                               lenet.loss,
                                               lenet.label,
                                               lenet.accuracy,
                                               lenet.ew,
                                               ],
                                              {
                                                  lenet.x1: input_x1,
```

```
                                        lenet.x2: input_x2,
                                        lenet.y1: input_y1,
                                        lenet.y2: input_y2
                                    })
            precision, recall, _thresholds = metrics.precision_recall_curve
(label, ew)
            auc = metrics.auc(recall, precision)

            if i % 5==0:
              print('setp:%d' % i, 'loss:', loss, ' ', 'auc', auc,"acc",ac
c)
            if acc > 0.95:
                break

        #开始测试
        input_x1, input_x2, input_y1, input_y2 = getTestTatas(1024)
        label, acc,ew, = sess.run([lenet.label,
                                   lenet.accuracy,
                                   lenet.ew],
                                      {
                                          lenet.x1: input_x1,
                                          lenet.x2: input_x2,
                                          lenet.y1: input_y1,
                                          lenet.y2: input_y2
                                      })
        precision, recall, _thresholds = metrics.precision_recall_curve(lab
el, ew)
        print("在测试集上的准确率：",acc)
        auc = metrics.auc(recall, precision)
        print(auc)
        plt.plot(recall, precision)
        plt.xlabel('recall')
        plt.ylabel('precision')
        plt.savefig("pr.jpg")
        plt.show()
        #AP：PR 曲线与 X 轴围成的图形面积
        # AUC：ROC 曲线下面的面积

if __name__ == '__main__':
    main()
```

3. 实验结果

（1）实验中训练数据和测试数据的选取参考了课程网站中讨论区里老师和同学们的讨论内容，在此十分感谢。本实验的训练数据和测试数据在训练次数足够的情况下可以

遍历 mnist 数据集中的所有内容，每一个 step 都是取的 mnist 数据集中 next_batch 里的内容。测试数据输出如图 1 所示。

```
Positive (0,0): 450
Positive (1,1): 450
Positive (2,2): 450
Positive (3,3): 450
Positive (4,4): 450
Positive (5,5): 450
Positive (6,6): 450
Positive (7,7): 450
Positive (8,8): 450
Positive (9,9): 450
Pos Total:4500
Positive (0,1): 100
Positive (0,2): 100
Positive (0,3): 100
Positive (0,4): 100
Positive (0,5): 100
Positive (0,6): 100
Positive (0,7): 100
Positive (0,8): 100
Positive (0,9): 100
Positive (1,2): 100
Positive (1,3): 100
Positive (1,4): 100
Positive (1,5): 100
Positive (1,6): 100
Positive (1,7): 100
Positive (2,8): 100
Positive (2,9): 100
Positive (3,4): 100
Positive (3,5): 100
Positive (3,6): 100
Positive (3,7): 100
Positive (3,8): 100
Positive (3,9): 100
Positive (4,5): 100
Positive (4,6): 100
Positive (4,7): 100
Positive (4,8): 100
Positive (4,9): 100
Positive (5,6): 100
Positive (5,7): 100
Positive (5,8): 100
Positive (5,9): 100
Positive (6,7): 100
Positive (6,8): 100
Positive (6,9): 100
Positive (7,8): 100
Positive (7,9): 100
Positive (8,9): 100
Neg Total:4500
Total:9000
```

图 1 测试数据输出示例

（2）实验采取 Lenet 网络对正负样本提取特征，然后计算距离，通过手动调节阈值来判断距离大于阈值则两个样本不为同一个数字，小于阈值则为同一个数字。最后通过计算样本的真实标签和预测标签来计算准确率。

实验中阈值在 0.3,0.4 左右时准确率最高只能在 0.8 左右，在 0.6-0.9 之间准确率不超过 0.7，但是在 0.2 时迭代 800 次就可达到 0.95 以上，效果还是不错的。

实验过程和结果如图 2 所示。

```
2019-06-27 20:35:12.804171: I T:\src\github\tensorflo
Start Training:
2019-06-27 20:35:17.250438: W T:\src\github\tensorflo
setp:0 loss: 14.4235    auc 0.456728981944 acc 0.5
setp:5 loss: 10.9452    auc 0.479530858212 acc 0.5
setp:10 loss: 7.41263    auc 0.47544229078 acc 0.5
setp:15 loss: 8.13732    auc 0.482939475348 acc 0.5
setp:20 loss: 7.37121    auc 0.503290798387 acc 0.5
setp:25 loss: 7.63126    auc 0.493673102103 acc 0.5
setp:30 loss: 7.36815    auc 0.497894742478 acc 0.5
setp:35 loss: 7.38887    auc 0.517254917677 acc 0.5
setp:40 loss: 7.34128    auc 0.503635777034 acc 0.5
setp:45 loss: 7.32339    auc 0.514852496331 acc 0.5
setp:50 loss: 7.28491    auc 0.528511836755 acc 0.5
setp:55 loss: 7.27237    auc 0.531132775593 acc 0.5
setp:60 loss: 7.22636    auc 0.549576253854 acc 0.5
setp:65 loss: 7.20407    auc 0.562609489573 acc 0.5
setp:70 loss: 7.15628    auc 0.583103391417 acc 0.5
setp:75 loss: 7.1745    auc 0.570627487878 acc 0.5
setp:80 loss: 7.11163    auc 0.598712743447 acc 0.5
setp:85 loss: 7.10288    auc 0.601401277957 acc 0.5
setp:90 loss: 7.06945    auc 0.619038539721 acc 0.5
setp:95 loss: 7.03855    auc 0.639796328048 acc 0.5
setp:100 loss: 6.99283    auc 0.663026730442 acc 0.5
setp:105 loss: 6.98505    auc 0.670112541545 acc 0.5
```

```
setp:110 loss: 6.87244    auc 0.700419978101 acc 0.5
setp:115 loss: 6.8307    auc 0.70664306471 acc 0.5
setp:120 loss: 6.82855    auc 0.705980933411 acc 0.500222
setp:125 loss: 6.75678    auc 0.72271892321 acc 0.500889
setp:130 loss: 6.69733    auc 0.739626852972 acc 0.502778
setp:135 loss: 6.67022    auc 0.750435808461 acc 0.504667
setp:140 loss: 6.57211    auc 0.768003608051 acc 0.506333
setp:145 loss: 6.48616    auc 0.780446919418 acc 0.506444
setp:150 loss: 6.3978    auc 0.7836198597 acc 0.518778
setp:155 loss: 6.21691    auc 0.802118758413 acc 0.525778
setp:160 loss: 6.07129    auc 0.805980519174 acc 0.543333
setp:165 loss: 5.88673    auc 0.816778020762 acc 0.559556
setp:170 loss: 5.7233    auc 0.827398916096 acc 0.583667
setp:175 loss: 5.60459    auc 0.828872334169 acc 0.632556
setp:180 loss: 5.52078    auc 0.834164262518 acc 0.687333
setp:185 loss: 5.41192    auc 0.840835289167 acc 0.723111
setp:190 loss: 5.26971    auc 0.854335454157 acc 0.716333
setp:195 loss: 5.18662    auc 0.858707887646 acc 0.748
setp:200 loss: 5.09392    auc 0.865248585996 acc 0.745556
setp:205 loss: 5.01028    auc 0.87000791148 acc 0.773111
setp:210 loss: 4.8749    auc 0.880871875141 acc 0.769444
setp:215 loss: 4.75713    auc 0.887883895095 acc 0.787778
setp:220 loss: 4.70841    auc 0.889877467059 acc 0.782778
setp:225 loss: 4.60834    auc 0.89650130342 acc 0.798444
```

```
setp:260 loss: 4.08065    auc 0.92457383186 acc 0.818444
setp:265 loss: 4.13452    auc 0.91753646712 acc 0.826556
setp:270 loss: 4.04365    auc 0.923468775304 acc 0.823667
setp:275 loss: 3.977    auc 0.926105906374 acc 0.832778
setp:280 loss: 3.87657    auc 0.932930103365 acc 0.833667
setp:285 loss: 3.77402    auc 0.936557013953 acc 0.850333
setp:290 loss: 3.79565    auc 0.937394978573 acc 0.842333
setp:295 loss: 3.71216    auc 0.938923303223 acc 0.854889
setp:300 loss: 3.72557    auc 0.94076808633 acc 0.842556
setp:305 loss: 3.62429    auc 0.94405720649 acc 0.855556
setp:310 loss: 3.59755    auc 0.944001085222 acc 0.859444
setp:315 loss: 3.62405    auc 0.941248669668 acc 0.86
setp:320 loss: 3.47146    auc 0.948913683275 acc 0.863556
setp:325 loss: 3.47781    auc 0.948997747336 acc 0.863556
setp:330 loss: 3.46513    auc 0.946559199964 acc 0.874556
setp:335 loss: 3.37917    auc 0.953450441988 acc 0.869444
setp:340 loss: 3.29376    auc 0.953748348123 acc 0.879333
setp:345 loss: 3.32287    auc 0.953031553367 acc 0.882778
setp:350 loss: 3.32234    auc 0.954621535098 acc 0.876667
setp:355 loss: 3.27338    auc 0.955896021312 acc 0.88
setp:360 loss: 3.24702    auc 0.955799572751 acc 0.877222
setp:365 loss: 3.19858    auc 0.957951644349 acc 0.887111
setp:370 loss: 3.19524    auc 0.958007650129 acc 0.886222
setp:375 loss: 3.08943    auc 0.961951590888 acc 0.895667
```

```
setp:630 loss: 2.25365    auc 0.983567475973 acc 0.940444
setp:635 loss: 2.29807    auc 0.981645520756 acc 0.935
setp:640 loss: 2.29296    auc 0.98232380144 acc 0.935778
setp:645 loss: 2.29049    auc 0.981846787921 acc 0.938889
setp:650 loss: 2.32381    auc 0.980026191531 acc 0.940222
setp:655 loss: 2.33827    auc 0.980112705489 acc 0.936556
setp:660 loss: 2.28822    auc 0.982078938558 acc 0.94
setp:665 loss: 2.28552    auc 0.981723252536 acc 0.938111
setp:670 loss: 2.23801    auc 0.982710628407 acc 0.938444
setp:675 loss: 2.19915    auc 0.983537813116 acc 0.940444
setp:680 loss: 2.21853    auc 0.982711912612 acc 0.940333
setp:685 loss: 2.23201    auc 0.983142408022 acc 0.941222
setp:690 loss: 2.21134    auc 0.982744419722 acc 0.94
setp:695 loss: 2.2638    auc 0.983207928946 acc 0.934556
setp:700 loss: 2.23725    auc 0.983512959864 acc 0.939889
setp:705 loss: 2.15864    auc 0.985122976717 acc 0.944222
setp:710 loss: 2.11426    auc 0.98566372039 acc 0.947222
setp:715 loss: 2.14321    auc 0.984871418643 acc 0.944222
setp:720 loss: 2.10579    auc 0.985023482184 acc 0.945778
setp:725 loss: 2.04994    auc 0.98747464217 acc 0.947
setp:730 loss: 2.18078    auc 0.983303448618 acc 0.941556
setp:735 loss: 2.15481    auc 0.984973971626 acc 0.943222
setp:740 loss: 2.12652    auc 0.985418244637 acc 0.944222
setp:745 loss: 2.22225    auc 0.982753633581 acc 0.941333
setp:750 loss: 2.02789    auc 0.989035291972 acc 0.950889
```

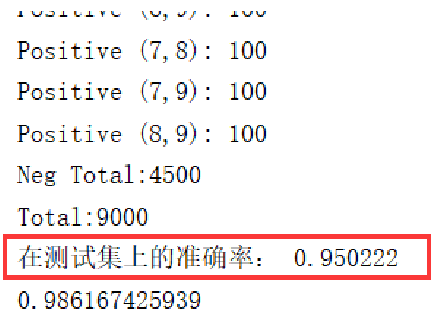
Positive (7,8): 100
Positive (7,9): 100
Positive (8,9): 100
Neg Total:4500
Total:9000
在测试集上的准确率： 0.950222
0.986167425939

图 2 训练过程和测试准确率

准确率和 loss 变化曲线如图 3 所示。

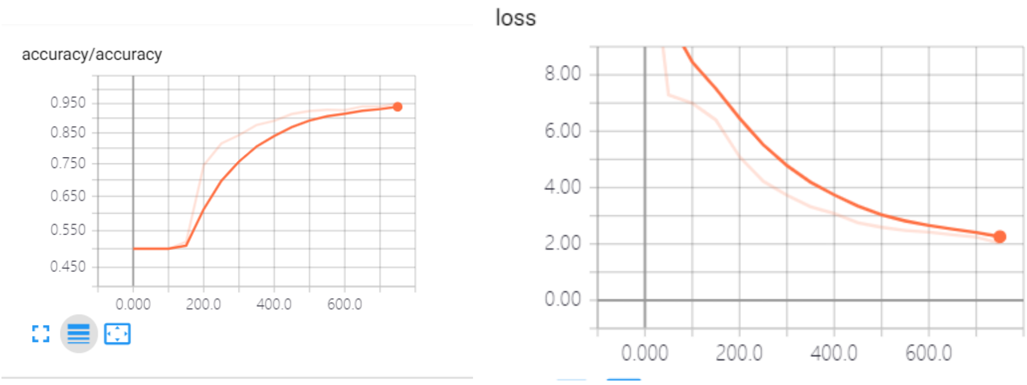

图 3 accuracy 和 loss 变化曲线

（3）pr 曲线和 AP 值

pr 曲线表示训练过程中准确率随召回率变化的曲线。pr 曲线如图 4 所示。

AP 值是 PR 曲线与 X 轴围成的图形面积。AP 值输出如图 5 所示。可见 AP 值可大 0.98，效果还是不错的。

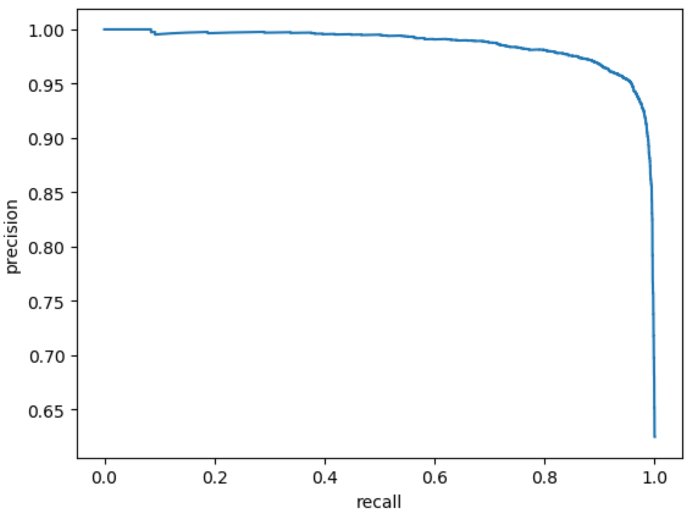

图 4 pr 曲线

```
Positive (6,8): 100
Positive (6,9): 100
Positive (7,8): 100
Positive (7,9): 100
Positive (8,9): 100
Neg Total:4500
Total:9000
在测试集上的准确率： 0.950222
0.986167425939
```

图 5 AP 值