```
import pandas as pd
import tensorflow as tf

INPUT_NODE = 256     # 输入节点
OUTPUT_NODE = 10     # 输出节点
LAYER1_NODE = 200    # 隐藏层节点数

BATCH_SIZE = 50     # 每次batch打包的样本个数

# 模型相关的参数
LEARNING_RATE_BASE = 0.8    #基础的学习率
LEARNING_RATE_DECAY = 0.99    #学习的衰减率
REGULARAZTION_RATE = 0.0001   #正则化项在损失函数中的系数
TRAINING_STEPS = 3000        #训练轮数
MOVING_AVERAGE_DECAY = 0.99    #滑动平均衰减率
START_POINT = 0
```

```
def inference(input_tensor, avg_class, weights1, biases1, weights2, biases2):
    # 不使用滑动平均类
    if avg_class == None:
        layer1 = tf.nn.relu(tf.matmul(input_tensor, weights1) + biases1)
        return tf.matmul(layer1, weights2) + biases2

    else:
        # 使用滑动平均类
        layer1 = tf.nn.relu(tf.matmul(input_tensor, avg_class.average(weights1)) +
 avg_class.average(biases1))
        return tf.matmul(layer1, avg_class.average(weights2)) + avg_class.average(
biases2)
```

```
def selectSample(data):
    global START_POINT
    #dataSample = data.sample(BATCH_SIZE).reset_index(drop=True)
    dataSample = data.iloc[START_POINT:min(data.shape[0],(START_POINT+BATCH_SIZE))
].reset_index(drop=True)
    START_POINT = (START_POINT + BATCH_SIZE)%data.shape[0]
    xSample = dataSample.iloc[:,0:256]
    ySample = dataSample.iloc[:,256:266]
    return xSample, ySample
```

```
def train(data, xIVali, yI_Vali):
    x = tf.placeholder(tf.float32, [None, INPUT_NODE], name='x-input')
    y_ = tf.placeholder(tf.float32, [None, OUTPUT_NODE], name='y-input')
    # 生成隐藏层的参数。
    weights1 = tf.Variable(tf.truncated_normal([INPUT_NODE, LAYER1_NODE], stddev=0.1))
    biases1 = tf.Variable(tf.constant(0.1, shape=[LAYER1_NODE]))
    # 生成输出层的参数。
    weights2 = tf.Variable(tf.truncated_normal([LAYER1_NODE, OUTPUT_NODE], stddev=0.1))
    biases2 = tf.Variable(tf.constant(0.1, shape=[OUTPUT_NODE]))

    # 计算不含滑动平均类的前向传播结果
    y = inference(x, None, weights1, biases1, weights2, biases2)

    # 定义训练轮数及相关的滑动平均类
    global_step = tf.Variable(0, trainable=False)
    variable_averages = tf.train.ExponentialMovingAverage(MOVING_AVERAGE_DECAY, global_step)
    variables_averages_op = variable_averages.apply(tf.trainable_variables())
    average_y = inference(x, variable_averages, weights1, biases1, weights2, biases2)

    # 计算交叉熵及其平均值
    cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y, labels=tf.argmax(y_, 1))
    cross_entropy_mean = tf.reduce_mean(cross_entropy)

    # 正则化损失函数的计算
    regularizer = tf.contrib.layers.l2_regularizer(REGULARAZTION_RATE)
    regularaztion = regularizer(weights1) + regularizer(weights2)
    loss = cross_entropy_mean + regularaztion

    # 设置指数衰减的学习率。
    learning_rate = tf.train.exponential_decay(
        LEARNING_RATE_BASE,
        global_step,
        data.shape[0] / BATCH_SIZE,
        LEARNING_RATE_DECAY,
        staircase=True)

    # 优化损失函数
    train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss, global_step=global_step)

    # 反向传播更新参数和更新每一个参数的滑动平均值
    with tf.control_dependencies([train_step, variables_averages_op]):
        train_op = tf.no_op(name='train')

    # 计算正确率
    correct_prediction = tf.equal(tf.argmax(average_y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    # 初始化会话，并开始训练过程。
    with tf.Session() as sess:
        tf.global_variables_initializer().run()
        validate_feed = {x: xIVali.head(150), y_: yI_Vali.head(150)}
        test_feed = {x: xIVali.tail(150), y_: yI_Vali.tail(150)}

        # 循环的训练神经网络。
        for i in range(TRAINING_STEPS):
            if i % 100 == 0:
                validate_acc = sess.run(accuracy, feed_dict=validate_feed)
                print("After %d training step(s), validation accuracy using averag
e model is %f " % (i, validate_acc))

            xs,ys=selectSample(data)
            sess.run(train_op,feed_dict={x:xs,y_:ys})

        test_acc=sess.run(accuracy,feed_dict=test_feed)
        print(("After %d training step(s), test accuracy using average model is %f
" %(TRAINING_STEPS, test_acc)))
```

```
def importData():
    data = pd.read_csv('./semeion.data', sep=' ', header=None)
    dataVali = data.tail(200)
    data.drop(data.tail(200).index,inplace=True)

    x = data.iloc[:,0:256]
    y_ = data.iloc[:,256:266]

    xVali = dataVali.iloc[:,0:256]
    y_Vali = dataVali.iloc[:,256:266]
    return data, xVali, y_Vali
```

```
def main(argv=None):
    data, xVali, y_Vali = importData();
    train(data, xVali, y_Vali)


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

```
After 0 training step(s), validation accuracy using average model is 0.146667
After 100 training step(s), validation accuracy using average model is 0.793333
After 200 training step(s), validation accuracy using average model is 0.200000
After 300 training step(s), validation accuracy using average model is 0.566667
After 400 training step(s), validation accuracy using average model is 0.306667
After 500 training step(s), validation accuracy using average model is 0.453333
After 600 training step(s), validation accuracy using average model is 0.540000
After 700 training step(s), validation accuracy using average model is 0.593333
After 800 training step(s), validation accuracy using average model is 0.833333
After 900 training step(s), validation accuracy using average model is 0.853333
After 1000 training step(s), validation accuracy using average model is 0.880000
After 1100 training step(s), validation accuracy using average model is 0.886667
After 1200 training step(s), validation accuracy using average model is 0.886667
After 1300 training step(s), validation accuracy using average model is 0.880000
After 1400 training step(s), validation accuracy using average model is 0.880000
After 1500 training step(s), validation accuracy using average model is 0.886667
After 1600 training step(s), validation accuracy using average model is 0.880000
After 1700 training step(s), validation accuracy using average model is 0.886667
After 1800 training step(s), validation accuracy using average model is 0.886667
After 1900 training step(s), validation accuracy using average model is 0.886667
After 2000 training step(s), validation accuracy using average model is 0.886667
After 2100 training step(s), validation accuracy using average model is 0.886667
After 2200 training step(s), validation accuracy using average model is 0.886667
After 2300 training step(s), validation accuracy using average model is 0.880000
After 2400 training step(s), validation accuracy using average model is 0.880000
After 2500 training step(s), validation accuracy using average model is 0.880000
After 2600 training step(s), validation accuracy using average model is 0.886667
After 2700 training step(s), validation accuracy using average model is 0.886667
After 2800 training step(s), validation accuracy using average model is 0.886667
After 2900 training step(s), validation accuracy using average model is 0.886667
After 3000 training step(s), test accuracy using average model is 0.873333
```