基于手机动作传感器的触屏位置学习

分工：
许思学：程序采集
席子义：机器学习
鲍习源：整理答辩与报告

手机传感器在日常使用中非常重要，比如赛车游戏和步数记录就无法离开它们，然而这背后暗藏隐患。

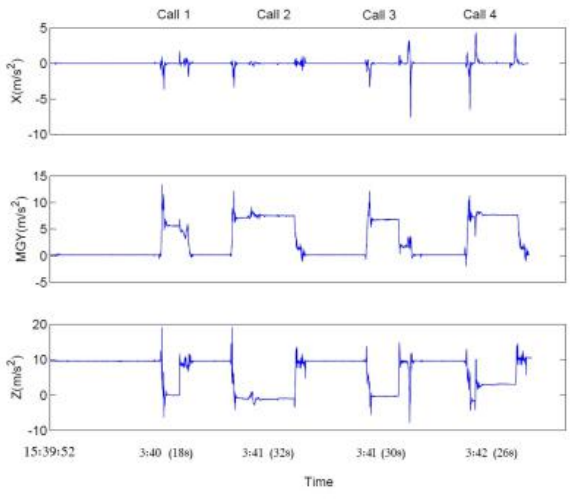
1.前人工作

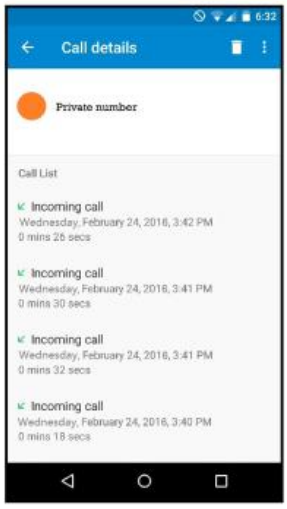
Stealing PINs via Mobile Sensors:
Actual Risk versus User Perception

Maryam Mehrnezhad, Ehsan Toreini, Siamak F. Shahandashti, Feng Hao

School of Computing Science, Newcastle University, Newcastle upon Tyne, UK

2017 年 4 月新堡大学研究人员在《国际资讯安全杂志》发布了论文，示范了用陀螺仪——追踪手腕旋转和方向的感测器，高精度猜测四位数 PIN 码，并指出只有少数传感器（GPS、相机）在使用时征询许可，APP 和网站可以自由监听用户传感器数据。原理大致是特定使用者在特定场景，按屏幕上数字按钮->手机微小晃动和角度变化->传感器上波形不同，下图是不同拨号事件对应加速度计的不同波形：





作者使用 JS 编写网页后台监听传感器，用 HTML5 编写 GUI 显示随机 PIN 码让用户输入，并考虑多种输入姿势。他们使用四传感器十二分量序列做特征提取，考虑序列最大、最小、平均值、能量，同时考虑时间、频率域（FFT）以及时间域不同方向相关各传感器相关系数，每个时间序列得到 114 特征，然后让十位用户输入了 2500 个 pin（12 个错误），进一步传入用 matlab 写的一个隐藏层 1000 节点的 ANN 网络训练，最后单次识别成功率不到 80%，多次尝试成功率增长，三次大于 90&，如下图：

Attempts	Multiple-users	Same-user
One	74%	79%
Two	86%	93%
Three	94%	97%

Table 1: PINlogger.js's PIN identification rates in different attempts.

而我们想改进的部分，是跳出数字按键的限制，直接学习触摸位置坐标，不但能识别 PIN 码，还可以进一步推测复杂密码，乃至于监视用户所有触屏行为，而且坐标不存在错误输入。

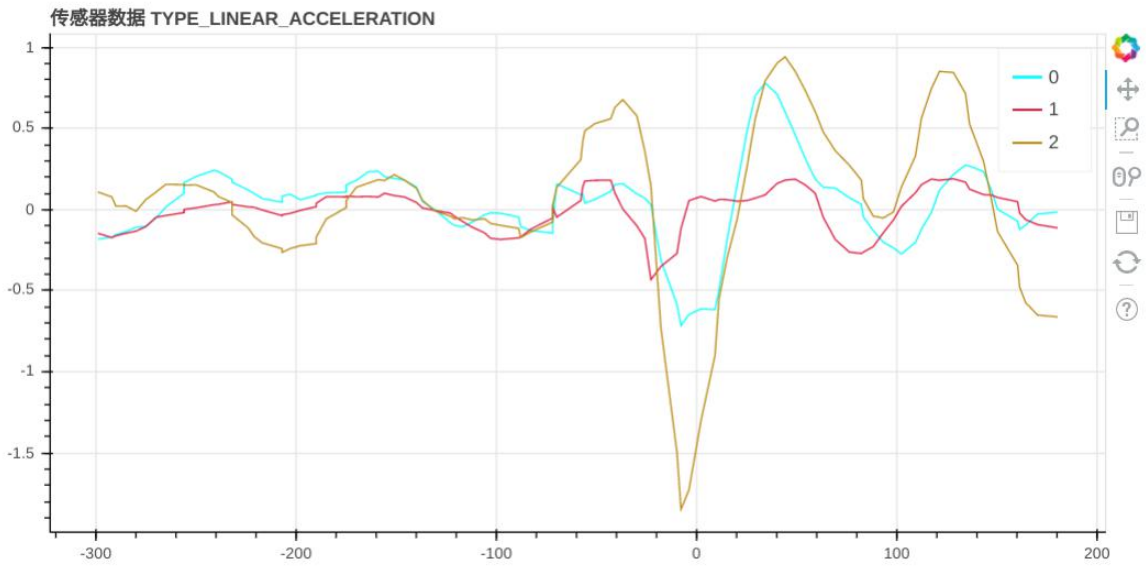
研究该问题对保障用户信息安全意义重大。

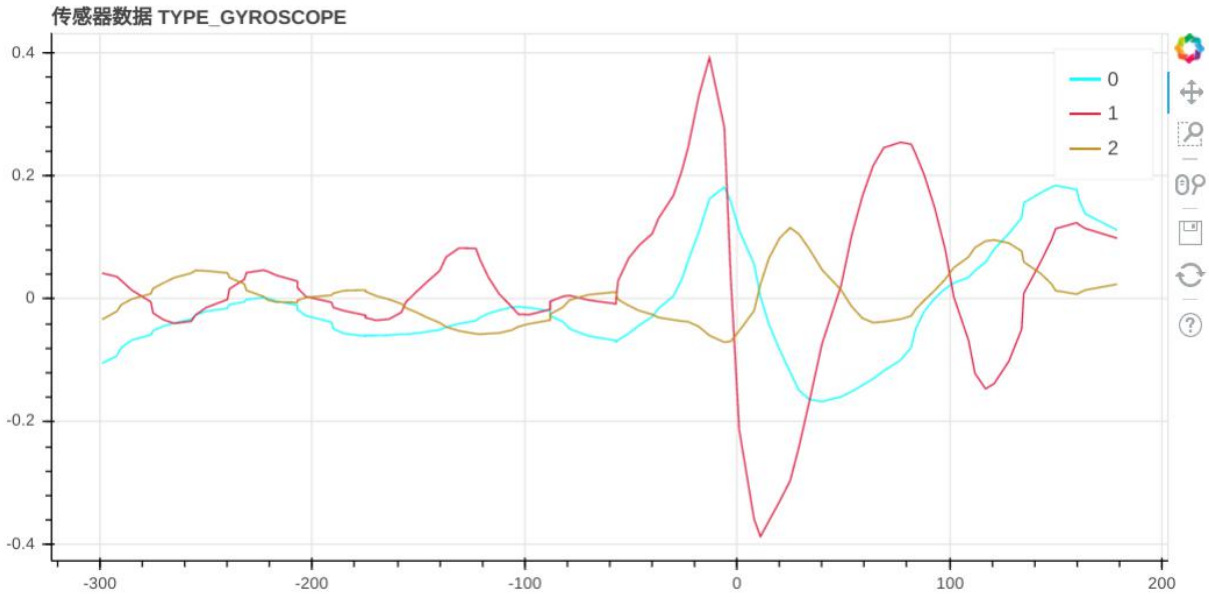
2.采集与学习

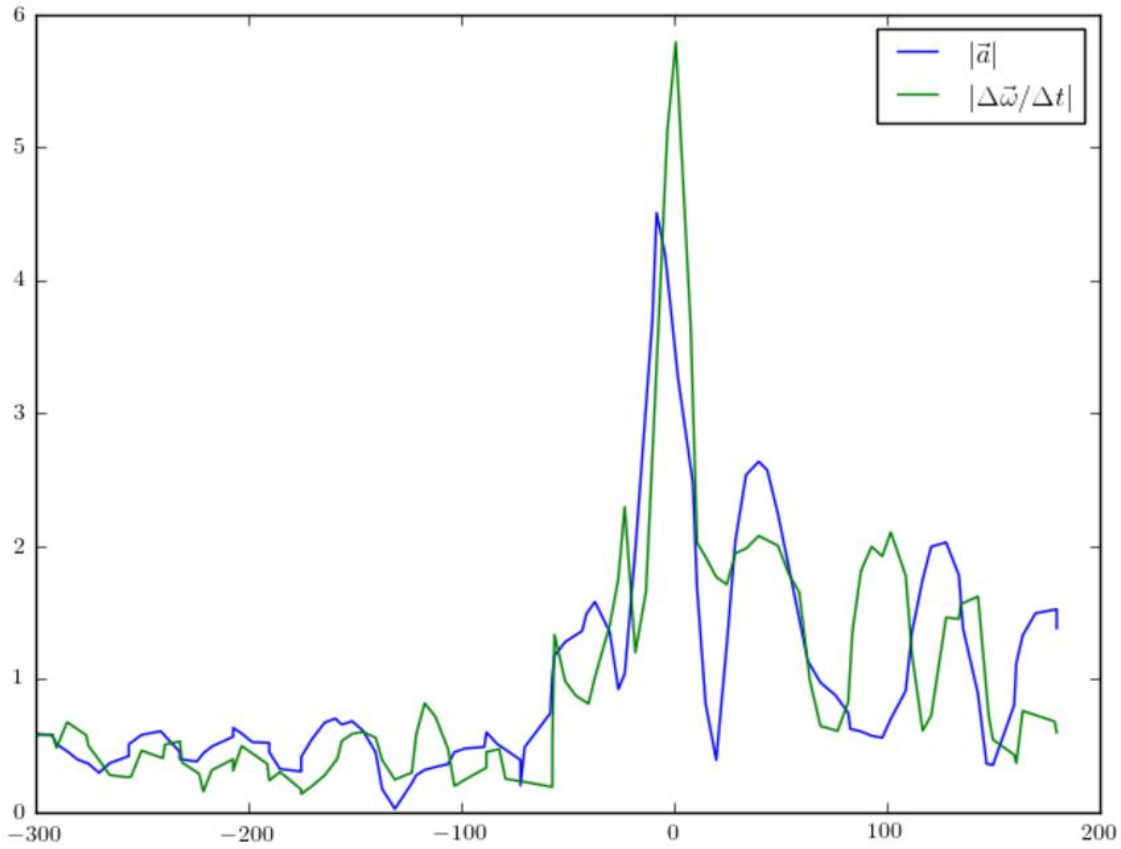
分四个步骤：编写采集程序，采集数据，机器学习和输入测试。

1)采集程序

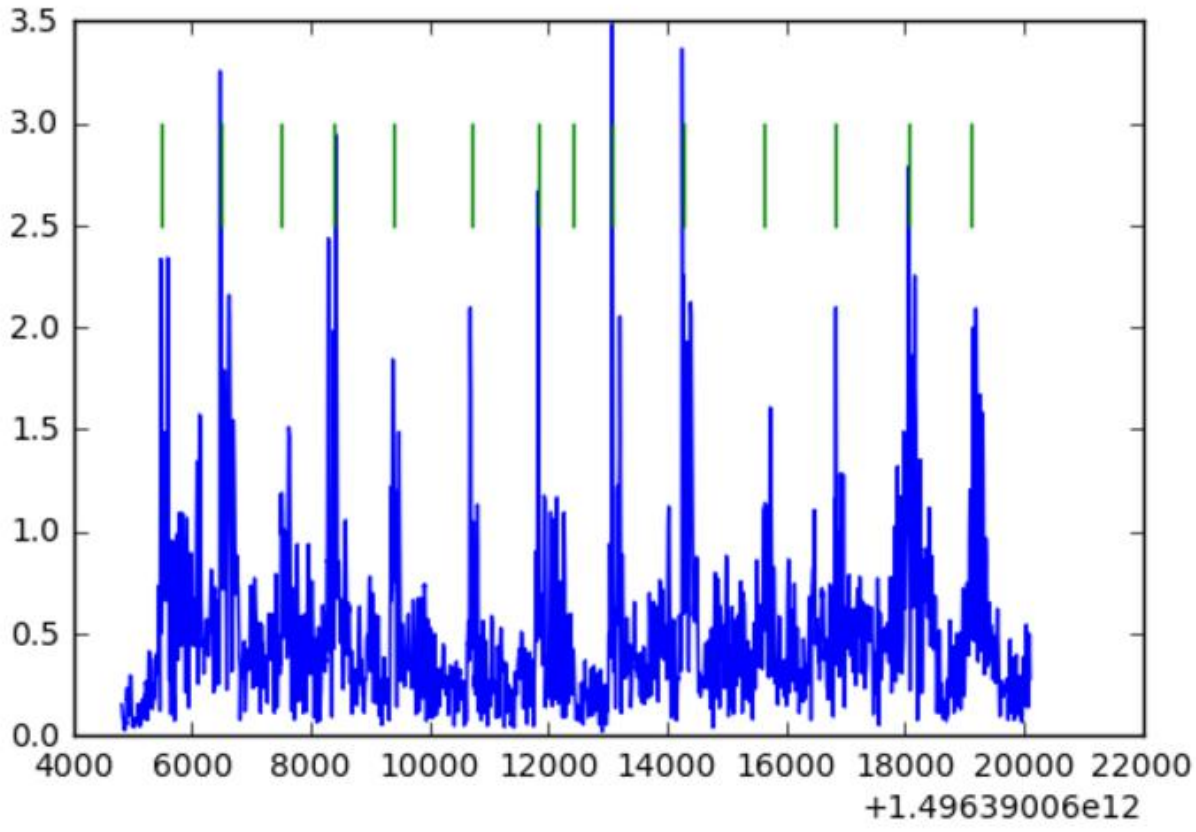
先分析一个典型触击事件，这将涉及到质心加速与旋转。下图中已将某次事件两种传感器波形时间轴零点平移到触击时刻。我们发现，触击时刻加速度最大，并且角速度变化最大，故同时考虑这两个量大于一定阈值时，即可判断发生一次触击，并推测出对应的触击时刻。


传感器数据 TYPE_LINEAR_ACCELERATION
1
0.5
0
-0.5
-1
-1.5
-300
-200
-100
0
100
200
0
1
2


传感器数据 TYPE_GYROSCOPE
0.4
0.2
0
-0.2
-0.4
-300
-200
-100
0
100
200
0
1
2



按照这种方法推测的触击时间（下图绿线）与实际波形最大值（蓝线）符合的非常好。

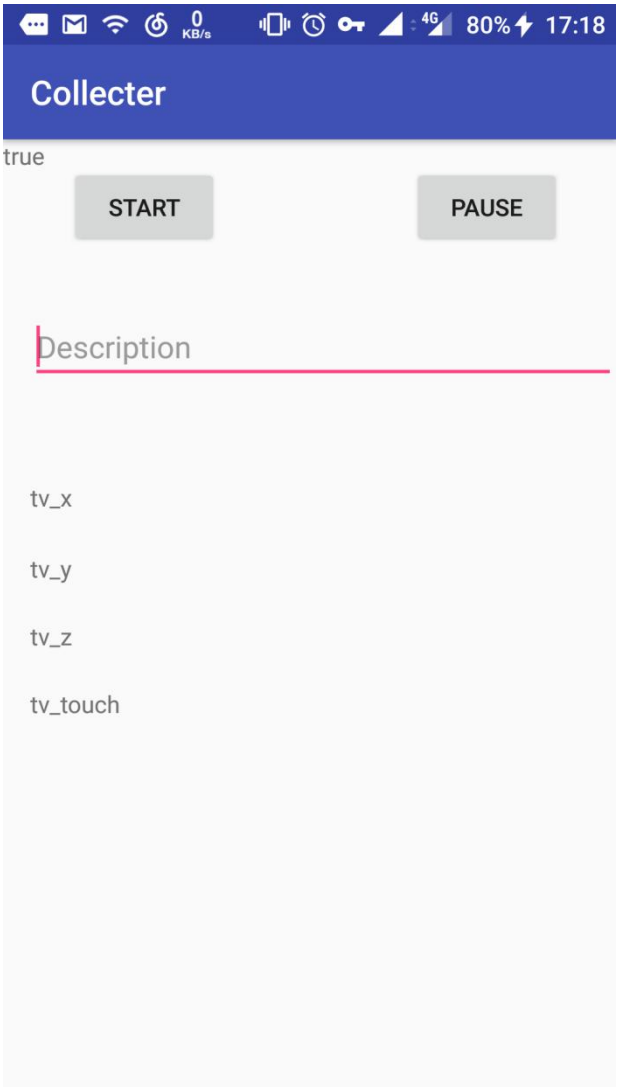


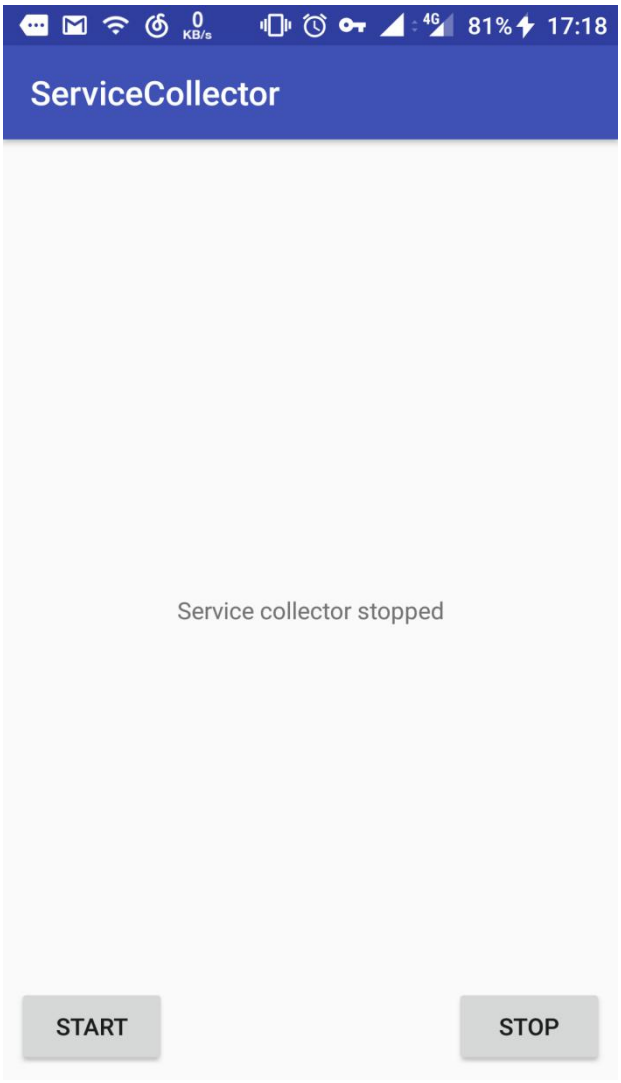
经过尝试，我们每个事件截取触击时刻前后 250ms。

选择传感器采用了下表四传感器十三分量：

Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity).	m/s^2
	SensorEvent.values[1]	Acceleration force along the y axis (including gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	
TYPE_GYROSCOPE	SensorEvent.values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation around the y axis.	
	SensorEvent.values[2]	Rate of rotation around the z axis.	
TYPE_LINEAR_ACCELERATION	SensorEvent.values[0]	Acceleration force along the x axis (excluding gravity).	m/s^2
	SensorEvent.values[1]	Acceleration force along the y axis (excluding gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (excluding gravity).	
TYPE_ROTATION_VECTOR	SensorEvent.values[0]	Rotation vector component along the x axis (x * sin(θ/2)).	Unitless
	SensorEvent.values[1]	Rotation vector component along the y axis (y * sin(θ/2)).	
	SensorEvent.values[2]	Rotation vector component along the z axis (z * sin(θ/2)).	
	SensorEvent.values[3]	Scalar component of the rotation vector ((cos(θ/2)).[1]	

我们用 java 编写了前台和后台传感器信息采集程序，分别如下图左右，START 按钮开始数据收集，PAUSE 按钮结束数据收集并写入数据文件，屏幕显示触碰坐标，左上角为 0,0。前台程序还可以添加描述。





我们采集的程序代码结构如下图：

主体代码

```
app/src/main
├── AndroidManifest.xml
├── java
│   └── com
│       └── example
│           └── hzxusx
│               └── collecter
│                   ├── CollectingActivity.java
│                   └── SensorBuffer.java
└── res
    ├── drawable
    ├── layout
    │   ├── activity_collecting.xml
```

时间数据结构如下图：

```
// 第 tap_index 个触碰事件
object[<tap_index>][0].tap_t // 触碰时刻，long
object[<tap_index>][0].tap_x // 触碰坐标x，tap_y 坐标y
object[<tap_index>][1-4].sensor //传感器类型名，String，"TYPE_ACCELEROMETER","TYPE_LINEAR_ACCELERATI
object[<tap_index>][1-4].timestamp //采集时间点，long [采集次数]
object[<tap_index>][1-4].data //采集的数据，float [采集次数][单次采集数据长度]
```

2）采集过程

理论上要对不同用户不同使用姿势/场景分别采集，但时间与计算资源有限，只采集一人右手触屏数据。我们尽量让触摸均匀分布在整个键盘区域，最后有 5000 次有效触击，每个事件包含触屏的两个坐标点以及共 13 组陀螺仪数据。

3）机器学习

基于 python3 完成，首先用 myinitprocess 函数把波形数据和 label 数据相分离：

```
def myinitprocess(filename):
    obj = {}
    with open(filename,"r") as f:
        obj = json.load(f)

    def add0(mylist):
        result=np.zeros(100)
        nnumber=np.shape(mylist)[0]
        if(nnumber<=100):
            result[:nnumber]=mylist
        else:
            result=mylist[:100]
        return result

    myinput=[[] for i in range(len(obj))]
    mylabel=np.zeros((len(obj),2))
    for i in range(len(obj)):
        for j in range(4):
            for k in range(3):
                myinput[i].append(np.array((add0(np.array(obj[i][j+1]['data'])[:,k])[start:])))
        myinput[i].append(np.array((add0(np.array(obj[i][4]['data'])[:,3])[start:])))
        mylabel[i][0]=obj[i][0]['tap_x']
        mylabel[i][1]=obj[i][0]['tap_y']
    myinput=np.array(myinput)

    def myprocess(mydata):
        fd=np.shape(mydata)[0]
        myresult=np.zeros_like(mydata)
        for i in range(13):
            temp=np.zeros(fd*(100-start))
    #         print(np.shape(temp))
            for j in range(fd):
                temp[j*(100-start):(j+1)*(100-start)]=mydata[j][i]
            preprocessing.scale(temp)
            for j in range(fd):
                myresult[j][i]=temp[j*(100-start):(j+1)*(100-start)]
        mydata=myresult
    myprocess(myinput)
    return myinput,mylabel
```

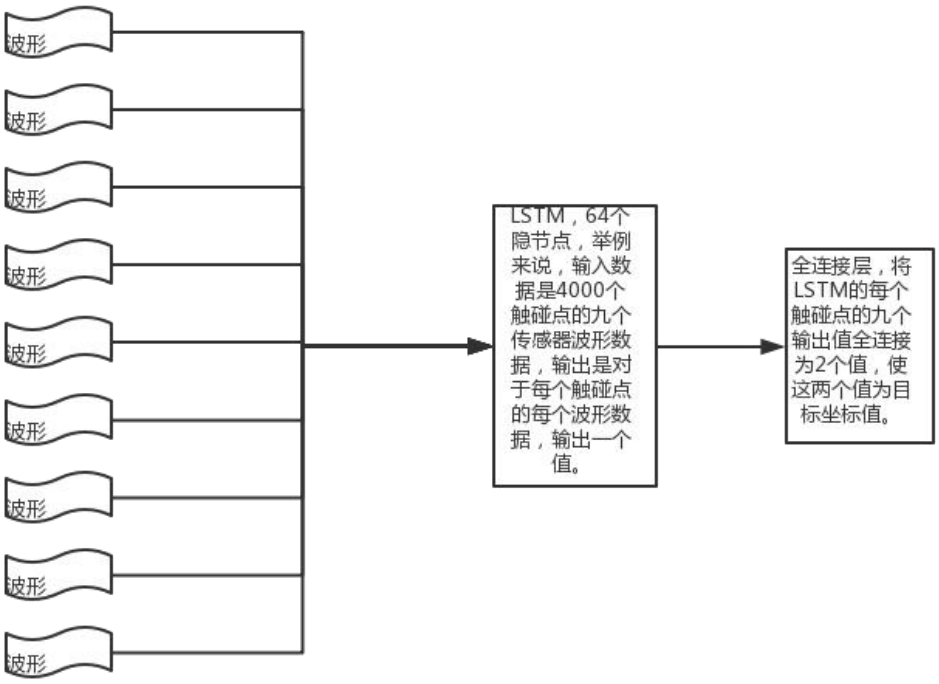
然后 getall 函数定义了从得到的数据进行预处理并划分训练集和测试集的过程。 数据预处理主要考虑到不同传感器信号强弱可能有差异，为了均衡考虑，就把各组波形数据统一计算其均值与方差，然后对每组中的单个事件数据减去均值除以方差以达到归一化的效果。这里有三个参数，filedir 是输入数据目录地址，cutplace 是训练集的百分比，crosslabel 是是否进行随机打乱数据。这里我的训练集取 0.8 的比例，测试集取 0.2 的比例，故 cutplace=0.2。另外还有一个十个数据点的未知位置数据等着预测，这个数据集取 cutplace=1.0，crosslabel=False。

```
def getall(filedir,cutplace,crosslabel):
    feathers=[]
    labels=[]
    jsonFiles = list(filter(lambda name:name.endswith(".json"), os.listdir(filedir)))
    print(jsonFiles)
    for item in jsonFiles:
        temp1,temp2=myinitprocess(filedir+item)
        feathers.append(temp1)
        labels.append(temp2)
    temp3,temp4=np.concatenate(tuple(feathers)),np.concatenate(tuple(labels))
    train=np.zeros((np.shape(temp3)[0],100-start,13))
    trainlabel=np.zeros((np.shape(temp3)[0],2))
    for i in range((np.shape(temp3)[0])):
        train[i]=temp3[i].T
        trainlabel[i]=temp4[i]
    test=0;testlabel=0
    if(crosslabel==True):
        train,test,trainlabel,testlabel=cross_validation.train_test_split(train,trainlabel,test_size=cutplace, random_state=
    return train,test,trainlabel,testlabel
```

下面是网络搭建和训练过程，训练用笔记本自带的 840M，X 和 netlabel 是输入训练数据与 label 的占位符，之后 tflearn.lstm 搭建 lstm 网络，其中使用的激活函数等参量都采用默认：

```
scope=None, name='LSTM')
for_pipe=1.0, return_seq=False, return_state=False, initial_state=None, dynamic=False, trainable=True, restore=True, reuse=False
tflearn.layers.recurrent.lstm (incoming, n_units, activation='tanh', inner_activation='sigmoid', dropout=None, bias=True, weights_init=None,
```

lstm 的 dropout 比例设置为 0.7 避免过拟合,return_seq=False 使得每个事件的每个波形返回一个值（64node），然后用 tflearn.fully_connected 制定全连接层，使得每个事件最后输出 xy，将其与实际的 label（坐标）进行取均方误差最小，然后开始训练，结构如下：



训练代码如下：

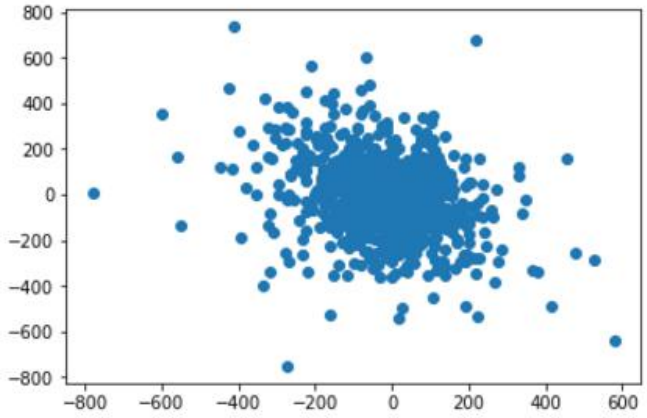
```
with tf.Graph().as_default():
#     save_path="modelfinal1"
    X=tflearn.input_data(shape=(None,100-start,9),dtype=tf.float32)
#     X = tf.placeholder(shape=(None,100-start,9),dtype=tf.float32)
    netlabel=tflearn.input_data(shape=(None,2),dtype=tf.float32)
#     netlabel=tf.placeholder(shape=(None,2),dtype=tf.float32)
    net = tflearn.lstm(X, 64, dropout=0.7,return_seq=False)
    net = tflearn.fully_connected(net, 2, activation='relu')
    loss = tf.reduce_mean(tf.pow(tf.subtract(net,netlabel),2))
    optimizer = tf.train.AdamOptimizer(learning_rate=0.01).minimize(loss)
    model = tflearn.DNN(net, tensorboard_verbose=3)
    init = tf.global_variables_initializer()
    with tf.Session() as sess:
        sess.run(init)
        cost=1
        qq=0
        while(qq<1000):
            sess.run(optimizer, feed_dict={X:train,netlabel:trainlabel/2000})
            cost = sess.run(loss, feed_dict={X:train, netlabel:trainlabel/2000})
            if(qq%40==0):
                a=sess.run(net, feed_dict={X:test})*2000-testlabel
                countx=0
                county=0
                for i in a:
                    if(abs(i[0])<150):
                        countx+=1
                    if(abs(i[1])<150):
                        county+=1
                print('xaccuracx:{:.6f}'.format(countx/np.shape(test)[0]),'yaccuracy:{:.6f}'.format(county/np.shape(test)[0]),
            qq+=1
        a=sess.run(net, feed_dict={X:test})*2000-testlabel
        b=sess.run(net,feed_dict={X:xsx1})*2000
```

其中 xaccuracy 和 yaccuracy 是阶段性训练好网络，作用在此时的测试集上的坐标准确度，cost 就是 loss ，随着 cost 减小，xy 准确度分别有 80%、70%，而评价标准按照半个手指对应的各坐标轴正负 150px 误差，以内为成功。最终训练结果：

```
countx=0
county=0
for i in a:
    if(abs(i[0])<150):
        countx+=1
    if(abs(i[1])<150):
        county+=1
print('x:',countx/np.shape(test)[0])
print('y:',county/np.shape(test)[0])
```

```
x: 0.8014184397163121
y: 0.7223910840932117
```

误差分布集中在原点附近：



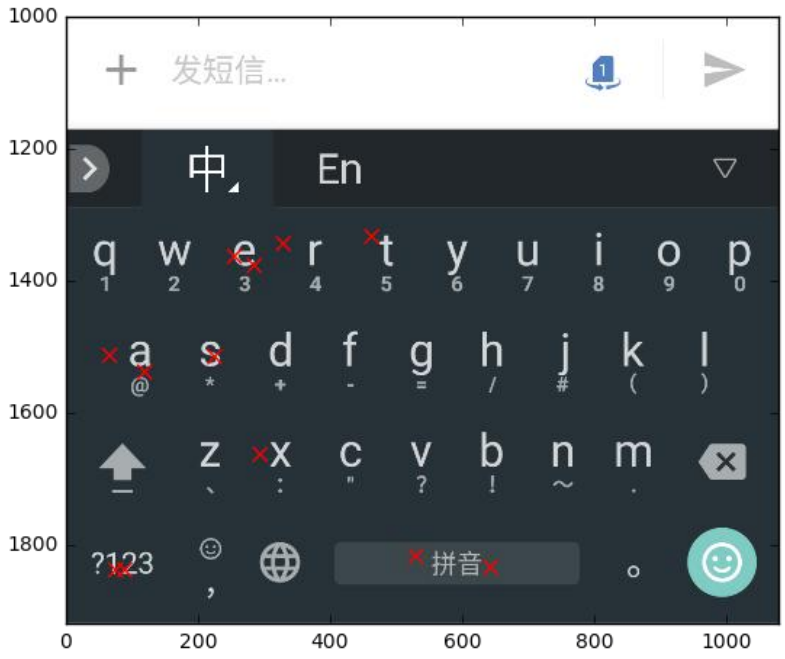
计算误差平均，发现绝对值很小，说明没有发生大的系统偏差：

```
print(np.average(a[:,0]))
print(np.average(a[:,1]))
```

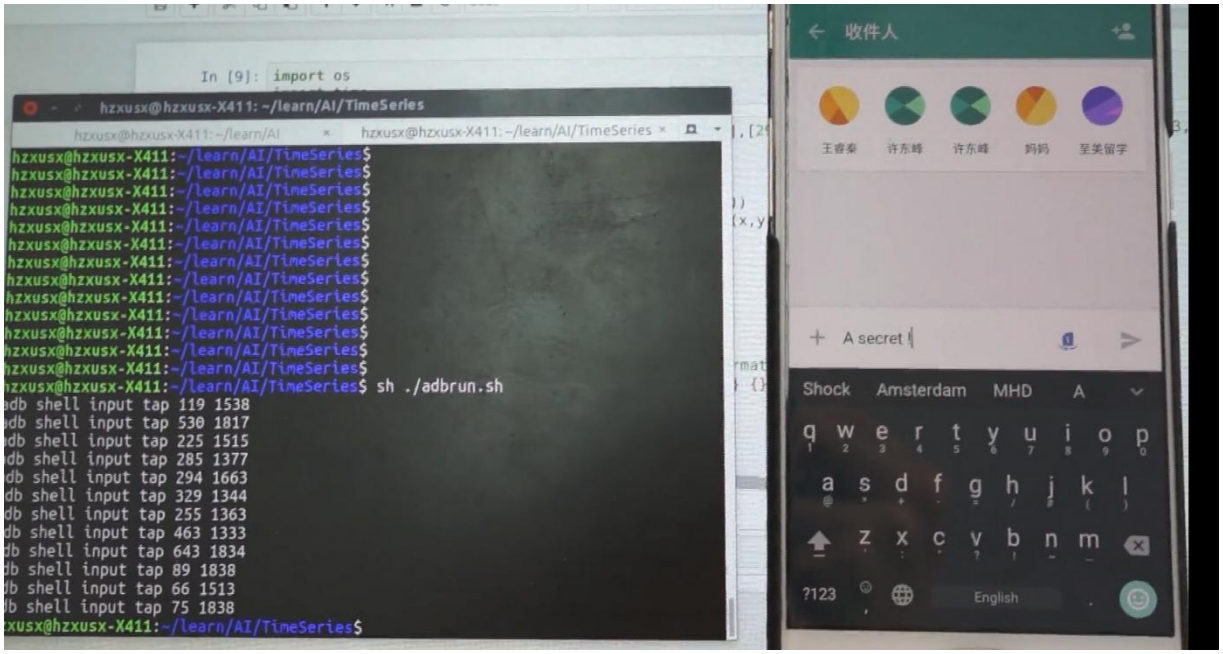
```
-17.2657368359
-8.93085081647
```

4）输入测试

因为没有对坐标具体内容做适配，我们做了一个输入测试，即输入一句话，用后台采集程序得到波形，处理后输入训练过的神经网络，将推测结果按推测时间输入手机，就按键识别率而言，错误 1/12，识别率 91.7%，下图红叉是推测的坐标：



这里有一个视频，截图可以看出虽然 secret 的 c 打成了 x，但是自动纠正的帮助使得整句话正确的输入了：



3.未来工作

首先指出我们的不足之处：时间与运算能力依然不足，坐标准确率有待提高，我们仅使用了低端 gpu 840m 计算，并且输入数据非常耗时间，没有对不同人不同姿势分别采集测试，对应可以有如下改进：

- 采集时间缩短，由更少的数据提取更多特征
- 利用更先进的神经网络
- 在更先进的机器上运算
- 采集各种情况下许多用户不同的数据
- 对坐标到内容做转换适配
- 学习坐标之外的内容

最后两点可以展开说说：

对坐标到内容做转换适配可以监测：

- PIN 码
- 滑动密码
- 全键盘、拼音九键

进一步：

- 打开了哪些 APP
- 浏览过哪些网页
- 购买过哪些产品

而学习坐标之外的内容指：

- 计步器，步态分析

- 赛车游戏中最佳路径
- 手机陀螺仪控制飞行器设定路径

进一步：

- 小脑紊乱病人康复监测
- 路径分析与导航，老人小孩定位

这说明相关工作前景广泛，完全不会局限在信息安全领域。