机器学习 (/tags/#机器学习)

基于手机传感器数据使用 CNN 识别用户行为的 Tensroflow 实现

Posted by 小辉 on November 8, 2016

智能移动设备、特别是手机，搭载了越来越多、越来越精确的传感器，利用这些传感器的数据，结合机器学习甚至深度学习的能力，可以识别出用户的行为，而用户的行为数据可以被用于像 UBI、反作弊解决方案、健身类 App 等很多领域，也可以作为 Real time customer engagement 的重要参考数据。

文章 Implementing a CNN for Human Activity Recognition in Tensorflow (http://aqibsaeed.github.io/2016-11-04-human-activity-recognition-cnn/) 介绍了在一个公开传感器数据集上面，使用 Tensorflow 实现一个 CNN 进行用户行为识别的方案，本文就是在阅读和测试这个方案过程中整理出来的。

TensorFlow 简介

首先还是先介绍一下目前最火的深度学习框架 Tensorflow，一推出就吸引了深度学习领域，从学术界到工业界的很多目光，这里简单回顾一下 Tensorflow 的历史：

- 2015 年 11 月 9 日，Google Research 发布了文章：TensorFlow - Google's latest machine learning system, open sourced for everyone (https://research.googleblog.com/2015/11/tensorflow-googles-latest-machine_9.html)，正式宣布其新一代机器学习系统开源。

> *If TensorFlow is so great, why open source it rather than keep it proprietary? The answer is simpler than you might think: We believe that machine learning is a key ingredient to the innovative products and technologies of the future. Research in this area is global and growing fast, but lacks standard tools. By sharing what we believe to be one of the best machine learning toolboxes in the world, we hope to create an open standard for exchanging research ideas and putting machine learning in products. Google engineers really do use TensorFlow in user-facing products and services, and our research group intends to share TensorFlow implementations along side many of our research publications.*

- 2016 年 4 月 13 日，Google 通过文章 Announcing TensorFlow 0.8 – now with distributed computing support! (https://research.googleblog.com/2016/04/announcing-tensorflow-08-now-with.html) 正式发布支持分布式的 TensorFlow 0.8 版本，结合之前对 CPU 和 GPU 的支持，TensorFlow 终于可以被用于实际的大数据生产环境中了。
- 2016 年 4 月 29 日，开发出目前最强围棋 AI 的 Google 旗下 DeepMind 宣布：DeepMind moves to TensorFlow (https://research.googleblog.com/2016/04/deepmind-moves-to-tensorflow.html)，这在业界被认为 TensorFlow 终于可以被当作 TensorFlow 在工业界发展的里程碑事件，极大提升了 TensorFlow 使用者的研究热情。

当然，随着使用 Tensorflow 的人越来越多，也出现了很多不同的声音，The Good, Bad & Ugly of TensorFlow (http://www.kdnuggets.com/2016/05/good-bad-ugly-tensorflow.html)（中文翻译 (https://mp.weixin.qq.com/s?

__biz=MzA3MzI4MjgzMw==&mid=2650715438&idx=2&sn=3dafb301ec8103fce7ad88d6039cb3ad)) 对目前 TensorFlow 的优缺点做了详细的分析，已经在使用和计划使用 Tensorflow 的同学们都可以看看。

传感器数据集

这个项目使用了 WISDM (Wireless Sensor Data Mining) Lab (http://www.cis.fordham.edu/wisdm/) 实验室公开的 Actitracker (http://www.cis.fordham.edu/wisdm/dataset.php) 的数据集。 WISDM 公开了两个数据集，一个是在实验室环境采集的；另一个是在真实使用场景中采集的，这里使用的是实验室环境采集的数据。

- 测试记录：1,098,207 条
- 测试人数：36 人
- 采样频率：20 Hz
- 行为类型：6 种
    - 走路
    - 慢跑
    - 上楼梯
    - 下楼梯
    - 坐
    - 站立
- 传感器类型：加速度
- 测试场景：手机放在衣兜里面

数据分析

从 实验室采集数据下载地址 (http://www.cis.fordham.edu/wisdm/includes/datasets/latest/WISDM_ar_latest.tar.gz) 下载数据集压缩包，解压后可以看到下面这些文件：

- readme.txt
- WISDM_ar_v1.1_raw_about.txt
- WISDM_ar_v1.1_trans_about.txt
- WISDM_ar_v1.1_raw.txt
- WISDM_ar_v1.1_transformed.arff

我们需要的是包含 RAW 数据的 WISDM_ar_v1.1_raw.txt 文件，其他的是转换后的或者说明文件。先看看这些数据分布：

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, r

if __name__ == "__main__":
    column_names = ['user-id', 'activity', 'timestamp', 'x-axis', 'y-axis', 'z-axis']
    df = pd.read_csv("actitracker_raw.txt", header=None, names=column_names)
    n = 10
    print df.head(n)
    subject = pd.DataFrame(df["user-id"].value_counts(), columns=["Count"])
    subject.index.names = ['Subject']
    print subject.head(n)
    activities = pd.DataFrame(df["activity"].value_counts(), columns=["Count"])
    activities.index.names = ['Activity']
    print activities.head(n)
    activity_of_subjects = pd.DataFrame(df.groupby("user-id")["activity"].value_counts()
    print activity_of_subjects.unstack().head(n)
    activity_of_subjects.unstack().plot(kind='bar', stacked=True, colormap='Blues', titl
    plt.show()
```

WISDM_ar_v1.1_raw.txt 文件不是合法的 CSV 文件，每行后面有个 ； 号，如果使用 Pandas 的 `read_csv` 方法直接加载会出错，需要先将这些分号全部删除。

这是一个不平衡的数据集，官方的描述文件中也提到：

- Walking: 424,400 (38.6%)
- Jogging: 342,177 (31.2%)
- Upstairs: 122,869 (11.2%)
- Downstairs: 100,427 (9.1%)
- Sitting: 59,939 (5.5%)
- Standing: 48,395 (4.4%)

接下来，我们看看对于每种行为来说，传感器数据有什么特征：

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 绘图使用 ggplot 的 style
plt.style.use('ggplot')

# 加载数据集
def read_data(file_path):
    column_names = ['user-id', 'activity', 'timestamp', 'x-axis', 'y-axis', 'z-axis']
    data = pd.read_csv(file_path, header=None, names=column_names)
    return data

# 数据标准化
def feature_normalize(dataset):
    mu = np.mean(dataset, axis=0)
    sigma = np.std(dataset, axis=0)
    return (dataset - mu) / sigma

# 绘图
def plot_axis(ax, x, y, title):
    ax.plot(x, y)
    ax.set_title(title)
    ax.xaxis.set_visible(False)
    ax.set_ylim([min(y) - np.std(y), max(y) + np.std(y)])
    ax.set_xlim([min(x), max(x)])
    ax.grid(True)

# 为给定的行为画出一段时间（180 × 50ms）的波形图
def plot_activity(activity, data):
    fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(15, 10), sharex=True)
    plot_axis(ax0, data['timestamp'], data['x-axis'], 'x-axis')
    plot_axis(ax1, data['timestamp'], data['y-axis'], 'y-axis')
    plot_axis(ax2, data['timestamp'], data['z-axis'], 'z-axis')
    plt.subplots_adjust(hspace=0.2)
    fig.suptitle(activity)
    plt.subplots_adjust(top=0.90)
    plt.show()

dataset = read_data('actitracker_raw.txt')
dataset['x-axis'] = feature_normalize(dataset['x-axis'])
dataset['y-axis'] = feature_normalize(dataset['y-axis'])
dataset['z-axis'] = feature_normalize(dataset['z-axis'])
```

```
# 为每个行为绘图
for activity in np.unique(dataset["activity"]):
    subset = dataset[dataset["activity"] == activity][:180]
    plot_activity(activity, subset)
```

这段代码会为每个行为画出 180 × 50ms 的波形图，慢跑的绘图结果如下：

不同行为的传感器数据波形图对比有助于帮助理解数据特征。

创建、训练并测试模型

完整的使用 Tensorflow 进行行为识别的代码如下，已经添加了必要的注释：

```
import pandas as pd
import numpy as np
from scipy import stats
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, r
import tensorflow as tf

# 加载数据集
def read_data(file_path):
    column_names = ['user-id', 'activity', 'timestamp', 'x-axis', 'y-axis', 'z-axis']
    data = pd.read_csv(file_path, header=None, names=column_names)
    return data

# 数据标准化
def feature_normalize(dataset):
    mu = np.mean(dataset, axis=0)
    sigma = np.std(dataset, axis=0)
    return (dataset - mu) / sigma

# 创建时间窗口，90 × 50ms，也就是 4.5 秒，每次前进 45 条记录，半重叠的方式。
def windows(data, size):
    start = 0
    while start < data.count():
        yield start, start + size
        start += (size / 2)

# 创建输入数据，每一组数据包含 x, y, z 三个轴的 90 条连续记录，
# 用 `stats.mode` 方法获取这 90 条记录中出现次数最多的行为
# 作为该组行为的标签，这里有待商榷，其实可以完全使用同一种行为的数据记录
# 来创建一组数据用于输入的。
def segment_signal(data, window_size=90):
    segments = np.empty((0, window_size, 3))
    labels = np.empty((0))
    print len(data['timestamp'])
    count = 0
    for (start, end) in windows(data['timestamp'], window_size):
        print count
        count += 1
        x = data["x-axis"][start:end]
        y = data["y-axis"][start:end]
        z = data["z-axis"][start:end]
        if (len(dataset['timestamp'][start:end]) == window_size):
            segments = np.vstack([segments, np.dstack([x, y, z])])
            labels = np.append(labels, stats.mode(data["activity"][start:end])[0][0])
    return segments, labels
```

```
# 初始化神经网络参数
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

# 初始化神经网络参数
def bias_variable(shape):
    initial = tf.constant(0.0, shape=shape)
    return tf.Variable(initial)

# 执行卷积操作
def depthwise_conv2d(x, W):
    return tf.nn.depthwise_conv2d(x, W, [1, 1, 1, 1], padding='VALID')

# 为输入数据的每个 channel 执行一维卷积，并输出到 ReLU 激活函数
def apply_depthwise_conv(x, kernel_size, num_channels, depth):
    weights = weight_variable([1, kernel_size, num_channels, depth])
    biases = bias_variable([depth * num_channels])
    return tf.nn.relu(tf.add(depthwise_conv2d(x, weights), biases))

# 在卷积层输出进行一维 max pooling
def apply_max_pool(x, kernel_size, stride_size):
    return tf.nn.max_pool(x, ksize=[1, 1, kernel_size, 1],
                          strides=[1, 1, stride_size, 1], padding='VALID')

dataset = read_data('actitracker_raw.txt')
dataset['x-axis'] = feature_normalize(dataset['x-axis'])
dataset['y-axis'] = feature_normalize(dataset['y-axis'])
dataset['z-axis'] = feature_normalize(dataset['z-axis'])

segments, labels = segment_signal(dataset)
labels = np.asarray(pd.get_dummies(labels), dtype = np.int8)
# 创建输入
reshaped_segments = segments.reshape(len(segments), 1,90, 3)

# 在准备好的输入数据中，分别抽取训练数据和测试数据，按照 70/30 原则来做。
train_test_split = np.random.rand(len(reshaped_segments)) < 0.70
train_x = reshaped_segments[train_test_split]
train_y = labels[train_test_split]
test_x = reshaped_segments[~train_test_split]
test_y = labels[~train_test_split]

# 定义输入数据的维度和标签个数
```

```
input_height = 1
input_width = 90
num_labels = 6
num_channels = 3

batch_size = 10
kernel_size = 60
depth = 60

# 隐藏层神经元个数
num_hidden = 1000

learning_rate = 0.0001

# 降低 cost 的迭代次数
training_epochs = 8

total_batchs = reshaped_segments.shape[0] // batch_size

# 下面是使用 Tensorflow 创建神经网络的过程。
X = tf.placeholder(tf.float32, shape=[None,input_height,input_width,num_channels])
Y = tf.placeholder(tf.float32, shape=[None,num_labels])

c = apply_depthwise_conv(X,kernel_size,num_channels,depth)
p = apply_max_pool(c,20,2)
c = apply_depthwise_conv(p,6,depth*num_channels,depth//10)

shape = c.get_shape().as_list()
c_flat = tf.reshape(c, [-1, shape[1] * shape[2] * shape[3]])

f_weights_l1 = weight_variable([shape[1] * shape[2] * depth * num_channels * (depth//10)
f_biases_l1 = bias_variable([num_hidden])
f = tf.nn.tanh(tf.add(tf.matmul(c_flat, f_weights_l1),f_biases_l1))

out_weights = weight_variable([num_hidden, num_labels])
out_biases = bias_variable([num_labels])
y_ = tf.nn.softmax(tf.matmul(f, out_weights) + out_biases)

loss = -tf.reduce_sum(Y * tf.log(y_))
optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate).minimize(lo

correct_prediction = tf.equal(tf.argmax(y_,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
cost_history = np.empty(shape=[1], dtype=float)

# 开始训练
with tf.Session() as session:
    tf.initialize_all_variables().run()
    # 开始迭代
    for epoch in range(training_epochs):
        for b in range(total_batchs):
            offset = (b * batch_size) % (train_y.shape[0] - batch_size)
            batch_x = train_x[offset:(offset + batch_size), :, :, :]
            batch_y = train_y[offset:(offset + batch_size), :]
            _, c = session.run([optimizer, loss], feed_dict={X: batch_x, Y: batch_y})
            cost_history = np.append(cost_history, c)
        print "Epoch {}: Training Loss = {}, Training Accuracy = {}".format(
            epoch, c, session.run(accuracy, feed_dict={X: train_x, Y: train_y}))
    y_p = tf.argmax(y_, 1)
    y_true = np.argmax(test_y, 1)
    final_acc, y_pred = session.run([accuracy, y_p], feed_dict={X: test_x, Y: test_y})
    print "Testing Accuracy: {}".format(final_acc)
    temp_y_true = np.unique(y_true)
    temp_y_pred = np.unique(y_pred)
    np.save("y_true", y_true)
    np.save("y_pred", y_pred)
    print "temp_y_true", temp_y_true
    print "temp_y_pred", temp_y_pred
    # 计算模型的 metrics
    print "Precision", precision_score(y_true.tolist(), y_pred.tolist(), average='weight
    print "Recall", recall_score(y_true, y_pred, average='weighted')
    print "f1_score", f1_score(y_true, y_pred, average='weighted')
    print "confusion_matrix"
    print confusion_matrix(y_true, y_pred)
```

结果为：

```
Precision 0.888409996548
Recall 0.884568153909
f1_score 0.880684696544
confusion_matrix
[[ 518   21    0    1   21   97]
 [  68 2163    0    0   15   20]
 [   3    1  368   14    5    0]
 [   3    0   11  332    1    1]
 [ 166   77    0   15  402  159]
 [  91    9    0    0   47 2700]]
```

已经数据准备和模型训练耗时较久，可以在每次运行后通过 Numpy 将结果保存到文件，然后可以使用下面的代码计算 True Positive Rate 以及 False Positive Rate，最后计算 ROC 和 confusion matrix，评估模型质量：

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, r
from sklearn.preprocessing import label_binarize

plt.style.use('ggplot')

def plot_confusion_matrix(cm, title='Normalized Confusion matrix', cmap=plt.cm.get_cmap(
    cm = cm / cm.astype(np.float).sum(axis=1)
    print "confusion_matrix: \n{}".format(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(6)
    plt.xticks(tick_marks, [1, 2, 3, 4, 5, 6], rotation=45)
    plt.yticks(tick_marks, [1, 2, 3, 4, 5, 6])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

def test_har():
    y_true = np.load("y_true.npy")
    y_pred = np.load("y_pred.npy")
    print "Precision", precision_score(y_true.tolist(), y_pred.tolist(), average='weight
    print "Recall", recall_score(y_true, y_pred, average='weighted')
    print "f1_score", f1_score(y_true, y_pred, average='weighted')
    print "confusion_matrix"
    print confusion_matrix(y_true, y_pred)
    plot_confusion_matrix(confusion_matrix(y_true, y_pred))

    if __name__ == "__main__":
        test_har()
```

Confusion matrix:

总结

可以看到使用 CNN 进行行为识别，即使只使用加速度传感器一种，识别的准确率已经挺高了，而且这个方案还有可以优化的地方，比如对输入数据进行更细致的准备；增加训练迭代的次数等。接下来，我们将尝试将这个模型移植到 Android 平台，测试一下进行实时用户行为识别的可行性，主要从 CPU 占用率以及耗电量等方面权衡。

PREVIOUS
TENSORFLOW 在 ANDROID 平台的移植
(/2016/10/18/ANDROID_TENSORFLOW/)

NEXT
神经网络快速入手
(/2016/11/17/NEURAL_NETWORK/)

FEATURED TAGS (/tags/)

调研 (/tags/#调研) iOS (/tags/#iOS) 机器学习 (/tags/#机器学习) 技术 (/tags/#技术)

FRIENDS


(https://zhuanlan.zhihu.com/talkingdata)


(http://weibo.com/TalkingData)


(https://github.com/TalkingData)

Theme by voyagelab (http://leopan.cn/) | Star 5