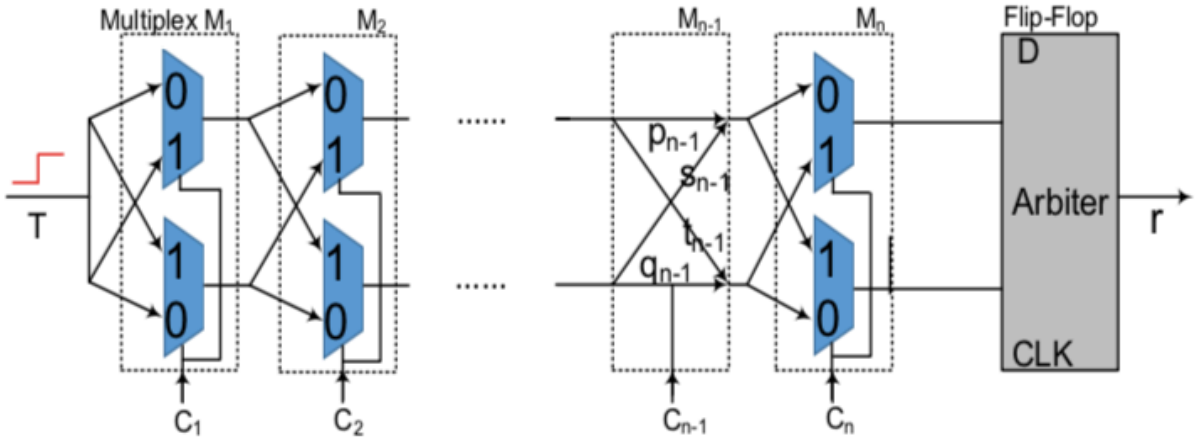
基于机器学习的物理不可克隆函数(PUF)建模攻击

信息安全1501 201508060103 张浩森

一、PUF

物理不可克隆函数(Physical Unclonable Function，PUF)是一种新的轻量级硬件安全原语。当输入一个激励时，PUF 利用芯片制造过程中难以预测的工艺偏差(Process Variation)，输出依赖于芯片的不可克隆的响应，非常适合资源受限环境下的设备认证。然而，攻击者可以收集一定数量的激励响应对将 PUF 进行建模，因此，PUF 易受基于机器学习建模攻击。下图是一种典型的PUF——Arbiter PUF，其中{C1,C2,...,Cn-1,Cn}共同组成激励，r为响应。其原理是:一个脉冲信号 T 会在 Arbiter PUF 上下两条路径同时传播，通过激励{C1,C2,...,Cn-1,Cn}改变路径(如 C1=1 时，在 M1 阶段交叉传播;C1=0 时，在 M1 阶段平行传播)，由于工艺偏差会影响不同路径的传播快慢，最终导致上下两条路径信号传播产生快慢差异，比较传播快慢生成激励响应r(0 或 1)。



二、建模攻击可行性分析

随着PUF的研究和应用日益增多，PUF的安全性也受到严重威胁，基于机器学习的 建模攻击方法就严重威胁着诸如仲裁PUF等强PUF的安全性。由于强PUF有大量CRP，若为每个CRP设计独立的电路，显然硬件开销十分巨大，因此，强PUF的不同CRP之间都有一定的关 联，建模攻击正是通过机器学习来推测运种关联，从而破解强PUF的CRP。建模攻击首先根据 强PUF的电路结构，建立W相关物理特性为未知数的CRP模型，然后通过机器学习从获取的 部分CRP数据推测运些相关物理特性，然后，对于未知响应的激励，则可W根据所推测的相 关物理特性，预测其响应，实现对强PUF的破解。常用的机器学习方法有支持向量机、逻辑回归、进化策略等。使用建模攻击破解仲裁型PUF和环形振荡PUF，CRP预测精度可达到99% W 上。即使前馈仲裁PUF、异或仲裁PUF、轻量仲裁PUF和电流镜PUF使强PUF的结构更加复杂，然 而使用建模攻击得到的CRP预测精度平均仍然可W达到90% W上，可见强PUF的安全性受到 严重威胁。本次实验采用逻辑回归和支持向量机来进行基于机器学习的建模攻击。

三、逻辑回归（logistic）和支持向量机（SVM）

1、logistic回归是一种广义线性回归（generalized linear model），因此与多重线性回归分析有很多相同之处。它们的模型形式基本上相同，都具有 w‘x+b，其中w和b是待求参数，其区别在于他们的因变量不同，多重线性回归直接将w‘x+b作为因变量，即y =w‘x+b，而logistic回归则通过函数L将w‘x+b对应一个隐状态p，p=L(w‘x+b),然后根据p 与1-p的大小决定因变量的值。如果L是logistic函数，就是logistic回归，如果L是多项式函数就是多项式回归。

2、SVM的分类思想本质上和线性回归LR分类方法类似，就是求出一组权重系数，在线性表示之后可以分类。我们先使用一组trainging set来训练SVM中的权重系数，然后可以对testingset进行分类。说的更加更大上一些：SVM就是先训练出一个分割超平面separation hyperplane, 然后该平面就是分类的决策边界，分在平面两边的就是两类。显然，经典的SVM算法只适用于两类分类问题，当然，经过改进之后，SVM也可以适用于多类分类问题。我们希望找到离分隔超平面最近的点，确保它们离分隔面的距离尽可能远。这里点到分隔面的距离被称为间隔margin. 我们希望这个margin尽可能的大。支持向量support vector就是离分隔超平面最近的那些点，我们要最大化支持向量到分隔面的距离。

四、建模分析

建模根据学长所提供的建模思路，并且加以创新。由于每一个独立的64位属于01串，都会产生一个唯一的对应的4 * 64矩阵，所以把这个4 * 64的矩阵变成一个数组，做256位输入，最后计算一位的输出，数据集由此产生。具体实现见下列代码。本次实验的逻辑回归和支持向量机用的相同的建模方式以及数据集。

五、实现具体代码

训练集产生代码：

```
import tensorflow as tf
import csv
import random
import numpy as np


def shape(M):
  return len(M), len(M[0])

out1 = open('traindatasetin.csv','w')
out2 = open('traindatasetout.csv','w')
csv_writer1 = csv.writer(out1)
csv_writer2 = csv.writer(out2)
csv_file = csv.reader(open('仿真Arbiter_PUF.csv','r'))

PUFdelay_np = []
PUFdelay = []
```

```
delay1 = 0
delay2 = 0

for i in csv_file:
    PUFdelay.append(i)

PUFdelay_np = np.array(PUFdelay, dtype=float)

C_np = [([0] * 64) for i in range(4)]
C_np_csv = []
seed = "01"
counter1 = 0
#counter0 = 0
counter = 0

while counter != 2000:
    for i in range(64):
        binary = random.choice(seed)
        if binary == "1":
            delaymiddle = delay2
            delay2 = delay1
            delay1 = delaymiddle
            delay1 += float(PUFdelay[1][i])
            delay2 += float(PUFdelay[2][i])
            C_np[0][i] = 0
            C_np[3][i] = 0
            counter1 += 1
            if counter1 % 2 != 0:
                C_np[1][i] = 1
                C_np[2][i] = -1
            if counter1 % 2 == 0:
                C_np[1][i] = -1
                C_np[2][i] = 1
        if binary == "0":
            delay1 += float(PUFdelay[0][i])
            delay2 += float(PUFdelay[3][i])
            C_np[1][i] = 0
            C_np[2][i] = 0
            #counter0 += 1
            if counter1 % 2 != 0:
                C_np[0][i] = -1
                C_np[3][i] = 1
            if counter1 % 2 == 0:
                C_np[0][i] = 1
                C_np[3][i] = -1
    for i in range(4):
```

```
        for j in range(64):
            C_np_csv.append(C_np[i][j])
    csv_writer1.writerow(C_np_csv)
    if counter1 % 2 == 0:
        if(delay1 > delay2):
            csv_writer2.writerow("1")
        else:
            csv_writer2.writerow("0")
    else:
        if(delay1 <= delay2):
            csv_writer2.writerow("1")
        else:
            csv_writer2.writerow("0")
    counter1 = 0
    delay1 = 0
    delay2 = 0
    C_np_csv = []
    counter += 1
```

测试集产生代码：

```
import tensorflow as tf
import csv
import random
import numpy as np


def shape(M):
  return len(M), len(M[0])


out1 = open('testdatasetin.csv', 'w')
out2 = open('testdatasetout.csv', 'w')
csv_writer1 = csv.writer(out1)
csv_writer2 = csv.writer(out2)
csv_file = csv.reader(open('仿真Arbiter_PUF.csv', 'r'))

PUFdelay_np = []
PUFdelay = []
delay1 = 0
delay2 = 0
```

```
for i in csv_file:
    PUFdelay.append(i)

PUFdelay_np = np.array(PUFdelay, dtype=float)

C_np = [([0] * 64) for i in range(4)]
C_np_csv = []
seed = "01"
counter1 = 0
counter11 = 0
#counter0 = 0
counter = 0

while counter != 1000:
    for i in range(64):
        binary = random.choice(seed)
        if binary == "1":
            delaymiddle = delay2
            delay2 = delay1
            delay1 = delaymiddle
            delay1 += float(PUFdelay[1][i])
            delay2 += float(PUFdelay[2][i])
            C_np[0][i] = 0
            C_np[3][i] = 0
            counter1 += 1
            if counter1 % 2 != 0:
                C_np[1][i] = 1
                C_np[2][i] = -1
            if counter1 % 2 == 0:
                C_np[1][i] = -1
                C_np[2][i] = 1
        if binary == "0":
            delay1 += float(PUFdelay[0][i])
            delay2 += float(PUFdelay[3][i])
            C_np[1][i] = 0
            C_np[2][i] = 0
            #counter0 += 1
            if counter1 % 2 != 0:
                C_np[0][i] = -1
                C_np[3][i] = 1
            if counter1 % 2 == 0:
                C_np[0][i] = 1
                C_np[3][i] = -1
    for i in range(4):
        for j in range(64):
```

```
            C_np_csv.append(C_np[i][j])
    csv_writer1.writerow(C_np_csv)
    if counter1 % 2 == 0:
        if(delay1 > delay2):
            csv_writer2.writerow("1")
        else:
            csv_writer2.writerow("0")
    else:
        if(delay1 <= delay2):
            csv_writer2.writerow("1")
        else:
            csv_writer2.writerow("0")
    counter1 = 0
    delay1 = 0
    delay2 = 0
    C_np_csv = []
    counter += 1
```

逻辑回归：

```
from __future__ import print_function, division
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.ensemble import RandomForestRegressor
import sklearn.preprocessing as preprocessing
from numpy import array
from sklearn.model_selection import train_test_split
#tensorflow 实现 Logistic Regression
#读取数据
x_test = pd.read_csv("testdatasetin.csv", header=None)  # 测试集特征
x_train = pd.read_csv("traindatasetin.csv", header=None)  # 训练集特征
y_train = pd.read_csv("traindatasetout.csv", header=None)  # 训练集标签
y_test = pd.read_csv("testdatasetout.csv", header=None)  # 测试集标签

y_train = tf.concat([1 - y_train, y_train], 1)
y_test = tf.concat([1 - y_test, y_test], 1)

#参数定义
learning_rate = 0.05  # 学习率
training_epochs = 300  # 训练迭代次数
batch_size = 100  # 分页的每页大小（后面训练采用了批量处理的方法）
display_step = 15  # 何时打印到屏幕的参量
```

```
n_samples = x_train.shape[0]  # sample_num 训练样本数量
n_features = x_train.shape[1]  # feature_num 特征数量 256
n_class = 2
#变量定义
x = tf.placeholder(tf.float32, [None, n_features])
y = tf.placeholder(tf.float32, [None, n_class])
#权重定义
W = tf.Variable(tf.zeros([n_features, n_class]), name="weight")
b = tf.Variable(tf.zeros([n_class]), name="bias")

#y=x*w+b 线性
pred = tf.matmul(x, W) + b

#准确率
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

#损失
cost = tf.reduce_sum(
    tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
#优化器
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

#初始化
init = tf.global_variables_initializer()


train_accuracy = []
test_accuracy = []
avg_cost = []
#训练
with tf.Session() as sess:
        sess.run(init)
        for epoch in range(training_epochs):
            #avg_cost = 0
            total_batch = int(n_samples / batch_size)
            for i in range(total_batch):
                _, c = sess.run([optimizer, cost],
                                feed_dict={x: x_train[i * batch_size: (i + 1)
* batch_size],
                                           y: y_train[i * batch_size: (i + 1)
* batch_size, :].eval()})

                train_accuracy.append(accuracy.eval(
                    {x: x_train, y: y_train.eval()}))
```

```
                #ax2.plot(epoch+1, avg_cost, 'c.')
                test_accuracy.append(accuracy.eval(
                    {x: x_test, y: y_test.eval()}))
                avg_cost.append(c / total_batch)
            #plt.plot(epoch + 1, avg_cost, 'co')

            if (epoch + 1) % display_step == 0:
                print("Epoch:", "%04d" % (epoch + 1), "cost=", c/total_batch)

        print("Optimization Finished!")
        print("Testing Accuracy:", accuracy.eval(
            {x: x_test, y: y_test.eval()}))

        plt.suptitle("learning rate=%f  training epochs=%i  sample_num=%i" % (
            learning_rate, training_epochs, n_samples), size=14)
        plt.plot(avg_cost)
        plt.plot(train_accuracy)
        plt.plot(test_accuracy)
        plt.legend(['loss', 'train_accuracy', 'test_accuracy'])
        plt.ylim(0., 1.5)
        #plt.savefig('AC8.png', dpi=300)
        plt.xlabel("Epochs")
        plt.ylabel("Rate")
        plt.show()
```

SVM：

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from sklearn import datasets

from pylab import mpl
mpl.rcParams['font.sans-serif'] = ['SimHei']

np.random.seed(1)
tf.set_random_seed(1)

sess = tf.Session()

#产生数据
x_vals = np.loadtxt(open(r"testdatasetin.csv", "r"), delimiter=",",
skiprows=0)
```

```
y_vals = np.loadtxt(open(r"testdatasetout.csv", "r"), delimiter=",",
skiprows=0)

#划分数据为训练集和测试集
train_indices = np.random.choice(
    len(x_vals), round(len(x_vals)*0.8), replace=False)
test_indices = np.array(list(set(range(len(x_vals))) - set(train_indices)))
x_vals_train = x_vals[train_indices]
x_vals_test = x_vals[test_indices]
y_vals_train = y_vals[train_indices]
y_vals_test = y_vals[test_indices]
#批训练中批的大小
batch_size = 100
# 初始化feedin
x_data = tf.placeholder(shape=[None, 256], dtype=tf.float32)
y_target = tf.placeholder(shape=[None, 1], dtype=tf.float32)

# 创建变量
W = tf.Variable(tf.random_normal(shape=[256, 1]))
b = tf.Variable(tf.random_normal(shape=[1, 1]))
# 定义线性模型
model_output = tf.matmul(x_data, W)+b
# Declare vector L2 'norm' function squared
l2_norm = tf.reduce_sum(tf.square(W))
#软正则化参数
alpha = tf.constant([0.2])
#定义损失函数
classification_term = tf.reduce_mean(tf.maximum(0., 1.-model_output*y_target))
loss = classification_term+alpha*l2_norm
# classification_term = tf.reduce_mean(tf.maximum(0., tf.subtract(1.,
tf.multiply(model_output, y_target))))
# loss = tf.add(classification_term, tf.multiply(alpha, l2_norm))
#输出
prediction = tf.sign(model_output)
accuracy = tf.reduce_mean(tf.cast(tf.equal(prediction, y_target), tf.float32))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)
#开始训练
sess.run(tf.global_variables_initializer())
loss_vec = []
train_accuracy = []
test_accuracy = []

for i in range(2000):
    rand_index = np.random.choice(len(x_vals_train), size=batch_size)
    rand_x = x_vals_train[rand_index]
    rand_y = np.transpose([y_vals_train[rand_index]])
```

```
    sess.run(train_step, feed_dict={x_data: rand_x, y_target: rand_y})
    temp_loss = sess.run(loss, feed_dict={x_data: rand_x, y_target: rand_y})
    loss_vec.append(temp_loss)
    train_acc_temp = sess.run(accuracy, feed_dict={
                              x_data: x_vals_train, y_target:
np.transpose([y_vals_train])})
    train_accuracy.append(train_acc_temp)
    test_acc_temp = sess.run(accuracy, feed_dict={
                             x_data: x_vals_test, y_target:
np.transpose([y_vals_test])})
    test_accuracy.append(test_acc_temp)
    if (i+1) % 100 == 0:
        print('Step #' + str(i+1) + ' W = ' +
              str(sess.run(W)) + 'b = ' + str(sess.run(b)))
        print('Loss = ' + str(test_acc_temp))  # test_acc_temp
plt.plot(loss_vec)
plt.plot(train_accuracy)
plt.plot(test_accuracy)
plt.legend(['损失', '训练精确度', '测试精确度'])
plt.ylim(0., 1.)
plt.show()
```

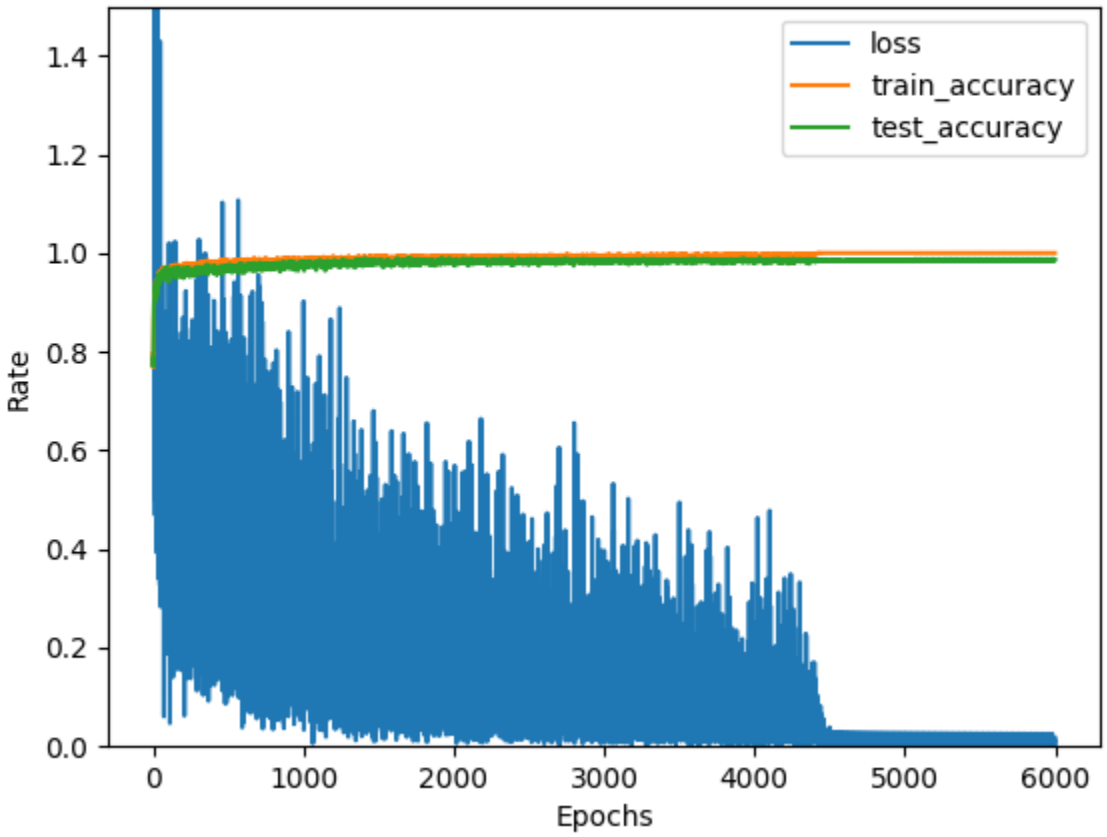
六、实验结果

本次实验的结果，逻辑回归测试正确率达到98.7%，SVM拟合程度合格。

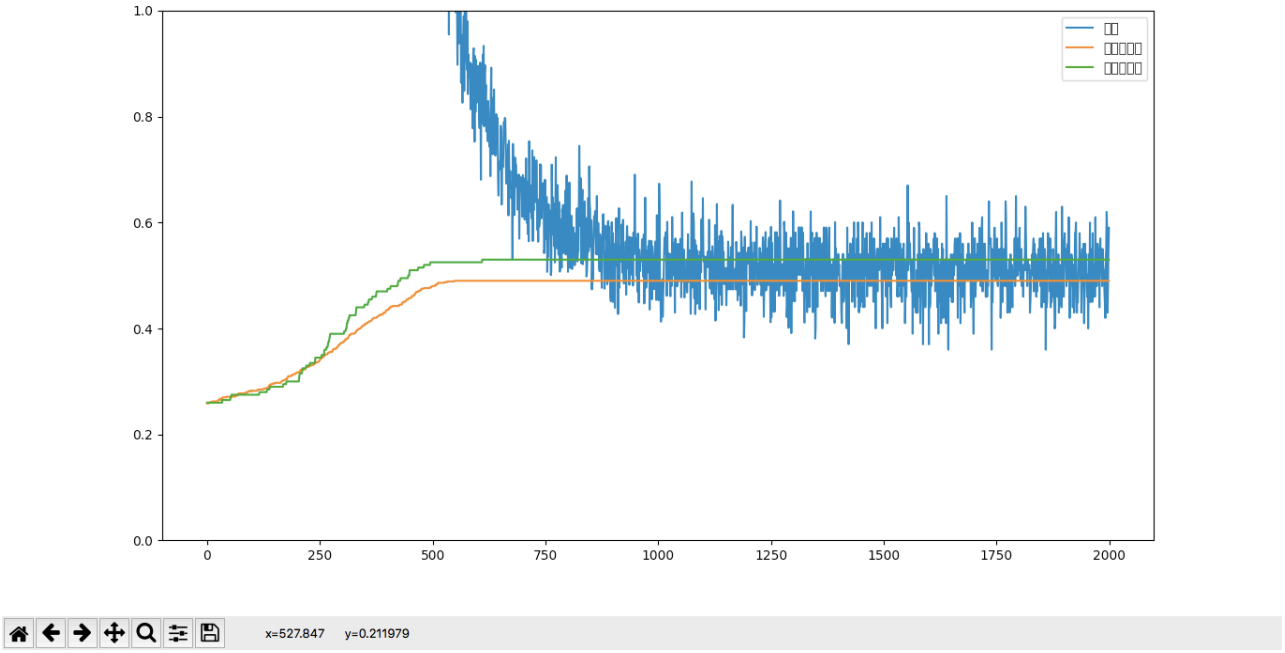
逻辑回归：

```
~/Desktop/PUF_attack 20:21:43
$ python3 logical.py
WARNING:tensorflow:From logical.py:46: softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecated
and will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See tf.nn.softmax_cross_entropy_with_logits_v2.

Epoch: 0015 cost= 0.35158629417419435
Epoch: 0030 cost= 0.3944033861160278
Epoch: 0045 cost= 0.29719927310943606
Epoch: 0060 cost= 0.24201819896697999
Epoch: 0075 cost= 0.26245918273925783
Epoch: 0090 cost= 0.16908435821533202
Epoch: 0105 cost= 0.2756670951843262
Epoch: 0120 cost= 0.355078911781311
Epoch: 0135 cost= 0.20120396614074706
Epoch: 0150 cost= 0.2468040943145752
Epoch: 0165 cost= 0.23929934501647948
Epoch: 0180 cost= 0.10441625118255615
Epoch: 0195 cost= 0.03925661444664001
Epoch: 0210 cost= 0.17751320600509643
Epoch: 0225 cost= 0.020880790054798128
Epoch: 0240 cost= 0.018718358874320985
Epoch: 0255 cost= 0.018308357894420625
Epoch: 0270 cost= 0.017967839539051057
Epoch: 0285 cost= 0.017645511031150817
Epoch: 0300 cost= 0.017330852150917054
Optimization Finished!
Testing Accuracy: 0.987
```

earning rate=0.050000 training epochs=300 sample_num=200



SVM：



七、实验分工

本次实验我主要负责建模工作，以及建模后实现数据采集算法的工作。张聪同学和陈夏润同学主要负责机器学习算法的研究以及实现。

八、实验心得

本次实验前期经历了从来没有经历过的痛苦。因为本组的所有成员以前从来没有任何建模的经历，也没有机器学习的基础，甚至说是一点也不了解相关的内容，而且PUF的工作原理在实验讲解的时候也听得云里雾里，最后思考了很长时间才把其工作原理思考清楚。在查找了相关资料找到了本组所要实现的逻辑回归和SVM算法，把数据丢进去进行计算后，发现每次正确率都只有50%多。在与其他组讨论的过程中，很长时间内也没有讨论出是哪里除了问题，一度怀疑是我们研究出来的算法出了问题。而机器学习的算法也真的是晦涩难懂，尽管使用了tensor-flow，也在很长时间内难以理解其精华。最后，我们大胆的猜测是我们的数据采集算法有一定的漏洞，并且经过长时间的缜密的思考，发现我们作为输出的数据结果是错误的。再经过思考之后，我们完善了一下数据采集的代码，最终用完美的数据集使得实验成功。

本次实验我认为试一次相当好的实验，感谢老师和学长能够为我们设计出此实验，让我们受益很多。但是学生想提一个小建议：

希望以后有此类实验，请老师和学长能够多为我们讲解一些内容，仅仅局限于实验文档的内容对于我们来说段时间内做出成果是很困难的，除非让我们把这个实验当成一个可交付的项目，而且在单个项目运行周期内交付。讲解更多的内容是为了让我们能够更快的入门。因为我们不仅仅是有这一个课程实验，我们还有其他很多的课程。也希望在每次实验之后老师和学长能够给我们一个标准的实验的正确答案，与我们自己做出的结果做一些比对，并且可以拓展我们的思路。之前的实验都没有给我们一个标准答案，尤其是JOP攻击那次实验。

感谢老师和学长，也感谢我的团队组员！学生敬上！