基于神经密码学的密钥交换协议研究综述

网络空间安全　赵正一

摘要

密钥交换协议是密码学研究的主要问题之一。神经密码学是最近发展起来的一种非经典的密码算法，他通过接受两个相同输入模式的神经网络之间的相互学习（Mutual Learning）来实现密钥交换，并使用特定的规则来更新彼此的权重。网络的每个权重分量都可以看作是权重空间中的一个随机游走。两个状态在同一个空间内运动，并在两个边界上吸收，以保证达成相同权重的更新结果。本文以树奇偶校验机为例，介绍了神经密码学在密钥交换研究领域上的贡献。并且通过使用预共享的机制对互相学习进行了一定程度上的改进。

关键字：神经密码学；相互学习；密钥交换协议；树奇偶校验机

Abstract

Key exchange is one of the major concerns in cryptology. Neural cryptography is a recent non-classical paradigm neural networks that receive the same input patterns and update their weights using specific rules. Each weight component of the network can be seen as a random walker in the weightspace. The two walkers move in the weights space and reflect at two boundaries to reach the identical weight as updating results. This paper introduced the contribution of neural cryptography on key exchange protocol field based on Tree Parity Machine. We also make some further improvment on mutual learning by using pre-shared machanism.

Keywords: Neural Cryptography; Mutual Learning; Key Exchange Protocol; Tree Parity Machine

目录

1 背景介绍

密钥交换问题是经典密码学 [1][2][3][4] 的主要研究内容之一，在经典密码学中得到了广泛的研究。首个公开的密钥交换协议是基于数论问题而提出的，Diffie-Hellman 密钥交换协议 [2] 就是在这个基础上开展研究的，并且 Diffie-Hellman 密钥交换协议使得密钥交换问题得到了更多人的关注。虽然它是依赖于离散对数的计算困难性，但是它很容易受到中间人攻击 [1]。

为了达成通信双方密钥相等，我们还可以采用神经网络的方式。两个神经网络通过对彼此的输出进行某种规则的更新权重，通过相互学习 [5] 的方式达成权重相等的目的。这个相等的权重，就可以拿来作为会话密钥使用。这就是神经密码学。神经密码学的优点是生成会话密钥的算法非常的简单快捷。训练中使用的网络可以是一个最简单的多层感知机。

神经网络的学习是一个经验化的行为。这个概念已经在统计学的模型和方法中得到了广泛的研究 [6][7]。模型的训练是一个动态的过程。对于训练语料的学习是由静态的网络一步一步生成出来的。模型在学习了训练语料后，在使用测试语料来验证模型效果，期望达到的目标就是能够无限逼近于曾经学习过的 “知识”。在形式化的证明中，对于一大类模型来说，学习和泛化的能力可以用由多个参数决定的常微分方程来描述。

相互学习是在这个基础上做了更深一步的发展。两个通信双方接收同样的输入向量，生成一个输出结果，并通过彼此的输出结果，在某种规则的约束下，更新自己神经网络的权重。最终达成通信双方的神经网络权重相等。或者也叫权重对齐。

2 模型介绍

2.1 多层感知机

人工神经网络的诞生源自于真实的神经网络。人们通过研究神经元的特性发现，学习可以被看作是在神经元之间建立新的连接或者是对已经有的类似于连接的物质进行修改的过程。所以对于人工神经网络来说，就要建立起权重与输入相乘最终得到某个结果的等式。其中 W 是权重矩阵，X 是输入矩阵，o 是对应的输出。

神经网络是有多层神经元群组成的，由泰勒展开公式可知，一个特定的函数可以分解为多个非线性函数的加和，也可以说多个非线性函数加和的时候，可以模拟出某一个特定的函数。

$$f(x) = \frac{f(x_0)}{0!} + \frac{f^{'}(x_0)}{1!}(x - x_0) + ... + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + R_n(x) \tag{1}$$

基于这个想法，我们将输入的 x，匹配某个权重 w，再增添某个偏置 b，就可以得到对于输入 x 的线性函数。再将结果输入某个激活函数中，得到最终结果。

$$f(x) = sign(wx + b) \tag{2}$$

我们将由输入空间到输出空间的到的函数称为感知机。多个神经元以全连接形式层次相连，形成的网络称为前馈神经网络，也称为多层感知机（MLP），由泰勒展开可以得知，MLP 理论上可以模拟所有函数。其中模型为 $y = F(x)$，训练数据为 $D = x_i, y_i{}_{i=1}^{n}$，预测数据为 $\hat{y_i} = F(x_i)$，训练目标为 $\min(|\hat{y_i} - y_i|)$。

通过多层感知机，我们可以得到最终的函数如下：

$$y = F(x) = f_3(W_3, f_2(W_2, f_1(W_1, x))) \tag{3}$$

并且通过梯度下降法优化目标。

$$E = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4}$$

梯度是误差对于权重的偏导数，误差通过下面的公式更新参数。

$$W^{t+1} = W^t - \eta_t \frac{\partial E}{\partial W} \tag{5}$$

由于偏导数存在链式法则，我们可以通过从后向前反向传播的方式计算梯度。

$$\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial f_3} \cdot \frac{\partial f_3}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial W_1} \tag{6}$$

但是梯度下降法存在以下问题：目标函数通常不是标准的凸函数，所以非常容易陷入局部最优解而不是全局最优解；网络层数增多后，容易出现梯度消失或者梯度爆炸问题。

2.2 树奇偶校验机

树奇偶校验机（Tree Parity Machine，TPM）由 K 个隐藏单元组成，其输入向量是 X，初始权重是 W，输出单元是 σ_k[5]：

$$X \subset x_{ij} \in \{-L, \cdots, 0, \cdots, +L\}$$

$$W \subset w_{ij} \in \{-L, \cdots, 0, \cdots, +L\}$$

$$\sigma_k = \text{sign}(\sum_{j=1}^{N} w_{ij}x_{ij}) \tag{7}$$

其中，sign(·) 函数代表取自变量的数学符号，用来表示自变量的正负形，所以有：

$$\text{sign}(\cdot) \in \{-1, 0, +1\}$$

TPM 最终的输出为 τ：

$$\tau^{A/B} = \prod_{k=1}^{K} \sigma_k^{A/B} \tag{8}$$

2.2.1 TPM 构建

TPM 的构建是一个动态的过程，通过多轮对比最终构建并同步好彼此的神经网络。其构建流程如下：

Algorithm 1 TPM 双方同步按照以下流程执行

1: 随机初始化权重矩阵 $w_{ij}^{A/B}$；
2: **while** $w_{ij}^{A} \neq w_{ij}^{B}$ **do**
3: 随机生成相同的输入矩阵 x_{ij}；
4: 计算 $\sigma_i^{A/B}$ 和 $\tau^{A/B}$ 的结果；
5: **for all** τ^A 和 τ^B，比较他们的结果，并按照如下结果继续执行 **do**
6: $\tau^A = \tau^B$：更新彼此的权重；
7: $\tau^A \neq \tau^B$：回到第三步；
8: **end for**
9: **end while**

2.2.2 TPM 学习规则

如 2.2.1 中的流程图所示，在进行到第 6 步的时候，需要按照特定的规则来更新神经网络的权重，通常我们会使用以下三种更新方法 [5][8]：

1. Hebbian 算法

$$w_i^+ = g(w_i + \sigma_i x_i \theta(\sigma_i, \tau^A)\theta(\tau^A, \tau^B)) \tag{9}$$

2. Anti-Hebbian 算法

$$w_i^+ = g(w_i - \sigma_i x_i \theta(\sigma_i, \tau^A)\theta(\tau^A, \tau^B)) \tag{10}$$

3. Random walk 算法

$$w_i^+ = g(w_i + x_i \theta(\sigma_i, \tau^A)\theta(\tau^A, \tau^B)) \tag{11}$$

其中，$\theta(x,y)$ 代表了 x 和 y 的相等关系，g 函数表示某种运算法则使得运算后的结果仍然保持在原运算空间内：

$$\theta(x,y)=\begin{cases}1, & x=y\\ 0, & x\neq y\end{cases} \tag{12}$$

2.3 收敛性分析

通过上述公式可以得到，如果双方的最终输出 $\tau^A=\tau^B$，而且对于每一个神经网络来说，每个 $\tau=\sigma_k$ 的神经元都有且只可能有以下三种运动情况：

共同运动 如果同一位置上隐藏层的输出是相等的，那么 Alice 和 Bob 双方的神经网络对应的神经单元都会发生权重更新，而且由于每一轮的输入矩阵是相同的，在相同的运算法则下，他们的运动方向一定是同向的。

$$\sigma_k^A=\sigma_k^B=\tau^{A/B} \tag{13}$$

某一方单独运动 如果同一位置上隐藏层的输出是不等的，那么 Alice 和 Bob 双方的神经网络对应的神经单元只有一方会发生权重更新。

$$\sigma_k^A\neq\sigma_k^B \tag{14}$$

不运动 如果同一位置上隐藏层的输出是相等的但是都不等于最终的数据结果，那么 Alice 和 Bob 双方的神经网络对应的神经单元都不会发生权重更新，他们不做运动，或者也可以近似看作他们的运动方向是同向的。

$$\sigma_k^A=\sigma_k^B\neq\tau^{A/B} \tag{15}$$

在此基础上，我们可以定义两个神经单元的距离。本文把同一位置上不同神经网络的隐藏单元看作是在同一条直线上运动的两个动点。由上面的运动情况分析有，他们要么同向运动，要么一方不运动，另一方运动。我们定义两点之间的距离为：

$$\rho_k=\frac{w_k^A\cdot w_k^B}{\sqrt{w_k^A\cdot w_k^A}\cdot\sqrt{w_k^B\cdot w_k^B}} \tag{16}$$

其中，$0<\rho_k<1$。当 $\rho_k=0$ 的时候，我们认为动点处于开始位置；当 $\rho_k=1$，我们认为同步结束，双方的权重矩阵相等。

因为神经网络的隐藏单元的深度 L 是一定的，所以对于上述我们简化撑的直线上两动点运动这个模型来说，动点的运动是有边界的。进入边界后，如果继续同向运动，动点被边界吸收，如果与边界反向运动，动点被边界反射。

图 1: 动点运动近似模拟模型

所以两动点的运动长度可以近似为 $m = 2L + 1$。故两动点一定会相遇，而且相遇的可能仅与时间相关。所以通信双方使用相同规模的神经网络，在一定时间内的相互学习后，权重矩阵一定会相等。

2.4 安全性分析

安全性分析的条件是，在每一次由 Eve 发起的攻击中，都认为 Eve 可以在 Alice 和 Bob 之间窃听消息，但没有机会更改他们。在这个条件下，我们讨论暴力破解和模拟攻击。

2.4.1 暴力破解

对于一个输入矩阵规模为 $K \times N$ 的输入，隐藏层的权重矩阵 W 的规模为 K，权重矩阵和输入矩阵的深度都是 L。在这样的条件下，一共有 $(2L+1)^{KN}$ 种会话密钥的可能性。假设当前神经网络仅仅是一个 $K = 3$，$N = 100$，$L = 3$ 的模型，它的计算规模也已经达到了 3×10^{253}。这对于目前的计算能力来说，暴力破解是不可能的。

2.4.2 模拟攻击

模拟攻击建立在窃听者 Eve 建立了一个与 Alice 和 Bob 相同的神经网络，并且在 Alice 与 Bob 的通信交流过程中全程窃听并记录下了内容。试图通过与 Alice 或 Bob 一起更新权重以达到三者权重相同的目的。下面我们分析这种攻击的可能性。

在这样的情况下，攻击者 Eve 一共有三种可能的选择：

1. $\tau^A \neq \tau^B$：三方均不更新权重。
2. $\tau^A = \tau^B = \tau^E$：Alice 和 Bob 双方因为具有相同的更新条件，所以采用约定好的更新规则更新权重。Eve 因为自身的结果和 Alice 以及 Bob 相等，所以 Eve 也根据窃听到的更新权重规则更新自己的权重。
3. $\tau^A = \tau^B \neq \tau^E$：Alice 和 Bob 双方因为具有相同的更新条件，所以采用约定好的更新规则更新权重。但是 Eve 的结果并不相同，所以 Eve 不更新自己的权重。

事实证明，由于概率存在，所以 Eve 的更新效率要远低于 Alice 和 Bob 双方更新权重的效率。所以 Eve 概率上永远也无法先于 Alice 或 Bob 同步权重。

3 实验验证

3.1 实验环境和设置

本文所做的实验是在如下环境下开展的：

1. 操作系统：Ubuntu 20.04.2
2. 语言环境：Python 3.6.12 :: Anaconda, Inc.
3. 依赖库：
   (a) numpy：1.19.5
   (b) matplotlib：3.3.4

3.2 实验过程

首先，我们需要搭建树奇偶校验机的神经网络。其中 k，n，l 是超参数，分别代表了神经网络的核数目，输入的维度以及神经网络的深度。这里的深度重点体现在了权重矩阵 W 和输入矩阵 X 的取值范围，也是神经密码学安全性的体现。在实验中，本文使用了 numpy 库中的 random 函数，来初始化神经网络的权重矩阵。权重矩阵是一个形状为 $k \times n$ 的矩阵，深度为 $[-l, l]$。

代码如下：

Listing 1: **BuildMachine.py**

```
import numpy as np


class Machine:
    '''
    To build a machine for synchrony the weight
    Param:
        k: the number of kernals
        n: the number of input neuron per kernal
        l: the depth of network
        W: weights in neural network
```

```
    '''

    def __init__(self, k=3, n=4, l=6):
        self.k = k
        self.n = n
        self.l = l
        self.W = np.random.randint(-l, l + 1, [k, n])
```

之后，根据树奇偶校验机的计算法则计算最终结果 τ。

代码如下：

Listing 2: **Calculation.py**

```
class Machine:
    def get_output(self, X):
    '''
    Calculate the final score
    Param:
        X: inputs of neural network
    Return:
        tau: final score
    '''
    k = self.k
    n = self.n

    # initialized with np.random.randint function
    W = self.W

    # reshape X in same size to W
    X = X.reshape([k, n])

    # np.sign function to get the sign of the value
    # np.sum function to find the sum of all elements in the matrix
    sigma = np.sign(np. sum(X * W, axis=1))

    # np.prod function to find the product of all elements in the matrix
    tau = np.prod(sigma)

    self.X = X
    self.sigma = sigma
    self.tau = tau

    return tau
```

最后，根据选定的更新权重规则来更新权重，通过不断的迭代循环，可以在可接受的运算规模上，完成双方权重的对齐。

代码如下：

Listing 3: **Update.py**

```
class UpdateRules():
    '''
    One of the following learning rules can be used for the synchronization:
       -Hebbian learning rule:
        w_i = w_i + \sigma_i\times x_i\times\theta(\sigma_i,\tau^A)\theta(\tau^A, \tau^B)
       -Anti-Hebbian learning rule:
        w_i = w_i - \sigma_i\times x_i\times\theta(\sigma_i,\tau^A)\theta(\tau^A, \tau^B)
       -Random walk:
        w_i = w_i + x_i\times\theta(\sigma_i,\tau^A)\theta(\tau^A, \tau^B)
    '''

    def theta(self, t1, t2):
        '''
        Param:
            t1:
            t2:
        Return:
            a bool value
        '''
        return 1 if t1 == t2 else 0

    def hebbian(self, W, X, sigma, tau1, tau2, l):
        '''
        Hebbian learning rule
        Param:
            W:     weights in neural network
            X:     inputs of neural network
            sigma: \sigma_i = \sum w_i\times x_i
            tau1:  tau^A \tau = \prod\sigma_i
            tau2:  tau^B \tau = \prod\sigma_i
            l:     the depth of network
        '''
        k, n = W.shape
        for (i, j), _ in np.ndenumerate(W):
            W[i, j] += X[i, j] * sigma[i] * \
                self.theta(sigma[i], tau1) * self.theta(tau1, tau2)
            W[i, j] = np.clip(W[i, j], -l, l)

```

```
39      def anti_hebbian(self, W, X, sigma, tau1, tau2, l):
40          '''
41          Anti-Hebbian learning rule
42          Param:
43              W:     weights in neural network
44              X:     inputs of neural network
45              sigma: \sigma_i = \sum w_i\times x_i
46              tau1:  tau^A \tau = \prod\sigma_i
47              tau2:  tau^B \tau = \prod\sigma_i
48              l:     the depth of network
49          '''
50          k, n = W.shape
51          for (i, j), _ in np.ndenumerate(W):
52              W[i, j] -= X[i, j] * sigma[i] * \
53                  self.theta(sigma[i], tau1) * self.theta(tau1, tau2)
54              W[i, j] = np.clip(W[i, j], -l, l)
55
56      def random_walk(self, W, X, sigma, tau1, tau2, l):
57          '''
58          Ranfom walk learning rule
59          Param:
60              W:     weights in neural network
61              X:     inputs of neural network
62              sigma: \sigma_i = \sum w_i\times x_i
63              tau1:  tau^A \tau = \prod\sigma_i
64              tau2:  tau^B \tau = \prod\sigma_i
65              l:     the depth of network
66          '''
67          k, n = W.shape
68          for (i, j), _ in np.ndenumerate(W):
69              W[i, j] += X[i, j] * \
70                  self.theta(sigma[i], tau1) * self.theta(tau1, tau2)
71              W[i, j] = np.clip(W[i, j], -l, l)
72
73
74  class Machine:
75      def update(self, tau2, update_rule='random_walk'):
76          '''
77          Update the weight matrix
78          Param:
79              tau2: tau^B
80              update_rule: learning update rules, default is random_walk
81          '''
```

```
        rules = UpdateRules()
        X = self.X
        tau1 = self.tau
        sigma = self.sigma
        W = self.W
        l = self.l

        if (tau1 == tau2):
            if update_rule == 'hebbian':
                rules.hebbian(W, X, sigma, tau1, tau2, l)
            elif update_rule == 'anti_hebbian':
                rules.anti_hebbian(W, X, sigma, tau1, tau2, l)
            elif update_rule == 'random_walk':
                rules.random_walk(W, X, sigma, tau1, tau2, l)
            else:
                raise Exception("Invalid update rule. Valid update rules are: " +
                                "\'hebbian\', \'anti_hebbian\' and \'random_walk\'.")
```

最终，实验结果如下所示。Alice 和 Bob 在更新了 718 轮权重后完成了权重对齐，并且将相等的权重矩阵作为协商好的会话密钥使用。

Listing 4: **Result.txt**

```
$ python TreeParityMachine.py

  Creating machines : k = 100, n = 10, l = 10
  Using hebbian update rule.
  Synchronization = 100 % Updates = 718
  Machines have been synchronized.
  Time taken = 12.529 seconds.
```

同时可以看到，Eve 只与 Alice 或 Bob 一起更新了 118 轮权重更新，还远远没有达到权重相等的地步。所以也验证了模拟攻击是无法破坏这一协议的。

Listing 5: **Attack.txt**

```
$ python TreeParityMachineAttack.py

  Eve's machine is only 63.15499 % synced with Alice's and Bob's
  she did 118 updates.
```

4 总结与展望

本文以采用 TPM 算法进行密钥交换的实验为基础，研究了神经密码学在密钥交换协议上的应用。通过相互学习的方法，最终在比较小的计算开销上完成了密钥交换的功能。同时，针对输入矩阵和权重矩阵边界的问题进行了一定程度上的改进。最终，通过实验验证了 Alice 和 Bob 秘密通信协商密钥的可能性，也验证了这种算法可以抵挡暴力破解和模拟攻击。

在未来，基于神经密码学的密钥交换协议仍然应当受到重视，通过在曲线空间上构建边界来使得同步攻击变得完全不可能。通过研究散列函数，确定适合 TPM 的权重矩阵的动态长度，而不是仅依赖与神经网络深度 l。同时，应该增强模型的可解释性，对于黑盒的神经网络，应该明确信息流在模型内部的传播，以形式化保证通信的安全性。

参考文献

[1] William Stallings. *Cryptography and Network Seurity-3rd edition*, Prentice Hall, 2003.

[2] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory, vol. 22, pp. 644-654, 1976.

[3] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[4] Bruce Schneier. *Applied Cryptography-2nd edition*. Addison Wiley, New York, 1996.

[5] Einat Klein, Rachel Mislovaty, Ido Kanter, Andreas Ruttor, and Wolf-gang Kinzel, *Synchronization of neural networks by mutual learning and its aplication to cryptography*, Advances in Neural Informaiton Processing Systems 17, MIT press pp. 689-696, 2005.

[6] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, Redwood City, 1991.

[7] A. Engel, and C. Van den Broeck, *Statistical Mechan-ics of Learning*, Cambridge University Press, 2001.

[8] Lanir Shacham, Einat Klein, Rachel Mislovaty, Ido Kanter, and Wolf-gang Kinzel, *Neural cryptography with feedback*, Phys. Rev. E 69, 046110, 2004.