Paillier算法

> 作者：lowlyli
>
> 时间：2021-12-3
>
> 选题原因：这部分是联邦学习里面的最基础的同态加密，机器学习其实也只是简单的线性运算$y = a * x + b$，而同态加密实现加密后运算保存一致，造就了在敏感数据行业机器学习，云计算的发展，采用联邦学习的分布式计算可以有效运用在金融，医疗等敏感信息上，随着个人隐私的加强，无法得到明文数据时，同态加密或许会成为互联网企业进行用户画像分析的下一个突破点。

同态加密介绍

同态加密

同态加密（英语：**Homomorphic encryption**）是一种加密形式，它允许人们对密文进行特定形式的代数运算得到仍然是加密的结果，将其解密所得到的结果与对明文进行同样的运算结果一样。换言之，这项技术令人们可以在加密的数据中进行诸如检索、比较等操作，得出正确的结果，而在整个处理过程中无需对数据进行解密。其意义在于，真正从根本上解决将数据及其操作委托给第三方时的保密问题，例如对于各种云计算的应用。

本质上，同态加密是指这样一种加密函数，对明文进行环上的加法和乘法运算再加密，与加密后对密文进行相应的运算，结果是等价的。由于这个良好的性质，人们可以委托第三方对数据进行处理而不泄露信息。

定义

简单定义

一种加密算法E()和相应的解密算法D()。

⊕和⊙为某种数学运算。

如果加密算法满足：$E(x+y) = E(x) \oplus E(y)$，我们将这种加密函数叫做加法同态 。

如果加密算法满足：$E(x*y) = E(x) \odot E(y)$，我们将这种加密函数叫做乘法同态 。

严格定义

同态加密的思想起源于私密同态，代数同态和算术同态是私密同态的子集。

R 和 S 是域，称加密函数 E：$R \to S$为：

加法同态，如果存在有效算法⊕，$E(x+y) = E(x) \oplus E(y)$或者 $x+y = D(E(x) \oplus E(y))$成立，并且不泄漏 x 和 y。

乘法同态，如果存在有效算法⊗，$E(xy) = E(x) \otimes E(y)$或者 $xy = D(E(x) \otimes E(y))$成立，并且不泄漏 x和 y。

混合乘法同态，如果存在有效算法⊙，$E(x \times y) = E(x) \odot y$ 或者 $xy = D(E(x) \odot y)$成立，并且不泄漏x。

减法同态，如果存在有效算法⊖，$E(x-y) = E(x) \ominus E(y)$或者 $x-y = D(E(x) \ominus E(y))$成立，并且不泄漏 x 和 y，则称$E()$为减法同态。

除法同态，如果存在有效算法⊘，$E(x/y) = E(x) \oslash E(y)$或者 $x/y = D(E(x) \oslash E(y))$成立，并且不泄漏$x$和 y，则称$E()$为除法同态。

代数同态，如果 E 既是加法同态又是乘法同态。

算术同态，如果 E 同时为加法同态、减法同态、乘法同态和除法同态。

分类

半同态加密 (Partial Homomorphic Encryption, PHE) ：只支持某些特定的运算法则 f ,PHE 的优点是原理简单、易实现，缺点是仅支持一种运算(加法或乘法)；

层次同态加密（Liveled HE，LHE）：一般支持有限次数的加密算法,LHE 的优点是同时支持加法和乘法，并且因为出现时间比 PHE 晚，所以技术更加成熟、一般效率比 FHE 要高很多、和 PHE 效率接近或高于 PHE，缺点是支持的计算次数有限。

全同态加密 （Fully Homomorphic Encryption, FHE）：支持无限次的任意运算法则 f，FHE 有以下类别：基于理想格的 FHE 方案、基于 LWE/RLWE 的 FHE 方案等等。FHE 的优点是支持的算子多并且运算次数没有限制，缺点是效率很低，目前还无法支撑大规模的计算。

第一个满足加法和乘法同态的同态加密方法直到2009年才由Craig Gentry提出。目前来说，全同态加密算法性能较差，应用较少。

比较常用的是半同态加密算法，实现方式有 RSA （乘法同态）、Elgamal（乘法同态）、Paillier （加法同态）等。

Paillier算法介绍

Paillier同态加密是由Pascal Paillier于1999年提出并命名的密码学理论。它是一种基于公私钥密码学的概率非对称算法。

这套理论是一个加法同态加密算法，意味着，只要给定公钥和需要加密的信息和，就可以计算加密后的和之和，再可以用私钥解密结果，这整个过程精度没有损失。

算法理论

Paillier 加密算法作为一种适用性非常广泛的同态公钥加密算法，因基于合数阶剩余类的难解性问题，使得该算法能够应用于许多实际的场景中，下面就 paillier算法的难解问题进行说明。

设$n = pq$，其中p, q为 2 个大素数，$\varphi(n)$为欧拉函数，

$$\varphi(n) = (p-1)(q-1)$$

Caemichael 函数 $\lambda(n) = lcm(p-1, q-1)$，

$$|Z_{n^2}^*| = \varphi(n^2) = n\varphi(n), \forall \omega \in Z_{n^2}^*$$

为直观起见，以下所有的 $\lambda(n)$用λ表示。根据 Caemichael 理论有
整

$$\omega^{\lambda} = 1 \bmod \ n, \ \omega^{n\lambda} = 1 \bmod \ n^2$$

数值的函数ε_g定义如下：$Z_n \times Z_n^* \to Z_{n^2}^*, (x, y) \to g^x \cdot y^n \bmod n^2$，如果$g$在$Z_{n^2}^*$中阶为$n$的倍数，则$\varepsilon_g$是一一映射。既对于给定的

$$\omega \in Z_{n^2}^*, x \in Z_n, \exists y \in Z_n^*$$

使得$\varepsilon_g(x, y) = \omega$ 。这样的 $\varepsilon_g(x, y)$称为 ω 的 n-residousity class, 用 $[[\omega]]_g$表示。

目前认为, 对于给定的 n, g, ω , 计算 $[[\omega]]_g$ 的问题是困难的, 这也就是 Composite Residousity Assumption(CRA)。

但是依据 p, q 的知识, 也就是 λ , 可以计算出任意的$[[\omega]]_g$ 事实上, 设 $S_n = \left\{u < n^2 \mid u = 1 \bmod n\right\}, \forall u \in S_n, L(u) = \frac{u-1}{n}$, 则有如下的结果:

$$[[\omega]]_g = \frac{L\left(\omega^{\lambda} \bmod n^2\right)}{L\left(g^{\lambda} \bmod n^2\right)} \bmod n$$

> 补充：Carmichael function[卡迈克尔函数相关性质]
>
> **定义**
>
> 在数论中，Carmichael函数的定义为使得$a^m \equiv 1 \mod n$ 成立的最小正整数m 其中$(a, n) = 1$ 将m记作$\lambda(n)$。在抽象代数术语中，$\lambda(n)$是模n的乘法群的指数。
>
> Carmichael函数也被称为规约函数(reduced totient function)以及最小泛指数函数(least universal exponent function)。
>
> **用Carmichael定理计算$\lambda(n)$**
> 根据唯一因式分解定理，任何$n > 1$的整数都可以用唯一的方式写成
>
> $$n = p_1^{r_1} p_2^{r_2} \cdots p_k^{r_k}$$

其中，$p_1 < p_2 < \ldots < p_k$ 是有小到大排列的素数,$r_1, r_2, \ldots, r_k$是正整数。那么，$\lambda(n)$就是其中每一项的λ的最小公倍数，有：

$$\lambda(n) = \operatorname{lcm}\left(\lambda\left(p_1^{r_1}\right), \lambda\left(p_2^{r_2}\right), \ldots, \lambda\left(p_k^{r_k}\right)\right)$$

上述的公式可由中国剩余定理来证明。

Carmichael函数的性质

设 a 和 n 互素， m 是最小指数， $a^m \equiv 1 \bmod n$ ，那么有:

$m \mid \lambda(n)$

也就是说，模n整数环中任意元素 a 的阶$m := \operatorname{ord}_n(a)$ 整除$\lambda(n)$。同时还有:

$$\lambda(n) = \max\left\{\operatorname{ord}_n(a) : \gcd(a, n) = 1\right\}$$
$$a|b \Rightarrow \lambda(a)|\lambda(b)$$
$$\lambda(\operatorname{lcm}(a, b)) = \operatorname{lcm}(\lambda(a), \lambda(b))$$

算法过程

(1) 密钥产生

选取两个随机的大素数p, q， 计算 $n = p * q$ 和$\lambda = lcm(p-1, q-1)$.选取随机数g， $g \in \mathbb{Z}_{n^2}^*$ 且满足 $\mu = \left(L\left(g^\lambda \bmod n^2\right)\right)^{-1}$ 存在， 其中函数 $L(x)$ 的定义如下 $L(x) = \frac{x-1}{n}$ 。

此时，公钥为(n, g)， 私钥为 (λ, μ)。

(2) 加密过程

对于明文m， $m \in \mathbb{Z}_n$ ， 选择随机数 $r < n$ ， 加密过程为

$$c = g^m r^n \left(\bmod n^2\right)$$

(3) 解密过程

对于密文 c 解密过程为

$$m = L\left(c^\lambda \bmod \mathrm{n}^2\right) * \mu \bmod \mathrm{n} = \frac{L\left(c^\lambda \bmod \mathrm{n}^2\right)}{L\left(g^\lambda \bmod \mathrm{n}^2\right)}$$

证明过程

补充前提:

首先回顾一下二项式定理。 $n \in \mathbb{N}^*$

$$(a+b)^n = \sum_{r=0}^{n} C_n^r a^{n-r} b^r = C_n^0 a^n + C_n^1 a^{n-1} b + \cdots + C_n^r a^{n-r} b^r + \cdots + C_n^n b^n$$

当 $a = 1, b = n, n = x$时可以化成下面的形式:

$$(1+n)^x = 1 + nx + \frac{x(x-1)n^2}{2!} + \ldots$$

可以化为:

$$(1+n)^x \equiv 1 + nx \left(\bmod n^2\right)$$

令 $y = (1+n)^x \bmod n^2$, 可化简为$x \equiv \frac{y-1}{n} \quad \left(\bmod n^2\right)$, 再令 $L(u) = \frac{u-1}{n}$ 则

$$L\left((1+n)^x \bmod n^2\right) \equiv L(y) \equiv \frac{y-1}{n} \pmod n \equiv x \pmod$$

即: $L\left((1+n)^x \bmod n^2\right) \equiv x \pmod n$

由上式结论不难知道: $\mu = \lambda^{-1}, (g = n+1)$。

此外，还有 $r^{\lambda n} \bmod N^2 = 1$

证明过程如下：

因为

$$\lambda = lcm(p-1, q-1)$$

所以

$$p-1|\lambda, \quad q-1|\lambda => \lambda = a(p-1) = b(q-1)$$

由费马小定理可得

$$g^{\lambda} = g^{a(p-1)} \equiv 1 \bmod p => p \mid g^{\lambda} - 1$$

同理

$$g^{\lambda} = g^{b(q-1)} \equiv 1 \bmod p \Rightarrow q \mid g^{\lambda} - 1$$

又因为$gcd(p, q) = 1$

所以

$$pq \left| g^{\lambda} - 1 => n \right| g^{\lambda} - 1 \Rightarrow g^{\lambda} \equiv 1 \bmod n => g^{\lambda} = k_1 n + 1$$

同理

$$\begin{gathered} r^{\lambda} = k_2 n + 1 \\ c^{\lambda} = \left(g^m r^n\right)^{\lambda} \bmod \left(n^2\right) \\ = g^{m\lambda} \bmod \left(n^2\right) \cdot r^{n\lambda} \bmod \left(n^2\right) \\ = \left(k_1 n + 1\right)^m \bmod \left(n^2\right) \cdot \left(k_2 n + 1\right)^n \bmod \left(n^2\right) \end{gathered}$$

有以下性质

$$\begin{gathered} (kn+1) \bmod \left(n^2\right) = kn + 1 \\ (kn+1)^2 \bmod \left(n^2\right) = 2kn + 1 \\ (kn+1)^3 \bmod \left(n^2\right) = 3kn + 1 \\ \cdots \\ (kn+1)^m \bmod \left(n^2\right) = mkn + 1 \end{gathered}$$

所以

$$\begin{aligned} c^{\lambda} &= \left(mk_1 n + 1\right) \bmod \left(n^2\right) \cdot \left(k_2 n^2 + 1\right) \bmod \left(n^2\right) \\ &= \left(mk_1 n + 1\right) \bmod \left(n^2\right) \end{aligned}$$

所以

$$\frac{L\left(c^{\lambda} \bmod \left(n^2\right)\right)}{L\left(g^{\lambda} \bmod \left(n^2\right)\right)} = \frac{\frac{mk_1 n}{n}}{\frac{k_1 n}{n}} = m$$

同态分析

同时，该算法还满足加法同态和乘法同态的两个性质。

下面分别就加法同态和乘法同态的证明过程做具体介绍。

首先给定两个密文

$$E\left(M_1, pk\right) = g^{M_1} R_1^n \bmod n^2$$

和

$$E\left(M_2, pk\right) = g^{M_2} R_2^n \bmod n^2,$$

其中$R_1 \in Z_n^*, R_2 \in Z_n^*$

(1) paillier 算法的加法同态证明过程如下：

$$
\begin{aligned}
& D\left(E\left(M_1, pk\right) \cdot E\left(M_2, pk\right) \bmod n^2\right) \\
&= D\left(\left(g^{M_1} R_1^n\right)\left(g^{M_2} R_2^n\right) \bmod n^2\right) \\
&= D\left(E\left(M_1+M_2, pk\right)\right) \\
&= M_1+M_2
\end{aligned}
$$

(2) paillier 算法的乘法同态证明过程如下:

$$
\begin{aligned}
& D\left(E(M_1, pk)^{M_2} \bmod n^2\right) \\
=& D\left(\left(g^{M_1} R_1^n\right)^{M_2} \bmod n^2\right) \\
=& D\left(E\left(M_1 M_2, pk\right)\right) \\
=& M_1 M_2
\end{aligned}
$$

Paillier算法实现

这里采用现成的库进行演示，源码里面封装好了加密解密过程，并创建了加密后的对象，重写了相关的运算；

感兴趣可以看以下：

源码分析

https://python-paillier.readthedocs.io/en/stable/phe.html

用法介绍

https://python-paillier.readthedocs.io/en/develop/index.html

测试代码

```python
from phe import paillier # 开源库
import time # 做性能测试

# 测试paillier参数
print("默认私钥大小：",paillier.DEFAULT_KEYSIZE) #2048
# 生成公私钥
public_key,private_key = paillier.generate_paillier_keypair()
# 测试需要加密的数据
message_list = [3.1415926,100,-4.6e-12]
print("原始数据：")
print(message_list)
# 加密操作
print("开始加密")
time_start_enc = time.time()
encrypted_message_list = [public_key.encrypt(m) for m in message_list]
time_end_enc = time.time()
print("加密耗时ms：",time_end_enc-time_start_enc)

encrypted_message_list_ciphertext = []
encrypted_message_list_len = []
for num in encrypted_message_list:
    encrypted_message_list_ciphertext.append(num.ciphertext())
    encrypted_message_list_len.append(num.ciphertext().bit_length())

print("加密后封装类:")
print(encrypted_message_list)
print("加密后数据：")
print(encrypted_message_list_ciphertext)
print("加密后数据长度")
print(encrypted_message_list_len)

```

```python
# 解密操作
print("开始解密")
time_start_dec = time.time()
decrypted_message_list = [private_key.decrypt(c) for c in encrypted_message_list]
time_end_dec = time.time()
print("解密耗时ms：",time_end_dec-time_start_dec)
print("解密数据:",decrypted_message_list)



# 测试加法和乘法同态
print("测试同态：")
a,b,c = encrypted_message_list # a,b,c分别为对应密文

a_sum = a + 5 # 密文加明文
a_sub = a - 3 # 密文加明文的相反数
b_mul = b * 1 # 密文乘明文,数乘
c_div = c / -10.0 # 密文乘明文的倒数

print("a:",a.ciphertext()) # 密文a的纯文本形式
print("a_sum：",a_sum.ciphertext()) # 密文a_sum的纯文本形式
# print(a_sum.ciphertext(False))
print("a+5=",private_key.decrypt(a_sum))
print("a-3",private_key.decrypt(a_sub))
print("b*1=",private_key.decrypt(b_mul))
print("c/-10.0=",private_key.decrypt(c_div))

##密文加密文
print("密文加密文")
print("明文相加")
print((private_key.decrypt(a)+private_key.decrypt(b)))
print("密文相加再解密")
print(private_key.decrypt(a+b))

#报错，不支持a*b，因为通过密文加实现了明文加的目的，这和原理设计是不一致的，只支持密文加！
#print((private_key.decrypt(a)+private_key.decrypt(b))==private_key.decrypt(a*b))
```

结果：

```
默认私钥大小： 2048
原始数据：
[3.1415926, 100, -4.6e-12]
开始加密
加密耗时ms： 0.04004788398742676
加密后封装类：
[<phe.paillier.EncryptedNumber object at 0x000002A5A5626520>, <phe.paillier.EncryptedNumber object at 0x000002A5A5648400>, <phe.paillier.EncryptedNumber object at 0x000002A5A7234C70>]
加密后数据：
```

[524291991382170452217704475980691776837614405188522037968068175026564759427912320918307453096000382119350489461565751654667188742927175709616936014994449725543996253181424504133113093704476842502121509404600704704401020343657955051139488993394696510864569944395376107274831208305297772744375254428029352127714153970661925734298887549106462250028475185047600901566232815901730690679594599732103375886312485088441992712816214151517489650303414853949493050648291376095351556437141922062712607363860043601898555564600998987018381122670560202868977113629267250394289271307071768894375207856525112730764648552403804056635121599918143847892529515739827570196829130981763379171478096053156763491864533735562542904363358041636200671073380326111231382706038175694844342345290261787254748389348410687268685858444786488671767893206001963801449392389648930342085919849556045695572133259745582479963206406380994047951250058959853589560058962477910483298184902450736920563613951952547910853893303937322618704979371386401229968669808155025000319713893612636379277847559556088628875403182870219822406238306335177716502535762098493866878014836214249807055797477782352753227476858275742534314681553372057747251377368724755085915645471865275377623886459,
269416424940886258693607260919488899845323764158616743749562006350639334858860427630659801338175678237124059137369627102051692452708091412719454022760515608223168361771711511747659126348989200942282546213019031501276065278364500888142766122517486053501206909354206670771945142628130707930654117834933893563876174302284605869037968954680433036310127553156824506084239452451725133622811971552446088378848338372034890331747119906390948627406273834264276845477660299119138603749781908689329085253292439523208528310438945333432983538267254740717013631946838120879332465184659686000301180662539102218024205358009140930273691260050183731830759440852865440742561544056141841979048430743911163852158845831858580757386884886402388858888211858821791968248589222950428572193200605881706047318018918551026896635566765386440926309183799721788073420579647226409581581112359939183851016788169669633614025777403325802025116007374430448105053688356467589927500134144872912575227027663354970924471933252614007929165708738565059332030526084180726286433089665082262477021640930039185640779877964368351443207035365443809983778188070146916089108323645588170174174728982137244143439173569978383981968082501105831184832488139182969566677450260491360575645497,
672086925542542340573658739233885056856091941371617257703258915736298300514171772636717108421375759472775458681276207475314687398083503106638117450737504893446605614216462813245601521825257222263727854788570044544981360850404970798986701963648802587870534152356529103294070868611321598972813206914606402653822445499954546528904242307905640120763114515468142099265611607887434612059052136900124495714592832523096078291984447819361584916677431878078166876060621264099494393637800047438781631353465174659844223331831729620531227301878054036564312918320946913594427253369637429340625001431712787325849640427716021850287819329842030461887139273564044212158986457725931102311159223582956676354648689347560064482793345002710056633043449671992874058955133835052485898106259930732407094339688659178264393215356629538972278971996772194186706655152384274844763263037100159537155046907187267094350979373364485631280863316542813593637073721800360632785281837538907801122409752631040556982597895243396500510624242181556717122746237272452063078477963523780388052869203131444811420155833305683886280761177152245962801086818868886001963847219304047857088892198781366886214250037109057272433786423187397743994951389815335116762978030925823681501191227]
加密后数据长度
[4096, 4095, 4096]
开始解密
解密耗时ms： 0.011508464813232422
解密数据: [3.1415926, 100, -4.6e-12]
测试同态：
a:
524291991382170452217704475980691776837614405188522037968068175026564759427912320918307453096000382119350489461565751654667188742927175709616936014994449725543996253181424504133113093704476842502121509404600704704401020343657955051139488993394696510864569944395376107274831208305297772744375254428029352127714153970661925734298887549106462250028475185047600901566232815901730690679594599732103375886312485088441992712816214151517489650303414853949493050648291376095351556437141922062712607363860043601898555564600998987018381122670560202868977113629267250394289271307071768894375207856525112730764648552403804056635121599918143847892529515739827570196829130981763379171478096053156763491864533735562542904363358041636200671073380326111231382706038175694844342345290261787254748389348410687268685858444786488671767893206001963801449392389648930342085919849556045695572133259745582479963206406380994047951250058959853589560058962477910483298184902450736920563613951952547910853893303937322618704979371386401229968669808155025000319713893612636379277847559556088628875403182870219822406238306335177716502535762098493866878014836214249807055797477782352753227476858275742534314681553372057747251377368724755085915645471865275377623886459

```
a_sum:
6860260575106684021987295297013981304877258923251203501072588587030006283940684306126577693249866034166449383357958208903070726625382669099972021176832206413087352135720502263378677742078173108415561584558070115525459744753860771623215205047249853698341862503041627939759276970004605405651034830888922380917628554395155055778860557445498100663421009101603016212617430770710359605064001370653943036930623442451876297105758010424009971148319382247545374047635980454508905324950046265190719244763709146426676452330995171477314787053573213372547481958243809489066214494754886110418015193739908533626025790638974751637810668589284223103473628313359915110308550491597403874842012031917877478640020043177616750583061302554251199845594199203275324285120528556680489441026928923108462646102793680323845566648739254395739749505599213191090218232698675633536003749530233624271017498475521507754412099191984406224218400610990600911017695179271344983538343661559976665069787931802598555601046353547144212616491467289229359579949927246144225226657350713634354792515234658152147149005772649722281235177831007349529869027599354869661444302870557513701825987540103467179942555846731265138103386824043145434385893297781140698917099417744796241485793800
a+5= 8.1415926
a-3 0.14159260000000007
b*1= 100
c/-10.0= 4.6e-13
密文加密文
明文相加
103.1415926
密文相加再解密
103.1415926

Process finished with exit code 0
```

同态加密扩展

同态加密是密码学领域自1978年以来的经典难题，也是实现数据隐私计算的关键技术，在云计算、区块链、隐私计算等领域均存在着广泛的应用需求和一些可行的应用方案。

这里介绍一下我感兴趣的联邦学习（夹带私货）

联邦学习

介绍

联邦学习（Federated Learning）是一种新兴的人工智能基础技术，在 2016 年由谷歌最先提出，原本用于解决安卓手机终端用户在本地更新模型的问题，其设计目标是在保障大数据交换时的信息安全、保护终端数据和个人数据隐私、保证合法合规的前提下，在多参与方或多计算结点之间开展高效率的机器学习。其中，联邦学习可使用的机器学习算法不局限于神经网络，还包括随机森林等重要算法。联邦学习有望成为下一代人工智能协同算法和协作网络的基础。

联邦学习的系统构架

以包含两个数据拥有方（即企业 A 和 B）的场景为例介绍联邦学习的系统构架。该构架可扩展至包含多个数据拥有方的场景。假设企业 A 和 B 想联合训练一个机器学习模型，它们的业务系统分别拥有各自用户的相关数据。此外，企业 B 还拥有模型需要预测的标签数据。出于数据隐私保护和安全考虑，A 和 B 无法直接进行数据交换，可使用联邦学习系统建立模型。联邦学习系统构架由三部分构成，如图所示。

第一部分：加密样本对齐。由于两家企业的用户群体并非完全重合，系统利用基于加密的用户样本对齐技术，在 A 和 B 不公开各自数据的前提下确认双方的共有用户，并且不暴露不互相重叠的用户，以便联合这些用户的特征进行建模。

第二部分：加密模型训练。在确定共有用户群体后，就可以利用这些数据训练机器学习模型。为了保证训练过程中数据的保密性，需要借助第三方协作者 C 进行加密训练。以线性回归模型为例，训练过程可分为以下 4 步（如图 所示）：

第①步：协作者 C 把公钥分发给 A 和 B，用以对训练过程中需要交换的数据进行加密。

第②步：A 和 B 之间以加密形式交互用于计算梯度的中间结果。

第③步：A 和 B 分别基于加密的梯度值进行计算，同时 B 根据其标签数据计算损失，并把结果汇总给 C。C 通过汇总结果计算总梯度值并将其解密。

第④步：C 将解密后的梯度分别回传给 A 和 B，A 和 B 根据梯度更新各自模型的参数。

迭代上述步骤直至损失函数收敛，这样就完成了整个训练过程。在样本对齐及模型训练过程中，A 和 B 各自的数据均保留在本地，且训练中的数据交互也不会导致数据隐私泄露。因此，双方在联邦学习的帮助下得以实现合作训练模型。

第三部分：效果激励。联邦学习的一大特点就是它解决了为什么不同机构要加入联邦共同建模的问题，即建立模型以后模型的效果会在实际应用中表现出来，并记录在永久数据记录机制（如区块链）上。提供数据多的机构所获得的模型效果会更好，模型效果取决于数据提供方对自己和他人的贡献。这些模型的效果在联邦机制上会分发给各个机构反馈，并继续激励更多机构加入这一数据联邦。以上三部分的实施，既考虑了在多个机构间共同建模的隐私保护和效果，又考虑了以一个共识机制奖励贡献数据多的机构。所以，联邦学习是一个「闭环」的学习机制。

联邦学习优势

1. 数据隔离，数据不会泄露到外部，满足用户隐私保护和数据安全的需求；
2. 能够保证模型质量无损，不会出现负迁移，保证联邦模型比割裂的独立模型效果好；
3. 参与者地位对等，能够实现公平合作；
4. 能够保证参与各方在保持独立性的情况下，进行信息与模型参数的加密交换，并同时获得成长。

参考网站

FedAI 中文站

https://cn.fedai.org/

参考资料

论文：

[1]廖祥宇. 基于paillier算法的谓词加密密文索引方案[D].湖北民族大学,2021.DOI:10.27764/d.cnki.ghbmz.2021.000096.

[2]张乐峰. 基于同态加密的空间众包隐私保护研究[D].中南财经政法大学,2019.

[3]崔建京,龙军,闵尔学,于洋,殷建平.同态加密在加密机器学习中的应用研究综述[J].计算机科学,2018,45(04):46-52.

博客：

https://blog.csdn.net/qq_34793644/article/details/118760670?utm_medium=distribute.wap_relevant.none-task-blog-2~default~baidujs_title~default-0.wap_blog_relevant_default&spm=1001.2101.3001.4242.1

https://blog.csdn.net/qq_40589204/article/details/116310125?spm=1001.2101.3001.6650.2&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-2.no_search_link&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-2.no_search_link

https://blog.csdn.net/qq_33885461/article/details/86555560?spm=1001.2101.3001.6650.2&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-2.no_search_link&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-2.no_search_link

知乎：

https://www.zhihu.com/question/27645858

https://zhuanlan.zhihu.com/p/77478956

代码：

https://github.com/FederatedAI/DOC-CHN/blob/master/%E6%9C%89%E5%A5%96%E5%BE%81%E9%9B%86%E6%B4%BB%E5%8A%A8/%E6%95%99%E7%A8%8B%E7%B1%BB/Paillier%20Cryptosystem%E7%90%86%E8%AE%BA%E4%B8%8E%E5%AE%9E%E8%B7%B5.pdf

https://github.com/wdxtub/federated-learning-note

https://python-paillier.readthedocs.io/en/stable/phe.html

https://python-paillier.readthedocs.io/en/develop/index.html