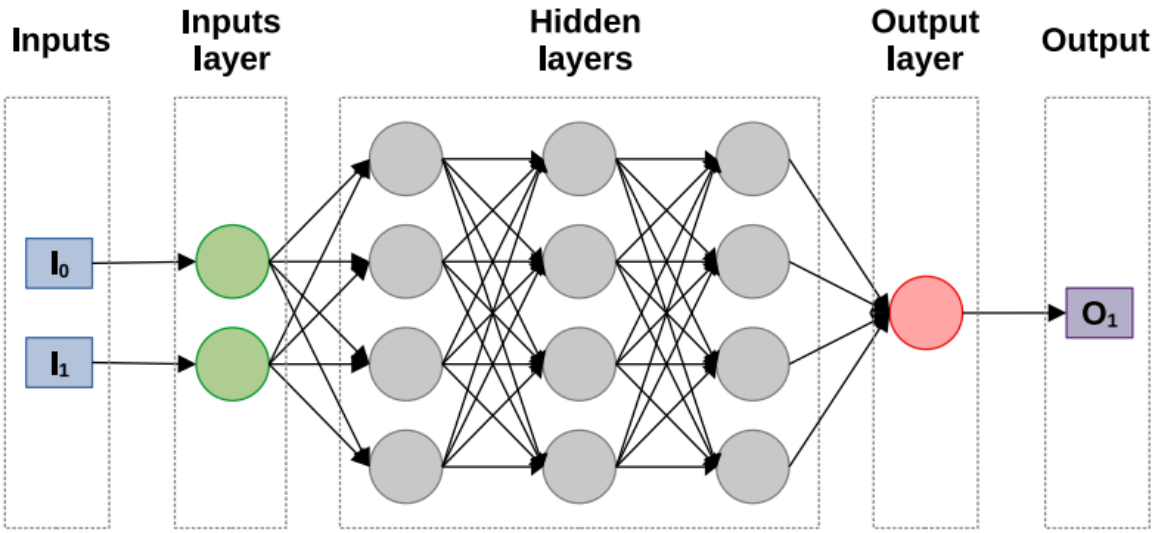
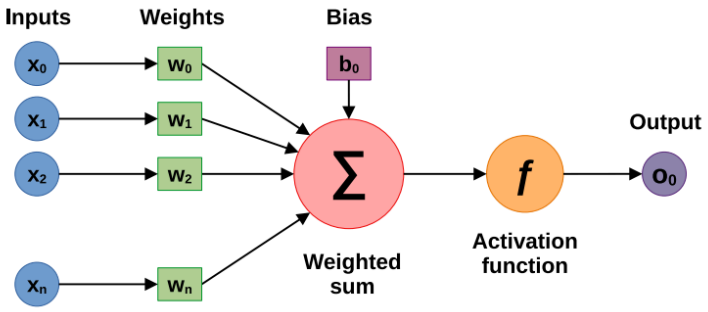
Sare Neuronalak eta Kriptografia Homomorfikoa

1. Sare Neuronala

Erabilitako sare neuronala sailkapen bitarra egiten duen MLP (Multilayer Perceptron) bat da. Aktibazio funtzio gisa sigmoidea erabiltzen da bai geruza ezkutuetan, bai irteera-geruzan.





Perceptron

2. Kriptografia Homomorfikoa

OpenFHE liburutegia, C++

2.1 CKKS

Programa honetan CKKS (Cheon-Kim-Kim-Song) zifratze-eskema guztiz homomorfikoa erabiltzen da “double” motako balioekin lan egiteko beharrezkoak diren eragiketak eskaintzen dituelako. CKKS enkriptazio guztiz homomorfikoko eskema bat da, zenbaki errealen hurbilketekin eragiketak egiteko bereziki diseinatua. Hori dela eta, oso egokia da ikasketa automatikorako.
CKKS eragiketa homomorfiko zifratuen emaitzak ez dira zehatzak, hurbilketak kalkulatzen dira eraginkortasun handiagoa eskainiz.

SIMD paketatzea eskaintzen duenez, Ciphertext motako aldagai bakar batek milaka zenbaki errealen bektore bat izan dezake (adb 32K edo 64K slot). Horri esker, inferentzia zifraturako eraginkortasun handia lor daiteke, adibidez, irudien prozesamendurako.

2.2 Aktibazio funtzioak

Sigmoid, ReLU edo tanh bezalako aktibazio funtzio ez-linealak kalkulatzea zaila da zifratze-homomorfikoko eskemetan, ezin baitira zuzenean kalkulatu soilik batuketa eta biderketak erabiliz. Kasu horietan, aktibazio funtzioen hurbilpen polinomikoak edo LUT taulak erabiltzen dira. OpenFHE liburutegiak Chebyshev polinomioetan oinarritutako hurbilpenak kalkulatzeko funtzioak eskaintzen ditu. [1]

2.3 Bootstrapping

CKKS-n bootstrapping-a beharrezkoa da zenbaki zifratuak “freskatzeko”, horien zarata pilatua gehiegi hazi bada, eragiketa homomorfiko gehiago aukera ematen duelako. Gainerako CKKS eragiketak bezala, bootstrapping-a jatorrizko zenbakiaren hurbilketa kalkulatzen du.

Bootstrapping-a eragiketa multzo bat da. Ez da beti beharrezkoa eragiketa guztiak exekutatzea, baina eragiketen ordena errespetatu behar da:

1. Funtzio linealaren ebaluazioa
2. Erredukzio modularra

Lehenik, funtzio lineala kalkulatzen da, ciphertext modulua handituz. Modulu handiago batek konputazio homomorfiko gehiago egin daitezkeela esan nahi du. Hori egiteak, zaborra gehitzen dio mezu zifratuari, zeina proportzionala den zifratze-moduluarekin. Ondoren, erredukzio modularra egiten da zaborra kentzeko.

Erabili diren OpenFHE liburutegiko bootstrapping funtzioak:

GetLevel()

ZH-an datu zifratu baten eragiketa bakoitzak “mulitiplicative depth” mailak kontsumitzen ditu. Maila hori, datu zifratu bati baliagarri izateari utzi aurretik geratzen zaizkion eragiketa kopurua adierazten du.

Maila zenbat eta altuagoa izan, gero eta eragiketa gutxiago egin daitezke bootstrapping egin behar izn aurretik. Maila threshold zehatz bat baino altuagoa denean, ciphetext-aren errorea/zarata altuegia dela esan nahi du, eta zabor bihurtuko da. Aldiz, bootstrapping egiten bada hori gertatu baino lehen, ciphertext-aren zarata-maila jaitsiko da eta eragiketak egiten jarraitzeko erabili ahal izango da.

EvalBootstrap()

Bootstrapping-a osatzen duten eragiketen multzoa. Ciphertext bat “freskatzen” du, hau da, maila jaisten du zabor bilaka ez dadin. Bootstrapping-ik gabe, eragiketa kopuru oso mugatua egin litzateke soilik. Eragiketa kopurua CryptoContext-an aurredefinitutako **multiplicativeDepth**-aren araberakoa izango da. Gero eta depth altuagoa, orduan eta eragiketa gehiago egin daitezke bootstrapping egin behar izan aurretik.

Horregatik, bootstrapping-a funtsezkoa da sare-neuronal zifratuen entrenamenduan, input eta epoch guztietan zehar weight eta bias aldagai berak erabiltzen baitira.

Rescale()

Biderketa bakoitzak maila bakarra kontsumitzen duenez, bootstrapping eragiketa osoa egin ordez, Rescale() funtzioa erabil dezakegu ciphertext-aren eskala jaisteko, zarata gehiegi igo ez dadin. bootstrappng osoa egitea denbora askoz gehiago behar du.

enc_inference_with_bootstrapping.cpp eta enc_train_functions.cpp kodeetan bootstrapping edo bootstrapping-aren eragiketa zehatz bat aplikatzen da zarata gehiegi duten ciphertext-etan. Inferentziaren kasuan, EvalMerge() egiten dela aprobetxatuz, EvalBootstrapping funtzioa gehienez behin erabiliko dugu layer bakoitzean, neurona kopurua handiegia ez den kasuetan. Ciphertext baten errore-maila eskuratzeko GetLevel() funtzioa erabiltzen da. cY aldagaiaren maila depth bat (aldagai globala) gainditu baino lehen bootstrapping-a egiten da.

```
cY = cc->EvalMerge(cNeuronak);

if(depth - cY->GetLevel() < 4){
      cY = cc->EvalBootstrap(cY);
}
```

Training-ean aldiz, Rescale() eta EvalBootstrap() funtzioak erabiltzen dira. Chebyshev hurbilpenaren kalkuluaren ondoren, biderketak “nested” moduan egiten direnez, derrigorrez EvalBootstrap() aplikatu behar zaio aktibazio funtzioaren irteerari. Gainerako operazioak Reduce()-ren bidez kudea daitezke. Orokorrean, biderketen ondoren Reduce() bat egitea nahikoa da zarata gehiegi ez handitzeko.

2.4 Crypto Context

multiplicative depth edo biderketa-sakonera ciphertext baten gainean modu sekuentzialean egin daitezkeen biderketa homomorfiko kopuru maximoa ciphertext-a zabor bihurtu aurretik adierazten du. multDepth handiago batek eragiketa gehiago egitea ahalbidetzen badu ere, aldi berean denbora igoarazten du.
scalingModSize-ak, eragiketa zifratuen emaitzen zehaztasuna definitzen du.
Ezinbestekoa da bi aldagai hauek bata bestearekin orekan definitzea.

numSlots Plaintext eta Ciphertext motako aldagaiak paketatzean gehienezko slot kopurua definitzen du. numSlots handiago bat, modu bektorialean egin daitezkeen eragiketa paralelo kopurua handitzen du. Ondorioz, ondo aprobetxatzen bada, eragiketa gehiago denbora tarte txikiago batean egitea posible izango da.

2.4.1 Inferentzia

Erabili diren sareak txikiak direnez, enc_inference.cpp kodean bootstrapping-a egitea ez da beharrezkoa, konfiguratutako multdepth-a nahikoa delako. multDepth-a handitzeak eragiketa homomorfikoa denbora gehiago eramatea egiten du, beraz, layer eta neurona gehiago dituzten sareetan bootstrapping-a erabili behar da denbora optimizatzeko. [2]

```
void init_crypto_context_ckks(CryptoContext<DCRTPoly> &cc){
    uint32_t multDepth = 12; // xor 12; diabetes 18;
    uint32_t scaleModSize = 50;
    CCParams<CryptoContextCKKSRNS> parameters;
    parameters.SetMultiplicativeDepth(multDepth);
    parameters.SetScalingModSize(scaleModSize);
```

```
    cc = GenCryptoContext(parameters);

    cc->Enable(PKE);
    cc->Enable(KEYSWITCH);
    cc->Enable(LEVELEDSHE);
    cc->Enable(ADVANCEDSHE);
}
```

Biderketa bakoitzak level bat gastatzen du eta zarata sartzen du emaitza zifratuan. Chebyshev hurbilpenak x sarrera ba jasotzen du eta y aldiz biderkatzen du bere buruaz, hurbilpenaren graduaren arabera (ciphertext · ciphertext), horrek errorea handitzen duelarik. XOR sarean 3 neuronetan produktuaren eta sigmoidearen chebyshev hurbilpenaren kalkulua egiten da. Gradua 5 izanik, multDepth-a 3·4 = 12 izan beharko da gutxienez errorea handiegia izan ez dadin.

Degree	Multiplicative Depth
3-5	4
6-13	5
14-27	6
28-59	7
60-119	8
120-247	9
248-495	10
496-1007	11
1008-2031	12

[3]

rotations: inferentzian EvalMerge erabiltzeko errotazioen gakoak sortu behar dira layer bakoitzarako. Plaintext-ak PackedPlaintext-etan paketatu daitezkenez, PackedPlaintext bat zifratzen denean PackedCiphertext bat lortzen da. Nahiz eta aldagai zifratu hori hainbat balioz osatutako ciphertext bat izan, ezin dira posizio bakoitzaren balioak zuzenean atzitu c[i] moduan, datuak paketatuta daudelako. Horretarako, EvalMerge erabiltzen da. EvalMerge ciphertext anitz batzen ditu PackedCiphertext bakar batean.

SIMD eragiketa bektorialak modu eraginkorrean aprobetxatzeko, bi packed plaintext-en ciphertext-en biderketa kalkulatzen da neurona bakoitzean wij·xi. Eragiketa hori modu berean errepikatzeko geruza bakoitzaren neurona guztien emaitzak ciphertext batean elkartu behar direnez (EvalMerge) EvalAtIndex gakoak sortu behar dira:

```
KeyPair<DCRTPoly> generate_keys(CryptoContext<DCRTPoly> &cc, nn_t *nn){
    KeyPair<DCRTPoly> keys = cc->KeyGen();
    cc->EvalMultKeysGen(keys.secretKey);
    cc->EvalSumKeyGen(keys.secretKey);
    // EvalMerge erabiltzeko beharrezkoak diren RotationKey-ak sortu
    std::vector<int32_t> indexList;
    int max = 0;
    for(int i = 0; i < nn->n_layers; i++){
        if (nn->layers_size[i] > max)
            max = nn->layers_size[i];
```

```
    }
    for(int i = 1; i < max; i++){
        indexList.push_back(i);
        indexList.push_back(-i);
    }
    cc->EvalAtIndexKeyGen(keys.secretKey, indexList);
    return keys;
}
```

[4]

2.4.2 Training

Entranamenduan bootstrapping-a egiteko aukera aktibatu behar dugu crytpoContext-ean. Hori egiteko, OpenFHE proiektuko advanced-ckks-bootstrapping.cpp adibidean azaltzen den prozedura jarraitu da.

```
KeyPair<DCRTPoly> init_crypto_context_ckks_train(CryptoContext<DCRTPoly> &cc, nn_t *nn, uint32_t &numSlots){
    ...
    parameters.SetSecretKeyDist(UNIFORM_TERNARY);
    parameters.SetSecurityLevel(HEStd_128_classic);

    #if NATIVEINT == 128
        parameters.SetScalingTechnique(FIXEDAUTO);
        parameters.SetScalingModSize(78);
        parameters.SetFirstModSize(89);
    #else
        parameters.SetScalingTechnique(FLEXIBLEAUTO);
        parameters.SetScalingModSize(59);
        parameters.SetFirstModSize(60);
    #endif

    std::vector<uint32_t> levelBudget = {4, 4};
    std::vector<uint32_t> bsgsDim = {0, 0};

    uint32_t levelsAfterBootstrap = 20;
    depth = levelsAfterBootstrap + FHECKKSRNS::GetBootstrapDepth(levelBudget, UNIFORM_TERNARY);
    parameters.SetMultiplicativeDepth(depth);
    parameters.SetBatchSize(numSlots);

    cc = GenCryptoContext(parameters);

    cc->Enable(FHE); // Bootstrapping gaitu
    ...
}
```

levelBudget: “We set a budget for the number of levels we can consume in bootstrapping for encoding and decoding, respectively. Using larger numbers of levels reduces the complexity and number of rotation keys, but increases the depth required for bootstrapping. We must choose values smaller than ceil(log2(slots)). A level budget of {4, 4} is good for higher ring dimensions (65536 and higher).”

bsgsDim: “We give the user the option of configuring values for an optimization algorithm in bootstrapping. Here, we specify the giant step for the baby-step-giant-step algorithm in linear transforms for encoding and decoding, respectively. Either choose this to be a power of 2 or an exact divisor of the number of slots. Setting it to have the default value of {0, 0} allows OpenFHE to choose the values automatically.”

2.4.3 Segurtasuna

4 segurtasun maila daude openFHE-n:
HEStd_NotSet: Ez da segurua

```
parameters.SetSecurityLevel(HEStd_NotSet);
parameters.SetRingDim(1 << 12);
```

HEStd_128_classic: 128 biteko segurtasuna

```
parameters.SetSecurityLevel(HEStd_128_classic);
```

HEStd_192_classic: 192 biteko segurtasuna

```
parameters.SetSecurityLevel(HEStd_192_classic);
```

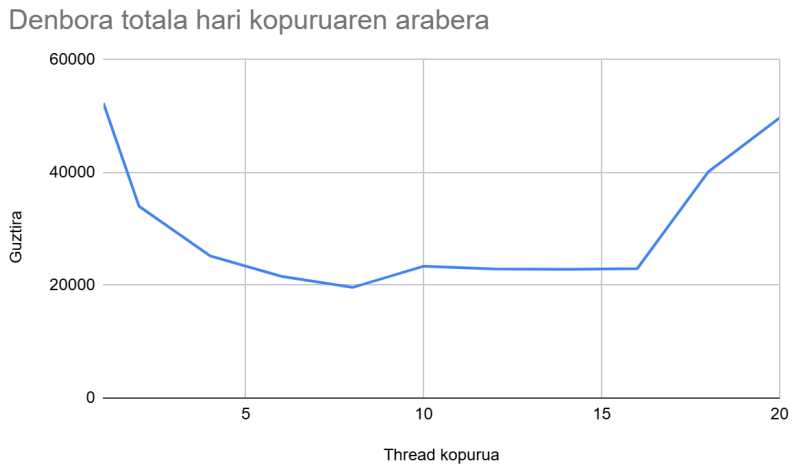
HEStd_256_classic: 256 biteko segurtasuna

```
parameters.SetSecurityLevel(HEStd_256_classic);
```

2.5 OpenMP

OpenFHE-k hariak erabiltzen ditu bootstrapping, biderketa, batuketa eta bestelako eragiketak azkartzeko. Hari kopuru desberdinekin egindako proben arabera, ondorioztatu da CryptoContext-ak modu eraginkorrenean 8 hari erabilita funtzionatzen duela. Denbora ms-tan (XOR HStd_NotSet):

Thread kopurua	Input 0	Input 1	Input 2	Input 3	Guztira
1	1393	21349	14768	14772	52303
2	887	13891	9606	9618	34016
4	636	10094	6998	7473	25210
6	558	8645	5977	6409	21598
8	**517**	**8125**	**5461**	**5504**	**19615**
10	498	9466	6693	6722	23387
12	498	9286	6543	6549	22884
14	493	9290	6518	6514	22823
16	500	9353	6566	6522	22950
18	621	15828	11960	11732	40150
20	605	19192	14834	15068	49710



Kodearen beste zatiak azkartzeko hari gehiago erabili nahiez gero, hariak banatzeko modu bat erabili beharko da (pool-etan banatu edo eskuz kudeatu banaketa). Bestela, CryptoContext-ak behar baino thread gehiago erabiliko ditu eta grafikoak erakusten duen bezala, thread-banaketak dakarren denbora-gainkarga gehituko zaie bootstrapping, biderketa, … eragiketei, batez-besteko eraginkortasuna jaitsiz.

nn-kripto proiektuan eskuz kudeatzen da hari-banaketa hori:

```
int hardware_threads = std::thread::hardware_concurrency();
int omp_threads = omp_get_max_threads();
int extra_threads = hardware_threads - omp_threads;
int available_threads = std::max(1, extra_threads);
```

eta paralelizatu nahi den kode zatian:

```
#pragma omp parallel for num_threads(available_threads)
for(int j = 0; j < nn->layers_size[L+1]; j++){
    ...
}
```

Horrela, makinak 16 thread erabiltzeko gaitasuna badu, CryptoContext-ak 8 hari bakarrik erabiltzen dituela bermatzen dugu.

3. Inferentzia

3.1 Input zifratuak

Bezeroak gako publikoarekin datuak zifratzen ditu zerbitzarira bidali aurretik. Zerbitzariak inferentzia homomorfikoa egiten du eta emaitza zifratua, y iragarpena, bezeroari bidaltzen dio. Bezeroak bere gako pribatua erabiliz deszifratzen du.
Zerbitzaria: modeloa
Bezeroa: sk, pk
Input, x: ciphertext
Pisuak (w), bias-ak (b): plaintext

Testing-ean, input bektore bakoitza packed plaintext bihurtzen da eta zifratzen da ciphertext bakarrean (ciphertext paketatua). Ciphertext hori, cx, neurona bakoitzera doan weight bakoitzarekin biderkatzen da Ciphertext · Plaintext (packed) eta balio guztiak batu egiten dira. Ondoren, bias-a gehitzen zaio eta neurona bakoitzeko balio bakar bat lortzen da. Balio hori Ciphertext motakoa da, posizio bakarrekoa.

```
for(int neurona = 0; neurona < nn->layers_size[layer+1]; neurona++){
    cNeuronak[neurona] = cc->EvalAdd(cc->EvalInnerProduct(cY, pW[layer][neurona], input_size), pB[layer][neurona]);
    cNeuronak[neurona] = cc->EvalChebyshevFunction(sigmoid_chebyshev, cNeuronak[neurona], -10, 10, 5);
}
```

Neurona bakoitzak bere geruzaren aktibazio-funtzioa kalkulatzen du, sigmoid, Chebyshev erabiliz. epoch kopurua eta learning_rate-a eragina dute balioen espazio tartean. Entrenamenduan epoch edota learning_rate altuegiak ezartzen badira, w eta b balioak handiegiak bihur daitezke testing-erako.

3.2 Sare zifratua

Modeloa plaintext-ean entrenatzen da eta zifratuta bidaltzen da. Arazoa: zifratze-gakoa.
Bezeroaren gakoarekin zifratu → Bezeroak deszifratu dezake modeloa

1. aukera:
Zerbitzariaren pk(model), bezeroari bidali, bezeroak prediction zifratua kalkulatu eta Zri bidali.
Zerb-ak deszifratu eta bezeroari bidali iragarpenak plaintext-ean.
Arazoa: Zerbitzariak sk_z(pk_z(y)) = y predikzioa lor dezake.

enc_net_inference.cpp programan inplementazioa.

2.aukera: Proxy Re-Encryption
Helburua: Modeloa zerbitzariak bakarrik ezagutu eta y bezeroak soilik ezagutu.
Arazoa: Proxy-ak pk_b eta sk_z ezagutuko ditu → modeloa ezagutuko du

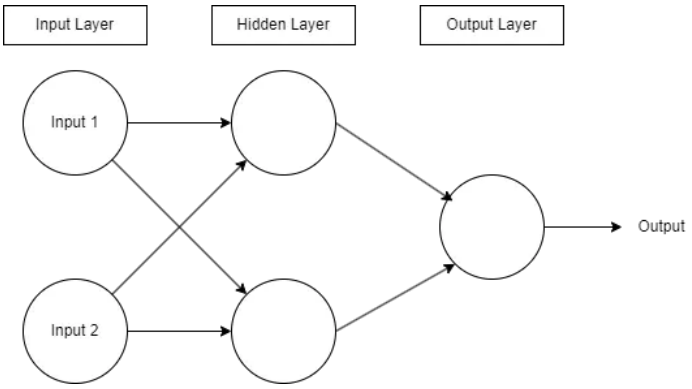
3. aukera: Re-Encryption
Helburua: Modeloa zerbitzariak bakarrik ezagutu eta y bezeroak soilik ezagutu
Komunikazioa eta kalkuluak:
B → **Z**: pk_b
Z: Re-Encryption Key = REK = pk_b "+" sk_z
Z → **B**: [REK, pk_z(model)]
B: nn(pk_z(model), x) = pk_z(y)
B: sk_b(REK(pk_z(y))) = y

enc_net_inference_pre.cpp programan inplementazioa.

3.3 Sare eta input zifratuak
Input: Zifratuta
Modeloa: Zifratuta
Helburua: Zerbitzari batean hornitzaile baten modelo zifratua eta bezero baten input zifratuak erabili inferentzia egiteko.
Zailtasunak: Ciphertext x Cipherext → Bootstrapping eragiketa asko eta mult depth altua

4 Inferentziaren Emaitzak
Programak exekutatzeko erabili den makina: Intel(R) Xeon(R) Silver 4309Y CPU @ 2.80GHz

3.1 eta 3.2 puntuen programak denbora berdinean exekutatzen direnez, 3.1 puntukoa erabiliko dugu probetarako, hau da, sarrerak zifratuta eta sarea plaintext-ean.

4.1 XOR


```
./bin/nn.exe --train --learning_rate 0.1 --epochs 10000 --batch_number 1 --dataset datasets/xor.csv
--layers 2,2,1 -s 1 --model models/model_xor.m --verbose --activation sigmoid --loss mse
```

```
./bin/nn --test  --dataset datasets/xor.csv --model models/model_xor.m --verbose --activation sigmoid
```

Pausu guztien denborak ms-tan (bootstrapping-ean OpenMP aktibatu gabe):

	Layer 0				Layer 1		
	1. Neurona		2. Neurona		1. Neurona		
Input	wx+b	Chebyshev	wx+b	Chebyshev	wx+b	Chebyshev	Guztira
1	82	259	84	267	49	98	936
2	86	255	82	266	45	91	908
3	92	274	82	256	48	92	927
4	85	247	86	300	43	139	939

Denborak 128bit ms-tan (OpenMP aktibatuta CryptoContext-ean):

Input	NotSet serie	NotSet par	NotSet SIMD	128bit serie	128bit par	128bit SIMD
0	154	157	155	3204	3151	3136
1	150	158	152	3105	3064	3396
2	150	160	153	3140	3099	3275
3	149	162	154	3160	3130	3275
BB	150,75	159,25	153,5	3152,25	3111	3270,5

TP: 2, FP: 0
FN: 0, TN: 2
Precision: 1
Recall: 1
F1: 1

4.2 Diabetes

Sare hau neurona gehiagorekin entrenatu behar dugunez, multDepth eta scaleModSize altuagoak erabili behar dira bootstrapping erabili nahi ez badugu: multDepth = 18 eta scaleModSize = 40 gutxienez. Horrek biderketa bakoitzak behar duen denbora handitzen duenez, neurona bakoitzaren batez besteko kalkulu-denbora handituko du. Hainbat sare desberdin probatu dira (8-2-2-1, 8-3-2-1, 8-2-1, …) baina emaitza hoberenak ematen dituena 8-3-1 sarea izan da.

4.2.1 Exekuzioa eta irteera

Plaintext entrenamendua:

```
./bin/nn --train --learning_rate 0.01 --epochs 2500 --batch_number 1 --dataset datasets/diabetes_train.csv
--layers 8,3,1 -s 1 --model model.m --verbose --scaling normalization
```

Inferentzia zifratua:

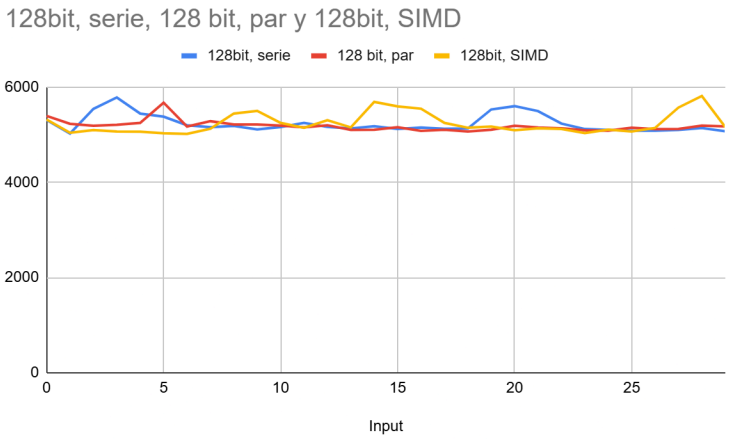
```
./bin/nn --test  --dataset datasets/diabetes_test.csv --model model.m --verbose --activation sigmoid
--scaling normalization
```

Plaintext inferentzia:
TP = 36, FP = 15
FN = 19, TN = 84
Precision = 0.705882,
Recall = 0.654545,
F1 = 0.679245,

Ciphertext inferentzia:
TP: 36, FP: 14
FN: 19, TN: 85
Precision: 0.72
Recall: 0.654545
F1: 0.685714

4.2.2 Denborak

	NotSet, serie	NotSet, par	NotSet, SIMD	128bit, serie	128 bit, par	128bit, SIMD
Input BB denbora (ms)	269,7666667	267,0666667	260,2333333	5236,166667	5182,766667	5239,566667



5 Training zifratua

5.1 Dataset zifratua

Input: Dataset zifratua
Modeloa: Plaintext
Helburua: Dataset-aren konfidentzialtasuna.

Supervised Learning ikasketa egingo denze, dataset-a eta input bakoitzari dagokion output-a zifratuta bidaliko dira entrenamendua egingo duen zerbitzarira.

Bezeroak training datuak zifratuko ditu eta zerbitzarira bidaliko ditu crypto context-arekin batera. Horretarako, Data Serialization egin beharko du. [5]

5.1.1 Chebyshev hurbilpenak

Nahiz eta inferentzia zifraturako tartea eta gradua nahikoak izan, entrenamendua zehatza izan dadin bi parametro horiek egokitu behar dira:

```
cA[layer][neurona] = cc->EvalChebyshevFunction(sigmoid_chebyshev, cZ[layer][neurona], -10, 10, 15);
```

Entrenamendurako, [-10,10] tartea eta n=15 graduaren erabilerak zehaztasun nahikoa eskaintzen du.

5.2 Sare zifratua

Input: Dataset, plaintext
Modeloa: Zifratua
Helburua: Modeloaren konfidentzialtasuna babestu entrenamenduan
Proxy Re-Encryption supongo

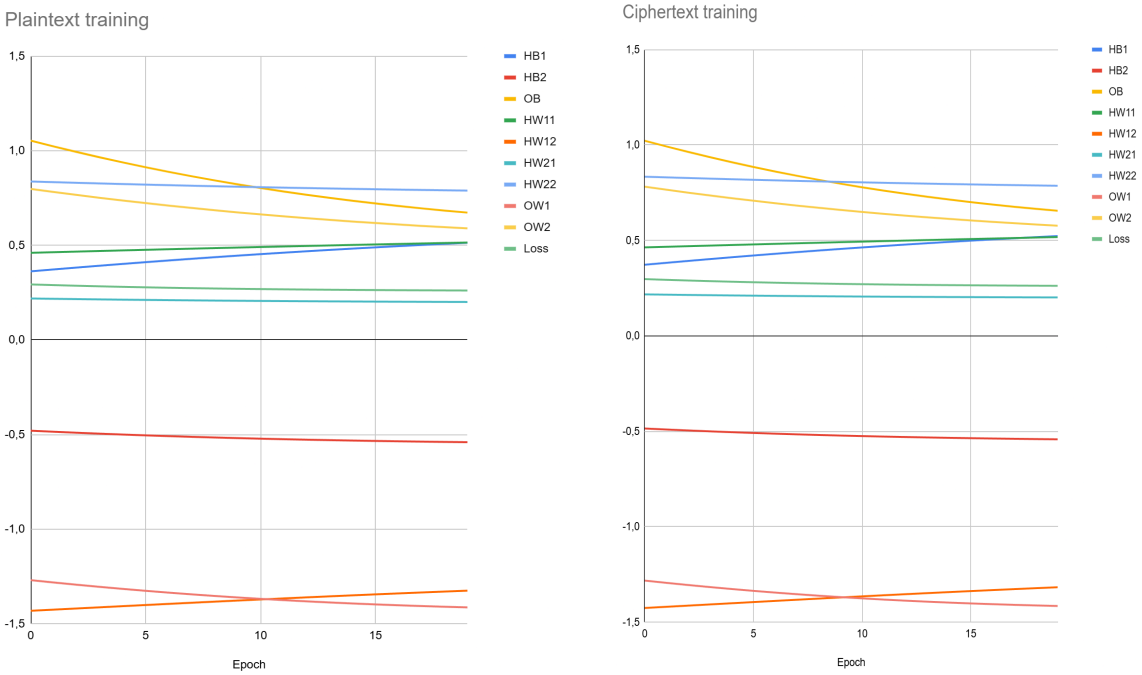
5.3 Sare eta dataset zifratuak

Input: Dataset, ciphertext
Modeloa: Zifratua
Helburua: ????
Zailtasunak: Ciphertext x Cipherext eta key !=-ak → Bootstrapping eragiketa asko eta mult depth altua

6 Entrenamenduaren Emaitzak

6.1 XOR

6.1.1 Balioak



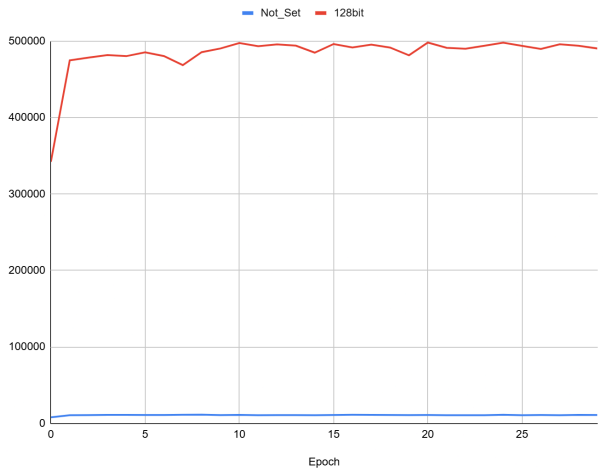
Lehen 20 epoch-en weight eta bias-en balioak plaintext eta ciphertext entrenamenduan.

CKKSn hurbilketak kalkulatzen direnez, ciphertext entrenamenduaren epoch bakoitzaren W eta B matrizeen balioak ez dira guztiz berdinak izango plaintext-arenarekin konparatuta. Hala ere, grafikoetan ikusten den moduan, lortu da chebyshev hurbilpen nahiko zehatz bat aukeratzea eta CKKS zehaztasun nahiko batera iristea W eta B balioak norabide zuzenean eta “rate” onargarri batean eguneratzea.

3400 epoch-ekin entrenatu da XOR sare bat eta lortu da inferentzia (plaintext eta ciphertext-ean) egitea. precision, recall eta f1 = 1.

6.1.2 Denborak

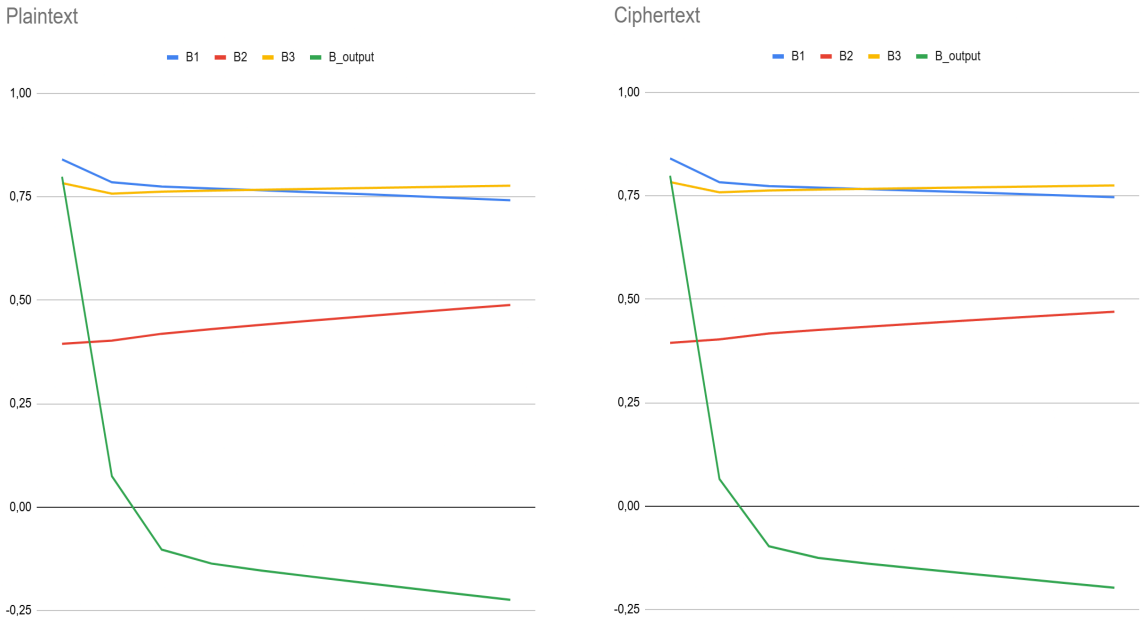
Denborak neurtzeko, Security = Not_Set eta 128bit erabili dira.



Input edota batch kopurua aldatzeak ez du denboran eraginik izango lineala delako.

6.2 Diabetes

6.2.1 Balioak



Lehen 20 epoch-en bias-en balioak plaintext eta ciphertext entrenamenduan.

Norabide egokian ikasten du.

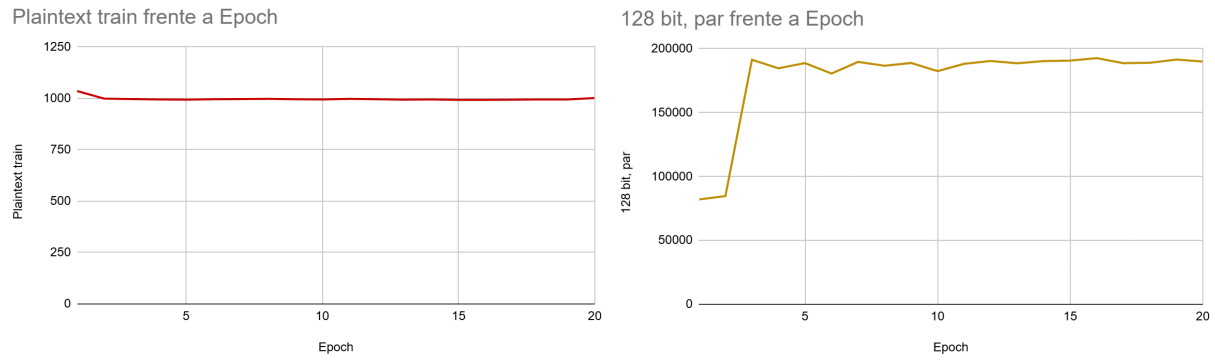
6.2.2 Denborak

Bootstrapping eragiketak optimizatu ondoren: eraginkortasuna = 2.48h/(41,44min/60) = 3,5907 azeleratu da.

614 input epoch bakoitzean.
BB (s) input:

NotSet	128 bit
4,0503	187,408

NotSEt: Input = 4,0503s = 0,067505min → Epoch = 614·0,067505 = 41,44807 min
128 bit-eko segurtasuna: Input = 187,408s = 3,1234 min → Epoch = 614·3,1234min = 31,9627h



Bi kasuetan lineala denez, input kopurua eta batch_size aldagaiaren balioa ez dute denboran eraginik izango. Lehen bi epoch-ak batez bestekoa baino gutxiago tardatzen dute beti, zarata orekatzen den arte biderketa kopuru finko bat egin behar duelako. W eta B ciphertext-en mailak orekatzen direnean, maila berdinean mantenduko dira entrenamendu guztian zehar. Beraz, gero eta zarata maila baxuagoa izna, orduan eta denbora gutxiago eramaten dute eragiketa homomorfikoek.

7 Ondorioak

Lortutako emaitzen arabera, ondoriozta daiteke kriptografia homomorfikoa bideragarria eta eraginkorra dela ikasketa automatikoan inferenzia zifratua burutzko, prozesamendu paraleloko teknikak erabiltzeari esker, hala nola SIMD. CKKS eskemak eskaintzen duen datu-paketatze gaitasunari esker, sare neuronal baten geruzetako eragiketak modu bektorialean exekuta daitezke, eta horrek eragiketa kopurua nabarmen murrizten du, baita exekuzio-denbora ere.
Hori dela eta, tamaina ertaineko sareetan inferentzia zifratua denbora onargarrietan burutu daiteke, eta tamaina handiko sareetan, gainera, prozesamendu paraleloaren eskalagarritasunari esker, errendimendua proportzionalki hobetzen da.

Entrenamendu zifratua CKKS eskema erabiliz egitea posible dela ere frogatu da. Hala ere, sortutako datu zifratuen gaineko neurona-sareen entrenamenduak muga handiak ditu oraindik, eta ezin da eraginkortzat jo praktikan erabiltzeko. Horren arrazoia da, neurri handi batean, SIMDren abantailak ez dituela guztiz aprobetxatzen. Inferentzian ez bezala, entrenamenduan ez dira SIMD modu eraginkorrean erabiltzeko egokiak liratekeen egiturak erabili weight eta bias-entzat, forward eta back propagation egiteko slot-ak lekuz aldatu beharko liratekeelako. Adibidez, matrize baten iraulia kalkulatu behar da forward-etik backpropagation-era pasatzeko, eta horretarako Ciphertext bektore guztien slot guztiak zatitu eta beste Ciphertext bektore batean ordenatu beharko lirateke [Future Work 1.]. Hori egitea lortuz gero, bideragarriagoa izango litzateke entrenamendu zifratua egitea. Bestalde, biderketa, batuketa, hurbilpenen ebaluazioak,… Weight eta bias egitura berdinetan modu iteratiboan egiteak, bootstrapping operazio asko egin behar izatea dakar, eta ondorioz denboraren hazketa handia. Arazo hori partzialki murriztu egin da Bootstrapping eragiketa osoa egin ordez Rescale erabili delako modu estrategikoan, baina hala ere, ez da training zifratuaren arazo nagusiena konpondu.

Laburbilduz, inferentzia zifratuak praktikoa eta eskalagarria dela erakusten du SIMD erabiliz, batez ere ereduak aldez aurretik plaintext-ean entrenatuta dauden eta inputak edo modeloa bakarrik zifratzen diren kasuetan. Entrenamendu zifratuak aldiz, hobekuntza nabarmenak eskatzen ditu eraginkortasuna lortzeko. Espero da etorkizuneko optimzazioek, hala nola entrenamendu osoan SIMD modu egokian erabiltzeak, egoera hori hobetzea.

8 Konparaketa

Work	HE Scheme	λ	Architecture	Training iteration time (s)
Yoo and Yoon [60]	TFHE	128	MLP[1-1-1]	1681.20
ReBoot	CKKS (bootstrapping)	128	MLP[1-1-1]	198.37
ReBoot	CKKS (bootstrapping)	128	*eMLP-1*	190.32
Colombo et al. [23]	TFHE	128	MLP[4-2-3]	646.72
ReBoot	CKKS (bootstrapping)	128	MLP[4-2-3]	193.77
ReBoot	CKKS (bootstrapping)	128	*eMLP-1*	190.32
Colombo et al. [23]	TFHE	128	MLP[16-4-2-3]	5034.00
ReBoot	CKKS (bootstrapping)	128	MLP[16-4-2-3]	621.15
ReBoot	CKKS (bootstrapping)	128	*eMLP-2*	629.65

Yoo and Yoon proiektuak ez du SIMD erabiltzen, eta ReBoot lortu du SIMD erabiltzea training-erako, denbora asko aurreztuz. MPL minimo baten ([1-1-1]) input baten denbora minimoa 200 segundu baino gutxiago da ReBoot proiektuan. [9]

nn-kripto proiektuko denborak arkitekturaren arabera:

Segurtasuna bit-etan	Arkitektura	Training iteration time (s)
128	MPL[1-1-1] (NOT)	38,3031
128	MPL[2-2-1] (XOR)	127,576
128	MPL[8-3-1] (diabetes)	187,408

Hau lehenengo biak baino azkarragoa da pruning erabiltzen duenean. Horregatik accuracy jaisten da pruning (kop?) igotzean
Bestela 1775.72 ordu baino gehiago epoch bat. Epoch batean zenbat input??

Table 3: Privately Training an MLP from scratch under different data pruning ratios.

Method		1%	5%	10%	20%	40%	50%	60%	70%	80%	90%
Acc.	Acc(%)	93.23	97.12	97.39	98.38	98.52	98.55	98.5	98.48	98.45	98.45
	$\Delta Acc.$	-5.26	-1.37	-1.1	-0.11	+0.03	+0.06	+0.01	-0.01	-0.04	-0.04
Runtime(h)	Time(h)	32.25	110.61	208.56	404.46	796.26	992.16	1188.06	1383.94	1579.88	1775.72
	speed up	60.8×	17.2×	9.4×	4.8×	2.5×	1.9×	1.7×	1.4×	1.2×	1.1×

Training from scratch. We show the performance of encrypted data pruning in the training from scratch setting in Table 3. We train a 3-layer MLP on the MNIST dataset. We set the pruning

[10]

9 Future Work

1. Training kodea azkartzeko agian funtzio hauek erabiltzen ikasi behar da matrize eragiketak azkartzeko:

```
Ciphertext<DCRTPoly> EvalLinearTransform(const std::vector<ReadOnlyPlaintext>& A, ConstCiphertext<DCRTPoly> ct) const;

Ciphertext<DCRTPoly> EvalCoeffsToSlots(const std::vector<std::vector<ReadOnlyPlaintext>>& A, ConstCiphertext<DCRTPoly> ctxt) const;

Ciphertext<DCRTPoly> EvalSlotsToCoeffs(const std::vector<std::vector<ReadOnlyPlaintext>>& A, ConstCiphertext<DCRTPoly> ctxt) const;
```

[1][8][9]
SIMD training hobetu baldin Split posiblea: *"CKKS bootstrapping has the best performance when the ciphertext contains a large number of slots (more than 100) and/or higher precision is sought (more than 3-8 bits)."* [6]
Bestela igual Ciphertext erraldoi bat sortu era errotazio arruntekin jolastu matrize egiturak simulatzeko.
Paper honetan W-ren matrize egiturak aldatzea lortzen du: [9]

2. Bibliografia bilatu: norbaitek erabiltzen du entrenatzeko? Bai, baina oso motela. [9][10]
3. OpenMP + SIMD hobetu daiteke?
4. Segurua al da bezeroari REK = rek_sortu(pk_b, sk_z) bidaltzea?
5. ReLU vs chebyshev hurbilpenak vs EvalPoly
6. Serializazioa: gakoak (pk, rotation, mult, …), CryptoContext, …
7. levelBudget for encoding is too large
8. Automatic Bootstrapping

10 Bibliografia

[1]https://github.com/openfheorg/openfhe-development/blob/02a8e9c76c3e2eff53392530199c63e4da53eb65/src/pke/include/cryptocontext.h#L2975
[2]https://github.com/openfheorg/openfhe-development/blob/main/src/pke/examples/advanced-ckks-bootstrapping.cpp

[3]https://github.com/openfheorg/openfhe-development/blob/02a8e9c76c3e2eff53392530199c63e4da53eb65/src/pke/examples/FUNCTION_EVALUATION.md
[4]https://github.com/openfheorg/openfhe-development/blob/main/src/pke/examples/rotation.cpp
[5]https://github.com/openfheorg/openfhe-development/blob/main/src/pke/examples/simple-real-numbers-serial.cpp
[6] Demystifying Bootstrapping in Fully Homomorphic Encryption by Ahmad Al Badawi and Yuriy Polyakov, August 24, 2023. https://eprint.iacr.org/2023/149.pdf
[7] https://eprint.iacr.org/2018/153.pdf
[8] https://eprint.iacr.org/2018/1043.pdf
[9] ReBoot: Encrypted Training of Deep Neural Networks with CKKS Bootstrapping. https://arxiv.org/pdf/2506.19693
[10] https://openreview.net/pdf?id=y2fAmldTIf