Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials (GETI)

Disseny i implementació d'una plataforma basada en intel·ligència artificial per a la validació de la robustesa envers atacs de canal lateral en circuits dígitals

MEMÒRIA

13 de setembre de 2021

Autor: Míriam Vila i Rodríguez

Director: Álvaro Gómez Pau

Codirector: Salvador Manich Bou

Convocatòria: 02/2021

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

Aquest Treball de Fi de Grau s'ha dut a terme a l'Escola Tècnica Superior d'Enginyeria Industrial de Barcelona (ETSEIB). Aquest projecte té com a objectiu la creació d'una plataforma basada en intel·ligència artificial per a la validació de la robustesa envers atacs de canal lateral. Es pretén desenvolupar un *software* des de zero capaç de realitzar prediccions i classificacions.

S'ha fet una recerca dels diferents mètodes estadístics més comuns avui dia i s'ha triat la xarxa neuronal com la més adient per a l'objectiu del projecte. Aquesta s'ha programat mitjançant el llenguatge C degut a la seva gran versatilitat i la facilitat d'arribar a nivells molt baixos dins de la programació. Per tant, poder capçar el complet funcionament de la xarxa neuronal. Aquesta eina, s'ha començat a comprovar amb conjunts de dades senzills i cada vegada se n'ha augmentat la complicació per tal d'assolir un nivell de complexitat elevat que permeti a la xarxa fer prediccions sobre conjunts de dades variatss. La plataforma dona llibertat a l'usuari per ser flexible en els seus paràmetres i així optimitzar-la al màxim per tal d'adaptar-se a cadascun dels conjunts de dades. S'aconsegueix una xarxa programada des de zero que funciona correctament per els diferents conjunts de dades que es presenten tot i que es deixa la porta oberta a optimitzar les dades d'entrenament per a realitzar un atac de canal lateral

Al llarg de les pàgines que segueixen, el lector podrà conèixer el procediment seguit en la realització del treball, així com els diferents entrenaments que s'ha realitzat amb els seus resultats. A més a més, a nivell personal, es vol plasmar la idea que una vegada es coneix un llenguatge de programació n'és prou semblant aprendre'n i saber utilitzar a un nivell prou complex un de desconegut.

Índex

Índex de figures

1 Introducció

1.1 Motivacions

Actualment, el gran increment de la capacitat d'emmagatzematge de la informació i una major implantació de les cadenes de subministrament mundials, han generat la necessitat a les empreses de detectar patrons i predir futurs escenaris per tal de ser més competitives en el mercat actual.

D'aquesta manera neix l'anomenada ciència de les dades. Una branca de l'estadística encarregada de modelar, entendre i interrelacionar grans quantitats de dades. També se la coneix habitualment com a aprenentatge estadístic. Aquesta àrea també està estretament relacionada amb les ciències computacionals i, l'anglicisme de moda, *machine learning*. Però, com aquests tecnicismes s'han imposat a les nostres vides i perquè no són només una moda passatgera?

La *revolució* tecnològica de les últimes dècades ha permès la massiva recopilació de dades de les empreses sobre les diferents activitats que realitzen. Això fa que les entitats puguin entendre i millorar els seus processos, i en conseqüència, estalviar en diners i temps. De totes maneres, la recopilació de dades en si mateixa és insignificant. Aquesta cal transformar-la en informació vàlida. És aquí on entra en joc el paper de l'aprenentatge estadístic. Per tal de poder prendre les decisions correctes cal estudiar i identificar els patrons amagats i les relacions en les dades i extreure'n conclusions.
Antigament, quan la quantitat de dades era limitat, aquest fet es podia analitzar amb l'estadística tradicional. Actualment, ja es parla de *Big Data*, on les dimensions sobrepassen amb creixes el que un humà pot gestionar en un temps raonable. És en aquest punt on aquest camp va de la ma amb la ciència computacional. Els ordinadors, són capaços de trobar aquestes relacions i aquests patrons de manera molt més eficient que no els humans.

A més a més, la ciència de dades és molt transversal i aplicable a diferents nivells de dificultat. Potser els exemples amb més ressò i que ja formen part de la vida quotidiana de les persones són els assistents de veu o els cotxes autònoms. De totes maneres, l'anàlisi estadístic està en gairebé tot allò que ens envolta. Des de la publicitat personalitzada fins a la sanitat.

El paper complicat i clau és saber programar els mecanismes necessaris per tal de trobar els patrons dins de les dades que es disposen i extreure'n conclusions. En la Figura 1 es mostra un esquema senzill del procés de presa de decisions. En el que s'intentarà centrar aquest Treball de fi de grau és en el tercer pas, és a dir, la programació i l'anàlisi de diferents bases de dades.

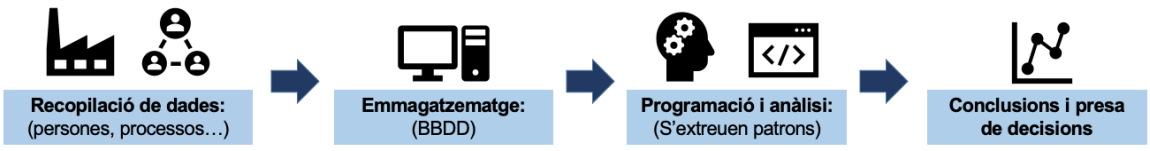

Figura 1: Procés d'anàlisi de dades

1.2 Objectius

Moltes de les infraestructures per realitzar anàlisis de dades ja estan dissenyades i simplement s'apliquen definint els paràmetres que són d'interès. En aquest Treball de fi de Grau s'ha volgut anar un pas més enllà i començar des de zero un model estadístic. Per tant, es defineixen els següents objectius principals:

- Desenvolupar una plataforma *software from scratch* capaç d'aprendre i realitzar prediccions amb diferents bases de dades orientat a avaluar la robustesa envers d'atacs de canal lateral.
- Estudiar en profunditat quines opcions hi ha dins de l'estadística computacional i quins són els seus usos i les diferents implementacions per a l'aplicació objectiu.
- Aplicació del disseny de la plataforma *software* per a la validació de la robustesa envers atacs de canal en circuits dígitals.

1.3 Abast del projecte

Aquest treball de final de grau pretén principalment desenvolupar una plataforma *software* d'intel·ligència artificial. En una primera fase s'analitzen tot el ventall de possibilitats d'infraestructures d'anàlisi exitents per tal de dissenyar aquella més adient pel projecte. Si bé és cert que existeixen diverses plataformes ja dissenyades i accessibles al públic, en aquest treball de final de grau s'ha decidit fer una implementació des de zero (*from scratch*). Així es podrà tenir un control total de l'algorisme a la vegada que es comprèn completament el seu funcionament.

Una vegada dissenyada, s'haurà d'entrenar amb diferents conjunts de dades, cada vegada més propers i semblants al conjunt de dades final que es vol entrenar.

Una vegada acabada la implementació, s'intentarà realitzar una predicció, per intentar realitzar un atac de canal lateral.

2 Mètodes d'anàlisi de dades i d'aprenentatge autònom

2.1 Introducció

En la majoria de processos tecnològics de presa de dades i especialment en aquells processos electrònics es disposa de dues parts ben diferenciades. La recollida de dades normalment es realitza mitjançant l'anomenat *hardware*. Un *hardware* és un conjunt d'elements físics o materials que poden interaccionar amb l'entorn i obtenir-ne informació. Usualment, envien aquesta informació a un ordinador. D'altra banda el *software* és el conjunt de programes i algoritmes que utilitza l'ordinador per a realitzar un determinat càlcul o acció.
Com ja s'ha dit anteriorment, en aquest treball de final de grau s'apostarà per a la implementació d'un *software* d'un model estadístic.

Per tant un *software* és un algoritme informàtic que utilitza l'ordinador per a realitzar càlculs. S'entén per algoritme doncs, aquella sèrie d'ordres amb les quals a partir d'una informació d'entrada es pot obtenir una informació de sortida en un temps acotat. Per exemple, a l'escriure la contrasenya per entrar a l'usuari de correu electrònic, a l'hora de clicar a accedir hi ha programada l'ordre de comprovar caràcter a caràcter si aquella clau donada coincideix amb la de l'usuari inserit.

Si es va un pas més enllà, es poden configurar algorismes per fer prediccions. Per exemple, si s'analitzen les cerques a internet d'un usuari se'n poden extreure certs patrons i per tant conèixer els interessos dels usuaris. Aquests algorismes prenen per nom algorismes d'aprenentatge estadístic o automàtic. Aquest àmbit s'engloba en l'anomenada intel·ligència artificial. Les màquines són capaces de fer prediccions en base a patrons i per tant són capaces de prendre decisions. És per aquest motiu, que se les considera intel·ligents i per tant tenir un comportament més o menys similar al que faria un humà.

De totes maneres, cal entrenar prèviament aquests algorismes amb conjunts de dades que continguin informació tant d'entrada com de sortida. Així la màquina pugui aprendre en base a uns exemples i més endavant, amb casos mai vists amb anterioritat, encertar les prediccions.

2.2 Aprenentatge estadístic

Com bé diu el propi nom, l'aprenentatge estadístic aprèn sobre com estimar un cert valor de sortida anomenat *output*. Així doncs, es distingeixen dos tipus d'aprenentatge: les prediccions i les inferències. S'entén per predir un cert valor quan s'estableix una conclusió sobre un futur esdeveniment o ocurrència. En aquest cas, s'intenta que l'algorisme sigui molt precís. D'altra banda, aquest aprenentatge pot comportar només la inferència entre dues variables, és a dir, com una variable Y pot estar afectada quan es varia X. Aquesta relació es detecta en base a les dades de les quals es disposen.

Existeixen dos tipus diferents de metodologies segons com es tracten les dades d'entrada. Aquests s'expliquen a continuació:

- **Mètodes paramètrics:**
  En aquest cas, s'assumeix quina és la tendència que presenten les dades del model a l'hora de calcular l'*output*. Per exemple, s'implementen models lineals, quadràtics, etc...

- **Mètodes no paramètrics:**

D'entrada, no es fa cap suposició sobre quina és la forma que presenten les dades.

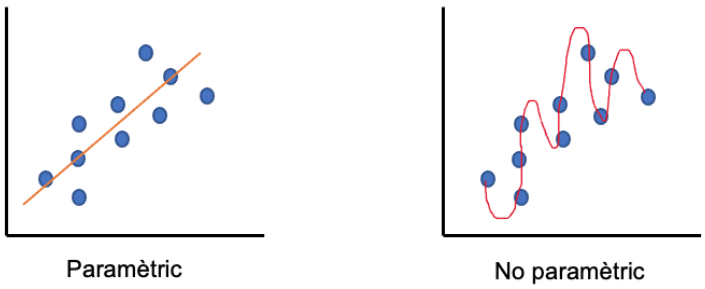

Figura 2: Mètode paramètric i no paramètric

A la Figura 2 es pot observar la diferència entre un mètode paramètric i no paramètric. Les principals desavantatges d'utilitzar els mètodes paramètrics és la gran quantitat d'observacions de dades que es necessiten i la gran afectació de les valors atípics. Tot i això, s'aconsegueix un model molt acurat. De totes maneres, si es segueix tant a prop les dades de les que es disposen es pot produir l'anomenat sobrefitament. Els valors atípics afecten de manera notable i distorsionen el model.

De totes maneres, si s'identifiquen els valors atípics i es descarten perquè no és millor sempre utilitzar un model més flexible i més realista?
Acostuma a ser molt més fàcil d'interpretar un model que utilitza un mètode més simple i a la vegada dona prediccions prou acurades.

Per tant, a l'hora de triar un model pot sorgir el dubte de que preferir, si precisió o simplicitat. Normalment, depèn de quin sigui l'objectiu principal. Si el que es pretén és **predir** s'intentarà que el model sigui flexible doncs no ens interessa la interpretabilitat d'aquest. Per contra, si el principal objectiu és detectar **inferències** entre variables, s'apostarà per un model més simple i relativament inflexiu. De totes maneres cal recordar que el model més flexible no sempre és el que prediu millor doncs es produeix el fenomen del sobrefitament.
Així doncs, tal i com s'observa a la Figura 3 quan s'augmenta la flexibilitat, es disminueix la interpretabilitat.

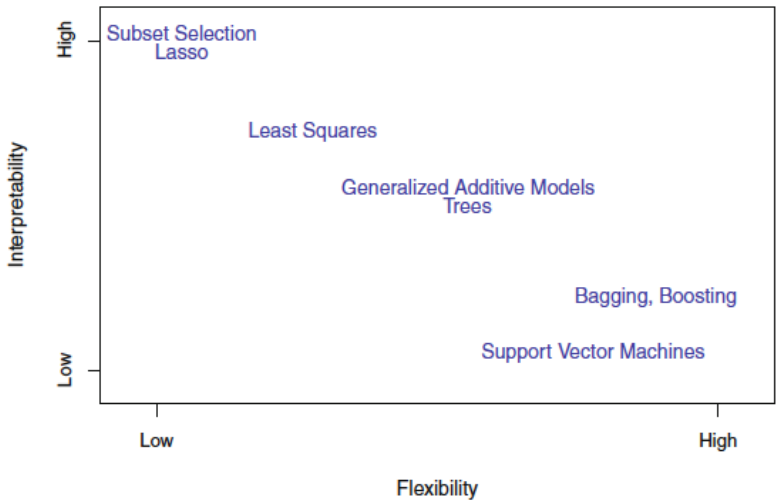

Figura 3: Representació de la interpretabilitat del model, amb diferents models d'aprenentatge estadístic. [2]

També es pot realitzar una classificació dels mètodes d'aprenentatge automàtic segons les dades obtingudes i la finalitat amb la que són utilitzades. Així doncs es troben:

- Aprenentatge supervisat
- Aprenentatge no supervisat
- Aprenentatge semisupervisat

A continuació, es detallen cada un d'ells.

2.2.1 Aprenentatge supervisat

Pel que fa a l'aprenentatge de supervisat, les dades a partir de les quals es crea el model estan etiquetades i classificades.
Per cada una de les observacions mesurades hi ha associada una resposta específica. L'objectiu principal és trobar un model que relacioni la resposta amb els diferents predictors de la manera més acurada possible per tal de predir la resposta per futures observacions. Alguns exemples de models d'aprenentatge estadístic són: la regressió lineal, la regressió logística, GAM, *boosting* i *supervided learning domain*.

Per una millor comprensió, s'exposa un paral·lelisme. S'agafa a un nen petit i se li mostra una imatge d'un tren i d'un cotxe. A continuació, se li ensenya diverses imatges de trens i de cotxes diferents i se li diu que els agrupi. En aquest cas es basarà en les dues primeres imatges per a classificar en dos grups de trens i cotxes.

2.2.2 Aprenentatge no supervisat

En aquest cas, les dades no estan etiquetades ni classificades. Hi ha observacions (*inputs*), però no hi ha cap resposta associada. En aquests casos no és possible d'aplicar models tals com la regressió logística doncs no hi ha cap variable de resposta a predir. En aquest cas, es treballa

d'alguna manera *'a cegues'*.
El que s'intenta doncs és establir relacions entre les variables i/o entre les diferents observacions. El mètode més clàssic d'estadísica no supervisada és el *clustering*.

Seguint amb el paral·lelisme de la secció anterior, ara al nen petit no se li ensenya primer les imatges d'un tren i un cotxe. Directament, se li donen imatges de trens i cotxes i se li indica que les classifiqui. El nen classificarà les imatges segons si tenen 4 rodes o més, segons si van per rails, etc... Buscarà quines són les similituds i les diferències però no arribarà a saber que una cosa són trens i l'altre cotxes.

2.2.3 Aprenentatge semisupervisat

De vegades, no està tan clar si un model ha de ser supervisat o no supervisat. Per exemple, de totes les observacions que es tenen, un nombre n d'aquestes tenen una resposta associada als predictors i un nombre m d'aquestes no. Aquests casos es poden donar quan, per exemple, és molt barat aconseguir dades dels predictors però molt car aconseguir-ne quina n'és la resposta associada.

2.3 Mètodes d'aprenentatge supervisat

En aquest apartat, s'expliquen els models més comuns dins de l'aprenentatge supervisat.

2.3.1 Regressió lineal

La regressió lineal és una eina molt útil per **predir** respostes quantitatives, és a dir, que prenen un valor numèric infinit.

Tal i com indica el seu nom, s'assumeix que les el *input* X i la resposta Y tenen una relació lineal aproximadament. Així doncs, matemàticament, es pot escriure la relació seguint l'expressió 1.

$$Y \approx \beta_0 + \beta_1 X \tag{1}$$

En aquest cas els valors de les β_0 i β_1 són dues constants que representen la intersecció i el pendent de la recta respectivament. Amb les dades d'entrenament s'estimen els valors d'aquests paràmetres.

Normalment, la resposta Y no només depèn d'un predictor X. Quan es té més d'un predictor, el mètode s'anomena **regressió lineal multiple**. El model doncs seguiria l'expressió 2.

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + ... + \beta_n X_n \tag{2}$$

2.3.2 Regressió logística

A diferència de la regressió lineal, en comptes d'intentar obtenir el valor exacte de Y, la regressió logística modela la provabilitat que Y pertanyi a una certa categoria.
Principalment, s'utilitza en problemes de classificació. L'expressió 3 es seguida pel model.

$$p(X) = \beta_0 + \beta_1 X \tag{3}$$

La raó per la qual és preferible usar una regressió logística en variables categòriques enfront a una regressió lineal és per l'aparició de provabilitats negatives o majors a 1.

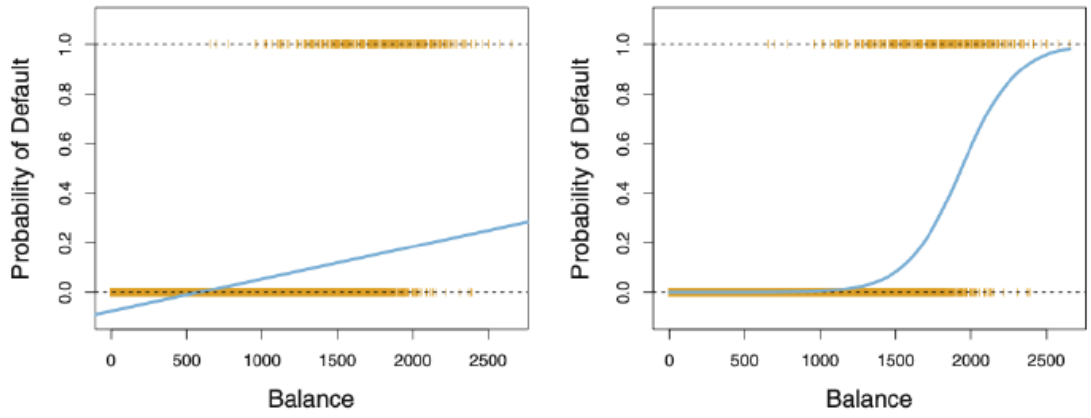

Figura 4: Regressió lineal vs. regressió logística [2]

A la Figura 4 es veu clarament un exemple. En aquest cas els valors de resposta només poden ser 0 o 1. En el cas de la regressió lineal, s'obtenen valors de provabilitat negativa, fet que no té sentit. En canvi, amb una regressió logística, la provabilitat queda acotada entre aquests dos valors.

Anàlogament, si es tenen diversos predictors, aquests també es consideren i el mètode utilitzat és la **regressió logística múltiple**. Aquest model es descriu per l'equació 4.

$$p(X) = \beta_0 + \beta_1 X + ... + \beta_n X_n \tag{4}$$

2.3.3 Anàlisi discriminant lineal (LDA)

És un mètode de classificació que prediu la provabilitat de la resposta d'estar dins d'un cert grup i està estretament relacionat amb la regressió logística. La diferència principal és que la regressió logística fa servir el mètode de la màxima provabilitat per predir els paràmetres i l'anàlisi discriminant lineal utilitza la mitjana i la variança de distribucions normals. Així mateix, la descripció de com es calculen els diferents Paràmetres de les equacions queda fora de l'abast d'aquest treball de final de grau.

A més a més, en l'anàlisi discriminant lineal no s'estudien les provabilitats de manera estàndard. Sinó que fent us del teorema de Bayes, si s'ha observat una determinada resposta calcula la provabilitat dels valors dels seus paràmetres.

D'altra banda, si no es vol assumir que la distribució és lineal sinó quadràtica, el mètode que cal utilitzar és **l'anàlisi quadràtic discriminant**.

2.3.4 K veïns més propers

No sempre es pot assumir quina serà la forma de les dades de que es disposen com en els mètodes anteriors. El mètode dels K veïns més propers és un clar exemple de un mètode no paramètric.

Aquest mètode estima la provabilitat de la resposta Y donats els predictors X i seguidament classifica una nova observació en aquell grup amb una major provabilitat de formar-ne part.
El seu funcionament és prou intuïtiu. S'explicarà a partir de l'exemple que es mostra a la Figura 5. En la part esquerra, s'ha graficat un simple conjunt de dades. Sis punts donen com a resposta taronja i els sis restants donen com a resposta blau. Així doncs, es vol predir quina serà la resposta del punt marcat amb una creu. Si es tria, per exemple, un valor del paràmetre K=3, es crearà un cercle que inclogui els tres veïns més propers al punt que es vol estudiar. En aquest cas, consisteix en dos punts blaus i un de taronja. Per tant les provabilitats estimades seran de 1/3 de ser taronja i de 2/3 de ser blau. Així doncs, el mètode predirà que la resposta del punt que es vol predir serà blau.
Per contra, en la part dreta de la Figura 5 s'ha seguit el mateix procediment per tots aquells possibles punts representats. Així doncs, si un punt està a la regió blava, es podrà predir com a resposta blava i si està en la zona taronja, es predirà com a taronja.

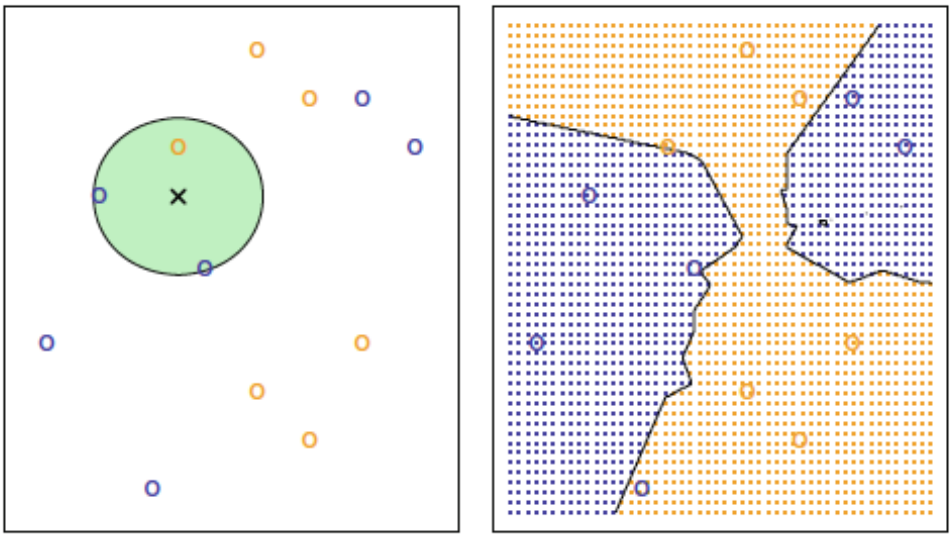

Figura 5: Representació del mètode de K veïns més propers[2]

La part més complicada és doncs, triar el paràmetre K adient. Si la K augmenta, el mètode és menys flexible i produeix que la frontera de decisió sigui pràcticament lineal. En canvi, si s'escull un valor de K molt baix, la frontera de decisió és extremadament flexible. Tal i com s'observa a la Figura 6 per una K molt petita el model és molt flexible i es creen unes petites illes que poden estar influenciades per valors atípics. En canvi, a la part dreta de la Figura 6 s'observa la tendència lineal de la frontera de decisió.
En tots dos casos, hi ha representada en una línia discontínua, per una K òptima, quina seria la frontera de decisió.

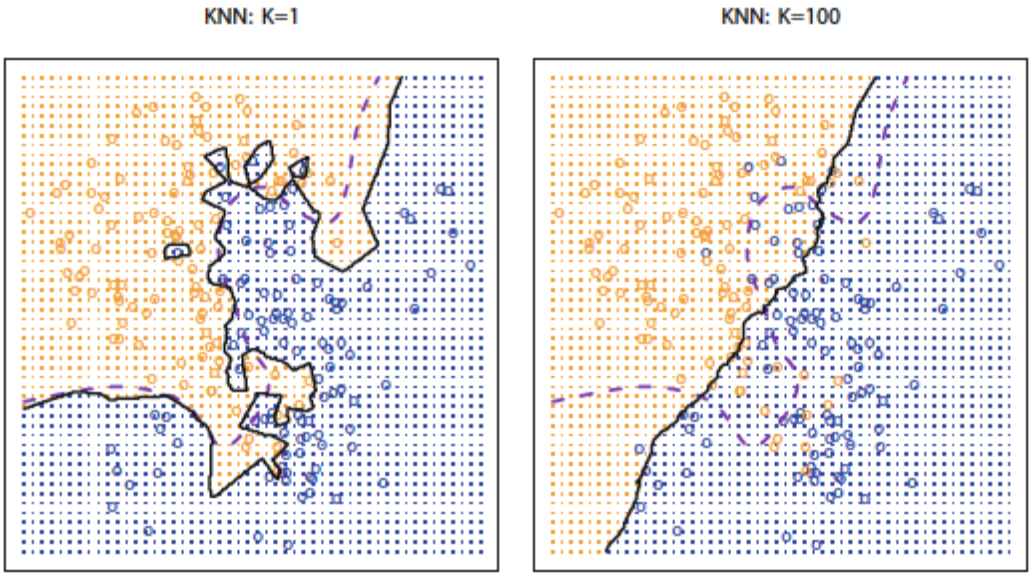

Figura 6: Representació del mètode de K veïns més propers per diferents valors de K[2]

2.3.5 Arbres de decisió

Aquest mètode comporta estratificar o segmentar l'espai de predicció en un nombre de regions més simples. Normalment, per tal de fer una predicció, s'utilitza la mitjana o la moda. Les ordres que es segueixen per tal fer les divisions de les regions recorda a un arbre, d'aquí el seu nom.

Aquests mètodes acostumen a ser simples i fàcils d'interpretar. Tot i això, normalment no acostumen a donar tan bons resultats com altres tipus de mètodes. Hi ha tres tipus d'arbres de decisió característics: *bagging*, *random forests* i *boosting*.

2.3.6 Xarxes neuronals

Aquest algorisme és un dels més comuns dins del món de la intel·ligència artificial. El seu nom es deu al fet que la seva estructura recorda a la manera de funcionar de les neurones del cervell. A grans trets, cada una de les entrades està relacionada en base a unes neurones que estan compostes per diferents pesos. Segons el valor d'aquests pesos les neurones s'activen o no fins a arribar a un resultat final. Més endavant, s'entra en detall en la implementació d'una xarxa neuronal des de zero.

2.3.7 *Support Vector Machines*

Aquest mètode permet realitzar classificacions i ha esdevingut cada vegada més popular en les últimes dècades. Tot i això, no es pot aplicar a qualsevol mena de sets de dades doncs cal que les diferents classes es puguin separar per una frontera de decisió lineal. Malgrat aquest fet, permet treballar amb un nombre elevat de dimensions.

2.4 Elecció del mètode d'aprenentatge estadístic

Com s'ha vist en l'apartat anterior, hi ha moltes possibilitats a l'hora de triar quin és el mètode idoni per a analitzar un conjunt de dades i extreure'n conclusions. És complicat respondre a la pregunta: *Quin és el millor mètode?* La resposta és depèn. La tria del mètode sempre ha d'anar lligada a les dades que es disposen i que es volen analitzar.

A la Figure 7 s'observa un resum de les diferents infraestructures que s'han exposat anteriorment.

Mètodes d'aprenentatge estadístic
Regressió lineal
Regressió logística
Anàlisi discriminant lineal
K veins més propers
Arbres de decisió
Xarxes neuronals
Support vector machines

Figura 7: Possibles mètodes d'aprenentatge estadístic

La plataforma que es vol implementar ha de ser capaç de suportar dades no parametritzades, doncs es vol realitzar una predicció sense cap assumpció prèvia de quina forma segueixen les dades. D'altra banda, tampoc interessa la simplicitat o la comprensió de la plataforma. Només cal que la xarxa faci una predicció correcta. Amb aquests premisses es poden descartar alguns dels mètodes. Així doncs, a la Figura 8 es poden veure quins mètodes queden descartats en vermell i quins segueixen sent candidats en verd.

Mètodes d'aprenentatge estadístic
Regressió lineal
Regressió logística
Anàlisi discriminant lineal
K veins més propers
Arbres de decisió
Xarxes neuronals
Support vector machines

Figura 8: Opcions més adients de mètodes d'aprenentatge estadístic per al disseny de la plataforma

De les tres opcions restants, en primer lloc es descarta el mètode de *Support vector machines* doncs aquest mètode dóna bons resultats si la frontera de decisió és lineal. Com no es vol fer

cap assumpció de les dades aquest mètode queda descartat. De les dues opcions restants, s'ha triat l'opció de les xarxes neuronals. Tot i que les dues opcions tenen alts rendiments a l'hora de realitzar prediccions, les xarxes neuronals només els cal treballar una vegada amb el conjunt de dades d'entrenament. Després, una vegada entrenada, es pot tan sols utilitzar les dades a predir.

Les xarxes neuronals són capaces d'aprendre a partir de nombrosos conjunts d'exemple i que no segueixen cap relació lineal entre ells. A més a més, a raó de la seva elevada complexitat, fa que no es vegi afectada per valors atípics.

Si bé pot semblar que el mètode de KNN és un mètode més ràpid ja que les xarxes neuronals cal entrenar-les, només cal fer-ho una vegada.

Així doncs, en els següents apartats s'explicarà com s'ha implementat una xarxa neuronal pas a pas i el seu posterior entrenament.

2.5 Programari necessari

Figura 9: Logotip C

La xarxa s'ha implementat mitjançant el llenguatge C. Si bé es cert que altres codis de programació ja tenen implementades llibreries que permeten treballar amb xarxes neuronals semidissenyades, al fer la implementació des de zero, el llenguatge C és molt més versàtil i flexible i permet accedir a memòries de baix nivell. Aquest llenguatge és el predecessor de Linux. Fou creat per Dennis Ritchie y Brian Kernighan als anys 70 [4] i va ser un dels primers llenguatges més estesos arreu del món. Després va derivar en els avui dia coneguts com a C++ i Java.

Aquest llenguatge no s'aprèn durant la carrera d'enginyeria en tecnologies industrials. Tot i que una vegada es coneix diversos llenguatges de programació n'és fàcil aprendre'n un altre, ha consistit en tot un repte personal. És per aquest motiu que el codi podria ser optimitzat per ser més curt. Aquest aprenentatge s'ha fet mitjançant la realització del curs en línia gratuït que ofereix Tutorialspoint. [8] La plataforma usada per programar en C ha estat **DEV C++**. Aquest editor de codi, permet programar tant en C++ com en C i s'ha instal·lat en un ordinador amb sistema operatiu Windows.

Figura 10: Logotip editor de codi DEV C++

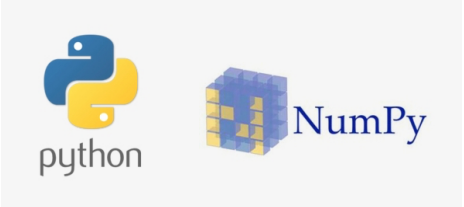

Figura 11: Logotip de Python i Numpy

Aiximateix, per a generar els conjunts de punts aleatoris s'ha emprat el llenguatge **Python**. Aquest llenguatge ha estat ensenyat a l'Escola Tècnica Superior d'Enginyeria Industrial de Barcelona. Permet l'obtenció d'un codi més senzill per generar nombres aleatoris. En aquests casos, no és d'interès tenir el control i entendre a profunditat com es generen els nombres per realitzar els diferents tests de la xarxa neuronal. S'ha utilitzat la llibreria Numpy. Aquesta llibreria permet realitzar càlculs matemàtics amb *Python* així com crear matrius i vectors. L'editor de codi utilitzat en Idle3. És un editor lliure que està disponible per gairabé tots els sistemes operatius.

Finalment, l'últim programari que s'ha emprat en el desenvolupament d'aquest treball ha estat **Matlab**. Aquest programa requereix llicència la qual la proporciona l'Escola d'Enginyeria en Tecnologies Industrials de Barcelona. Aquesta eina és molt potent per fer càlculs i a la vegada per fer gràfics que requereixen d'interpolacions. Així doncs, s'ha emprat per a representar els resultats de les xarxes neuronals.

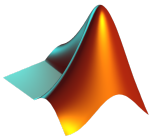

Figura 12: Logotip de Matlab

3 Disseny i implementació d'una xarxa neuronal *from scratch*

3.1 Introducció a les xarxes neuronals

Una xarxa neuronal intenta imitar el funcionament del cervell i les seves neurones. La informació arriba al nostre cervell a una neurona determinada, per exemple la ma toca alguna cosa molt calenta. Aquesta envia una serie de senyals de la ma fins al cervell. Aquest enviament es produeix a partir d'impulsos electroquímics. Si en rep suficients, la neurona es despolaritza, i per tant d'activa. Així doncs es transmet el senyal fins que al cervell s'obté la resposta d'apartar la ma.

Anàlogament, una xarxa neuronal consisteix d'un seguit d'unitats que estan interrelacionades i es comuniquen entre si enviant-se senyals segons unes connexions ponderades amb uns determinats pesos.

La idea més simple de neurona és el perceptró. El perceptró pren uns certs valors d'entrada i segons el valor de sortida s'activa o no. Aquesta resposta és binària. Aquesta resposta depèn dels anomenats pesos que caracteritzen les connexions entre les neurones. Quan s'uneixen molts perceptrons s'aconsegueix obtenir una xarxa neuronal.

Aquestes neurones estan organitzades en capes. Hi ha la capa d'entrada on l'*input* és les diferents observacions. Al mig, hi tenim les capes ocultes. N'hi poden haver tantes com sigui necessari. Finalment, hi ha la útlima capa, que és on s'obté el resultat. A la Figura 13 se'n mostra un exemple.

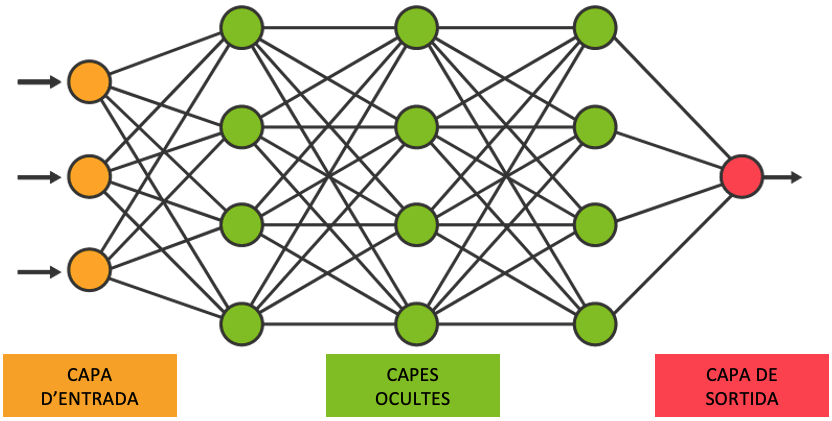

Figura 13: Xarxa neuronal amb les seves parts

Cada xarxa està formada mínim per els següents elements:

- **Neurones:** Unitats de processament.
- **Valor d'activació:** Cada neurona k té d'entrada un valor i n'emet un altre com a sortida. Y_k.

- **Pesos:** Les neurones estan connectades definides per uns pesos W_{ij}. Els subíndexs identifiquen la connexió entre la neurona i fins a la neurona j.

- **Biaix:** El biaix que es vol establir a cada neurona. Es simbolitza amb θ.

A continuació, s'explicarà més detalladament quines són totes les parts necessaries que conformen una xarxa neuronal tot seguint la xarxa neuronal programada en C.

3.2 Programació d'una xarxa neuronal

La programació de la xarxa neuronal des de 0 ha estat basada en l'article publicat a Internet per la Universitat de Birmingham [1] on s'explica com crear l'algorisme des de zero.

En un primer moment, es defineix la xarxa neuronal amb tan sols una capa oculta. Això podrà ser modificat posteriorment a convenir segons els resultats obtinguts.

En apartats anteriors s'ha explicat amb trets genèrics el funcionament d'una xarxa neuronal. Tot i això, a continuació, s'explica en detall quines són les característiques de l'algorisme que s'ha programat.
Tal i com el propi nom indica, una xarxa neuronal està formada per neurones connectades entre si. Cada una d'aquestes neurones, agafa el total de les seves entrades (IN) i en calcula una sortida determinada (OUT). La manera com es relacionen les diverses neurones ve donada per l'expressió 5.

$$OUT = f(\sum_{i=1}^{N} W_i IN_i - \theta) \tag{5}$$

La funció 6 és l'equació f d'activació. Aquesta és una funció graó. Per tant els valors de soritda de les neurones poden prendre valors binaris.

$$f(\mu) = \left.\begin{array}{l} 1 -> \mu \geq 0 \\ 0 -> \mu < 0 \end{array}\right\} \tag{6}$$

Per tant, si s'analitza l'equació 5, la neurona s'activa quan es supera el valor llindar θ.

Com es tracta d'una funció graó, produeix problemes al derivar-la. És per aquest motiu que cal trobar una funció que s'aproximi el màxim a la funció graó. Se'n poden aplicar diverses tot i que la més usada i amb millors resultats és la funció sigmoide.

- **Funció sigmoide**
  La funció sigmoide 7, també coneguda com a corba logística, permet acotar els valors dels resultats en un rang entre 0 i 1. Es mostra una representació a la Figura 14 A més a més també és derivable.

$$OUT = \frac{1}{(1 + e^{-IN})} \tag{7}$$

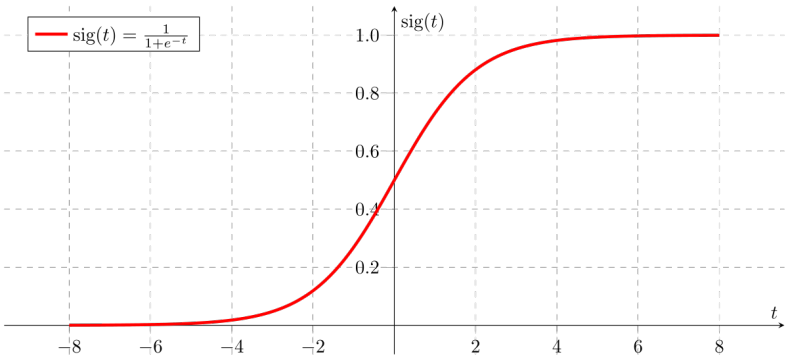

Figura 14: Funció sigmoide. Font: Wikipedia commons

A continuació, cal definir una funció que atribueixi valors aleatoris als pesos W de les diferents neurones i les inicialitzi. Aquesta funció s'aconsegueix amb la funció *rand*().

Cada una de les connexions entre neurones té un pes W el qual es multiplica per la suma de les seves entrades. Per tant si es té una primera capa oculta amb *NumInput* de neurones, la sortida de la primera capa oculta s'obté de la següent expressió:

$$SumH = Input[i]Weight[i] \tag{8}$$

En la Figura 15 es mostra un esquema del que s'està analitzant.

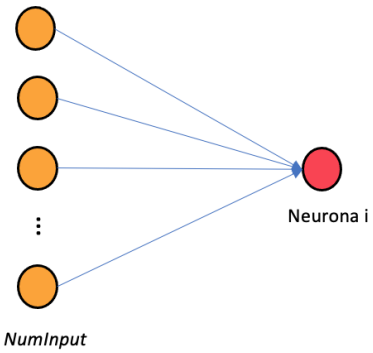

Figura 15: Xarxa neuronal sense capes ocultes i amb una neurona i de sortida

D'aquesta expressió el *Weight* $[i]$ és el pes que comunica cada una de les entrades amb la neurona *i*. A més a més, cada una de les neurones tindrà un biaix que es defineix al principi. Aquest biaix es guarda a la matriu *Weight* en la primera fila.

Tot i això, normalment hi haurà capes entremig que no estaran compostes de només una neurona. És per això que cal afegir un nou índex a la variable *Weight*. Per tant, es convenient crear la matriu *WeightIH*$[i][j]$, que indica el pes entre l'*input* de la neurona *i* fins a la neurona *j*.
Així doncs, a partir de l'expressió 8 es pot crear un bucle que recorri totes les neurones i d'entrada i calculi la sortida de les neurones *j*. Finalment, es calcula mitjançant la funció sigmoide 7 per tal de donar activació o no a les neurones.

Primer s'estableix el biaix en el vector SumH per cada una de les j neurones. Aquest, com ja s'ha mencionat anteriorment, està guardat en la primera fila de la matriu $WeightIH[i][j]$. Seguidament, es recorren totes les neurones d'entrada multiplicant els valors pels diferents pesos de les connexions. Finalment, per cada neurona j es calcula amb la funció sigmoide l'activació o no de la neurona. A continuació es mostra un exemple de codi:

```
SumH[j]=WeightIH[0][j];
for (i=1 ; i<=NumInput ; i++){
    SumH[j]+=Input[i]*WeightIH[i][j];
}
Hidden[j]=1/(1+exp(-SumH[j]))
```

A aquest bucle cal afegir-li un altre bucle que recorri totes les neurones j i així calculi els outputs de totes les neurones de la capa oculta.
El mateix codi es pot aplicar per calcular l'*output* del sistema. Simplement cal canviar les variables del sistema. En l'exemple que es mostra a continuació hi ha els pesos de la capa oculta fins a l'output.

```
for (k=1; k<=NumOutput ; k++){
    SumO[k]=WeightHO[0][k];
    for (j=1 ; j<=NumHidden ; j++){
        SumO[j]+=Hidden[j]*WeightHO[j][k];
    }
    Output[k]=1/(1+exp(-SumO[k]))
}
```

Per a fer-ho més entenedor, a la Figura 16 es mostra un esquema de com seria la xarxa neuronal amb la capa oculta introduïda. Cal destacar que totes les neurones d'una capa estan connectades amb totes les neurones de la següent capa. A l'esquema de la Figura 16 no es mostra així, per tal de fer-ho més entenedor.

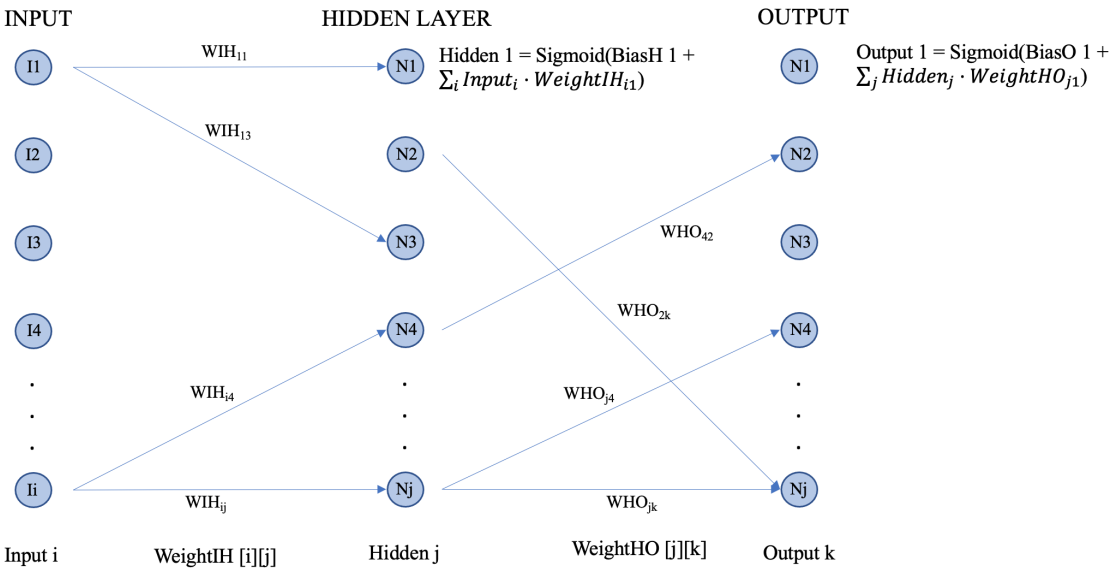

Figura 16: Esquema senzill xarxa neuronal

Normalment, per tal d'entrenar la xarxa neuronal i després procedir a fer-ne els càlculs corresponents, es disposa de diversos conjunts de dades. Cada un d'aquest conjunt estarà format per un $Input[i]$ i un $Target[k]$. Així doncs, s'anomena *NumPattern* al nombre p d'entrades amb el qual es pot entrenar el sistema. Aquestes entrades també s'anomenen observacions.

Però, com s'entrena aquesta xarxa i quan ha de deixar d'iterar i per tant es poden considerar els resultats com a correctes?

La xarxa neuronal s'entrena minimitzant els errors dels *outputs* obtinguts comparats amb el *target* esperat. Per calcular aquest error, s'utilitza l'expressió de la suma residual de quadrats. Aquest error s'inicialitza en 0 i va variant el seu valor per cada sortida k a partir de l'expressió següent:

```
Error=0.0;
// Per cada una de les sortides s'aplica:
Error += 0.5 * (Target[p][k] - Output [p][k]) * (Target[p][k] - Output [p][k]);
```

La següent etapa que cal plantejar és l'aprenentatge de la xarxa neuronal, és a dir, la modificació automàtica dels diferents pesos.

El procés d'entrenament es fa en dos passos. Aquests s'anomenen *feed-forward* i *back-propagation*. El primer avança en el sentit de la xarxa neuronal, és a dir, des de l'*input* cap a la sortida. En canvi, el *back-propagation* avança en sentit contrari. A continuació, s'analitzen els dos procediments amb més profunditat:

- ***Feed-forward***

El sentit d'anàlisi de la xarxa neuronal és d'entrada fins a sortida. Seguint les passes mencionades anteriorment, s'aconsegueix calcular la matriu $Target[p][k]$ on cal recordar que p és l'observació que s'està analitzant i k la neurona de sortida que s'estudia. Així doncs, simplement multiplicant els pesos de les connexions pels diferents valors de les neurones es pot obtenir una predicció de resposta. Aquesta predicció diferirà del *output* que realment s'esperava. És en aquest moment on entra en joc la *back-propagation*.

- ***Back-propagation***

  El sentit en el que es va en aquest cas és des de la resposta fins a la primera capa. Una vegada es té la predicció de la xarxa es pot calcular l'error que es comet i recalcular els pesos. Més endavant s'explica més detalladament el procediment que es segueix.

Una vegada s'ha aconseguit una predicció, aquesta cal comparar-la amb la resposta real per a reduir l'error. Per això, es calcula les derivades parcials *dError/dWeight* i es varia els valors dels pesos un valor *DeltaWeight* en la direcció que redueixi de major manera l'error. Així doncs, es recalcula la variació del pes amb l'expressió següent:

```
DeltaO[k]=(Target[p][k]-Output[p][k]) * Output [p][k] * (1.0 - Output [p][k])
```

Aquesta variació es calcula per cada una de les neurones i per cada una de les capes. Així doncs, cal tornar enrere els bucles per calcular els errors i calcular els nous pesos.

Aquest procediment cal fer-lo fins que l'error sigui menor a un valor desitjat. Aquest error mai serà 0 doncs com s'utilitza la funció sigmoide aquesta té dues asímptotestotes en y=0 i y=1. Aquest error es defineix a l'iniciar la xarxa neuronal. També pot passar que mai s'aconsegueixi un error suficientment petit i el programa faci totes les iteracions.

A més a més, tot el codi està englobat en un últim bucle que calcula èpoques. Aquest reodrena de manera aleatòria les p observacions. De no ser així, la xarxa neuronal sempre seguiria el mateix patró de càlcul. Aquest fet és especialment útil quan es tenen poques dades d'entrenament de la xarxa.

La xarxa neuronal dissenyada deixa llibertat a l'usuari d'escollir el nombre de neurones a l'entrada, la capa oculta i la sortida. També cal indicar el nombre total de conjunts de dades amb els quals s'entrenarà la xarxa neuronal.

En els següents apartats es mostrarà l'entrenament de la xarxa amb diferents conjunts de dades. En tots els casos, l'elecció del nombre de capes s'ha realitzat mitjançant el mètode de prova-error. Primer s'ha començat amb tan sols una capa oculta i s'ha anat augmentant el nombre de capes ocultes fins que s'ha obtingut el resultat desitjat.

4 Avaluació de l'implementació dissenyada

4.1 Porta XOR enters

Per tal de provar de manera simple la xarxa neuronal dissenyada s'ha definit una porta XOR. Les variables de les entrades es poden observar a la Taula 1.

Input 1	Input 2	Target
0	0	0
0	1	1
1	0	1
1	1	0

Taula 1: Porta lògica XOR

Les variables lliures de la xarxa s'han definit als valors següents:

- **NumIN=2** : S'entren dos inputs
- **NumHid=3** : Hi ha tres neurones a la capa oculta
- **NumOut=1** : Es vol tan sols un nombre com a output
- **NumPat=4** : Hi ha quatre combinacions diferents de valors
- **Nombre de capes ocultes=1** : Al ser un conjunt de dades ja s'obtenen bons resultats amb tan sols una capa oculta.
- **Paràmetres:** eta=0.5 alpha=0.9 i smallwt=0.9
- **Error=0.0004**

Cal remarcar, que aquest conjunt de paràmetres s'han definit en base al mètode de prova i error. Primerament, s'ha començat amb poques neurones a la capa oculta i s'ha anat agumentant el número de neurones per tal d'obtenir un error menor a 0,0004 en el menor nombre d'èpoques possible.

Tot el codi sencer per a l'entrenament d'aquesta xarxa es pot visualtizar aquí: XARXA XOR

A continuació, a la Figura 17 s'observen els resultats obtinguts d'entrenar aquesta xarxa neuronal amb els valors de la Taula 1.

```
Epoch 0     :   Error = 0.587471
Epoch 100   :   Error = 0.343867
Epoch 200   :   Error = 0.009983
Epoch 300   :   Error = 0.003573
Epoch 400   :   Error = 0.002131
Epoch 500   :   Error = 0.001506
Epoch 600   :   Error = 0.001159
Epoch 700   :   Error = 0.000940
Epoch 800   :   Error = 0.000789
Epoch 900   :   Error = 0.000681
Epoch 1000  :   Error = 0.000597
Epoch 1100  :   Error = 0.000531
Epoch 1200  :   Error = 0.000478
Epoch 1300  :   Error = 0.000435

NETWORK DATA - EPOCH 1397

Pat     Input1          Input2          Target1         Output1
1       0.000000        0.000000        0.000000        0.013063
2       1.000000        0.000000        1.000000        0.985879
3       0.000000        1.000000        1.000000        0.986598
4       1.000000        1.000000        0.000000        0.015806

Goodbye!
```

Figura 17: Resultats XOR amb 0 i 1

Tal i com s'observa a la Figura 17 l'algorisme atura l'entrenament de la xarxa neuronal a l'època 1397 on l'error ja és suficientment petit. A més a més, si es compara la columna *Target1* amb la de l'*Output1* veiem que per l'última iteració de l'algorisme prediu de manera correcta la resposta.

El codi té previst enregistrar en un arxiu de text l'error comès en cada època d'entrenament. Així doncs, se n'ha fet un anàlisi d'aquest fitxer mitjançant l'eina Excel per tal de veure'n l'evolució. A la Figura 18 es pot veure com l'error en un primer moment és prou elevat i descendeix notablement a partir d'aproximadament l'època número 100.

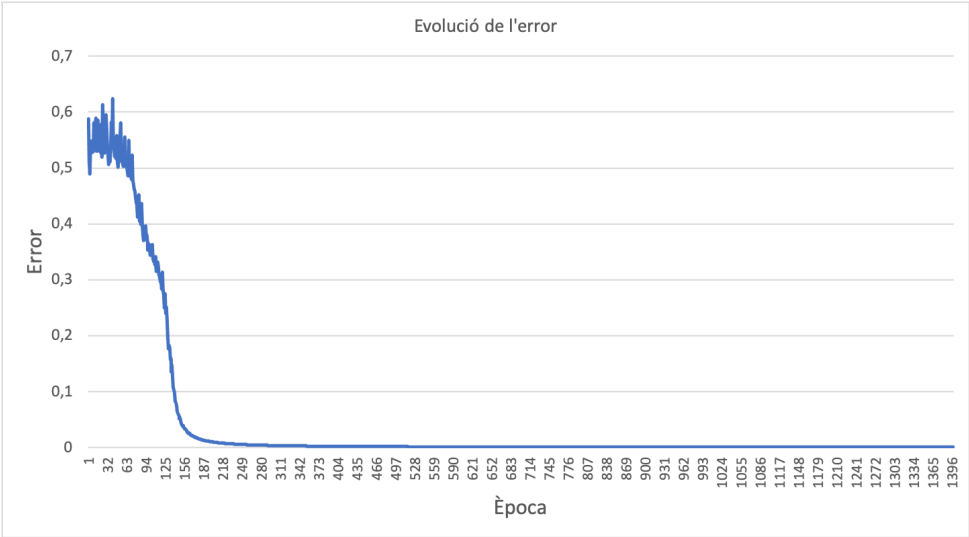

Figura 18: Evolució de l'error en l'entrenament

A continuació s'ha validat aquesta xarxa neuronal una vegada ja entrenada. El codi en C crea un arxiu de text amb cada una de les matrius dels pesos tal i com han quedat una vegada realitzat l'entrenament. Així es poden visualitzar i poden ser usades en futures ocasions.
Les dues matrius dels pesos es mostren a la Figura 19.

```
MATRIUIH
-0.000000       2.868438        7.092114        0.319102
0.000000        -6.774828       -4.652837       0.395935
0.000000        -7.145097       -4.757971       0.322815

MATRIUHO
0.000000        -1.813541
0.000000        -10.540757
0.000000        9.991955
0.000000        -4.353727
```

Figura 19: Matrius de pesos una vegada entrenada la xarxa neuronal

En aquest cas, per tal de testejar la xarxa neuronal i comprovar que aquesta funciona amb nous valors s'ha utilitzat d'entrada els valors que es mostren a la Taula 2.

Input 1	Input 2	Target
0.9	0.9	0
0.9	0	1
0	0	0
0	0.9	1

Taula 2: Valors test de la xarxa

Amb aquests valors, a la Figura 20 es pot observar els resultats obtinguts de realitzar aquest test a la xarxa neurnonal programada. Tal i com s'observa, la columna *Output1* s'aproxima clarament a la columna *Target1*. A més a més, l'error més gran comès és de 0,001847 al calcular la predicció pels valors 0 i 0.9. Així doncs es pot concloure que la xarxa neuronal té un bon funcionament per una porta lògica XOR.

```
Error1    = 0.001624

Error2    = 0.001702

Error3    = 0.001787

Error4    = 0.001847

Input1          Input2          Target1         Output1
1       0.900000        0.900000        0.000000        0.049483
2       0.900000        0.000000        1.000000        0.987540
3       0.000000        0.000000        0.000000        0.013054
4       0.000000        0.900000        1.000000        0.989008

Goodbye!
```

Figura 20: Resultats test xarxa neuronal

4.2 Porta XOR decimal

La xarxa neuronal inicialment ha estat dissenyada com a una porta XOR, però els valors d'entrada amb els quals s'ha entrenat són només combinacions de zeros i uns. Per tal d'anar un pas

més enllà, mitjançant el programari *Python* s'han generat 150 punts aleatoris diferents amb dues coordenades amb valors compresos entre 0 i 1. El codi utilitzat es pot observar a la Figura 43. Aquest script genera dos arxius de text diferents. En un primer crea 150 entrades de nombres amb coordenades x i y i en calcula el seu resultat segons una porta lògica XOR. Després, genera 50 parelles de nombres més per tal de poder realitzar una comprovació de la xarxa neuronal. Amb un senzill codi de Matlab, que es mostra a la Figura 44 s'han graficat els punts generats aleatòriament per tal de veure que estaven uniformement distribuïts i per verificar que el codi de Python calcula correctament la sortida de la porta lògica. A la Figura 21 i a la Figura 22 es pot observar com queden distributs els punts en els dos plans z=0 i z=1 i es compleixen també les condicions d'una porta XOR.

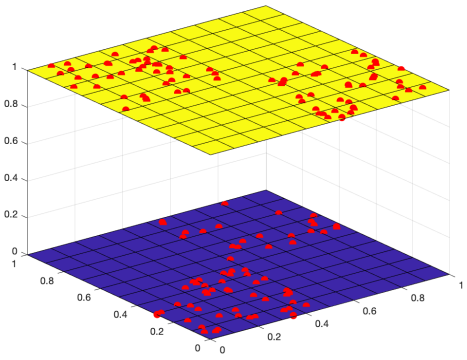

Figura 21: Grafic 3D amb els punts d'entrenament de la xarxa

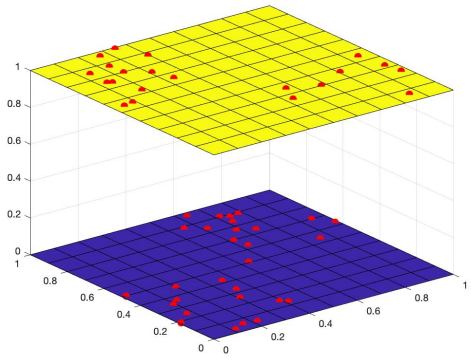

Figura 22: Grafic 3D amb els punts de testeig de la xarxa

A continuació, s'ha modificat el codi utilitzat en l'apartat anterior per tal que els *Inputs* i els *Targets* es llegeixin directament del fitxer de text. Primer de tot s'ha entrenat la xarxa amb els punts creats a l'arxiu d'entrenament. Tal i com s'ha fet en l'apartat anterior primer els paràmetres de la xarxa es defineixen en base al mètode prova-error. Sempre es comença a estudiar pels casos més senzills. Així doncs es distingeixen dos casos:

- **1 capa oculta**
  En aquest cas, s'ha dissenyat la xarxa neuronal amb tan sols una capa oculta formada per quatre neurones. El nombre de neurones òptim s'ha aconseguit amb el mètode prova/error. A continuació, es detallen els paràmetres òptims amb els quals s'ha entrenat la xarxa neuronal:

  - **NumIn=2**: L'entrada està formada per dos nombres
  - **NumHid=4**: Hi ha 4 neurones a la capa oculta
  - **NumOut=1**: El sistema tan sols ha de predir una resposta
  - **NumPat=150**: Hi ha 150 nombres generats aleatòriament
  - **Nombre de capes ocultes=1**: En una primera fase, s'estudia la xarxa neuronal amb només una capa oculta.
  - **paràmetres**: eta=0.1 alpha=0.3 i smallwt=0.5

– **Error**: 0.0004

En aquest cas, el programa realitza les 100000 iteracions doncs no s'aconsegueix un error menor a 0,0004. Així doncs a la Figura 23 s'observa el resultat dels deu primers nombres d'entrenar la xarxa amb 150 punts aleatoris entre 0 i 1.

El codi sencer per aquest cas es pot observar aquí: XARXA PUNTS

```
Epoch 99300 :   Error = 7.030362
Epoch 99400 :   Error = 7.031035
Epoch 99500 :   Error = 7.033249
Epoch 99600 :   Error = 7.025982
Epoch 99700 :   Error = 7.019248
Epoch 99800 :   Error = 7.027381
Epoch 99900 :   Error = 7.025552

Goodbye!


NETWORK DATA - EPOCH 100000

Pat     Input1          Input2          Target1         Output1
1       0.515738        0.751417        0.000000        0.828353
2       0.511377        0.350624        1.000000        0.857528
3       0.081484        0.769625        1.000000        0.860176
4       0.838845        0.096923        1.000000        0.855166
5       0.273330        0.513477        1.000000        0.841680
6       0.673115        0.834089        0.000000        0.026268
7       0.298533        0.041117        0.000000        0.000000
8       0.540529        0.649128        0.000000        0.817881
9       0.169460        0.828584        1.000000        0.842832
10      0.957699        0.722509        0.000000        0.021186
```

Figura 23: Resultats d'entrenar la xarxa neuronal amb punts aleatoris

Així doncs, l'algorisme s'atura en l'iteració 100000. L'error comès és significativament superior respecte a la porta lògica XOR. De totes maneres, es pot considerar l'entrenament com a correcte ja què si s'arrodoneix l'*Output* a la unitat més propera s'aconsegueix el *Target*.

Anàlogament a l'apartat anterior, també s'ha fet un anàlisi de l'evolució de l'error. Aquest es mostra a la Figura 24 el qual s'ha realitzat mitjançant *Excel*. Es pot observar com al principi és prou elevat i cap a l'iteració 1000 s'estabilitza en 7.

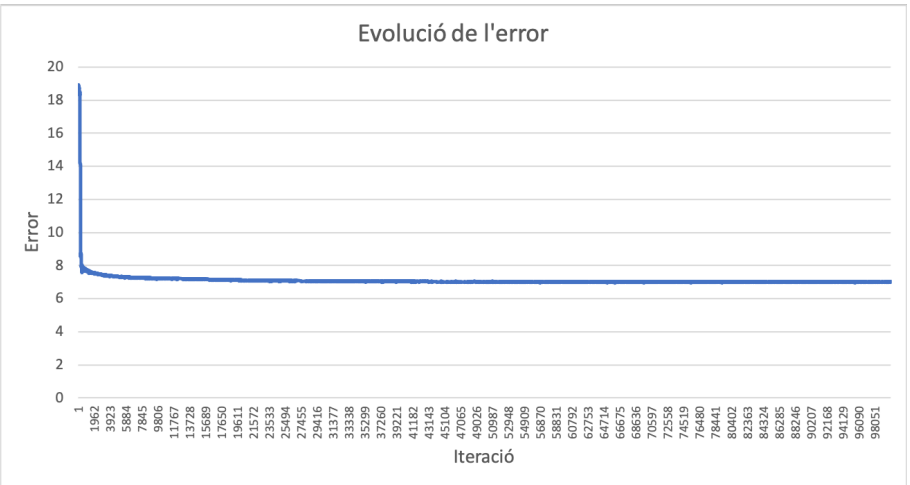

Figura 24: Evolució de l'error en l'entrenament de la xarxa neuronal

D'altra banda, per veure-ho de manera més visual, s'han creat dos gràfics amb els *Inputs* i l'*Output*. Els resultats que s'obtenen de la última iteració es mostren en les Figures 25 i 26.

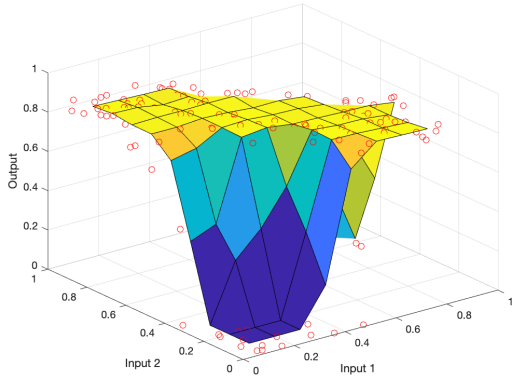

Figura 25: Grafic 3D amb els punts d'entrenament de la xarxa

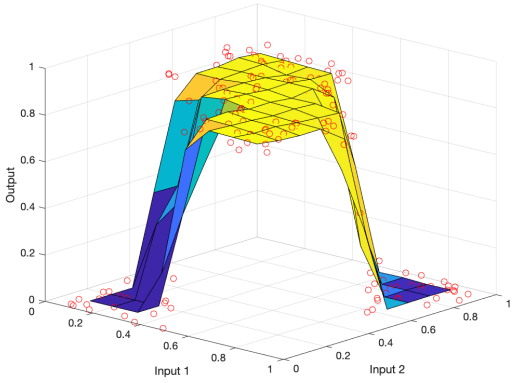

Figura 26: Grafic 3D amb els punts d'entrenament de la xarxa

Les Figures 25 i 26 han estat construides mitjançant Matlab amb el codi que es pot veure a la Figura 44. Així doncs, es pot veure com la xarxa prediu perfectament aquells punts on les dues coordenades són altes o baixes. En canvi, per aquelles coordenades on una de les variables es major a 0,5 i l'altra és menor a 0,5 la xarxa neuronal no s'apropa tant al valor de 1. A més a més, els valors intermitjos és on es crea una major confusió, tal i com era d'esperar.

A continuació, s'ha fet una comprovació d'aquesta xarxa neuronal amb un nou set de dades de 50 combinacions de nous punts. Els resultats de la xarxa neuronal per aquest conjunt de dades també s'han representat mitjançant el programa de Matlab 44 per tal de tenir una representació del seu comportament. Aquests es mostren a la Figura 27 i la Figura 28.

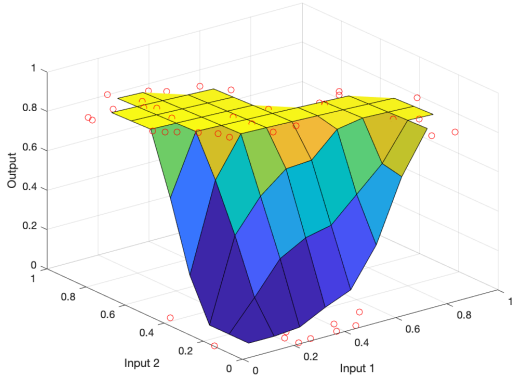

Figura 27: Grafic 3D amb els punts de test de la xarxa

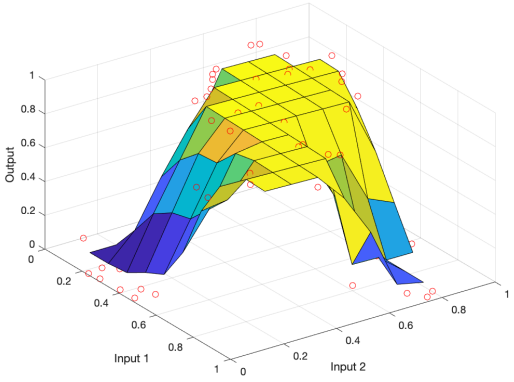

Figura 28: Grafic 3D amb els punts de test de la xarxa

Per tal d'obtenir més específicament el percentatge d'encert en la predicció de la xarxa, s'ha suposat que una predicció pren per valor 1 si el valor predit per la xarxa és major a 0,5. En canvi, la predicció prendrà per valor 0 si és menor a 0,5. Així doncs, comparant-ho també amb el resultat esperat, s'ha pogut crear la taula de confusió 3.

Target ——— **Prediction**	**1**	**0**
1	95,65%	37,04%
0	4,35%	62,96%

Taula 3: Matriu de confusió dels resultats del test de la xarxa neuronal

Així doncs, s'observa com la xarxa neuronal prediu molt millor les respostes que són 1 que no pas les respostes que són 0. De totes maneres, es pot afirmar que la xarxa és útil a l'hora de fer predicció doncs s'obtenen uns millors resultats que no pass fent-ho aleatòriament.

El següent pas és crear una nova capa oculta per mirar si aquests resultats es poden millorar, tot i que, a priori, poden ser acceptats.

- **2 capes ocultes**
  L'estructura és la mateixa que en les xarxes neuronals presentades anteriorment, simplement que se li afegeix una capa oculta de neurones. En la Figura 30 es mostra un esquema de la xarxa neuronal utilitzada i a continuació se'n detallen els paràmetres característics:
  - **NumIn=2**: L'entrada està formada per dos nombres
  - **NumHid1=4**: Hi ha 4 neurones a la primera capa oculta
  - **NumHid2=4**: Hi ha 4 neurones a la segona capa oculta
  - **NumOut=1**: El sistema tan sols ha de predir una resposta
  - **NumPat=150**: Hi ha 150 nombres generats aleatòriament

- **Nombre de capes ocultes=2**: S'augmenta el nombre de capes ocultes per tal d'intentar millorar el funcionament de la xarxa neuronal
- **paràmetres**: eta=0.1 alpha=0.5 i smallwt=0.5
- **Error**: 0.0004

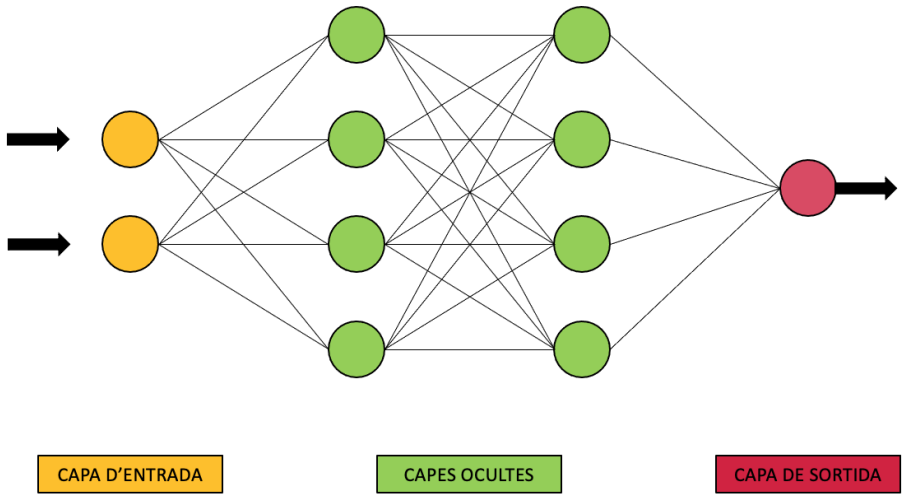

Figura 29: Esquema xarxa neuronal dues capes

Analògament a la xarxa amb una capa, el programa realitza les 100000 iteracions doncs no s'aconsegueix un error menor a 0,0004. Així doncs, a la Figura 30 s'observa el resultat dels deu primers nombres d'entrenar la xarxa amb els mateixos 150 punts aleatoris entre 0 i 1 utilitzats en la xarxa amb una capa oculta.

El codi sencer es pot observar aquí: XARXA PUNTS DUES CAPES

```
Epoch 99500 :   Error = 6.917952
Epoch 99600 :   Error = 6.781226
Epoch 99700 :   Error = 6.865141
Epoch 99800 :   Error = 6.818536
Epoch 99900 :   Error = 6.813885

Goodbye!


NETWORK DATA - EPOCH 100000

Pat     Input1          Input2          Target1         Output1
1       0.515738        0.751417        0.000000        0.860700
2       0.511377        0.350624        1.000000        0.843960
3       0.081484        0.769625        1.000000        0.828867
4       0.838845        0.096923        1.000000        0.843965
5       0.273330        0.513477        1.000000        0.802431
6       0.673115        0.834089        0.000000        0.041967
7       0.298533        0.041117        0.000000        0.000000
8       0.540529        0.649128        0.000000        0.877293
9       0.169460        0.828584        1.000000        0.839139
10      0.957699        0.722509        0.000000        0.032147
```

Figura 30: Resultats d'entrenar la xarxa neuronal amb punts aleatoris

Per tant, tal i com ja s'ha indicat, l'algorisme s'atura a l'iteració 100000. L'error comès és semblant a l'anterior tot i que lleugerament inferior. Concretament s'aconsegueix un error de 6,8138. Per tant, s'ha procedit com en l'apartat anterior i s'ha arrodonit la predicció. Si el valor és superior a 0,5 s'ha considerat un 1 de resposta i si és inferior s'ha considerat un 0. Es pot observar doncs com la xarxa prediu prou bé.

En aquest cas, també s'ha fet un analisi de l'evolució de l'error. Aquest és prou elevat en un primer moment però al valotant de l'iteració 1000 decau fins un valor de 7 aproximadament reduint-se fins a 6,813885. Aquest fenomen es mostra a la 31 realitzada mitjançant el programari *Excel*.

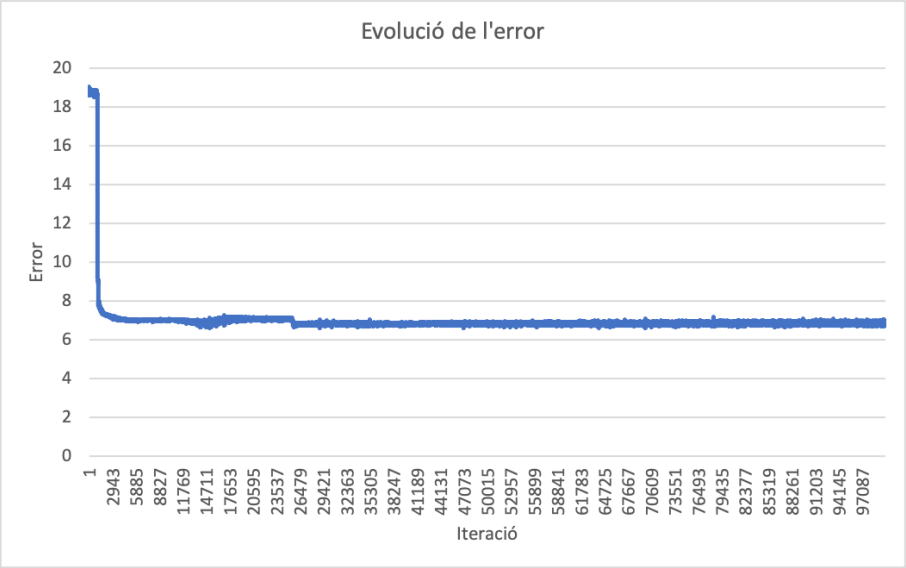

Figura 31: Evolució de l'error en l'entrenament de la xarxa neuronal

A més a més, mitjançant el mateix programa de *Matlab* que es pot observar a la Figura 44, també s'ha representat les prediccions del programa. Aquestes s'observen a les Figures 32 i 33. Així doncs, es veu com queda una forma similar a l'apartat anterior.

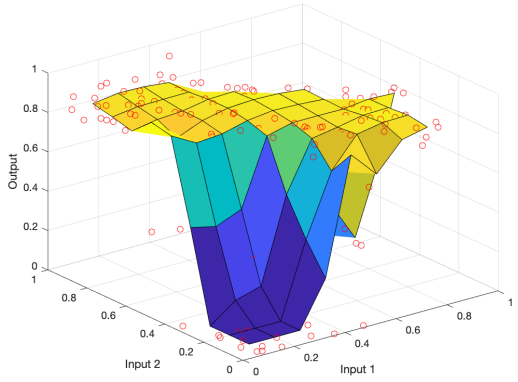

Figura 32: Grafic 3D amb els punts de test de la xarxa

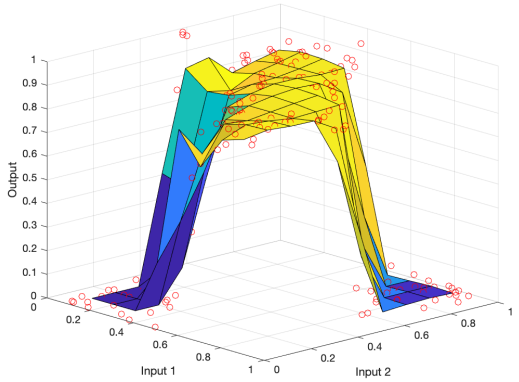

Figura 33: Grafic 3D amb els punts de test de la xarxa

A continuació, s'ha fet una comprovació d'aquesta xarxa neruonal amb el mateix conjunt de dades utilitzat anteriorment amb aquesta finalitat. Aquests, també s'han representat mitjançant el codi de *Matlab* 44. Es poden observar a les Figures 34 i 35.

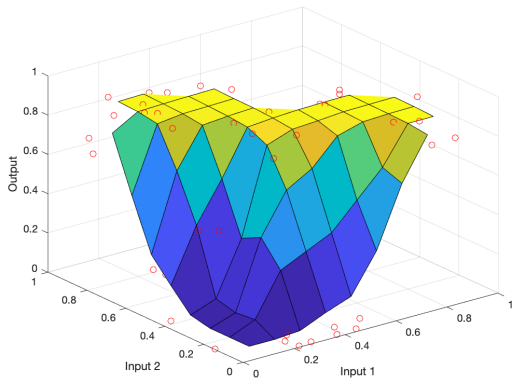

Figura 34: Grafic 3D amb els punts de test de la xarxa

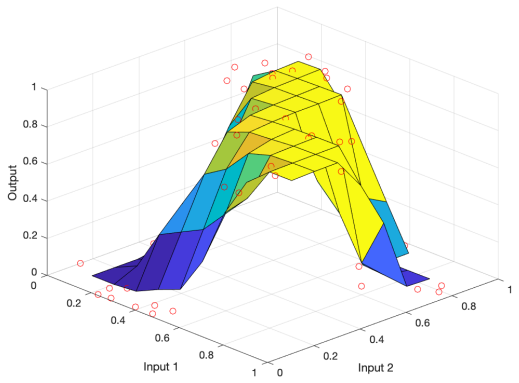

Figura 35: Grafic 3D amb els punts de test de la xarxa

A més a més, per tal d'obtenir més detalladament el percentatge d'encert de la xarxa neuronal amb dues capes ocultes. Així doncs, s'ha arrodonit la predicció de la xarxa a la unitat més propera. Els resultats es poden observar a la Taula 4.

Si es compara aquests valors amb la xarxa de tan sols una capa, es pot afirmar que aquesta prediu millor per a valors on s'espera un 0 però en canvi té un comportament pitjor quan la resposta és 1.

Així doncs, en global, és preferible utilitzar la xarxa neuronal amb dues capes doncs s'ob-

Target ——- Prediction	1	0
1	73,08%	23,08%
0	26,92%	76,92%

Taula 4: Matriu de confusió dels resultats del test de la xarxa neuronal

tenen millors resultats per qualsevol de les dues prediccions.

4.3 Predicció de correlacions 6 punts

Finalment, una vegada s'ha comprovat el correcte funcionament de la xarxa, s'ha realitzat un últim entrenament més encarat a l'objectiu final de la xarxa. Per tant, la xarxa haurà de predir si existeix alguna correlació entre sis punts compresos entre 0 i 1. Per a fer-ho s'han generat 100 conjunts de punts diferents. D'aquests cent grups, 50 són punts completament aleatoris i per tant no presenten cap correlació. Així doncs, la xarxa ha de predir un 0. El codi de *Python* utilitzat per generar aquest conjunt es pot observar a la Figura 45. També s'han generat conjunts de punts compresos entre 0 i 1 que formin part de rectes aleatòries que passen per l'origen de coordenades. De cada una de les rectes s'obtenen 6 punts amb dues coordenades. Aquest codi també s'observa a la Figura 45. Seguidament, s'ha ordenat de manera aleatòria cada un dels conjunts de dades mitjançant el codi de *Python* que es mostra a la Figura 46.

Cada línia dels fitxers de text conté informació sobre 6 punts i la predicció esperada segons la codificació següent:

- **1:** punts en una mateixa recta
- **0:** punts dispersos

Així doncs els sis primers nombres corresponen a les coordenades x dels punts i les sis darreres a les coordenades y. L'últim dígit és la resposta esperada de la xarxa neuronal.

En aquest cas, els paràmetres de la xarxa també han estat triats en base al mètode de prova/error. Els valors utilitzats han estat els següents:

- **NumIn=12**: L'entrada està formada per sis punts amb dues coordenades cada un
- **NumHid1=12**: Hi ha 12 neurones a la primera capa oculta
- **NumHid2=12**: Hi ha 12 neurones a la segona capa oculta
- **NumOut=1**: El sistema tan sols ha de predir una resposta
- **NumPat=75**: Hi ha 75 nombres generats aleatòriament
- **Nombre de capes ocultes=2**
- **paràmetres**: eta=0.25 alpha=0.3 i smallwt=0.5
- **Error**: 0.0004

En aquest cas, s'ha començat directament amb el cas que la xarxa neuronal tingui dues capes doncs tal i com s'ha observat en l'apartat anterior s'obtenen resultats millors. A més a més, no s'ha prosseguit a afergir-hi més capes doncs els resultats obtinguts fan una bona predicció. A la Figura 36 s'observa el resultat dels cinc primers nombres d'entrenament de la xarxa neuronal. La xarxa neuronal s'atura en l'època 549 doncs l'error comès és inferior a 0,004.

El codi sencer es pot observar aquí: XARXA CORRELACIÓ DUES CAPES

```
Epoch 0     :   Error = 9.934235
Epoch 100   :   Error = 0.061857
Epoch 200   :   Error = 0.017469
Epoch 300   :   Error = 0.009355
Epoch 400   :   Error = 0.006174
Epoch 500   :   Error = 0.004529

Goodbye!


NETWORK DATA - EPOCH 549

Pat     Input1          Input2          Input3          Input4          Input5          Input6          Input7
        Input8          Input9          Input10         Input11         Input12         Target1         Output1

1       -0.001365       -0.073253       0.018600        0.998451        -0.015759       -0.845953       0.010539
        0.565716        0.002437        0.130809        -0.013238       -0.710610       1.000000        0.999033

2       -0.000589       -0.420845       -0.000734       -0.525196       0.000962        0.687951        -0.000028
        -0.020372       -0.001325       -0.947093       -0.000853       -0.609813       1.000000        0.999156

3       0.000396        0.141324        -0.000494       -0.176350       -0.002374       -0.847860       0.002096
        0.748534        0.002465        0.880094        0.000171        0.060993        1.000000        0.998717

4       0.317902        0.271124        0.345736        0.623485        0.701559        0.561747        0.765626
        0.920686        0.381890        0.433705        0.722575        0.903035        0.000000        0.003089

5       0.080746        0.262288        0.623782        0.856420        0.255406        0.761909        0.582913
        0.457695        0.186237        0.088497        0.550272        0.419311        0.000000        0.006349
```

Figura 36: Resultats d'entrenar la xarxa neuronal

Així doncs, també s'ha realitzat un anàlisi de l'evolució de l'error. Aquest es redueix fins a 0,004 tal i com es desitja. Així es mostra a la Figura 37 realitzada amb *Excel*.

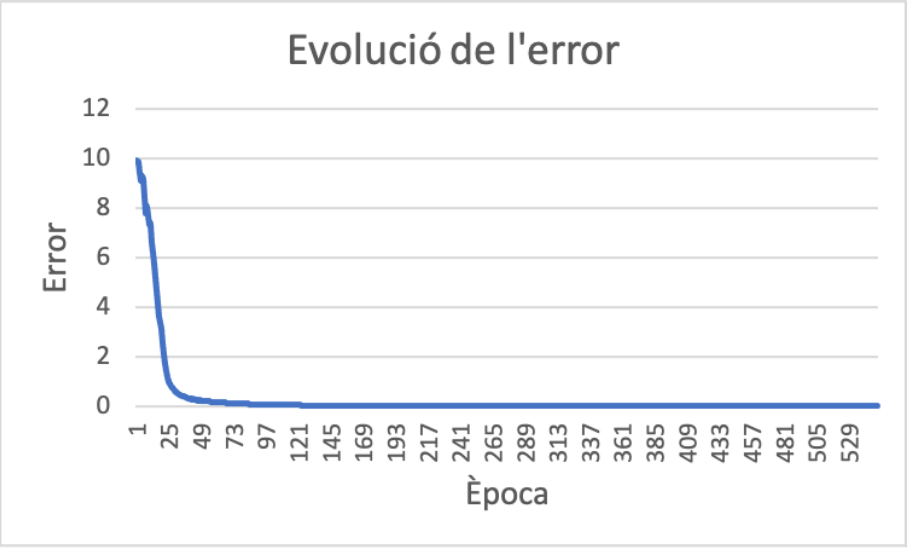

Figura 37: Evolució de l'error durant l'entrenament

Així doncs, l'entrenament de la xarxa neuronal s'ha donat per bo i es produeix el *testing* de l'algorisme.

Com ja s'ha vist en els casos plantejats anteriorment, la xarxa neuronal no prediu exactament els 1 i els 0. Així doncs, seguint el mateix criteri, si un nombre és major a 0,5 s'ha considerat com un 1 i si és inferior a 0,5 s'ha considerat 0. Per tant, la matriu de confusió resultat es pot observar a la Taula 5.

Target ——- Prediction	**1**	**0**
1	100%	0%
0	0%	100%

Taula 5: Matriu de confusió dels resultats de la comprovació de la xarxa neuronal

Per tant aquesta matriu es pot afirmar que prediu bé un 100% dels casos estudiats.

4.4 Predicció de correlacions 100 punts

Per tal d'aconseguir fer un entrenament el més semblant al set de dades final amb el qual es vol treballar, s'ha augmentat el nombre de punts per conjunt. Simplement, ara en comptes de tenir 6 punts per recta, se'n tenen 100. El codi emprat es pot observar a la Figura 47. En aquest cas, els 100 primers nombres de cada línia fan relació a les variables x i els 100 darrers nombres a les variables y.

Totes aquests conjunts de punts s'han separat en dos arxius mitjançant el codi de *Python* 46. Així doncs, 75 observacions són per fer l'entrenament de la xarxa i les 25 restants són per fer-ne la comprovació.

Els paràmetres de la xarxa també han estat triats en base al mètode prova/error i són els que es

mostren a continuació:

- **NumIn=200**: L'entrada està formada per cent punts amb dues coordenades cada un
- **NumHid1=10**: Hi ha 10 neurones a la primera capa oculta
- **NumHid2=10**: Hi ha 10 neurones a la segona capa oculta
- **NumOut=1**: El sistema tan sols ha de predir una resposta
- **NumPat=75**: Hi ha 75 nombres generats aleatòriament
- **Nombre de capes ocultes=2**
- **paràmetres**: eta=0.25 alpha=0.3 i smallwt=0.5
- **Error**: 0.0004

Aquesta xarxa no fa les 100000 iteracions sinó que s'atura a l'iteració 435 tal i com es mostra a la Figura 38. Així doncs, s'aconsegueix un error inferior a 0,004 i es dona com a acceptable.

El codi sencer es pot observar aquí: XARXA CORRELACIÓ 100 PUNTS

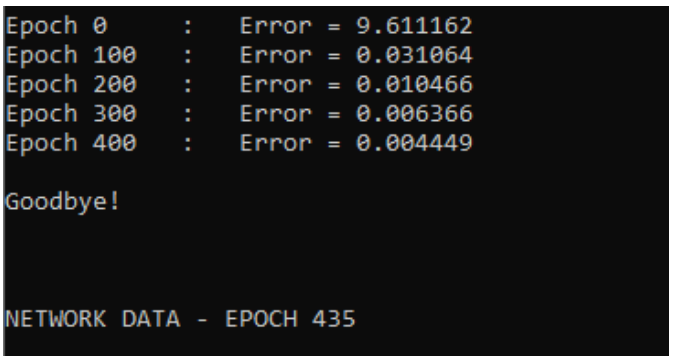

Figura 38: Resultats correlació 100 punts

S'ha procedit a realitzar la comprovació de la xarxa neuronal. Anàlogament als altres casos, s'ha arrodonit a la unitat més propera (0 o 1) els valors predits de la xarxa i s'han comparat amb el valor esperat. Aquesta xarxa també prediu de manera perfecta tots els casos plantejats. Així es mostra a la Taula 6.

Target ——- Prediction	**1**	**0**
1	100%	0%
0	0%	100%

Taula 6: Matriu de confusió dels resultats de la comprovació de la xarxa neuronal

Aquest comportament tan bo de la xarxa pot ser degut a que els punts generats estan tots sobre una mateixa recta i per tant només retorna 1 quan els punts tenen una correlació de 1. És a dir,

és una xarxa neuronal que detecta donats 100 punts si estan o no sobre la mateixa recta.

4.5 Predicció de correlacions diferents a 1

Per portar un pas més enllà la xarxa, s'ha decidit crear conjunts de punts que no estiguin sobre una mateixa recta però, en canvi, tinguin una correlació elevada. Així doncs, els primers 50 punts s'han obtingut mitjançant la primera part de codi mostrat a la Figura 48. Els altres s'ha aconseguit mitjançant la llibreria *Numpy* de *Python*. Aquesta té la funció *random.multivariate_normal*() que permet generar conjunts de punts com els que s'observen a la Figura 39.

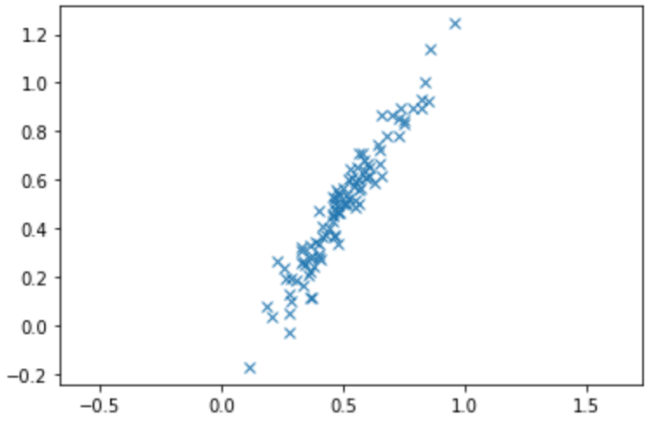

Figura 39: Exemple de conjunt de punts generats

Per tal de triar una covariància adient s'ha fixat una variància petita de 0,035. Se n'ha calculat la seva desviació estàndard seguint l'equació 9 on s'expressa com la variança és el quadrat de la desviació estàndard.

$$\sigma = \sqrt{\sigma^2} \tag{9}$$

Així doncs, per calcular les covariàncies, s'ha fet ús de l'expressió 10. Aquesta és l'expressió del coeficient de correlació de Pearson.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{10}$$

Com la matriu de covariàncies és simètrica $cov(X,Y) = cov(Y,X)$. Així doncs, s'han creat distribucions amb coeficients de correlacions aleatoris entre els valors desitjats.

S'ha procedit de la mateixa manera i s'ha fet l'entrenament de la xarxa una vegada utilitzat el codi de la Figura 46 per a dividir les dades de manera aleatòria en 75 punts d'entrenament i 25 punts de comprovació. Els paràmetres triats en base al mètode de prova/error es mostren a continuació:

- **NumIn=200**: L'entrada està formada per cent punts amb dues coordenades cada un

- **NumHid1=50**: Hi ha 50 neurones a la primera capa oculta
- **NumHid2=50**: Hi ha 50 neurones a la segona capa oculta
- **NumOut=1**: El sistema tan sols ha de predir una resposta
- **NumPat=75**: Hi ha 75 nombres generats aleatòriament
- **Nombre de capes ocultes=2**
- **paràmetres**: eta=0.25 alpha=0.3 i smallwt=0.5
- **Error**: 0.0004

Així doncs, tal i com s'observa a la Figura 40, l'algorisme s'atura a la iteració 2077 doncs s'aconsegueix un error menor a 0,004.

El codi emprat és el mateix que en l'apartat anterior, simplement canviant-hi els arxius amb els quals s'alimenta amb les dades.

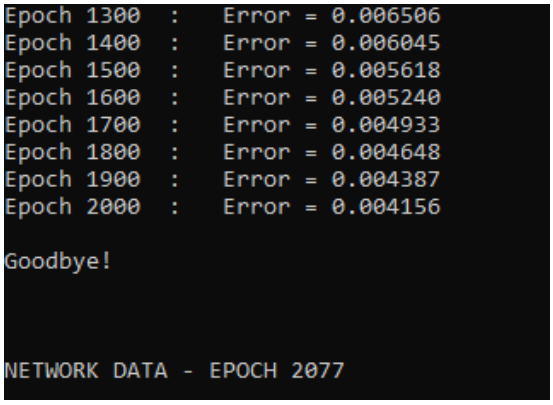

Figura 40: Resultat entrenament xarxa neuronal amb núvol de punts

S'ha procedit a la comprovació de la xarxa neuronal. A la Taula 7 s'observa com aquesta xarxa també és capaç de predir perfectament tots els casos presentats.

Target ——- Prediction	**1**	**0**
1	100%	0%
0	0%	100%

Taula 7: Matriu de confusió dels resultats de la comprovació de la xarxa neuronal

4.6 Predicció final

S'ha generat un conjunt de 5367 casos de punts. Cada cas conté informació normalitzada en el rang de 0 a 1 sobre 100 coordenades x i 100 coordenades y. L'objectiu doncs és, amb l'entrenament previ fet amb la xarxa neuronal, que aquesta predigui quin és l'únic cas on els cent punts

tenen una correlació considerablement gran. Aquest punt correspondria a un byte correcte de la clau i per tant es podria realitzar un atac de canal lateral. Aquesta predicció s'ha provat amb els valors dels últims dos entrenaments fets amb la xarxa. A continuació s'expliquen els resultats de cada una d'elles:

- **Correlació de 1**
  En aquest cas la xarxa ha estat entrenada amb nombres completament aleatoris que prenen valors d'entre 0 i 1 i valors sobre rectes aleatòries que també es troben dins del rang de 0 i 1.
  En aquest cas, la xarxa no és capaç de trobar cap valor amb una correlació prou elevada i tots els resultats que en retorna són 0. Això es deu a que la xarxa ha estat entrenada per a trobar punts completament alineats, i en el conjunt de dades del que es disposa, es busca una correlació elevada però no perfecte.

- **Correlació menor a 1**
  En aquest cas, s'ha generat un conjunt de nombres. S'ha fixat una desviació estàndard de 0,035 i una covariància d'entre 0,5 i 1. Tot i que la xarxa té un bon comportament en fer el test amb dades generades de la mateixa manera que les de l'entrenament, no té un bon funcionament a l'hora de realitzar l'atac de canal lateral. No només reconeix una sola variable com a el byte correcte sinó que dóna múltiples respostes cometent un error prou gran.

5 Pressupost

A continuació s'intentarà detallar el cost d'aquest projecte. Existeixen tres tipus de cost tal i com s'exposa a l'expressió 11.

$$Cost_{total} = Cost_{hum} + Cost_{amortitzaci} + Cost_{electricitat} \tag{11}$$

A continuació es detallen cada un d'ells:

- **Cost humà**

  El cost humà inclou el cost de les persones que treballen per al desenvolupament del projecte. En aquest cas, només s'ha considerat el sou d'un enginyer. Aquest és de 30€/hora. Aquest sou és l'estàndard per un enginyer que es dedica a la programació.

  Per calcular aquest cost cal saber el total d'hores invertides en el projecte. A la Taula 8 es mostren les diferents tasques del projecte i el nombre d'hores invertides.

Etapes del projecte	**Hores totals [h]**
Elecció del tema i registre del TFG	10
Recerca teòrica sobre MAE	25
Aprenentatge en programació en C	45
Programació de la xarxa neuronal en C	55
Redacció del capítol 3 de la memòria	22
Adaptació de la xarxa i creació dels diferents conjunts de dades	40
Redacció del capítol 4 de la memòria	25
Finalització de la memòria	30
Revisió de la memòria	15
Dipòsit dígital de la memòria	1
Preparació i defensa del TFG	22
Reunions	10
TOTAL	**300**

Taula 8: Hores de producció

  Així doncs, tenint en compte que la durada d'aquest projecte ha estat 300 hores, s'ha calculat com mostra l'equació 12 el cost humà.

$$Cost_{hum} = 300h \times 30\frac{€}{hora} = 9000€ \tag{12}$$

- **Cost amortització**
  Aquest es refereix al cost d'amortització dels portàtils utilitzats en aquest treball. Es tracta d'un Macbook Pro retina 13" 2015 el qual tenia un preu de 1800€ i un ASUS X756 el qual tenia un preu de 600€. Considerant que la vida útil dels ordinadors són 12 i 7 anys

respectivament s'ha calculat el cost amb l'expressió 13.

$$Cost_{amortitzacio} = \left(\frac{1800€}{12anys} + \frac{600€}{7anys}\right) \times \frac{1any}{365dies} \times 120dies = 77,50€ \quad (13)$$

- **Cost electricitat**

  Finalment, es calcula el cost de l'electricitat consumida pels dispositius electrònics. S'ha considerat que el preu mig de la llum és 1,5968 €/ KWh segons la pàgina d'Endesa [7]. S'ha considerat que els dos ordinadors consumeixen 50 W/h, per tant el consum ha estat de 100 W/h. Així doncs, a l'expressió 14 es mostra el cost de l'electricitat.

$$Cost_{electricitat} = 1,5968\frac{€}{kWh} \times 300h \times 0,1kWh = 47,90€ \quad (14)$$

A la Taula 9 es mostren els valors dels costos així com el del cost total.

Cost enginyer	9000€
Cost d'amortització	77,50 €
Cost electricitat	43,09 €
Cost TOTAL	**9120,59 €**

Taula 9: Taula dels costos del projecte

Per tant, s'estima que el cost total del projecte ha estat de 9120,59€.

6 Impacte ambiental

Pel que fa a l'impacte ambiental, aquest és reduït. Aquest projecte desenvolupa una plataforma *software*. Així doncs, l'únic impacte ambiental que es genera és la producció de l'energia en forma d'electricitat que utilitzen els dispositius electrònics emprats en aquest treball.

L'institut català de l'energia només disposa de dades de l'any 2020 doncs el 2021 encara no ha finalitzat. De totes maneres, no es creu que les estadístiques hagin variat considerablement d'un any respecte l'altre. La producció bruta d'energia a Catalunya l'any 2020 ha estat de 45315,2GWh. [5]. L'energia nuclear ha estat la principal font energètica per a la producció d'energia elèctrica a Catalunya, representant un 54,9% de la producció total. A més a més, els cicles combinats i la cogeneració han suposat un 12,2% i un 10,9% respectivament de la producció total. Per tant, en conjunt, un **80,2%** de l'energia total prové de fonts energètiques no renovables. Així es mostra en la Figura 41.

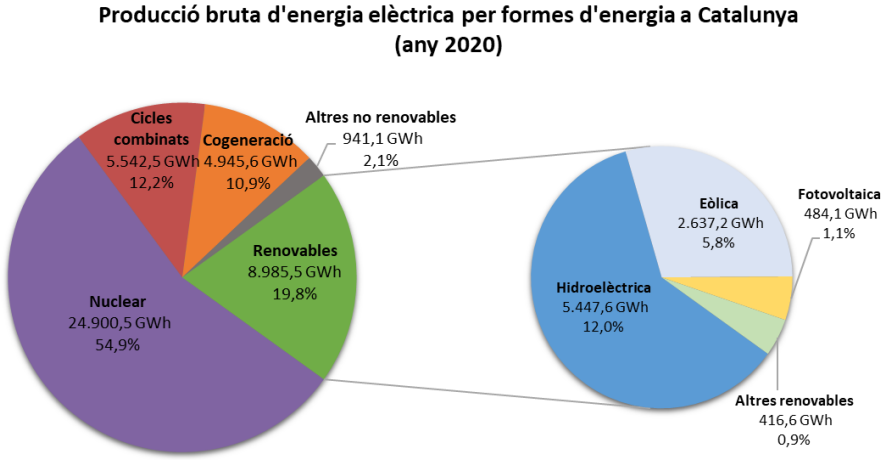

Figura 41: Estructura de la producció bruta de l'electricitat a Catalunya l'any 2020. [5]

Així doncs, la generació d'energia no prové principalment de fonts renovables. Així doncs, l'ús d'aparells electrònics i per tant dels dispositius necessaris per a la realització d'aquest treball suposen un cost pel medi ambient. Per tant, cal que cada usuari en faci un us correcte i intenti malgastar la menor quantitat d'energia possible. Així doncs es creu que actualment la indústria TIC consumeix un 7% de l'energia total produïda al món [6].

Finalment, no només cal tenir en compte la vida útil d'un ordinador. La petjada ecològica de cada objecte va des de que se'l comença a fabricar fins que desapareix per complet. Se'n mostra un esquema a la Figura 42.

Figura 42: Cicle de vida d'un dispositiu electrònic. Font: Wikipedia.org

Es calcula que un 66% de la contaminació que produeix un dispositiu dígital sorgeix a raó del procés de fabricació. Un 27% del seu ús, un 6% de la seva distribució i un 2% en el final de la seva vida útil [6].

Així doncs, cal fer-ne un consum responsable dels dispositius i intentar allargar-ne el màxim possible la seva vida útil. Algunes pràctiques que els usuaris poden fer són apagar els dispositius per tal que no consumeixin bateria o connectar el mode ecològic o baix consum.

Conclusions

Finalment, tenint en compte els objectius plantejats a l'inici d'aquest projecte es conclou:

- S'ha desenvolupat un algorisme des de zero capaç de fer prediccions envers diferents conjunts de dades. Aquesta plataforma, és una xarxa neuronal programada amb el llenguatge C que permet tenir una comprensió i un domini complet sobre la xarxa. No s'ha fet cap ús de cap llibreria prèviament preparada en el decurs de la programació de la xarxa.

- S'ha realitzat un estudi sobre la programació en el llenguatge C. S'han analitzat quines són totes les possibilitats que ofereix i s'ha orientat a aquelles eines que permeten el funcionament d'una xarxa neuronal.

- S'ha fet un anàlisi en profunditat sobre els diferents mètodes estadístics existents, especialment, de les xarxes neuronals. A més a més, s'ha estudiat en detall tot el funcionament de la xarxa neuronal en la seva implementació en C. Entendre com funciona una xarxa neuronal, ajuda a la comprensió dels resultats que prediu i els possibles errors.

- El sistema és adaptable a diferents dades d'entrenament. Dona llibertat a l'usuari per triar el nombre de neurones de les capes ocultes i el nombre d'entrades i sortides que es desitgen així com el nombre de casos amb els quals s'entrena la xarxa

- En primer lloc, la xarxa és capaç de predir amb una fiabilitat bona conjunts de dades senzills. S'ha augmentat el nivell de complexitat de les dades amb les quals la xarxa ha estat entrenada i aquesta segueix donant resultats amb elevada fiabilitat. És capaç de treballar amb diferents conjunts de dades d'entrenament diferents entre si.

- Si es realitza un entrenament adequat es pot arribar produir un atac de canal lateral amb la xarxa neuronal presentada. Cal que les dades amb les que es fa l'entrenament siguin exhaustivament semblants a les dades a predir. De no ser així, l'entrenament de la xarxa no es pot donar com a vàlid i les prediccions que realitza tenen baixa fiabilitat.

Futur treball

Optimització de l'algorisme

Com ja s'ha comentat, abans del desenvolupament d'aquest treball de fi de grau no tenia cap mena de coneixença sobre el llenguatge en el que s'ha programat la xarxa neuronal. Si bé aquesta s'ha anat fent més complexa i a la vegada més optimitzada a mesura que ha augmentat els diferents entrenaments sempre es pot millorar per tal que el codi sigui el més ràpid i comprensible possible.

Atac de canal lateral

Aquesta xarxa neuronal pot servir per realitzar diferents atacs de canal lateral. Cal doncs, refinar les dades d'entrenament de la xarxa perquè siguin el més semblants a les dades de predicció final. Una vegada amb l'entrenament correcte fet, la xarxa ha de ser capaç de predir correctament quina és aquella combinació de nombres que tenen una correlació més elevada i per tant correspon a un dels bytes correctes de la clau.

Creació d'una interfície d'interacció

Per tal que l'usuari que vulgui fer servir aquesta xarxa neuronal no hagi de buscar en el codi on canviar els arxius i els paràmetres de la xarxa, es podria crear una plataforma que fes de comunicadora entre l'usuari i la xarxa. Aquesta hauria de contenir informació sobre amb quins paràmetres es vol entrenar la xarxa i quins són els arxius d'entrenament. Finalment, la interfície hauria de demanar valors a l'usuari i retornar la predicció amb el percentatge de fiabilitat d'aquesta.

Agraïments

A l'Álvaro, el tutor del treball, per l'assessorament i l'ajuda proporcionada. Al Salvador, el codirector, per aportar un altre punt de vista sobre el treball.

A la mare, el pare, el Ricard i tota la familia per l'ajuda i l'acompanyament constant en el camí que m'ha dut fins aquí.

A l'Assemblea d'Estudiants, a l'Aula Lliure i a ESN, per deixar-me trobar el meu lloc dins la universitat.

A tots els amics i amigues, pels que han estat i els que encara hi son.

A tots ells, moltes gràcies.

Bibliografia

[1] Dr. John A. Bullinaria, *Step by Step Guide to Implementing a Neural Netorwork in C* URL: https://www.cs.bham.ac.uk/ jxb/INC/nn.html

[2] James, G.; Witten, D.;Hastie, T.;Tibshirani, R. *An introduction to statistical learning*, SPRINGER 2013

[3] Ben Krose, Patrick van der Smagt *An introduction to neural networks*, The univeristy of Amsterdam 1996

[4] Andrés Marzal, Isabel Gracia *C* (*Introducción a la programación con C*, Deparamento de Lenguajes y Sistemas Informáticos Universitat Jaume I 2003

[5] Desconegut *Balanç d'energia elèctrica de catalunya* URL: http://icaen.gencat.cat/ca/energia/estadistiques/resultats/anuals/balanc_energia/

[6] Desconegut *Clicking clean. Who is winning the race to build a green internet?* URL: http://www.clickclean.org/spain/es/

[7] Desconegut *Precio Kw/h Endesa* URL:https://www.endesa.com/es/blog/blog-de-endesa/luz/cuanto-cuesta-electricidad.

[8] Desconegut *Learning C programming* URL: https://www.tutorialspoint.com/cprogramming/index.htm i https://www.youtube.com/watch?v=JOfK95bzwCAlist=PLWPirh4EWFpEK1RbpruFXrwgzcGUjZ4nV

7 Annex

7.1 Generació de nombres aleatoris

```
import random

file=open("nombres_train.txt",mode="w",encoding="utf-8")
i=0
while i<150:
    x=random.random()
    y=random.random()
    if x<0.5 and y<0.5:
        z=0
    elif x>0.5 and y>0.5:
        z=0
    else:
        z=1
    file.write(str(x)+";"+str(y)+";"+str(z)+";\n")
    i=i+1
file.close()

file_2=open("nombres_test.txt",mode="w",encoding="utf-8")
j=0
while j<50:
    x=random.random()
    y=random.random()
    if x<0.5 and y<0.5:
        z=0
    elif x>0.5 and y>0.5:
        z=0
    else:
        z=1
    file_2.write(str(x)+";"+str(y)+";"+str(z)+";\n")
    j=j+1
file_2.close()
```

Figura 43: Script de python per tal de generar parelles de nombres aleatoris

7.2 Gràfic 3D amb els punts obtinguts

```
t=readtable('nombres_test.txt');
[x y] = meshgrid(0:0.1:1);
z = 0*x;
surf(x,y,z);
hold on
z2=1-x*0;
surf(x,y,z2);
hold on
plot3(t.Var1,t.Var2,t.Var3,'.r','MarkerSize',20);
hold on
z0=0*x;
y0=0.5-0*x;
plot3(x,y0,z0,'-r')
```

Figura 44: Script de matlab per tal de graficar en 3D el conjunt de nombres trobats

7.3 Generació de sis nombres correlacionats i nombres aleatoris

```
import random

file=open("nombres_correlaci_random.txt",mode="w",encoding="utf—8")



#Generar 50 conjunts de nombres aleatoris

i=0
while i<50:
    x1=random.uniform(—1,1)
    x2=random.uniform(—1,1)
    x3=random.uniform(—1,1)
    x4=random.uniform(—1,1)
    x5=random.uniform(—1,1)
    x6=random.uniform(—1,1)
    file.write(str(x1)+" "+str(x2)+" "+str(x3)+" "+str(x4)+" "+str(x5)+" "+str(x6)+" "+str(0)+"\n")
    i=i+1
file.close()

#Generar 50 conjunts de nombres amb una correlaci

file_1=open("nombres_correlacio_rectes.txt",mode="w",encoding="utf—8")
i=0
while i<50:
    m=random.random()
    x1=random.uniform(—1,1)
    y1=m*x1
    x2=random.uniform(—1,1)
    y2=m*x2
    x3=random.uniform(—1,1)
    y3=m*x3
    file_1.write(str(x1)+" "+str(y1)+" "+str(x2)+" "+str(y2)+" "+str(x3)+" "+str(y3)+" "+str(1)+"\n")
    i=i+1
file_1.close()
```

Figura 45: Script de python per a generar nombres correlacionats i nombres aleatoris. A cada línia, les posicions 1,3,5 són les coordenades x i les posicions 2,4,6 són les coordenades y dels punts.

7.4 Ordenació aleatòria de les dades

```
import random
with open('arxiu_XN.txt','r') as file:
    dades = [ (random.random(), line) for line in file ]
dades.sort()
with open('arxiu_def_train.txt','w',encoding="utf−8") as file2:
    with open('arxiu_def_test.txt','w',encoding="utf−8") as file3:
        i=0;
        for _, line in dades:
            if i<75:
                file2.write( line )
                i=i+1
            elif i>=75:
                file3.write( line )
                i=i+1
```

Figura 46: Script de python per a ordenar de manera aleatòria els conjunts de punts.

7.5 Generació de cent nombres correlacionats i nombres aleatoris

```
import random
file=open("nombres_correlaci_random_100_dif.txt",mode="w",encoding="utf-8")
#Generar 50 conjunts de nombres aleatoris
i=0
while i<50:
    for d in range(200):
        x=random.random()
        file.write(str(x)+" ")
    file.write(str(0)+"\n")
    i=i+1
file.close()
#Generar 50 conjunts de nombres amb una correlaci
file_1=open("nombres_correlacio_rectes_100_2.txt",mode="w",encoding="utf-8")
i=0
while i<50:
    x=[]
    y=[]
    m=random.uniform(0,1000)
    for d in range (100):
        if m<1:
            x1=random.random()
            y1=m*x1
            x.append(x1)
            y.append(y1)
        elif m>=1:
            y1=random.random()
            if d%5==0:
                x1=(y1)/m
            else:
                x1=y1/m
            x.append(x1)
            y.append(y1)
    for elem in x:
        file_1.write(str(elem)+" ")
    for elem in y:
        file_1.write(str(elem)+" ")
    file_1.write(str(1)+"\n")
    i=i+1
file_1.close()
```

Figura 47: Script de python per a generar 100 nombres correlacionats i aleatoris. A cada línia dels dos fitxers, les 50 primeres coordenades són les variables x dels punts i les 50 coordenades següents són les variables y.

7.6 Generació de cent nombres amb una correlació menor elevada però menor a 1

```
import numpy as np
import random
import math
with open('nombres_normalitzats.txt','w',encoding="utf-8") as file:
  i=0
  while i<50:
    mean = [0, 0]
    std=math.sqrt(0.035)
    covXY=std*std*random.uniform(0.5,1)
    cov = [[0.035, covXY], [covXY, 0.035]]
    x, y = np.random.multivariate_normal(mean, cov, 100).T

    ma=[max(x),max(y)]
    mi=[min(x),min(y)]
    matotal=max(ma)
    mitotal=min(mi)

    x1=[]
    y1=[]
    for num in x:
      nou=(num - mitotal )/(matotal-mitotal)
      x1.append(nou)
    for num in y:
      nou=(num - mitotal )/(matotal-mitotal)
      y1.append(nou)

    for elem in x1:
      file.write(str(elem)+" ")
    for elem in y1:
      file.write(str(elem)+" ")
    file.write(str(1)+"\n")
    i=i+1
```

Figura 48: Script de python per a generar 100 nombres amb una correlació elevada però menor a 1. A cada línia dels dos fitxers, les 50 primeres coordenades són les variables x dels punts i les 50 coordenades següents són les variables y.