Joan Ribé Vidal

COMPUTACIÓ ANALÒGICA D'UNA XARXA NEURONAL

TREBALL DE FI DE GRAU

dirigit per Carles Aliagas Castell

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2022

Resum.
En aquest Treball de Fi de Grau s'explica com entrenar una xarxa neuronal simple des de zero i com, un cop entrenada, la podem implementar amb un circuit analògic, que estarà compost principalment per amplificadors operacionals. S'estudiarà també la viabilitat de computar el model amb un circuit analògic en comptes de fer-ho amb un de digital.

Resumen.
En este Trabajo se explica cómo entrenar una red neuronal simple desde cero y cómo, una vez entrenada, la podemos implementar con un circuito analógico, compuesto principalmente de amplificadores operacionales. También se estudiará la viabilidad de computar el modelo con un circuito analógico frente al digital.

Abstract.
The goal of this Project is to explain how to train from scratch simple neural networks. Once trained, I also will show how we can compute its output using an analog circuit. I will study viability of those implementations, comparing them to digital computers.

Índex de taules

1. Introducció. 3
2. Descripció general del projecte 4
2.1 Hardware 4
2.2 Software 4
3. Marc teòric Referencial i Conceptual 5
3.2 Xarxa Neuronal 5
3.3 Computació Analògica 12
4. Requisits 16
4.1 Xarxa Neuronal 16
4.1.1 Requisits Funcionals 16
4.1.2 Requisits no Funcionals 19
4.2 Circuit Analògic 20
4.2.1 Requisits Funcionals 20
4.2.2 Requisits no Funcionals 21
5. Implementació 23
5.1 Xarxa Neuronal 23
Avaluació.(Jocs de prova) 28
5.2 Circuit Analògic 30
Perceptró simple, And 30
Perceptró multicapa 36
Avaluació (Jocs de prova). 41
6. Conclusions 47
7. Recursos utilitzats 50
7.1 Part I Xarxa Neuronal 50
7.2 Part II Computació Analògica 51
8. Annexes: 52
8.1 Codi 52
8.1.1 Codi c# 52
8.1.1.1 DataSet.cs 52
8.1.1.2 Neuron.cs 54

8.1.1.3 Layer.cs 56
8.1.1.4 NeuralNet.cs 58
8.1.1.5 Program.cs 65
8.1.2 Codi d'Implementació Aleatòria en Python 69
8.1.3 Codi Test Circuit 71
8.1.3.1 Codi Arduino 71
8.1.3.2 Codi Python 72
8.2 Imatges del la terminal del programa d'entrenament. 73
8.4 Esquemes Circuits per Importar a Falstad. 75
Esquema AND 75
Esquema Xarxa Donut 76

1. Introducció.

En aquest treball s'implementa de zero un algorisme per tal d'entrenar una xarxa neuronal.

Una **xarxa neuronal** és una xarxa o circuit de neurones de nodes artificials. Aquestes xarxes artificials es poden utilitzar per a models predictius, control adaptatiu i aplicacions. Les neurones estan interconnectades i aquestes connexions tenen uns pesos que realcen o inhibeixen la ponderació de les anteriors neurones. De trobar els pesos correctes en diem entrenar.

Un cop haurem entrenat la xarxa neuronal, s'explicarà com computar-la analògicament. L'objectiu és mirar si té alguna finalitat fer-ho, i ensenyar com es faria.

Inicialment havia enfocat el treball a fer un ordinador analògic. A mig camí vaig veure que era senzill implementar una xarxa neuronal ja entrenada, i el treball viraria cap aquest sentit.

Els ordinadors analògics electrònics estan composts principalment d'amplificadors operacionals. Més endavant hi entrarem en més detall, però principalment ens permeten realitzar sumes, multiplicacions, derivar, integrar, funcions logarítmiques i antilogarítmiques, i qualsevol combinació d'aquestes.

Els objectius de caire més acadèmic són aprendre com funciona una xarxa neuronal des de les estructures que poden tenir, paràmetres, etc fins la implementació de l'algorisme de *BackPropagation.*

Entre aquests propòsits també estan el aprendre com funcionen els amplificadors operacionals i resoldre tots els dubtes d'electrònica que han aparegut durant aquests mesos.

2. Descripció general del projecte

En aquest apartat hi ha descrit tot el material físic i software necessari utilitzat per totes les proves i experiments fets durant el treball. Servirà doncs d'orientació per si algú el vol replicar totalment o parcial.

2.1 Hardware

Necessitarem:

- Un ordinador des del que poder utilitzar programar en c# i accedir al navegador. Es recomana que tingui almenys 8Gb de Ram i un processador de 64 bits d'almenys 1.8 GHz de freqüència.
- Una RaspberryPi 3b+.
- Una placa arduino o similars.
- Plaques de proves (protoboard).
- Multímetre.
- Soldador, pasta per soldar (flux), estany, i demés estris necessaris per poder soldar components electrònics..
- Plaques perforades de Baquelita per soldar-hi a sobre.
- Amplificadors Operacionals. Concretament he utilitzat el model LM358P LM358, a l'annex hi adjuntaré el datasheet.
- Díodes rectificadors Schottky. Concretament el model IN5818.
- Resistències de 20 fins a 4700 Ohms. Amb 1W n'hi hauria d'haver prou. És important que tinguin màxim un 1% d'error ja que aquest error afectarà directament a la precisió dels resultats.
- Generador de senyals. El model concret utilitzat és GF-230 de PROMAX.
- Font d'alimentació. El model concret és FAC-363B també de PROMAX.

2.2 Software

A l'apartat de software, s'ha utilitzat:

- Visual Studio 2022 Community, que ja inclou el framework .NET, per executar codi en c#
- Raspbian Lite, sistema operatiu per a la Raspberry pi. Ja inclou python3, que també l'hem necessitat.
- Un navegador qualsevol, des del qual podrem utilitzar el simulador de circuits CircuitJS, a l'annex s'hi troba la referència .
- Flourish, software per a fer gràfiques.

3. Marc teòric Referencial i Conceptual

Aquest document té dues parts ben diferenciades.

Primer de tot, s'explica com entrenar una xarxa neuronal partint de 0, maneres de fer-ho i diferents observacions que n'he extret durant la implementació.

En aquesta primera part, concretaré què és una neurona, com un conjunt d'aquestes compon una xarxa neuronal, presentaré una implementació que l'entrena amb pesos aleatoris, i com funciona l'algorisme de *Backpropagation*[1] sempre acompanyat del de *Gradient Descend*[2], que també ensenyarem.

A la segona fase, s'implementarà el model entrenat a la primera part com un ordinador analògic capaç de computar-lo un cop definits els pesos.

A continuació, s'introdueixen els conceptes teòrics sobre els que s'ha construït aquest.

3.2 Xarxa Neuronal

Com ja s'ha dit, definim una xarxa neuronal com una malla de neurones connectades entre elles. L'objectiu que persegueixen és emular el cervell humà i les xarxes neuronals biològiques que utilitza, per processar així dades d'entrada i, en funció d'aquesta, donar-ne una resposta o sortida.

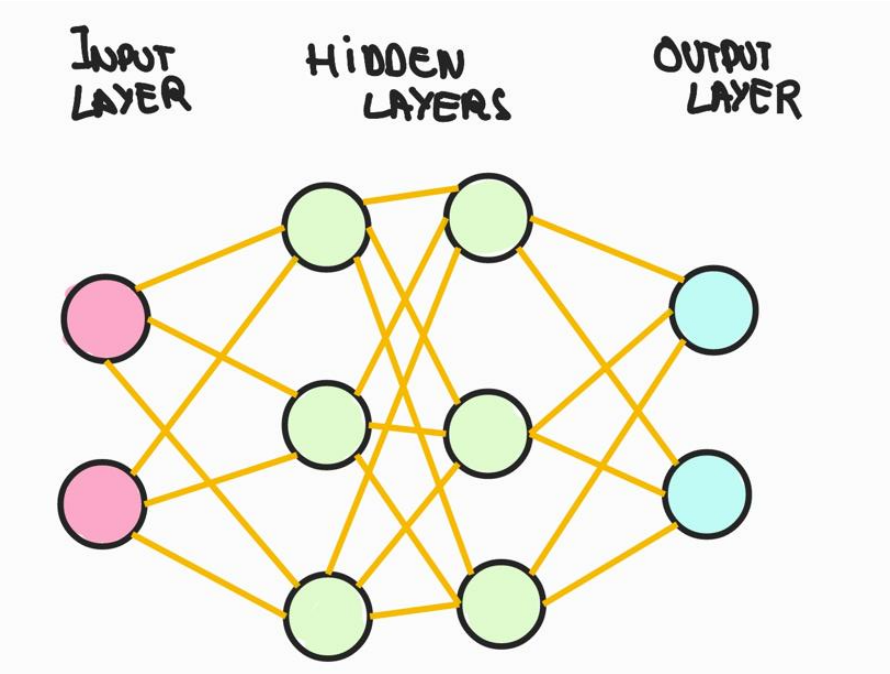


Figura 1. Xarxa Neuronal

[1] A l'apartat 5.1, pàgina 21, s'explica detalladament.
[2] A l'apartat 5.1, pàgina 24, s'explica detalladament.

A la Figura 1, cada cercle representa una neurona. Cadascuna de les connexions representa el pes que se li dóna al valor de la neurona anterior. I és que els valors que s'indueixen a les neurones d'entrada es traslladen, ponderats pel pes de les seves connexions, a les adjacents, on es fa la suma ponderada de tots els valors.

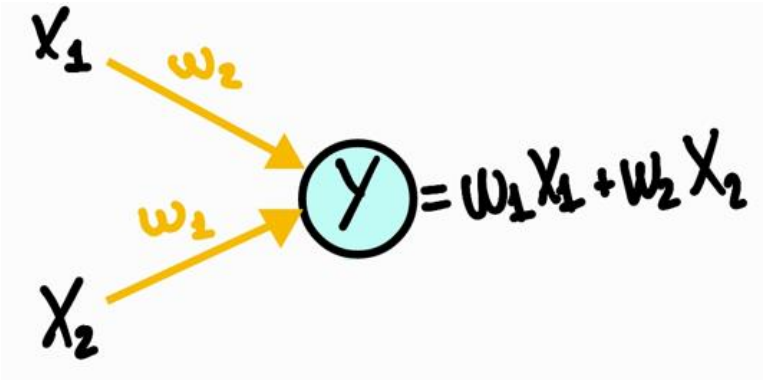


Figura 2. Perceptró Simple

Tal com hem explicat doncs, en aquesta figura podem veure com la neurona Y està connectada a dues neurones, i com el valor que prendrà serà:

$$Y = w_1X_1 + w_2X_2 \quad (1)$$

Així, del procés d'ajustar els pesos per cadascun dels enllaços entre neurones de la xarxa neuronal n'hi direm entrenar. Com podem observar a la Figura 3, per a una xarxa formada per una única neurona i amb una única entrada, estem davant d'un model de regressió lineal on, per exemple, de donar un llindar que separés la sortida la tindríem classificada en dos grups.

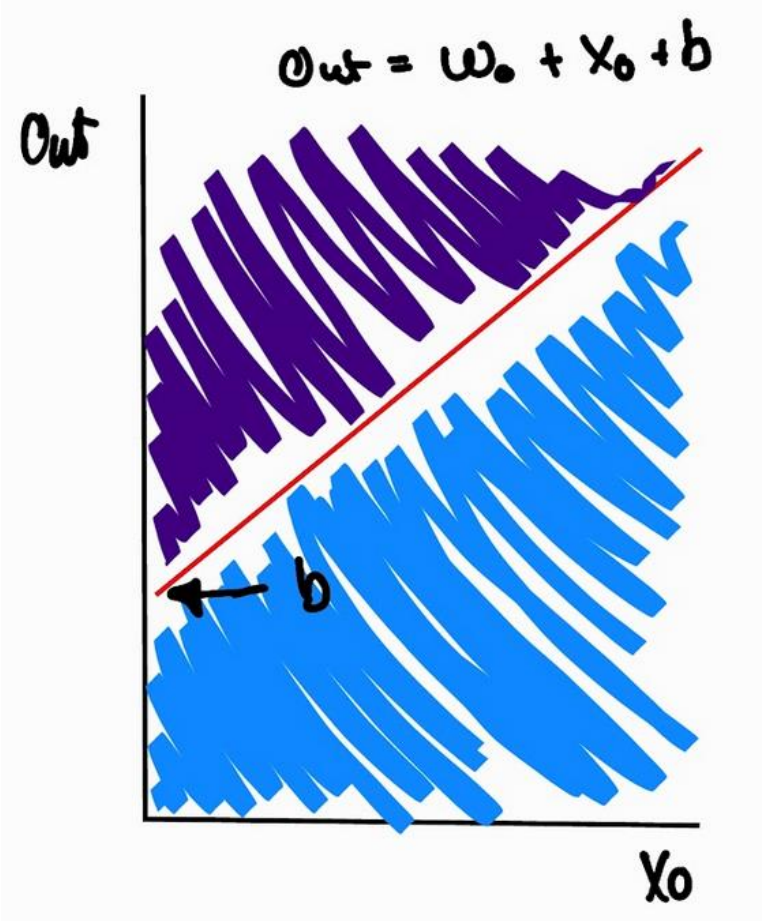


Figura 3, Regressió lineal.

En cas de tenir dues entrades en comptes d'una, en podríem treure el pla que separés un món 3d, i així per n dimensions.

A continuació hi ha un exemple d'un perceptró[3] simple que resol la porta lògica AND.

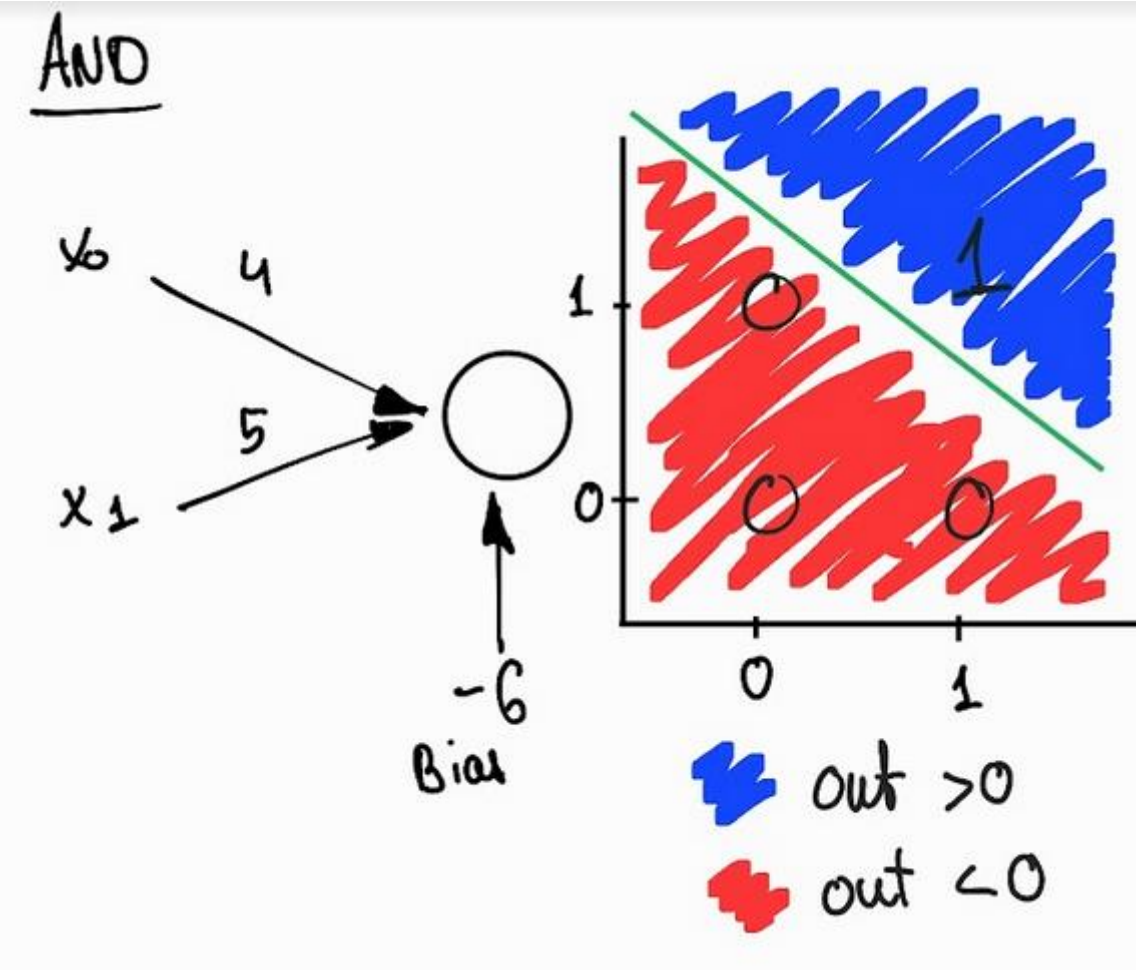


Figura 4, Perceptró simple.

Hem vist com classificar diferents conjunts d'entrada en dos grups. És intuïtiu afirmar que si afegim neurones, podem resoldre regressions més complexes i que, en cas de tenir més neurones de sortida, podrem arribar a agrupar amb més de dues classes. I així serà, sempre que tinguem en compte les següents consideracions:

Tota equació de recta y = mx + b té el seu terme independent, un biaix. A partir d'ara m'hi referiré com a *Bias*. El mateix trobem en un pla, i així en N dimensions.

A la Figura 4 ho podem veure.

[3] El **perceptró**, en l'àmbit d'aprenentatge automàtic, és un algorisme per a classificadors binaris. Aquesta classificació és lineal i en la forma més simple està compost d'una sola neurona.

També hem de fer esment del fet següent. I és que tant la multiplicació d'un vector per un escalar (aplicar la ponderació del pes), com la suma de vectors, resulta en un altre vector. Per tant, de tenir una xarxa de mil neurones i una sortida, s'obtindrà una xarxa neuronal equivalent al que podem obtenir amb un perceptró simple.

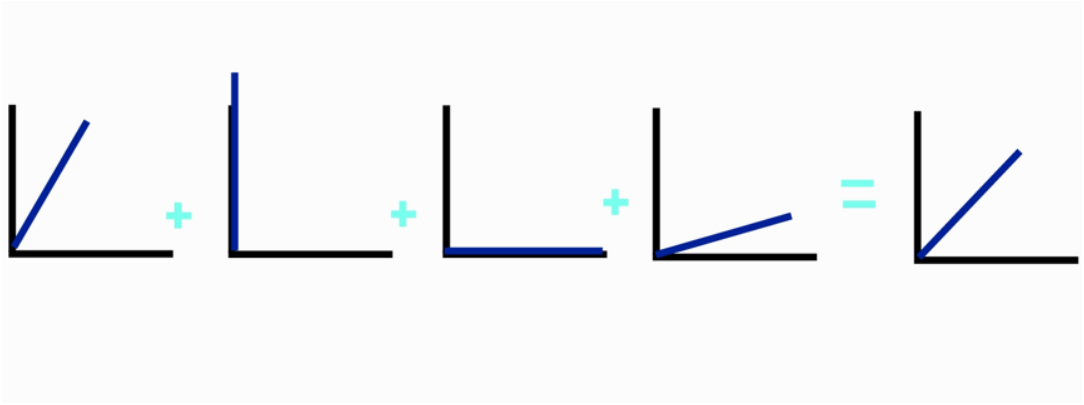


Figura 5, Suma de vectors.

Aquesta nova premissa però, no acaba amb les aspiracions de resoldre regressions no lineals.

Una manera de resoldre el contratemps descrit és aplicant el que se'n diu una funció d'activació a cada neurona. Així, ara la sortida a cada neurona serà:

$$Y = f\left(w_1 X_1 + w_2 X_2 \ldots\right) \quad (2)$$

On f(x) pot ser qualsevol funció no lineal que sigui diferenciable[4]. Algunes de les funcions més utilitzades són la Sigmoide, la Tan(x), la funció que se'n diu ReLU, etc.

[4] En essència una funció és diferenciable si admet derivades en qualsevol direcció i pot aproximar, com a mínim, fins a primer ordre per una aplicació afí.

Responent a la fórmula general :

$$y = \frac{1}{1 + e^{-x}} \tag{3}$$

Aquesta és la fórmula de la funció sigmoidal. Moltes corbes d'aprenentatge de sistemes complexes responen a una progressió semblant a la de la Figura 6, de la què en destaca l'acceleració d'enmig. La funció sigmoide generalment pot descriure aquesta evolució i per això és àmpliament utilitzada. És de les funcions d'activació que acostumen a resultar en millors prediccions un cop entrenades les xarxes que les contenen.

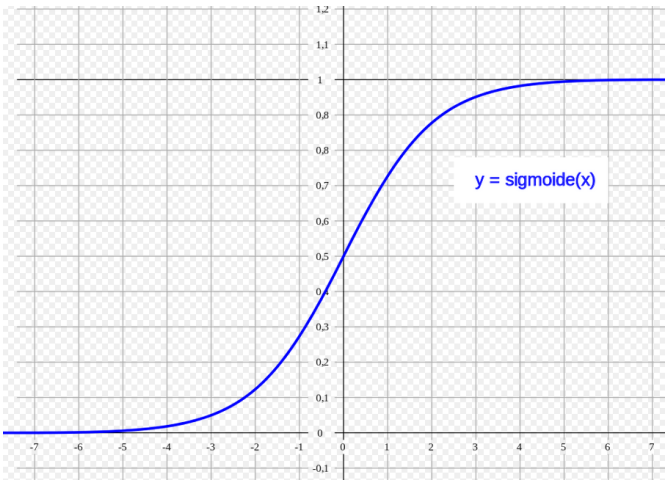


Figura 6. Funció Sigmoide

A mode de resum doncs, una funció d'activació és qualsevol funció que serveixi perquè les sortides de les neurones a les que s'apliquen no siguin lineals o proporcionals a l'entrada.

Un altre exemple de funció d'activació és la que es coneix com a *Rectifier Linear Unit* (ReLU), i respon a la fórmula:

$$Max\ (0, x) \tag{4}$$

Aquesta funció és especialment ràpida d'entrenar ja que només cal fer una comparació del resultat de la suma ponderada amb 0. Precisament per això és de les més utilitzades.

Té els seus inconvenients, com que no és diferenciable per a x=0, cosa que anteriorment havia dit que era imperatiu. Com que només és per a 0, el podem tractar de manera aïllada.

Alguns dels altres desavantatges que comporta els tractarem més endavant, ja que han anat sortint durant la implementació de l'entrenament.

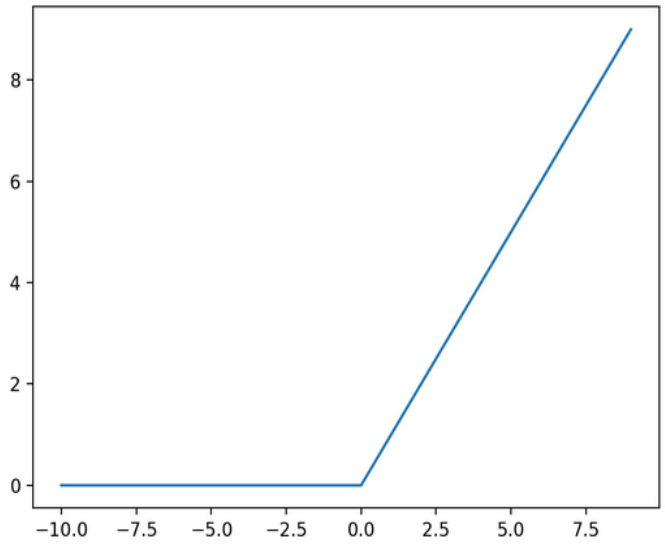


Figura 7, ReLU.

Més endavant s'explicarà el per què d'utilitzar aquesta funció d'activació per al nostre propòsit.

3.3 Computació Analògica

Un computador analògic és aquell que opera amb nombres reals, a diferència dels digitals, que s'han de limitar a aquells nombres computables. Per tant, els computadors analògics tenen una taxa de dimensió de la informació infinitament més gran fet que, d'estar davant d'un computador analògic ideal seríem capaços de resoldre problemes impossibles per la computació digital.

N'hi ha de tot tipus, des dels purament mecànics, datats pels 100 anys aC basats en engranatges diferencials capaços de reproduir els moviments orbitacionals del Sol i la Lluna, fins als que tractarem en aquest document, els computadors analògics electrònics basats principalment en Amplificadors Operacionals i que a continuació detallarem.

Els Amplificadors Operacionals són un circuit integrat amb dues entrades, una que en direm negativa i una altra de positiva, una sortida i dos terminals que l'alimenten. Per tal d'entendre el seu funcionament i com utilitzar-los, hem de tenir en compte el següent:

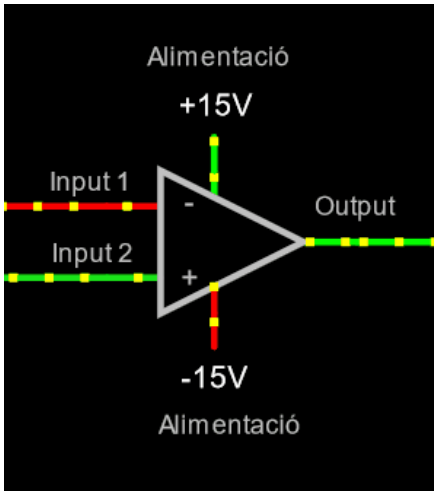
- El voltatge de sortida sempre oscil·larà entre els valors dels terminals que proporcionen l'alimentació.

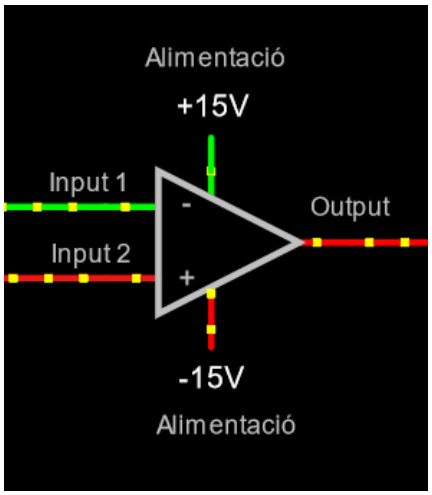
- Si el voltatge de l'entrada positiva, per poc que sigui, és menor a la que hem titllat de negativa, la sortida resultant serà equivalent al valor mínim dels dos terminals d'alimentació. En cas contrari, i per tant que sigui major, el resultat serà el màxim dels voltatges que l'alimenten.

- El corrent no flueix mai per les entrades. El proporcionen les fonts que l'alimenten. Sí que ho farà per la sortida.

A l'hora de treballar amb els amplificadors operacionals de manera pràctica hem de tenir en compte que internament ofereixen una petita resistència que ens farà perdre precisió.

A les següents figures es poden veure bé les propietats descrites.





Figures 8 i 9 Amplificador Operacional

A continuació, detallarem les aplicacions d'aquest circuit integrat que s'utilitzaran en aquest treball.

La següent configuració inverteix la senyal que se li posa a l'entrada negativa. Com veiem, l'altra entrada la tenim connectada al terra. Si les resistències r1 i r2 a la Figura 11 tenen el mateix valor, la sortida resultant serà, com ja hem dit i com també indica el nom, l'entrada invertida.

I és que aquí no acaben les aplicacions d'aquest component actiu[5], ja que la sortida respon a la fórmula:

$$Vout = -Vin\,\frac{r2}{r1} \quad (5)$$

Així doncs, variant el valor de les resistències, serem capaços de multiplicar el voltatge d'entrada per un escalar:

[5] Considerem component actiu aquell capaç de produir energia i no només consumir-ne.

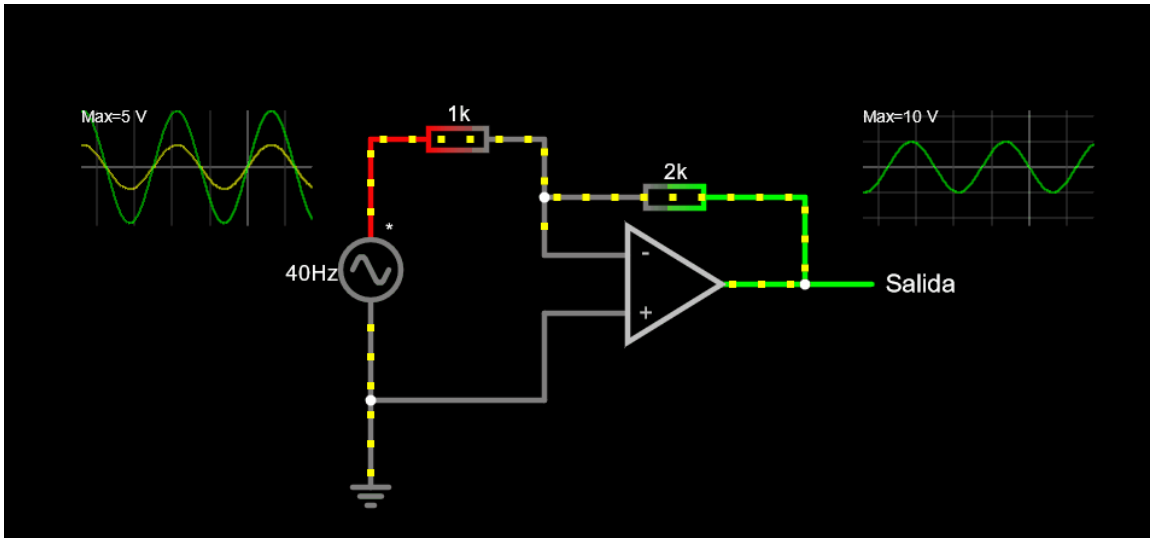


Figura 10, Amplificador Inversor.

I d'aquest circuit passem al següent i últim que utilitzarem per ara, molt semblant a l'anterior. Un sumador de tensió o sumador analògic.

A la configuració de la Figura 11, podem afegir-li més "entrades", i el resultat derivat serà la suma d'ambdues, novament negat. Concretament:

$$Vout = Vin1\ \frac{r3}{r1} + Vin2\ \frac{r3}{r2} \tag{6}$$

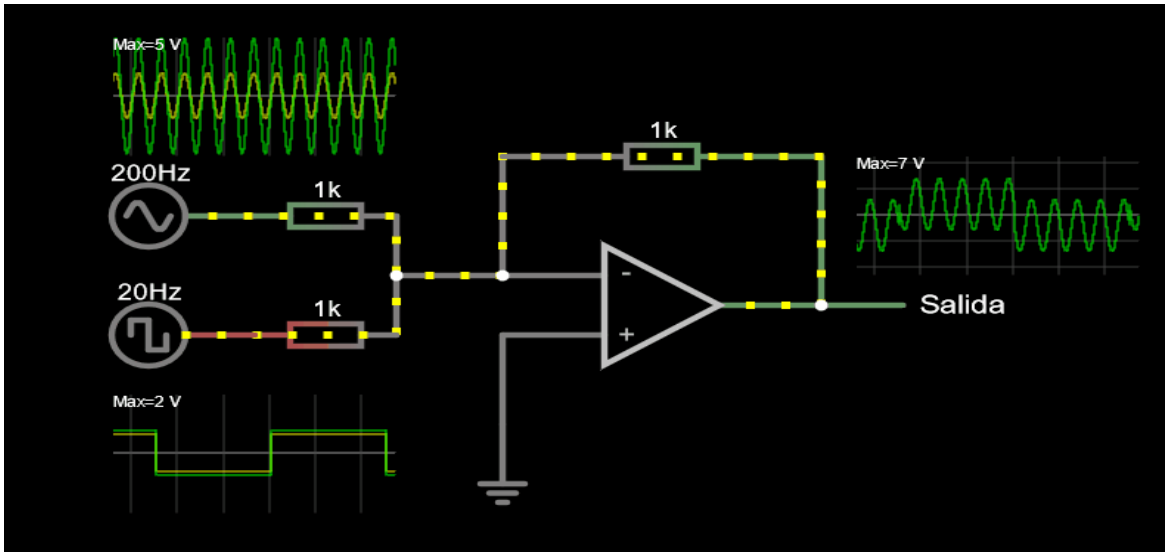


Figura 11, Amplificador Sumador.

I així per tantes entrades com convingui. En aquest circuit concret està basat tot el Treball, ja que la fórmula (3) que correspon al circuit de la Figura 12 no deixa de ser una suma ponderada, i aquesta, per definició una neurona.

4. Requisits

4.1 Xarxa Neuronal

4.1.1 Requisits Funcionals

Respecte els requisits, podem començar pel llenguatge de programació a utilitzar, que serà el c#. És un llenguatge Orientat a Objectes. Forma part de la plataforma .NET de Microsoft. Semblant a Java i si fa no fa igual de ràpid que aquest. L'he escollit entre les milers d'opcions per motius exclusivament de caire laboral. Qualsevol altre també podria fer la feina.

A continuació definirem el problema que ha de resoldre la xarxa un cop creada. Si bé està feta per acceptar qualsevol topologia[6], a l'hora de provar-la i implementar-la de manera analògica necessitava generar dades a classificar. He provat diferents *datasets*[7] i tests, però només es documentarà el que s'utilitzarà a la segona part del treball.

Concretament, he generat un conjunt de punts que podem classificar en dos grups. Tenim, doncs, un grup de punts (x,y) que pertanyen a un cercle de radi 1 i centre (0,0), i un altre amb tot de punts compresos entre la circumferència de centre (0,0) i radi 1.5 que també té el seu centre a l'origen de coordenades i 2.5 per radi.

[6] La topologia és el patró de connexions que presenta una xarxa neuronal
[7] Conjunt de dades

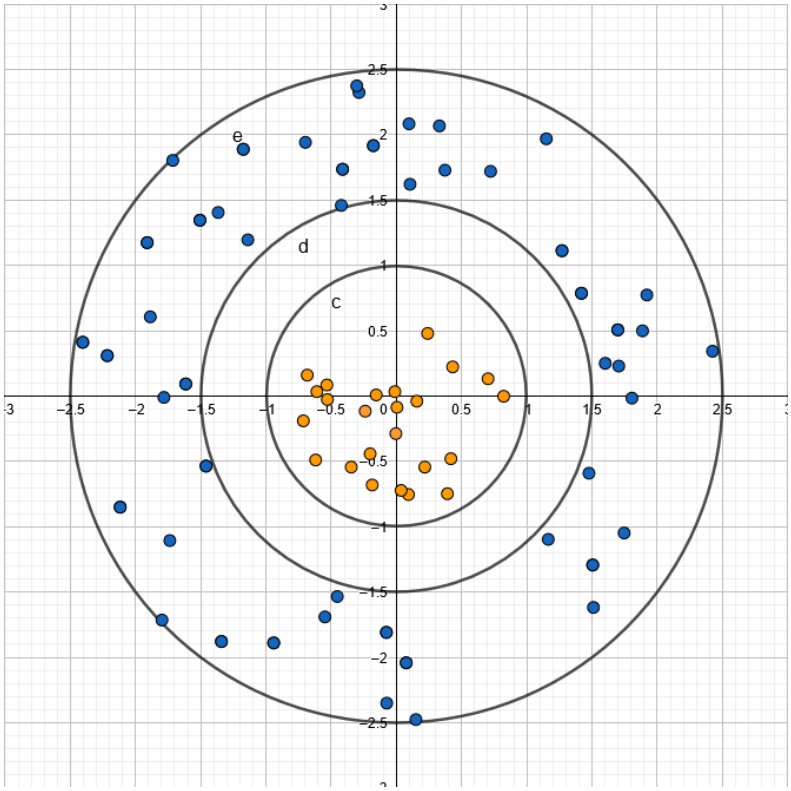


Figura 12, Dades utilitzades.

A l'Annex hi ha adjuntat el codi c# que genera el *dataset*.

Tornant cap als requisits de la Xarxa Neuronal pròpiament dita. Necessitarem que tingui dues entrades i ens retorni una sortida. L'entrenarem de manera que retorni un 0 si el punt (X,Y) que se li dona com a input està comprès al cercle interior i un 1 si està dins de l'anell exterior.

Respecte a la topologia, n'hi ha moltes capaces de resoldre aquest entrellat. Com ja hem dit, la capa d'entrada tindrà dues neurones, una per cada coordenada del punt, la de sortida d'una neurona. Ho he provat amb diferents configuracions. A continuació, hi ha descrites les més simples amb les que ho he aconseguit.

Amb una sola *Hidden Layer* de 4 neurones ha eixit amb nota l'entrenament, amb un error de menys d'un 1% en nombrosos entrenaments.

També amb dues capes interiors. Una de 3 neurones i una altra de dues.

Per motius que a la part d'implementació acabarem de tractar, s'ha utilitzat aquesta darrera topologia.

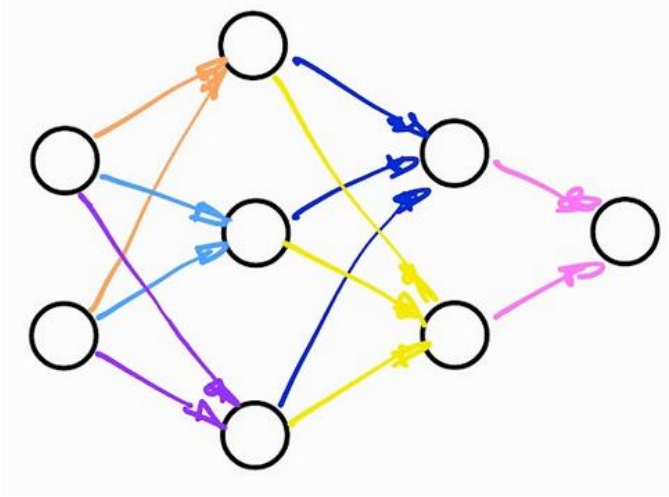


Figura 13, Topologia implementada.

I fins aquí els requisits que definiran la forma de la xarxa. Els algorismes per entrenar-la i les diferents observacions es tracten a l'apartat de la implementació, punt 5.2.

4.1.2 Requisits no Funcionals

Com que la xarxa a entrenar és senzilla, no hi ha requisits ni exigències en quant a la eficiència. Tampoc en termes d'escalabilitat.

Busquem un codi funcional, però sobretot flexible, per anar provant les diferents variacions dels algorismes d'entrenament i formes de la xarxa. Per això és que s'hi farà una Classe per les neurones, una per les connexions, una per cada capa i una última per la xarxa neuronal que les contindrà a elles i en funcions encapsulades les funcions d'error, funcions d'activació, i derivades d'aquestes.

He considerat important poder generar un *dataset* nou, canviar la topologia de la xarxa, reentrenar-la de nou i poder tornar a testejar-la sense necessitar tancar i obrir el programa de nou per tal d'agilitzar les proves.

Com no podia ser d'altra manera, també s'ha de poder guardar els pesos de la xarxa en qualsevol moment.

Com que és innecessari, serà una aplicació de consola, sense interfície gràfica. Així podrem dedicar el temps a provar un rere altre els paràmetres i forma de la xarxa.

Per comprovar que realment tenen sentit els algorismes aplicats, ho compararem amb el cost temporal d'estar-ho entrenant generant pesos totalment aleatoris a cada passada i guardant els resultats més bons obtinguts per aquests.

4.2 Circuit Analògic

4.2.1 Requisits Funcionals

Com hem vist fins ara, en la primera fase d'aquest Treball hem aconseguit entrenar un model que resol la classificació que tenim a la Figura 13. Ara, volem passar aquests pesos a un circuit analògic que el computi. Per això, farem servir el sumador analògic ja mostrat, així com el circuit inversor, també format per amplificadors operacionals.

Primer de tot, implementarem la porta lògica AND, després la xarxa neuronal que hem entrenat.

S'ha explicat que utilitzarem díodes per aplicar la funció d'activació ReLU a la suma ponderada dels sumadors. Aquests però, tenen la seva resistència interna i provoquen una lleugera pèrdua de tensió. Com que se'n posaran uns quants no és un fet que es pugui ignorar. El resultat del circuit doncs, que en un món perfecte serà 0 o 1, com a molt deixarem que es desviï 0.25 punts per sobre o per sota. Si hi ha més error que aquest, considerarem que la implementació no és vàlida.

Els díodes seran del tipus Schottky[8], que són, d'entre tots els tipus de díodes, dels que tenen una pèrdua d'energia més negligible.

Per descomptat, tant a l'AND com a la implementació de la xarxa neuronal tindrem dues entrades i dues sortides.

Als Amplificadors Operacionals se'ls subministrarà +-15V.

[8] També dit díode de barrera Schottky, és un dispositiu semiconductor que proporciona commutacions molt ràpides entre els estats, així com tensions llindar baixes.

4.2.2 Requisits no Funcionals

Volem utilitzar el mínim d'amplificadors operacionals possibles. Per aconseguir això, haurem de seguir diverses directrius que han anat sortint i que es poden observar a la implementació. Totes giren al voltant que, en un inici, com que el sumador li canvia el signe, tenia un mínim de dos amplificadors operacionals per cada neurona, un per fer l'operació i l'altre per revertir-ne el canvi de signe. Per tant, per a optimitzar l'àrea, potser ens convindrà tenir un *Bias* amb el seu valor invertit.

Al mateix temps, hem de tenir en compte també que necessitaríem soldar moltes resistències per assolir els valors exactes de cada pes. Per això és que li hem donat un marge d'error generós al resultat en l'anterior apartat, on parlàvem dels díodes i les pèrdues de tensió que portava implícitament la seva utilització. Així, tot i que el resultat teòric sigui d'1V, donarem com a bo un resultat de 0.85V, per exemple.

Tornem ara al tema de com s'alimenten el amplificadors operacionals. Es pot fer amb una sola font de tensió però, jo he decidit utilitzar-ne dues, que resulta més fàcil de preparar i entendre.

El circuit per aconseguir-ho és el següent:

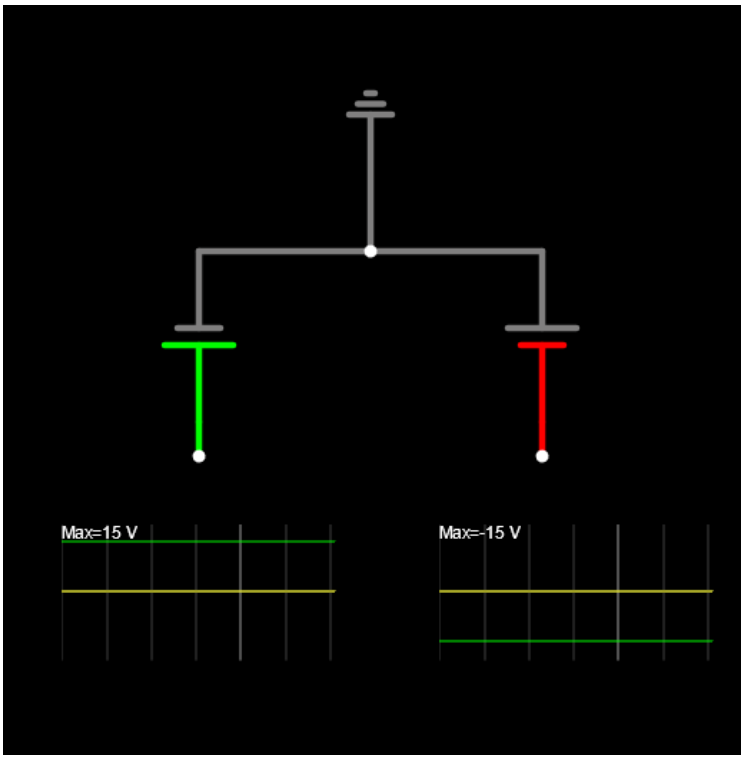


Figura 14, Circuit que subministra +-15V

Que a la pràctica resulta en:

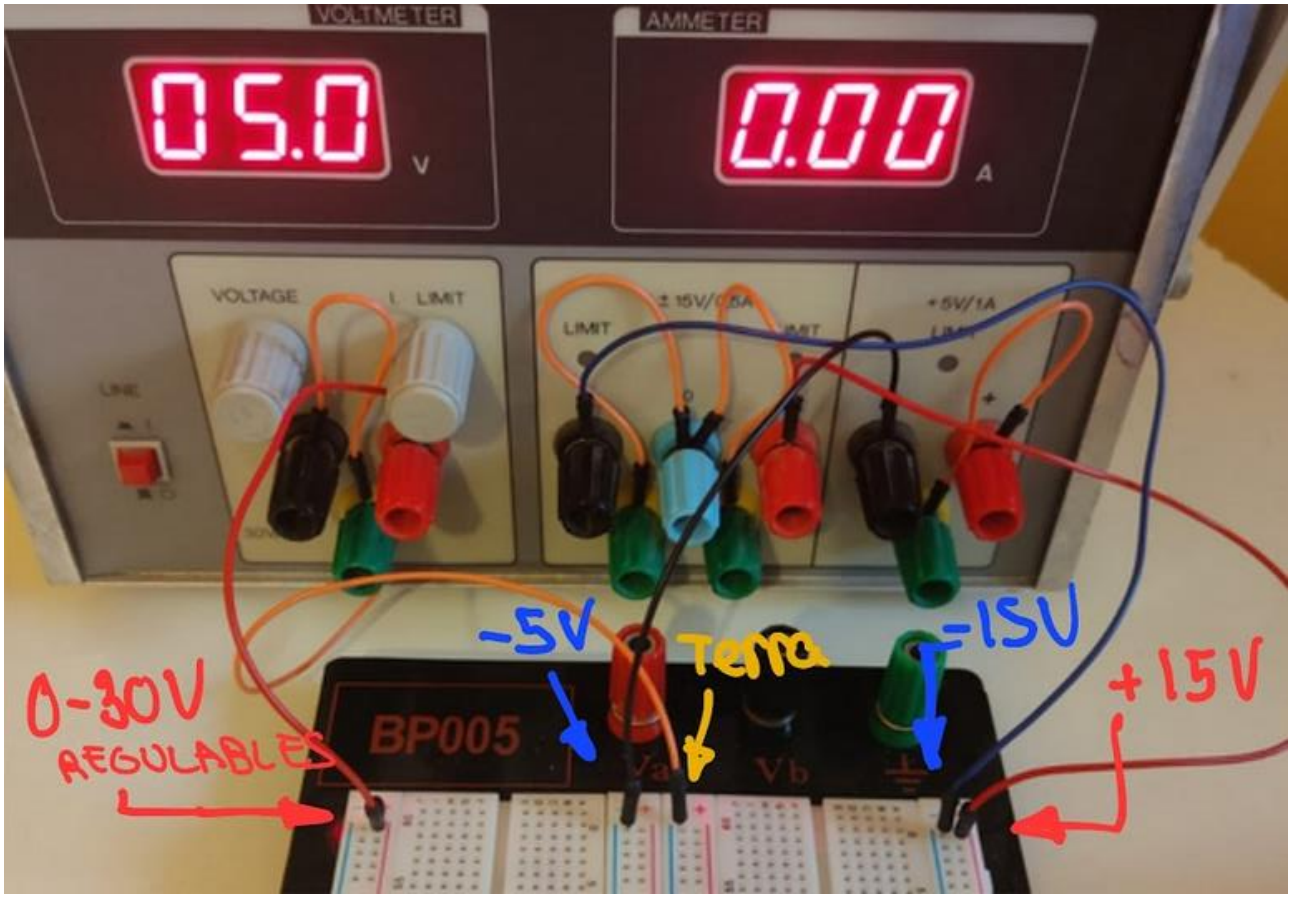


Figura 15. Alimentació Proporcionada

5. Implementació

5.1 Xarxa Neuronal

Fins ara, s'ha descrit com serà la xarxa neuronal. Seguidament tractarem de l'implementació dels algorismes d'entrenament. La totalitat del codi es pot trobar als annexes.

Implementació de l'algorisme de *Backpropagation*.

El primer pas per entrenar una xarxa neuronal és fer el que se'n diu el *forward*. És a dir, recórrer capa a capa fent les sumes ponderades des de l'entrada fins a obtenir els resultats de la capa de sortida. Un cop ho hem aconseguit, toca avaluar com d'equivocats estan els pesos que tenim, que abans de la primera passada prendran valors aleatoris.

Un cop s'ha fet el *forward*, toca saber com de bo o dolent ha estat el resultat, per saber en quina direcció tocarà arreglar els pesos del model que estem entrenant. Per això és que es necessita una funció de cost. Aquesta ens permetrà quantificar com de lluny s'està de tenir la xarxa completament entrenada.

La funció d'error que es farà servir serà l'Error Quadràtic Mitjà (EQM). És utilitzat en molts camps de les matemàtiques i computació. En aquest cas l'he escollit com a funció de cost perquè és molt fàcil fer-ne la derivada, cosa que facilita les proves i depuració de l'algorisme.

$$EQM = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}i - Yi)^2 \quad (6)$$

Aplicant la funció de cost doncs, es pot saber com de dolent és el resultat de cadascuna de les neurones de l'última capa. Però, a qui s'ha de culpar? A l'últim pes que l'ha ponderat? a l'anterior?

Necessitem una manera de propagar l'error a cada neurona. I per això tenim el *Back Propagation*.

La finalitat d'aquest serà trobar com varia el cost respecte un canvi de cadascun dels pesos de la nostra xarxa neuronal. Com era d'esperar, es farà el mateix pels *Bias*. D'aquests darrers en tindrem definit un per a cada capa en comptes d'un per neurona, per simplificar-ho tot. Així tenim que volem trobar la derivada del cost respecte els paràmetres de pes (w) i bias (b).

$$\frac{\partial C}{\partial W} \qquad\qquad \frac{\partial C}{\partial b}$$

(7)

Comencem doncs a calcular la derivada dels pesos a l'última capa L.

En aquesta última capa cada cost resultant depèn de la suma ponderada dels diferents pesos, a la que ens referirem com a Z^L i l'aplicació de la funció d'activació a la que podem dir-n'hi a(). Així:

$$C(a(Z^L)) = ERROR$$

(8)

Per calcular la derivada de la funció de cost, sobre la funció d'activació, sobre la suma ponderada, tocarà aplicar-hi la regla de la cadena[9].

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial w^L}$$

$$\frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial b^L}$$

(9)

Així doncs, ens tocarà calcular la derivada del cost respecte la funció d'activació. En el nostre cas, com hem explicat abans, la derivada de l'error quadràtic mitjà serà:

$$\frac{\partial C}{\partial a_j^L} = (\, a_j^L - \, y_j)$$

[9] En càlcul infinitesimal, la regla de la cadena és una manera de calcular la derivada de la composició de dues o més funcions. Diu així: una variable, y, depèn d'una segona variable, u, i aquesta alhora depèn d'una tercera variable, x, llavors la velocitat de canvi de y respecte de x és la velocitat de canvi de y respecte de u multiplicada per la velocitat de canvi de u respecte de x.

(10)

Seguidament, haurem de fer la derivada de la funció d'activació respecte la sortida de la suma ponderada. Per a la funció ReLU, serà 0 per a valors menors a 0 i 1 per a valors més grans. Tenim que, si la suma ponderada dóna 0, no és derivable. Es retornarà un 0 en cas que de tenir un 0, que treballant amb reals ja és poc probable que ens toqui. Més endavant analitzarem les implicacions directes de fer servir aquesta funció d'activació i no d'altres.

$$H(x) = \begin{cases} 0, & per\ a, x \leq 0 \\ 1, & per\ a, x > 0 \end{cases}$$

(11)

Ara només ens queda trobar la derivada dels pesos Wj i de B respecte la Z, la suma ponderada. Com que el Bias és un terme independent, la seva derivada serà 1. Respecte W, seran els valors de la capa anterior a_i^{l-1}.

$$\frac{\partial z^L}{\partial b^L} = 1 \qquad \frac{\partial z^L}{\partial w^L} = a_i^{l-1}$$

(12)

Així doncs, a mode de resum:

$$\delta^L = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$$

$$\frac{\partial C}{\partial b^L} = \delta^L$$

$$\frac{\partial C}{\partial w^L} = \delta^L \cdot a_i^{l-1}$$

(13)

I fins aquí per calcular l'error de les neurones de l'última capa. Ara toca tractar la resta de capes.

Si δ^L pren valors grans, voldrà dir que per petit que sigui el canvi que rebi l'input de la neurona, afectarà dràsticament al resultat. Passarà el contrari si pren valors petits. Hem trobat el gradient, qui marca la responsabilitat de cada neurona a la xarxa, i amb aquest podem estimar l'error de cada pes, com podem veure a l'última de les fórmules del formulari (13).

I com es calcula l'error a la capa L-1? Doncs de la mateixa manera que hem fet abans, només cal aplicar la regla de la cadena.

$$\frac{\partial C}{\partial w^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial w^{L-1}}$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial b^{L-1}}$$

(14)

No és gaire difícil veure que la meitat de la feina ja la tenim calculada d'abans i que es pot simplificar l'expressió.

$$\delta^{L-1} = W^L \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}}$$

(15)

Com es pot observar, es va arrossegant l'error de les neurones de les capes finals cap endavant de la xarxa, d'aquí el nom de *BackPropagation*.

Implementació de *Gradient Descend*

Per tal d'implementar-lo es necessita el que se'n diu el *learning rate*. I és que trobar quant afecta cada neurona a l'error final, no és en va. Estem parlant de trobar el gradient de l'error. És a dir, sabem la direcció que hem de prendre per disminuir-lo. Quants passos fem en aquesta direcció determinarà la velocitat d'entrenament. La longitud d'aquest pas és el *learning rate*, que es parametritzarà. La idea és anar recalculant els gradients amb el *BackPropagation* a mesura que es va avançant, fins trobar un mínim relatiu de la funció d'error, moment en què considerarem que hem entrenat la xarxa.

Fins aquí ho he implementat fil per randa. Però l'algorisme de *BackPropagation* explicat aquí és una variació de l'original. I és que m'he saltat un pas. Generalment es repetiria el *BackPropagation* descrit per moltes entrades diferents. D'això se'n diu una època. Es van guardant el grau de responsabilitat de cada neurona per cadascuna de les entrades, se'n fa una mitjana i a aquesta s'aplicarà el *Gradient Descend*. Entre diferents avantatges, trobem que no afecta tant el soroll, etc.

Si ho he fet així és per simplicitat i per estudiar-ne el comportament i veure'n el gradient a cada pas. Com que no hi ha marge per tenir errors al *Dataset*, he considerat que no era necessari.

Més característiques d'aquest fer. La funció d'activació. Com ja s'ha avançat, serà la ReLU. Els motius són exclusivament que és especialment senzilla d'implementar sobre el circuit analògic, ja que amb un simple díode podem anul·lar el corrent quan és negatiu i deixar-lo passar quan és positiu.

Aquesta funció però, té els seus inconvenients. I és que per a sumes ponderades menors a 0, la seva derivada serà 0 i per tant se'ns anul·la la Delta(δ^L) del *Gradient Descend* i no arribarem mai a entrenar els pesos que hi arriben. Aquest és un dels motius pels quals, a vegades, la implementació presentada no arriba a entrenar el model quan els pesos inicials derivin en moltes sumes ponderades menors a zero. D'això se'n diu *Dying Relu Problem*, ja que pot ser que per tots els inputs possibles les neurones quedin inactives.

El *learning rate*. Aquest paràmetre ha donat molt de joc també. D'agafar-ne un de molt gran, es tardarà menys a entrenar la xarxa. Això no en el cas que tractem no té gaire trascendència ja que la xarxa és petita i, a més a més, no es fan servir èpoques, que evidentment multiplicaria per el nombre de dades a mirar en cada època el temps d'entrenament.

És per això que té sentit agafar un *learning rate* baix.

Avaluació.(Jocs de prova)

El programa encarregat d'entrenar la xarxa ha estat dissenyat per generar tantes dades com es vulguin per a poder comprovar el resultat un cop entrenat.

Generalment, es dedica un 80% de les dades del *dataset* per a entrenar i el 20% per a validacions. Com que podem generar dades il·limitades no he tingut gaire en compte aquest habitual procedir.

Una característica de l'entrenament és que generalment s'espera a tenir un error ínfim, i a esperar que aquest deixi de reduir-se a parar l'entrenament. En el meu cas en canvi, li passava N dades i deixava que les fes totes, llavors feia una passada amb dades noves per comprovar si s'havia entrenat bé o no.

Alguna que altra vegada l'entrenament s'estancava. Depurant he vist que es deu a diferents raons. Primera, el ja mencionat *Dying ReLU problem*. Després, accentuat per no fer servir èpoques s'encallava jugant a un ping pong del que li costava moltes iteracions sortir-ne, ja que els gradients d'una execució i la següent es neutralitzaven entre ells. Jugant amb el learning rate podem fer que passi més o menys.

No he trobat cap cas amb *Overfit*[10].

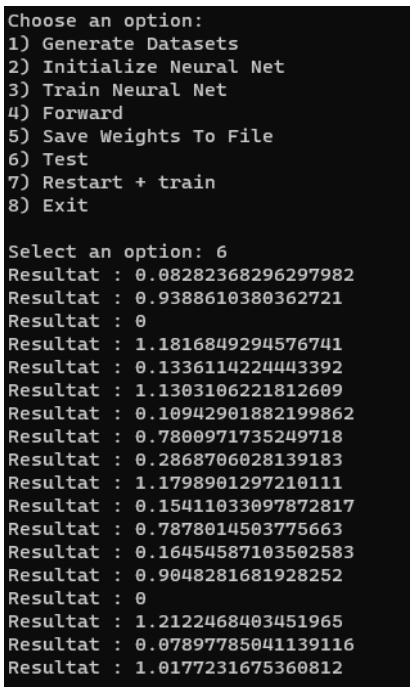


Figura 16, Opció Test del programa en c#, prova alternant les dues classe binàries a cada test.

[10] L'*overfitting* (sobreajustament) és l'efecte de sobreentrenar un algorisme d'aprenentatge, fet que causa que només funcioni per les dades amb les que s'ha entrenat i no resulti en una solució generalitzada.

És gairebé instantani el temps d'entrenament per a la xarxa descrita.

Durant 30 dies també he deixat entrenant-se sense parar la mateixa xarxa neuronal de manera totalment aleatòria. De totes les inicialitzacions de pesos amb nombres pseudoaleatoris, n'he guardat aquelles que de 10000 tests la mitjana del EQM era inferior a 0.1.

El codi, que es troba a l'annex, és un script de python executat com a servei de Systemd. Podria haver deixat diferents serveis executant el codi en paral·lel.

Concretament s'ha executant des del 2 d'agost fins el 3 de setembre. Durant aquest temps ha provat un total de 8 900 000 configuracions de pesos diferents, de les quals només 13 han donat un error quadràtic mitjà per a 10000 entrades menor al 0,1.

Queda confirmat per tant que, fins i tot per una topologia extremadament petita i simple com aquesta, no és una opció viable entrenar-la amb valors aleatoris sense cap algorisme que ajudi a acotar-los.

5.2 Circuit Analògic

Perceptró simple, And

A continuació s'explica com s'ha implementat un perceptó simple que resolgui la porta lògica And amb un circuit analògic format per un únic amplificador operacional.

Per tal de resoldre-la, implementarem els següent dibuix, veure també el de la figura 4:

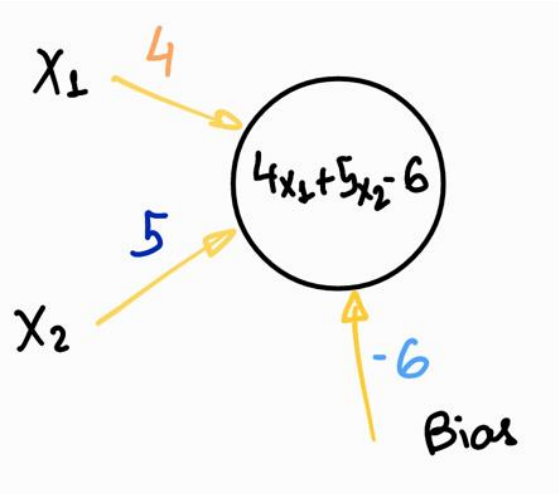


Figura 17, Perceptró que implementa l'AND.

Així, per a cada possible entrada:

$$Y(0, 0) = 4 * 0 + 5 * 0 - 6 = -6$$
$$Y(0, 1) = 4 * 0 + 5 * 1 - 6 = -1$$
$$Y(1, 0) = 4 * 1 + 5 * 0 - 6 = -2$$
$$Y(1, 1) = 4 * 1 + 5 * 1 - 6 = +3$$

Com podem veure, classifica amb valors menors a 0 aquells valors que haurien de donar 0 i amb nombres positius els que estan per sobre. Si apliquem la funció d'activació ReLU obtindrem 0 per als negatius. Així doncs, només ens queda implementar el circuit.

En aquest cas, ja que no deixa de ser un exemple, els resultats estaran negats. Per obtindre el resultat descrit a dalt hauriem d'utilitzar un altre amplificador operacional per tal de canviar el signe de la sortida general. A la implementació ja més complexa del punt 4.2 ja veurem com ho solucionem.

Per tant, de moment:

$$Y\ (0, 0) = (4 * 0 + 5 * 0 - 6) * -1 = + 6$$
$$Y\ (0, 1) = (4 * 0 + 5 * 1 - 6\ ) * -1 = + 1$$
$$Y\ (1, 0) = (4 * 1 + 5 * 0 - 6) * -1 = + 2$$
$$Y\ (1, 1) = (4 * 1 + 5 * 1 - 6) * -1 = -3$$

Aquest serà l'esquema del circuit que necessitem:

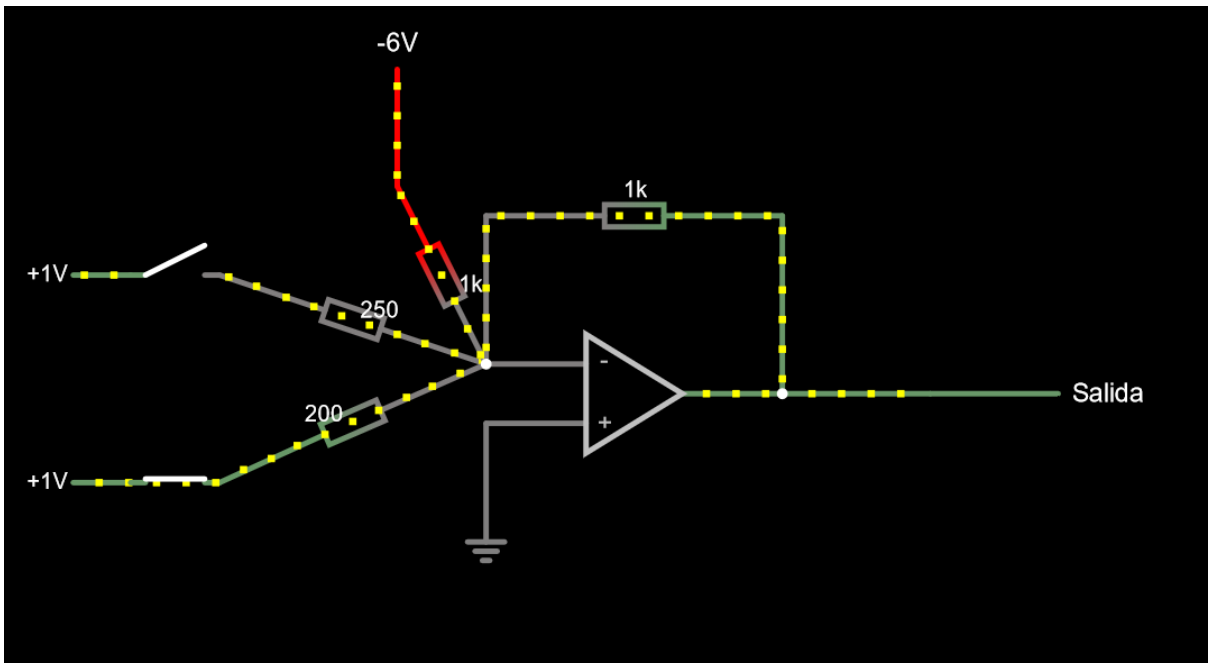


Figura 18, Circuit que implementa la porta lògica AND.

Com per tots els circuits, el fitxer per carregar-los al simulador es pot trobar a l'annex.

Si no ha quedat clar com funciona el circuit de la Figura 18, està ben explicat a l'apartat 3.2, ja que es tracta d'un sumador analògic simple.

A l'hora d'implementar el circuit, s'han adaptat els valors de les resistències per convertir els valors de les entrades de 5V a 1V. A continuació mostrem el circuit final que s'ha implementat.

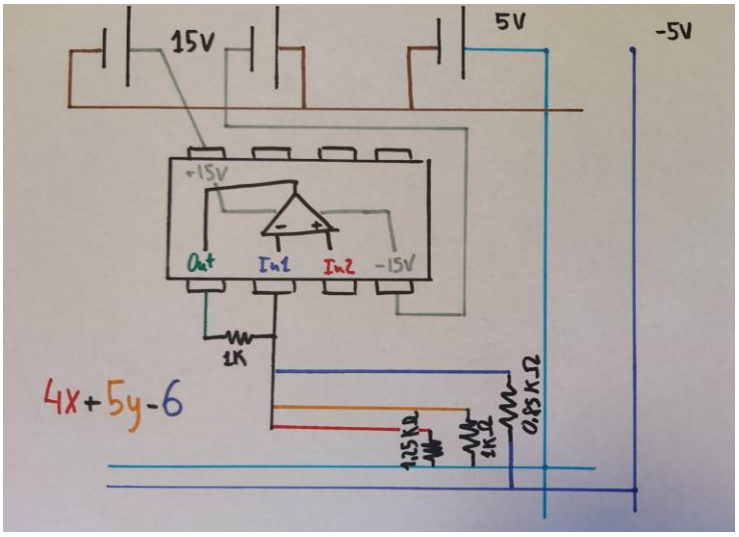


Figura 19, Circuit que implementa la porta lògica AND.

Ara ja es pot materialitzar aquest circuit. Les següents imatges són del circuit soldat.

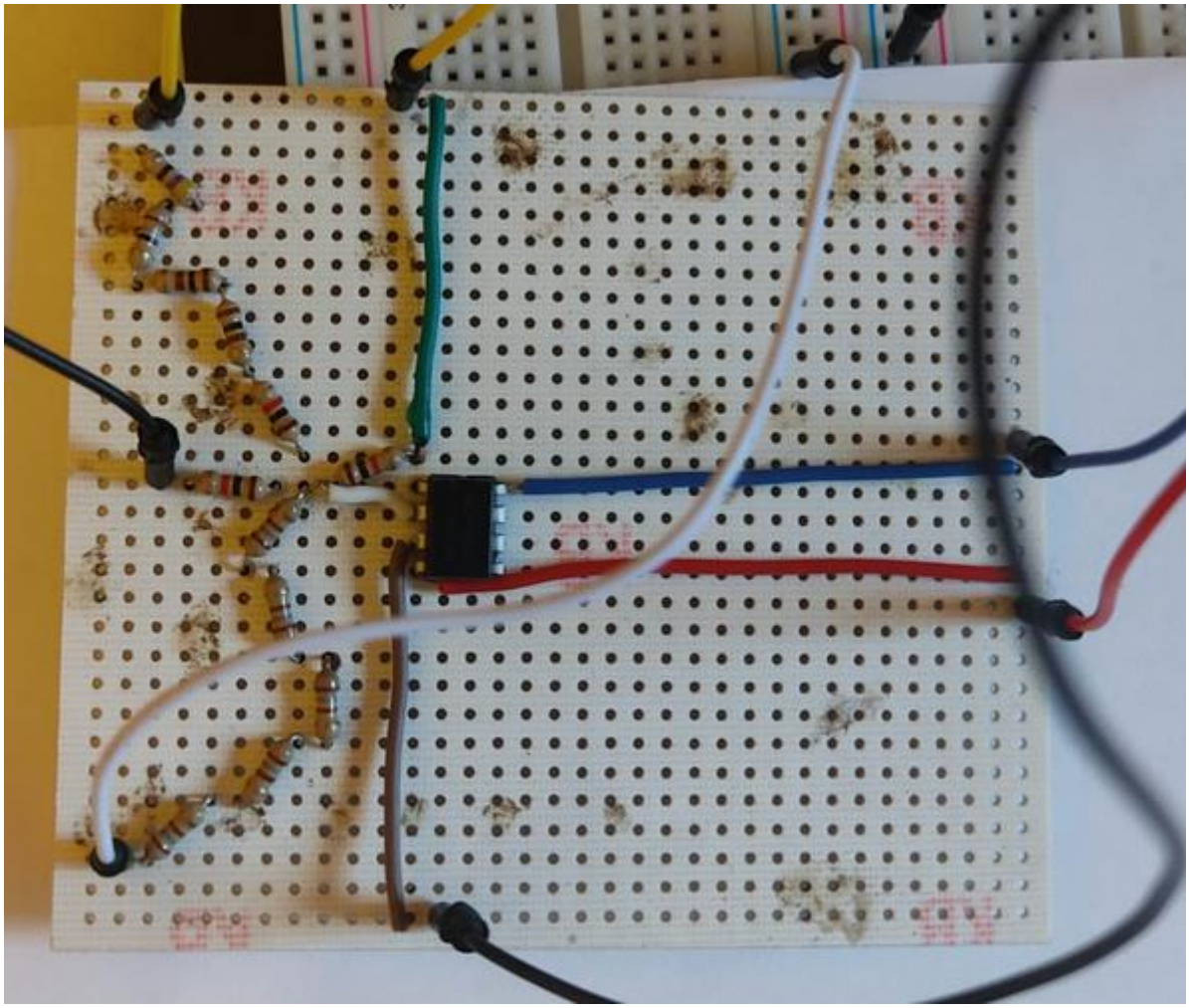


Figura 20, Circuit soldat que implementa la porta lògica AND.

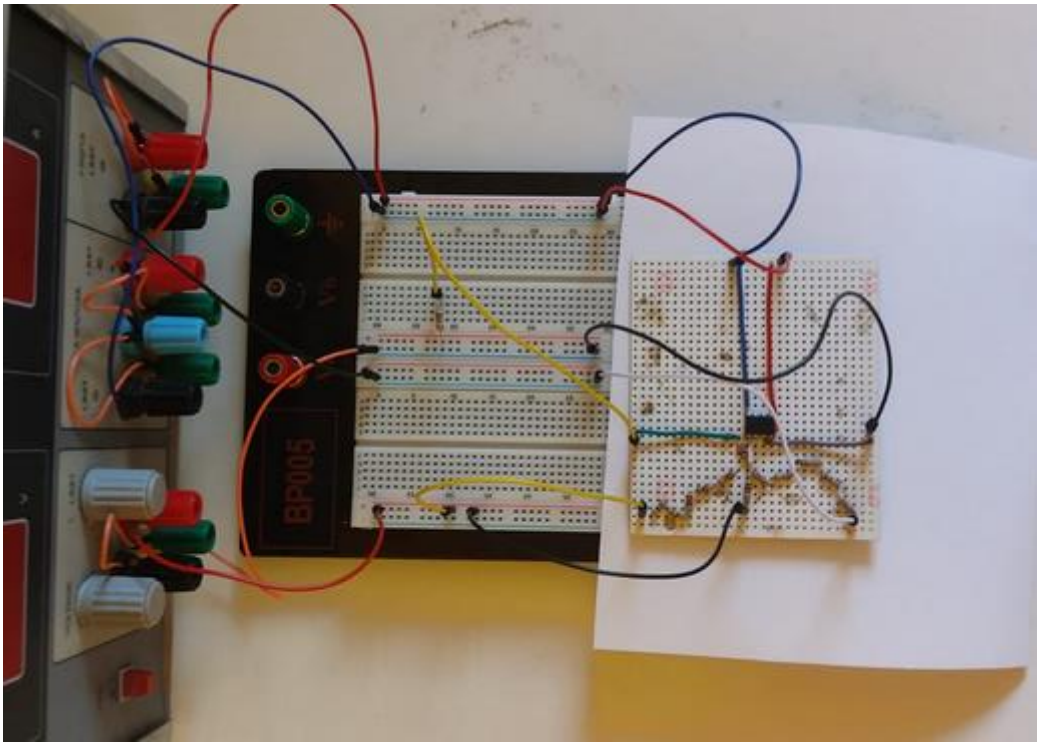


Figura 21, Circuit soldat i connectat.

Avaluació (Jocs de prova).

Sortida per a les dues entrades connectades, cables blanc i negre.

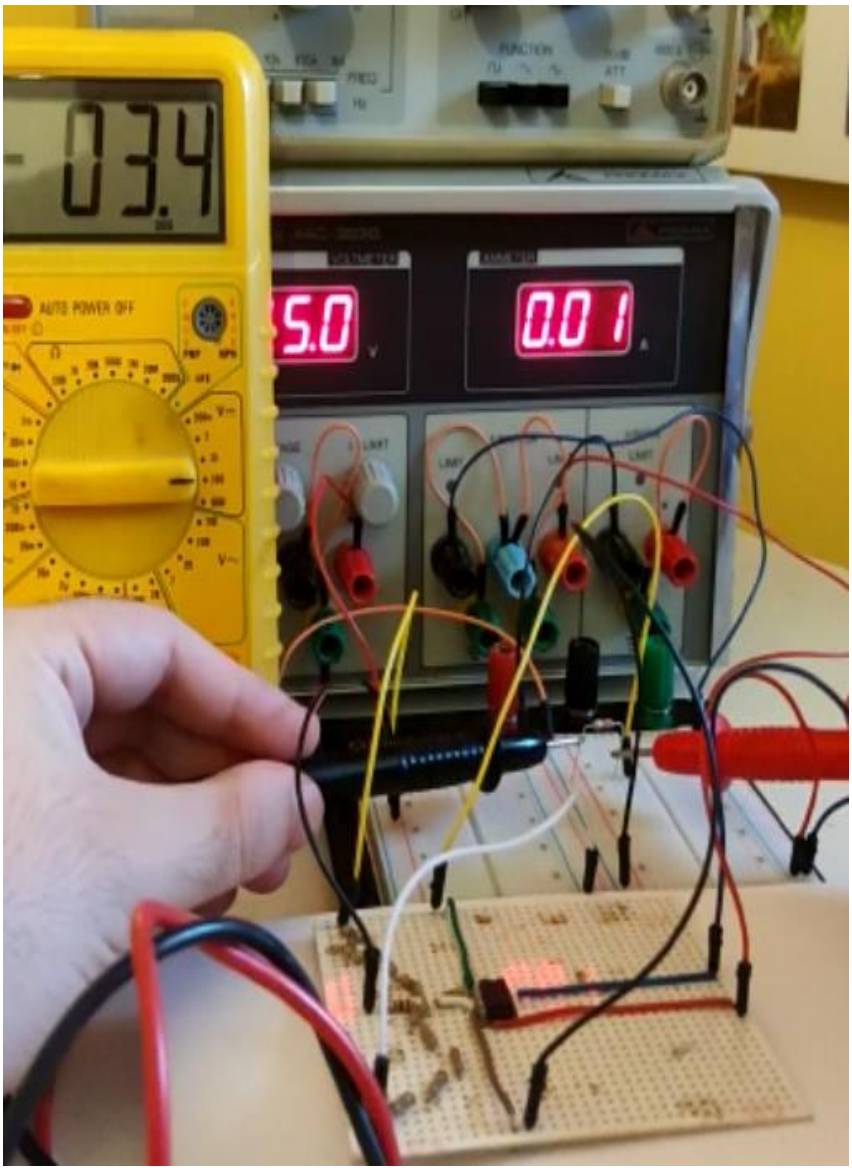


Figura 22, Test Y(1,1).

$$Y(1, 1) = (4 * 1 + 5 * 1 - 6) * -1 = -3$$

Com toca, ens queda més o menys el resultat que havíem estimat tant en aquest com per la resta d'entrades.

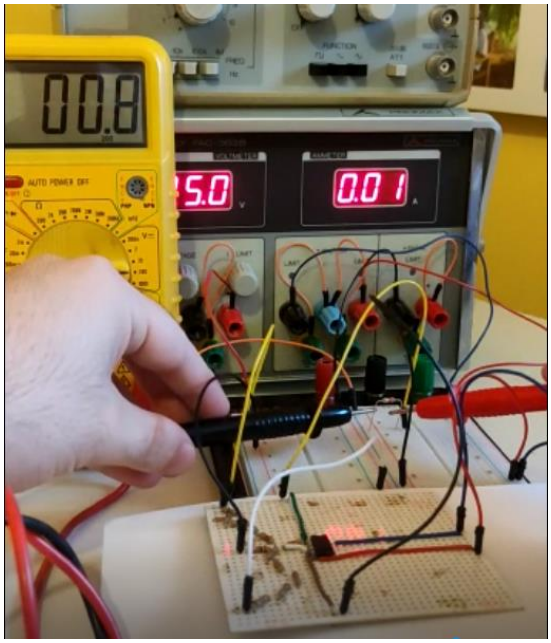


Figura 23, Test Y(0,1).

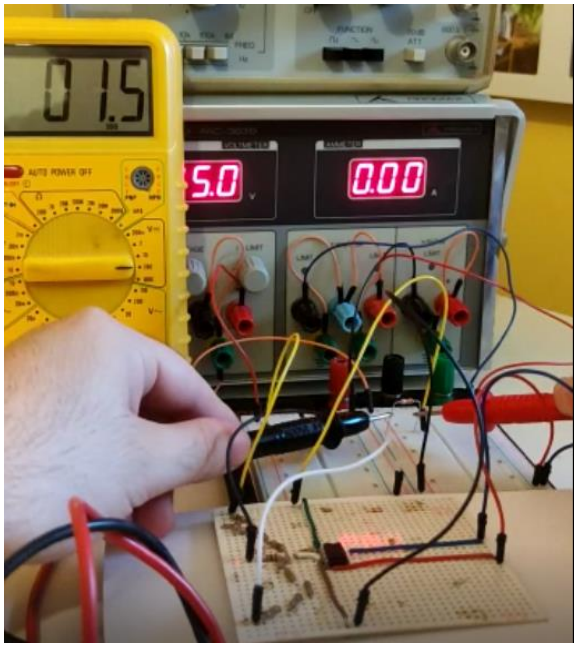


Figura 24, Test Y(1,0).

Y (0, 1) = (4 * 0 + 5 * 1 - 6) * -1= + 1;

Y (1, 0) = (4 * 1 + 5 * 0 - 6) * -1 = + 2

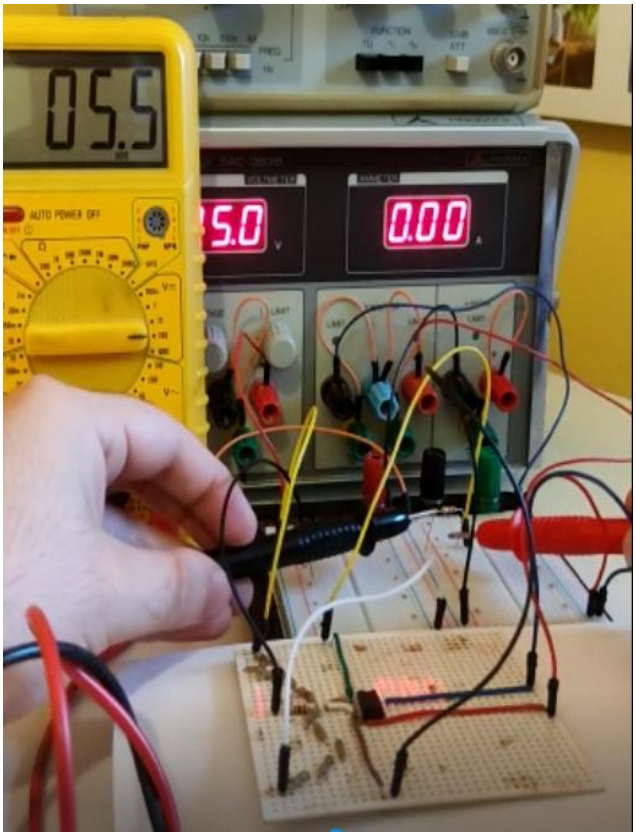


Figura 25, Test Y(0,0).

Y (0, 0) = (4 * 0 + 5 * 0 - 6) * -1 = + 6

El primer pas serà entrenar una xarxa. Els de la següent figura seran els pesos que utilitzaré. De totes les execucions que he fet amb backpropagation i totes les aleatòries, m'he quedat amb aquest pels següents motius:

El primer de tots és que l'error quadràtic mitjà és realment petit, sent del 0.067021542.

Els segon és que si ens fixem en les connexions entre la segona i tercera, i la tercera i última capa, tenen totes el mateix signe. Això ens permet una optimització que d'altra manera no podríem aplicar.

I és que amb un únic amplificador operacional per neurona en tenim prou, ja que el canvi de signe de l'amplificador ja resol que tots els pesos siguin negatius.

En les connexions que a la figura estan pintades de color taronja per exemple, haurem de negar l'input de baix amb un amplificador operacional per tal que la ponderació que apliquem amb les resistències sigui negativa, fer la suma ponderada amb un nou amplificador i finalment invertir el signe que se'ns haurà girat amb la darrera suma amb un altre amplificador operacional.

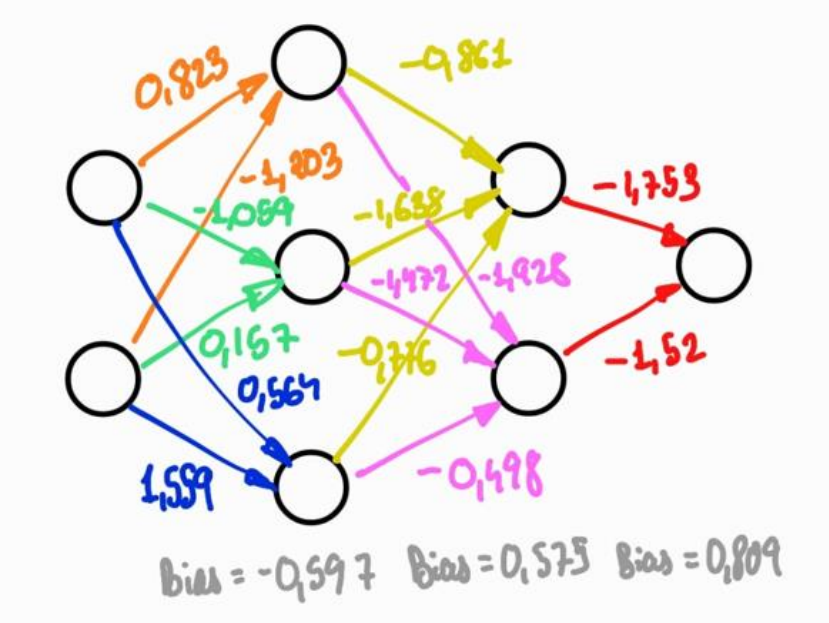


Figura 26, Model implementat.

Un dels altres exemples també estudiats era el següent:

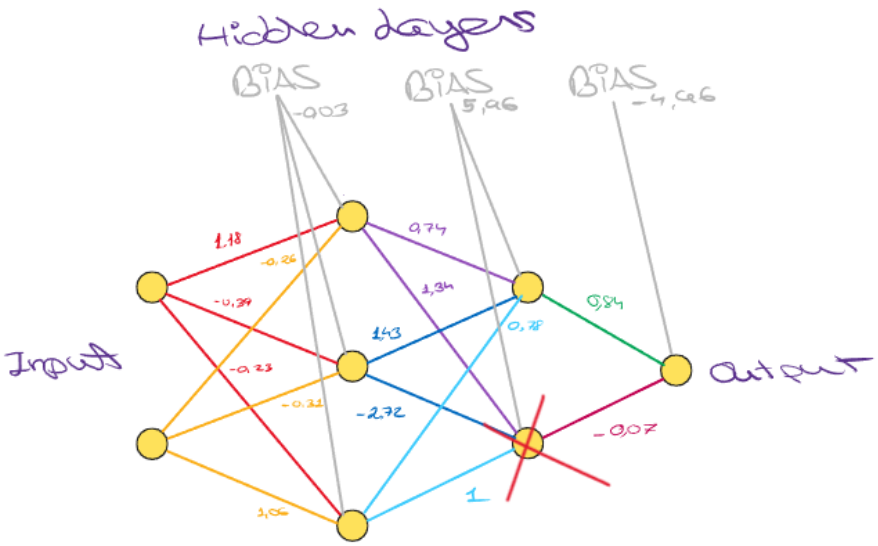


Figura 27, Model Alternatiu estudiat.

Va estar apunt de ser la candidata escollida pel fet que podia eliminar-ne una neurona de la última capa ja que està connectada a la sortida per un pes tan petit que és negligible si la utilitzem o no. L'error però, era notablement superior a la seleccionada i per això ha estat la implementada. Utilitzant els anteriors raonaments podem fàcilment crear el circuit que implementi la xarxa seleccionada:

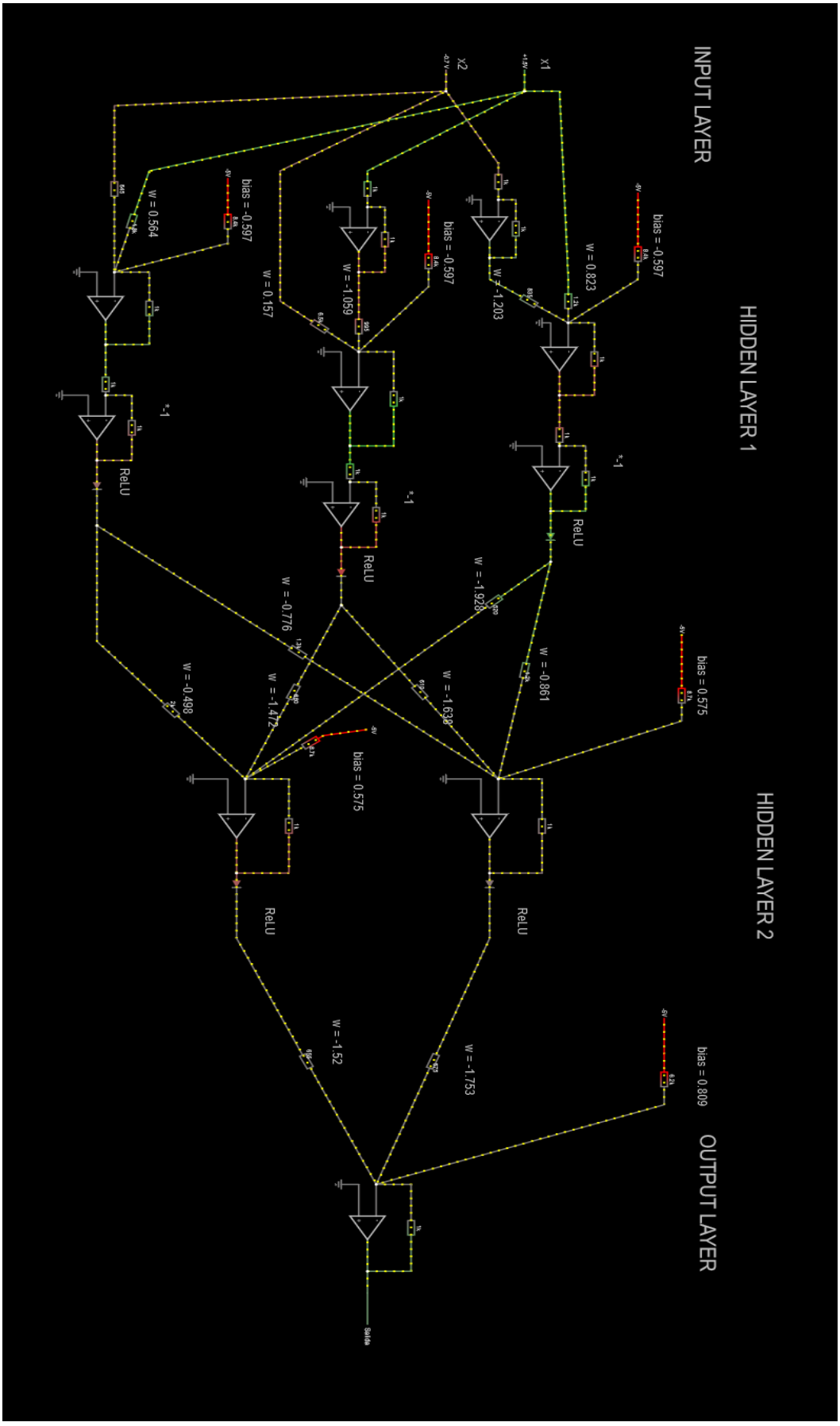


Figura 28, Circuit analògic del model de la Figura 26

A continuació he deixat una fotografia del circuit soldat en diferents mòduls amb etiquetes explicant cada component a què fa, juntament amb la imatge equivalent de l'esquema de la Figura 28.

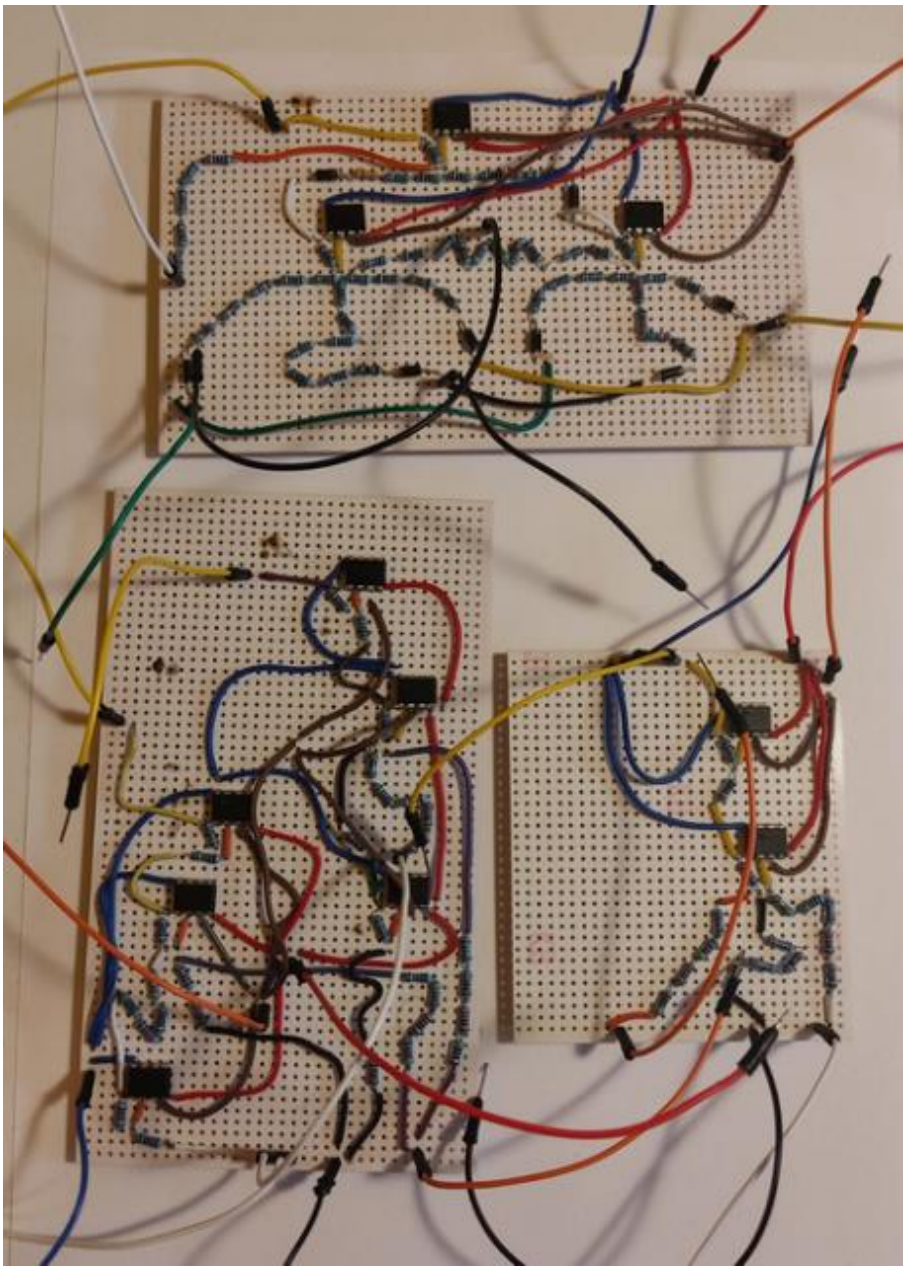


Figura 29, Circuit analògic del model de la Figura 26 soldat.

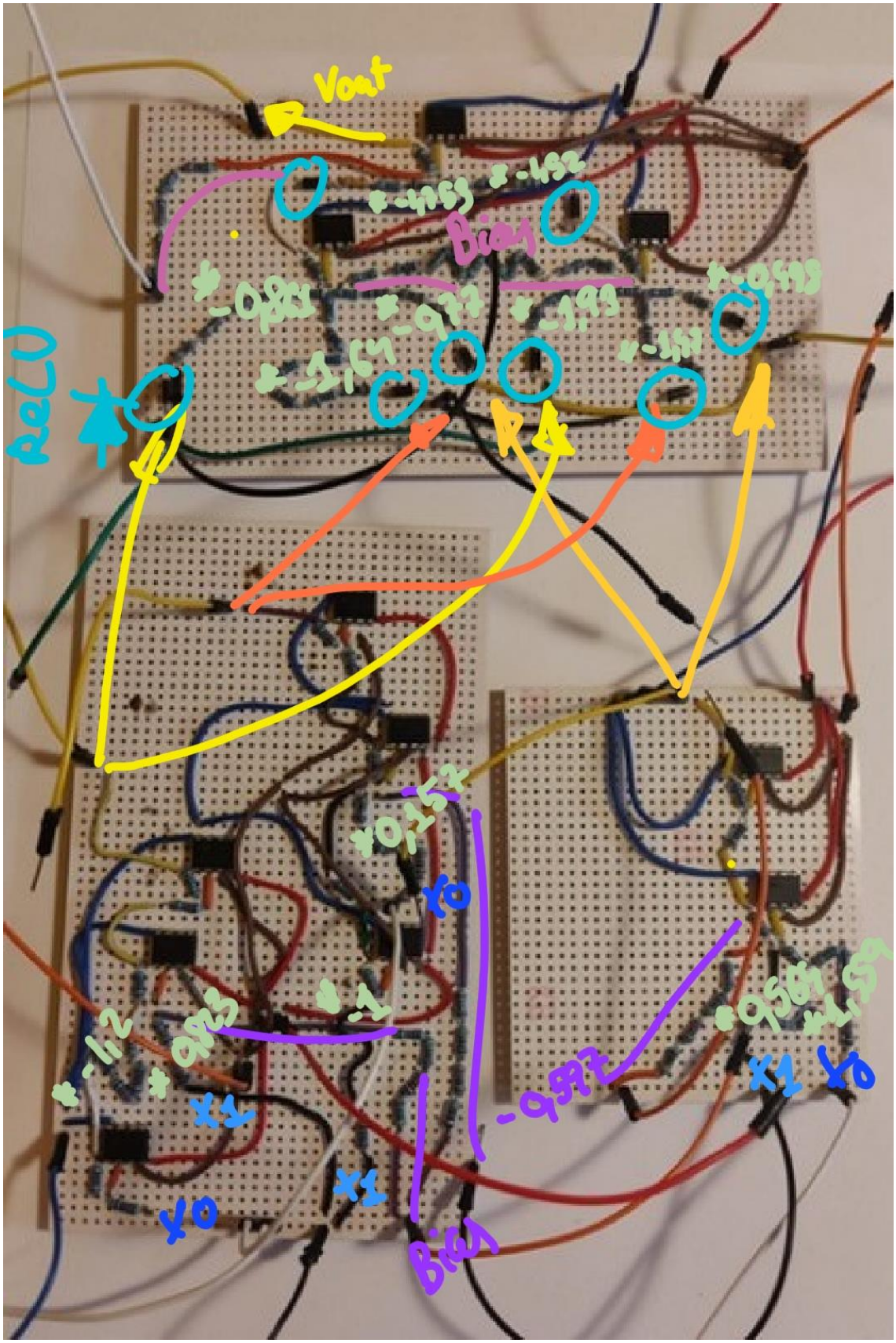


Figura 30, Circuit analògic de la Figura 29 amb anotacions.

Avaluació (Jocs de prova).

Per tal de provar el circuit s'ha utilitzat un "arduino uno" per llegir els valors de voltatge de les dues entrades, que amb dos potenciòmetres anirem variant. Al mateix temps, també llegirà la sortida per comprovar si està funcionant correctament. Amb la llibreria pyserial de python podrem recollir i tractar els resultats. Els guardo en un fitxer separats per comes per poder llegir com a csv.

Exemple del fitxer:

Vteoric,	Vx0,	Vx1,	Vout, Radi
0,	0.39,	-0.85,	0.0, 0.9352005132590551
-,	0.85,	-0.85,	0.44, 1.2020815280171306
1,	1.31,	-0.85,	0.79, 1.5616017418023074
1,	1.31,	-1.15,	0.79, 1.7431580536486069
1,	1.31,	-2.5,	0.79, 2.822428032740605
-,	1.31,	-0.35,	0.79, 1.355949851580065
-,	1.31,	-0.3,	0.79, 1.3439121995130487
0,	0.74,	-0.17,	0.18, 0.7592759709091287
0,	0.57,	-0.13,	0.0, 0.5846366392897386
0,	0.84,	-0.19,	0.0, 0.8612200647918045
-,	1.06,	-0.24,	0.72, 1.0868302535354821

A l'hora de fer les proves per voltatges negatius, hi ha un problema. L'arduino només llegeix entrades analògiques de 0V a 5V. És per això que s'han utilitzat amplificadors analògics inversors per llegir la senyal i després negar-la.

Aquest fet fa que per punts de diferents quadrants de l'eix de coordenades cartesians s'hagin de provar separats, ja que haurem de fer canvis mínims al circuit de test i afegir el signe a l'script de python que guarda els resultats al fitxer.

A l'Annex hi ha adjunts els codis dels dos programes fets en python i arduino.

A continuació s'adjunten fotografies de la configuració del circuit.

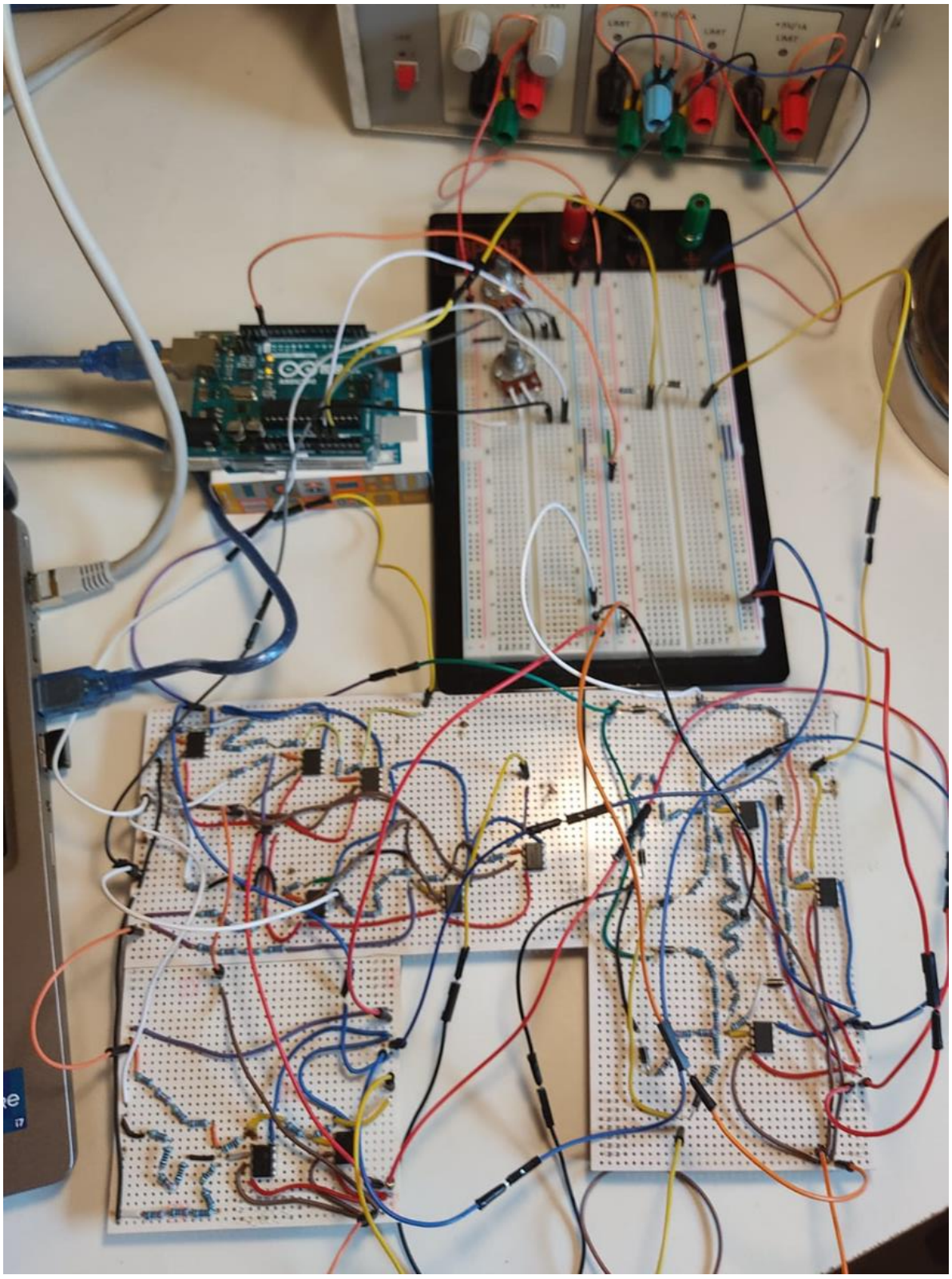


Figura 31, Circuit analògic del model de la Figura 26 soldat i connectat al circuit de test.

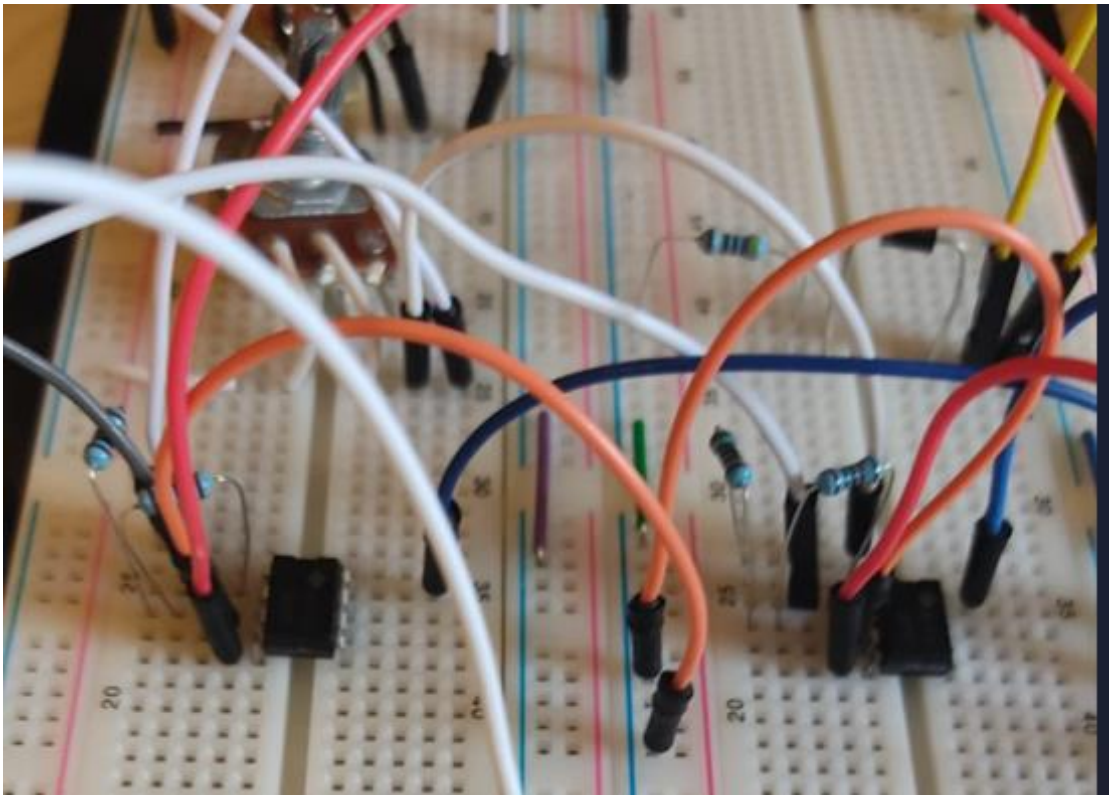


Figura 32, Part del circuit de test on es nega el Voltatge que es fa servir d'input amb amplificadors operacionals.

A partir del fitxer csv generat amb les proves s'ha pogut fer la següent gràfica. Cal recordar que la xarxa implementada havia de retornar 0 per als punts de dins del cercle de radi 1 i retornar 1 per a aquells que estan a dins l'anell d'entre les circumferències d'1.5 i 2.5.

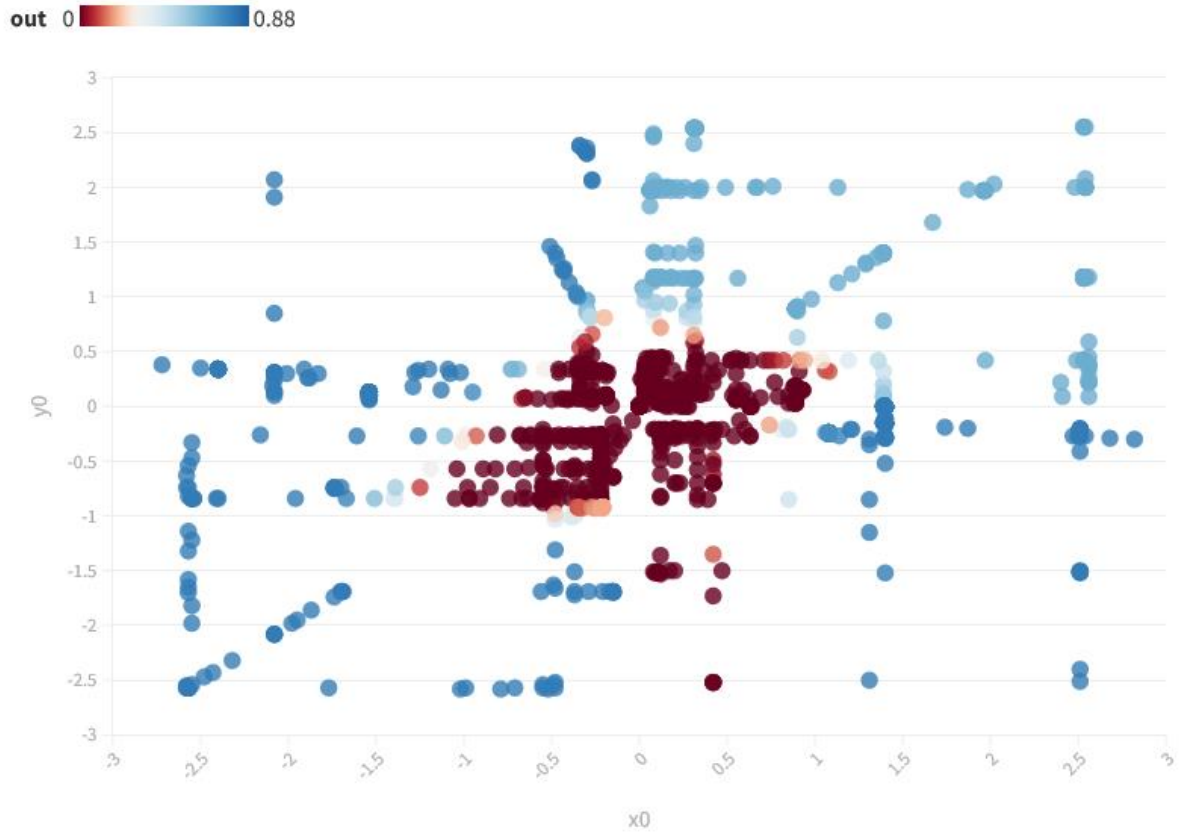


Figura 33, Resultats obtinguts.

Es pot veure que s'ha classificat correctament la gran majoria de punts, a excepció d'algun punt del quadrant de baix a la dreta.

A continuació deixo també els resultats teòrics que s'esperaven.

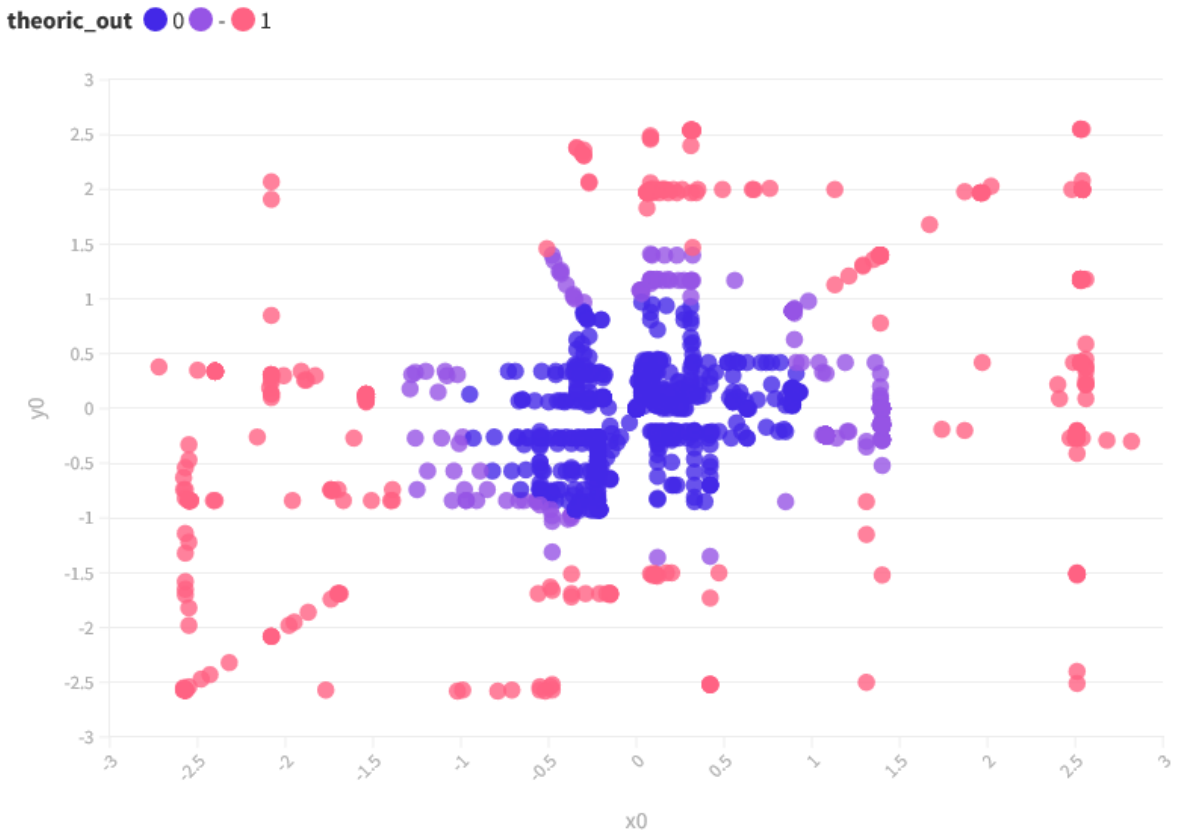


Figura 34, Resultats teòrics.

Com es pot observar quan esperem un 1V el circuit retorna 0.88V. Això és perquè el díode dissipa voltatge. És un dels inconvenients del circuit analògic implementat. Complicant més el circuit es podria solucionar, però no s'hi entrarà en aquest treball.

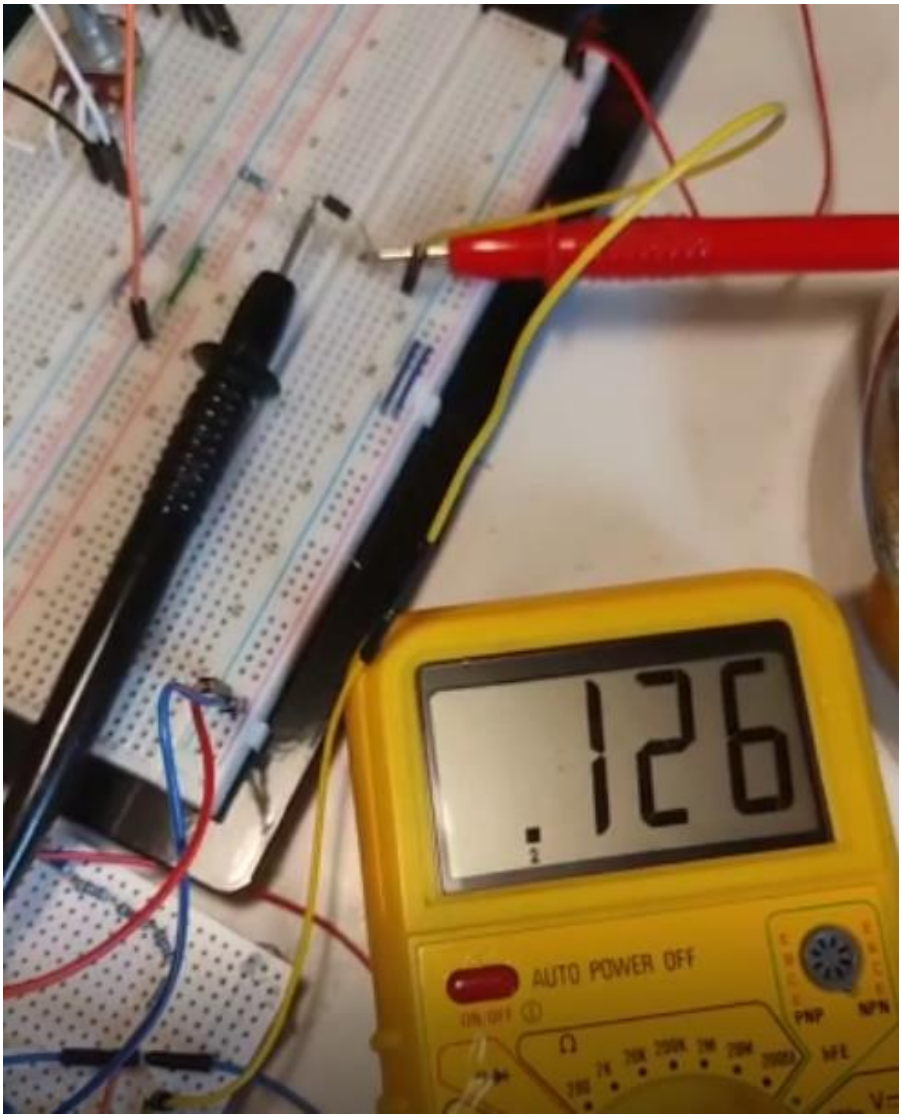


Figura 35, Diferència de tensió en els terminals de l'últim díode del circuit.

6. Conclusions

Per tal de poder treure'n conclusions sobre si té el algun sentit o ús el circuit ensenyat, primer parlaré dels avantatges i desavantatges de la implementació d'una xarxa neuronal analògica com la mostrada.

Desavantatges:

No és gens flexible. Com que els pesos els dicten les resistències que hem soldat, no es pot reprogramar i s'ha de dissenyar una pcb[11] ad hoc per a cada entrenament. Si bé es pot automatitzar aquest disseny, segueix sent un clar desavantatge.

El cost capital, sempre i quant no en fem moltes tirades d'un mateix circuit, és superior al que tindríem fent servir un microcontrollador qualsevol, sigui un esp32, arduino, etc, que es poden aconseguir a preus propers als d'1€ la unitat perfecte per l'economia d'escala. Tot i així, si s'inverteix en una línia per imprimir els circuits al final valdria la pena.

De manera simple només podem implementar la funció d'activació ReLU, que si bé va molt bé per a entrenar ràpidament les xarxes, els resultats obtinguts sempre són més inexactes que si podem utilitzar-ne d'altres com la sigmoide o fins i tot barrejar-ne.

Avantatges:

El consum és ínfim. Si utilitzem amplificadors operacionals per a un circuit com el que presentem el consum seria aproximadament d'uns 10mA * els 15 V que té d'alimentació cadascun dels amplificadors operacionals. Així doncs, per el circuit on en tenim 11, estaríem consumint 150mW.

Cal remarcar que aquest consum és una estimació per a generalitzar-ne el cost. L'ideal és, manualment, mirar el corrent que es subministra als terminals d'alimentació , i afegir-hi la que utilitza també el corrent en repòs que en resta a l'interior. Aquesta darrera informació la podem trobar als respectius *datasheets*.

També s'ha de tenir en compte que l'ús d'un microcontrolador com pot ser una placa arduino també consumeix poca potència. Més o menys necessiten uns 45 mA. Per un mínim voltatge requerit de 7V, la potència consumida serà de 0.3 Watts, molt proper al consum estimat per la implementació analògica descrita a sobre.

[11] Una pcb és una placa de prototipat de circuits.

Per a un circuit més gran igualarem i inclús es podria arribar a superar el cost de 0.3 Watts que dèiem que podia utilitzar un microcontrolador senzill, fet a tenir en compte.

El temps d'execució és gairebé instantani. Mentre que un microcontrolador ha d'executar instrucció a instrucció a una freqüència determinada, en cas del circuit analògic no cal cap rellotge.

Per a amplificadors operacionals ideals, el retard de l'output serà nul. Però per als reals no n'és el cas.

I és que els amplificadors operacionals reals tarden fraccions de microsegon a variar el voltatge. Per això als fulls de dades dels mateixos hi podem trobar el que se'n diu *l'slew rate*. En el cas del lm358 tenim un slew rate de 0,3 V/microSegon. Així doncs, per un circuit amb un sol amplificador operacional, si l'augment és d'1V tardarà a donar el resultat corresponent gairebé 1 microsegon. En el nostre circuit, com a molt hi ha increments de 2V, i no són habituals. Per tal d'estimar quant tarda haurem de tenir en compte que tots amplificadors comencen a regular el voltatge alhora, pel que el temps mitjà total no s'allunya gaire de 2V*0.3 microsegons. És a dir, 6.6 microsegons.

El processador de l'arduino uno, té una freqüència de 16 MHz. Per tant, en 6.6 microsegons pot executar 16*6.6 instruccions, arribant a un total de 105. El nombre d'instruccions que es necessiten per a executar el *forward* de la xarxa descrita és lleugerament inferior, però per a xarxes més grans, aquest nombre augmentarà, mentre que el temps d'un circuit analògic equivalent es mantindria gairebé als 6.6 microsegons donats (Tenint en compte una variació de voltatge de 2V a cada neurona com a màxim).

Un processador de 3Ghz en canvi, trigarà 0,3 nanosegons per instrucció, podent arribar a les 20000 instruccions en aquests 6.6 segons. En aquest cas però, el consum es dispararia. Per exemple, un i5-3330, de 3.2 GHz, té un TDP[12] de 77W, un consum notable.

Ras i curt, els avantatges són circumstancials, i els desavantatges molt notoris. Això fa que sigui difícil pensar-ne una aplicació concreta realment efectiva.

Una aplicació on tindria especialment sentit, és tractar directament els valors que capturi un sensor analògic i tenir-los com a entrada, per acabar donar-ne sortides a altres dispositius directament o fins i tot enviar aquesta sortida a un microcontrolador.

Al final el propi circuit analògic ja pot actuar com a ADC[13] si s'entrena per retornar 0 o 1 com a l'exemple implementat en aquest treball.

[12] TDP significa Potència de Diseny Tèrmic i es refereix al consum d'alimentació de la càrrega teórica máxima del procesador.
[13] Un ADC (Analog to Digital Converter) serveix per convertir una senyal analògica a digital.

Com a conclusió des del prisma acadèmic, aquest treball m'ha servit per entendre a fons com funciona una xarxa neuronal, així com a veure conceptes d'electrònica que no havia vist a la carrera fins ara.

7. Recursos utilitzats

7.1 Part I Xarxa Neuronal

Aioobe et al., 1958. Generate a random point within a circle (uniformly). *Stack Overflow*. Available at: https://stackoverflow.com/questions/5837572/generate-a-random-point-within-a-circle-uniformly [Accessed April 15, 2022].

Anon, 2018. ¿Qué es una red neuronal? parte 3.5 : Las Matemáticas de Backpropagation | DotCSV. *YouTube*. Available at: https://www.youtube.com/watch?v=M5QHwkkHgAA&list=PL-Ogd76BhmcC_E2RjgIIJZd1DQdYHcVf0&index=10 [Accessed April 7, 2022].

Aioobe et al., 1958. Generate a random point within a circle (uniformly). *Stack Overflow*. Available at: https://stackoverflow.com/questions/5837572/generate-a-random-point-within-a-circle-uniformly [Accessed September 3, 2022].

Mizutani, E., Dreyfus, S.E. & Nishio, K., On derivation of MLP backpropagation from the Kelley-Bryson optimal-control gradient formula and its application. Available at: https://coeieor.wpengine.com/wp-content/uploads/2019/03/ijcnn2k.pdf [Accessed April 8, 2022].

Smilkov , D.S., Tensorflow - Neural Network Playground. *A Neural Network Playground*. Available at: https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=4,2&seed=0.14678&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false [Accessed March 2, 2022].

Aioobe et al., 1958. Generate a random point within a circle (uniformly). *Stack Overflow*. Available at: https://stackoverflow.com/questions/5837572/generate-a-random-point-within-a-circle-uniformly [Accessed April 8, 2022].

7.2 Part II Computació Analògica

Falstad, P., Simulador Circuits Electrònics CircuitJS. *Paul Falstad*. Available at: https://www.falstad.com/circuit/circuitjs.html [Accessed March 6, 2022].

Anon, Gràfiques Online. *Flourish*. Available at: https://flourish.studio/ [Accessed September 3, 2022].

Khutoryansky, E., 2016. OP amp circuits: Analog Computers from operational amplifiers. *YouTube*. Available at: https://www.youtube.com/watch?v=_o4ScgRZtNI&list=LL&index=99&t=253s [Accessed April 1, 2022].

Anon, 2022. Operational amplifier. *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Operational_amplifier [Accessed March 1, 2022].

Mancini, R., Understanding basic analog ideal OP AMPS (rev. B) - texas instruments. *Understanding Basic Analog – Ideal Op Amps*. Available at: https://www.ti.com/lit/an/slaa068b/slaa068b.pdf [Accessed April 20, 2022].

8. Annexes:

8.1 Codi

8.1.1 Codi c#

Codi de la implementació per consolades de zero de l'estructura de la xarxa neuronal i els algorismes d'entrenament.

8.1.1.1 DataSet.cs

Codi per generar les dades.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace ZeroToNeuralNet
{
	internal class DataSet
	{
	private Tuple<double, double>[] type1;
	private Tuple<double, double>[] type2;
	private readonly Random random;
	private double radius1;
	private double radius2;
	private int dim = 100000;

	public DataSet(double radius1, double radius2, int dim)
	{
		this.random = new Random();
		this.dim = dim;
		this.type1 = new Tuple<double, double>[dim];
		this.type2 = new Tuple<double, double>[dim];
		for (int i = 0; i < dim; i++)
		{
		this.type1[i] = RandomPoint(0, radius1);
		this.type2[i] = RandomPoint(radius1 + 0.5, radius2 + 0.5);
		}
		this.radius1 = radius1;
		this.radius2 = radius2;
	}

	/*
```

```
		Retorna un punt aleatori comprès entre les circumferències 2*PI*radius1
i 2*PI*radius2, on radius1 < radius
	*/
	public Tuple<double, double> RandomPoint(double radius1, double radius2)
	{
		double r = radius2 * Math.Sqrt(random.NextDouble() * (1 -
Math.Pow((radius1 / radius2), 2)) + Math.Pow((radius1 / radius2), 2)); //  radi
random;	2PIr PIr^2 creixen diferent per això l'arrel quadrada
		double theta = random.NextDouble() * 2 * Math.PI;
	// angle random
		double x = r * Math.Cos(theta);  // passem a coordenades cartesianes
		double y = r * Math.Sin(theta);

		return new Tuple<double, double>(x, y);

	}

	public void printData()
	{
		Console.WriteLine("Points within 0-1 radius");
		for (int i = 0; i < dim; i++)
		{
		Console.Write(" " + type1[i] + ",");
		}
		Console.WriteLine("\nPoints within 1.5-2.5 radius");
		for (int i = 0; i < dim; i++)
		{
		Console.Write(" " + type2[i] + ",");
		}
		Console.WriteLine("\n");
	}

	public Tuple<double, double>[] GetType1()
	{
		return this.type1;
	}

	public Tuple<double, double>[] GetType2()
	{
		return this.type2;
	}
	}

}
```

8.1.1.2 Neuron.cs

Classse Neurona

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ZeroToNeuralNet
{
	internal class Neuron
	{
	private List<Connection> fromConn;
	private List<Connection> toConn;
	private static int count=0;
	private int id;
	private double value;
	private double aValue;

	public Neuron()
	{
		this.fromConn = new List<Connection>();
		this.toConn = new List<Connection>();
		this.id = count;
		count++;
		this.value = 0;
		this.aValue = 0;
	}

	public void SetNewConnectionFrom(Neuron neuron)
	{
		this.fromConn.Add(new Connection(neuron, this.id));
	}

	public void SetNewConnectionTo(Neuron neuron)
	{
		this.toConn.Add(new Connection(neuron));
	}

	public List<Connection> GetFromConn()
	{
		return this.fromConn;
	}
	public List<Connection> GetToConn()
	{
		return this.toConn;
	}

	public int GetId()
	{
```

```
		return this.id;
	}

	public double GetValue()
	{
		return this.value;
	}
	public double GetAValue()
	{
		return this.aValue;
	}

	public void SetValue(double val)
	{
		this.value = val;
	}

	public void SetAValue(double val)
	{
		this.aValue = val;
	}
	}
}
```

8.1.1.3 Layer.cs

Classe per les capes

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ZeroToNeuralNet
{
	internal class Layer
	{
	private static readonly Random random = new Random();

	private List<Neuron> neurons;
	private double bias;
	private int n_neurons;

	public Layer(int n_neurons)
	{
		this.neurons = new List<Neuron>();
		for (int i = 0; i < n_neurons; i++)
		{
		this.neurons.Add(new Neuron());
		}
		this.n_neurons = n_neurons;
		this.bias = random.NextDouble() * 2 - 1;

	}

	public void ConnectLayerFrom(Layer from)
	{
		foreach (Neuron a in neurons)
		{
		foreach (Neuron b in from.neurons)
		{
			a.SetNewConnectionFrom(b);
		}
		}

	}

	public void ConnectLayerTo(Layer to)
	{
		foreach (Neuron a in neurons)
		{
		foreach (Neuron b in to.neurons)
		{
			a.SetNewConnectionTo(b);
```

```
                }
                }

        }

        public double GetBias()
        {
                return this.bias;
        }

        public void SetBias(double bias)
        {
                this.bias = bias;
        }
        public List<Neuron> GetNeurons()
        {
                return this.neurons;
        }

        }

}
```

8.1.1.4 NeuralNet.cs

Classe de la xarxa neuronal.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ZeroToNeuralNet
{
	internal class NeuralNet
	{
	private List<Layer> layers;
	private int[] topology;
	private double learningRate;

	public NeuralNet(List<int> topology, double learning_rate)
	{
		this.layers = new List<Layer>();
		this.topology = new int[topology.Count];
		topology.CopyTo(this.topology);
		this.layers.Add(new Layer(topology[0]));
		for (int i=1; i<topology.Count; i++)
		{
		layers.Add(new Layer(topology[i]));
		layers[i - 1].ConnectLayerTo(layers[i]);
		layers[i].ConnectLayerFrom(layers[i - 1]);
		}
		this.learningRate = learning_rate;
	}

	public int Forward(double[] input)
	{
		double sum;

		// Save input in Input Layer Neurons
		for (int i = 0; i < this.layers[0].GetNeurons().Count; i++)
		{
		this.layers[0].GetNeurons()[i].SetValue(input[i]);
		this.layers[0].GetNeurons()[i].SetAValue(input[i]);
		}

		// Go trough each layer applying weights and activation funct
		for (int i = 1; i < this.layers.Count; i++)
		{
		foreach (Neuron neuron in this.layers[i].GetNeurons())
		{
```

```
                sum = this.layers[i].GetBias();
                foreach(Connection conn in neuron.GetFromConn())
                {
                        sum += conn.GetNeuron().GetAValue()*conn.GetWeight();
                }
                neuron.SetValue(sum);
                neuron.SetAValue(Relu(sum));
        }
        }

        return 0;
}

public double Train(double[] input, double[] expected_output)
{
        double[] output = new double[layers.Last().GetNeurons().Count];
        List<double[]> deltas = new List<double[]>();
        double[] d;
        double[] dmse;
        double aux, id;
        double deltamean;

        // Forward
        this.Forward(input);
        // Save Output
        for (int i=0; i< output.Length; i++)
        {
        output[i] = layers.Last().GetNeurons()[i].GetAValue();
        }


        /*  Backward
        *
        *  Per a Capa l
        *  dC/dW = deltaL = Dmse(expected_output, output) * DRelu(value)
        *
        *  Per a Capes l-1 -> 0
        *  deltaL-1[] = deltaL[] * WL[] * Drelu(valueL-1)
        *  deltaL-2[] = deltaL-1[] * WL-1[] * Drelu(valueL-2)
        */
        // Output layer Delta
        d = new double[this.layers.Last().GetNeurons().Count];
        dmse = Dmse(expected_output, output);
        for (int i=0; i < this.layers.Last().GetNeurons().Count; i++)
        {
        d[i] = dmse[i] * DRelu(this.layers.Last().GetNeurons()[i].GetValue());
        }
        deltas.Add(d);

        // Hidden Layers Delta

        // Per cada capa restant
```

```
            for (int i=1; i< this.layers.Count-1; i++)
            {
            // Deltas[dim]
            d    =    new    double[this.layers[this.layers.Count    -    1    -
i].GetNeurons().Count];

            // Per cada neurona en trobem la seva delta
            for  (int  j  =  0;  j  <  this.layers[this.layers.Count  -  1  -
i].GetNeurons().Count; j++)
            {
                  d[j] = 0;
                  //delta L * WL per cada neurona a la capa anterior
                  for   (int   k=0;   k   <   this.layers[this.layers.Count   -
i].GetNeurons().Count; k++)
                  {
                        d[j]   +=   this.layers[this.layers.Count   -   1   -
i].GetNeurons()[j].GetToConn()[k].GetNeuron().GetAValue() * deltas[i - 1][k];
                  }
                  //  * Drelu
                  d[j]  =  d[j]  *  DRelu(this.layers[this.layers.Count  -  1  -
i].GetNeurons()[j].GetValue());
            }
            deltas.Add(d);
            }

            //Console.WriteLine("DEBUG1");
            /*  Gradient Descend
            *
            *  Per cada deltas[l]
            *  Bias[l] = Bias[l] - mean(Deltas L) * learning_rate
            *  Weights[l] = Weights[l] - values[l-1] * Deltas[l] * lr
            *
            */
            for (int i = 0; i < this.layers.Count - 1; i++)
            {
            deltamean = 0;
            // weigths
            for   (int   j=0;   j<   this.layers[this.layers.Count   -   1   -
i].GetNeurons().Count; j++)
            {
                  for  (int  k=0;  k  <  this.layers[this.layers.Count  -  1  -
i].GetNeurons()[j].GetFromConn().Count; k++ )
                  {
                        //Weights[l] = Weights[l] - values[l-1] * Deltas[l] * lr
                        aux    =    this.layers[this.layers.Count    -    1    -
i].GetNeurons()[j].GetFromConn()[k].GetWeight() - this.layers[this.layers.Count -
2 - i].GetNeurons()[k].GetAValue() * deltas[i][j] * this.learningRate;
                        this.layers[this.layers.Count       -       1       -
i].GetNeurons()[j].GetFromConn()[k].SetWeight(aux);

                        //Ho hauriem hem de guardar també a l'anterior capa al
connectedTO però com que no mirem mai els pesos ena questa direcció no cal
```

```
                }
                deltamean += deltas[i][j];
            }
            deltamean     /=     this.layers[this.layers.Count     -     1     -
i].GetNeurons().Count;
            // update Bias
            this.layers[this.layers.Count            -            1            -
i].SetBias(this.layers[this.layers.Count   -   1   -   i].GetBias()   -   deltamean*
learningRate);
            }

            //Console.WriteLine("DEBUG2");
            // retornem mse
            return MeanSquaredError(expected_output, output);
      }

      public double Relu(double sum)
      {
            return Math.Max(0, sum);
      }

      public double DRelu(double sum)
      {
            if (sum > 0)
            {
            return 1;
            }
            else
            {
            return 0;
            }
      }
      public double MeanSquaredError(double[] teoric, double[] output)
      {
            double sum = 0;
            for (int i=0; i < output.Length; i++)
            {
            sum += Math.Pow((teoric[i] - output[i]), 2);
            }
            return sum / output.Length;
      }

      public double[] Dmse(double[] teoric, double[] output)
      {
            double[] dmse = new double[teoric.Length];
            for (int i=0; i< teoric.Length; i++)
            {
            dmse[i] = output[i] - teoric[i];
            }
            return dmse;
      }
```

```
	public void PrintWeights()
	{
		Console.WriteLine("\tWeights");
		foreach (Layer layer in this.layers)
		{
		if (this.layers.First() != layer) {
			foreach (Neuron neuron in layer.GetNeurons())
			{
				foreach (Connection conn in neuron.GetFromConn())
				{
				Console.Write("\t id:" + neuron.GetId() + "From " +
conn.GetNeuron().GetId() + " w: " + conn.GetWeight());
				}
			}
			Console.WriteLine("\t\t\t Bias: " + layer.GetBias());
			Console.WriteLine();
		}
		}
	}


	public void PrintWeightsToFile(String path)
	{
		StreamWriter sw = new StreamWriter(path);

		sw.WriteLine("\tTopology");
		for (int i = 0; i < this.layers.Count; i++)
		{
		sw.WriteLine("\tLayer " + i);
		foreach (Neuron neuron in this.layers[i].GetNeurons())
		{
			sw.Write("\tid: " + neuron.GetId());
		}
		sw.WriteLine();
		}
		sw.WriteLine("\n\n\tWeights");
		foreach (Layer layer in this.layers)
		{
		if (this.layers.First() != layer)
		{
			foreach (Neuron neuron in layer.GetNeurons())
			{
				foreach (Connection conn in neuron.GetFromConn())
				{
				sw.Write("\t id:" + neuron.GetId() + "From " +
conn.GetNeuron().GetId() + " w: " + conn.GetWeight());
				}
			}
			sw.WriteLine("\t\t\t Bias: " + layer.GetBias());
			sw.WriteLine();
		}
```

```
        }
        sw.Close();
}

public void PrintValues()
{
        Console.WriteLine("\tValues");
        foreach (Layer layer in this.layers)
        {
        foreach ( Neuron neuron in layer.GetNeurons())
        {
                Console.Write("\t"+neuron.GetAValue());
        }
        Console.WriteLine();
        }
}

public void PrintNoActivationValues()
{
        Console.WriteLine("\tValues Before Activation Function");
        foreach (Layer layer in this.layers)
        {
        foreach (Neuron neuron in layer.GetNeurons())
        {
                Console.Write("\t" + neuron.GetValue());
        }
        Console.WriteLine();
        }
}

public void PrintTopology()
{
        for (int i=0; i< this.layers.Count; i++)
        {
        Console.WriteLine("\tLayer " + i);
        foreach(Neuron neuron in this.layers[i].GetNeurons())
        {
                Console.Write("\tid: "+ neuron.GetId());
        }
        Console.WriteLine();
        }
}

public List<Layer> GetLayers()
{
        return this.layers;
}

public int[] GetTopology()
{
        return this.topology;
}
```

```
	public void SetLearningRate(double learingRate)
	{
		this.learningRate = learingRate;
	}

	public double GetLearningRate()
	{
		return this.learningRate;
	}
	}
}
```

8.1.1.5 Program.cs

Main del programa de consola.

```
using ZeroToNeuralNet;

namespace Program
{
	class Program
	{
	static void Main(String[] args)
	{
		List<int> topology = new List<int> { 2, 3, 2, 1 };
		NeuralNet xarxa = new NeuralNet(topology, 0.05);
		DataSet trainData = new DataSet(1, 2, 1000);
		DataSet testData = new DataSet(1, 2, 20);
		double x,y;

		bool end = false;
		while (!end)
		{
		Console.Clear();
		Console.WriteLine("Choose an option:");
		Console.WriteLine("1) Generate Datasets");
		Console.WriteLine("2) Initialize Neural Net");
		Console.WriteLine("3) Train Neural Net");
		Console.WriteLine("4) Forward");
		Console.WriteLine("5) Save Weights To File");
		Console.WriteLine("6) Test");
		Console.WriteLine("7) Restart + train");
		Console.WriteLine("8) Exit");
		Console.Write("\r\nSelect an option: ");

		switch (Console.ReadLine())
		{
			case "1":
				Console.WriteLine("Enter dimension of Training
Dataset:");
				trainData = new DataSet(1, 2,
Convert.ToInt32(Console.ReadLine()));
				Console.WriteLine("Enter dimension of Test Dataset:");
				testData = new DataSet(1, 2,
Convert.ToInt32(Console.ReadLine()));
				Console.WriteLine("Test data ");
				testData.printData();
				Console.WriteLine("Press any key to continue");
				Console.ReadLine();
				break;

			case "2":
				int a;
```

```
                        List<int> topo = new List<int> { };

                        while (true)
                        {
                        Console.WriteLine("Enter Neural Net topology. ex:  2\n
3\n 2\n 1\n 0\n  (0 to end)");
                        a = Convert.ToInt32(Console.ReadLine());
                        if (a != 0)
                        {
                              topo.Add(a);
                        }
                        else
                        {
                              break;
                        }
                        }
                        Console.WriteLine("Enter learning Rate");
                        xarxa          =          new          NeuralNet(topo,
Convert.ToDouble(Console.ReadLine()));
                        break;

                  case "3":
                        for (int i = 0; i < trainData.GetType1().Length; i++)
                        {
                        Console.WriteLine("mse: " + xarxa.Train(new double[] {
trainData.GetType1()[i].Item1, trainData.GetType1()[i].Item2 }, new double[] { 0
}));
                        Console.WriteLine("mse: " + xarxa.Train(new double[] {
trainData.GetType2()[i].Item1, trainData.GetType2()[i].Item2 }, new double[] { 1
}));
                        }
                        xarxa.Forward(new double[] { 1.2, -1.1 });
                        Console.WriteLine("Resultat        :        "        +
xarxa.GetLayers().Last().GetNeurons()[0].GetAValue());
                        xarxa.Forward(new double[] { -0.5, -0.2 });
                        Console.WriteLine("Resultat        :        "        +
xarxa.GetLayers().Last().GetNeurons()[0].GetAValue());
                        Console.WriteLine("Press any key to continue");
                        Console.ReadLine();
                        break;

                  case "4":
                        Console.WriteLine("enter input x: ");
                        x = Convert.ToDouble(Console.ReadLine());
                        Console.WriteLine("enter input y: ");
                        y = Convert.ToDouble(Console.ReadLine());
                        xarxa.Forward(new double[] {x,y});
                        Console.WriteLine("Resultat        :        "        +
xarxa.GetLayers().Last().GetNeurons()[0].GetAValue());
                        Console.WriteLine("Print values");
                        xarxa.PrintValues();
                        Console.WriteLine("Print values before activation");
```

```
                        xarxa.PrintNoActivationValues();
                        Console.WriteLine("Print weights");
                        xarxa.PrintWeights();

      Console.WriteLine("##########################################");
                        xarxa.PrintTopology();
                        Console.WriteLine("Press any key to continue");
                        Console.ReadLine();
                        break;
                  case "5":
                        xarxa.PrintWeights();

      xarxa.PrintWeightsToFile("C:\\Users\\OuDret\\Desktop\\output.txt");
                        Console.WriteLine("Press any key to continue");
                        Console.ReadLine();

                        break;
                  case "6":
                        for (int i = 0; i < testData.GetType1().Length; i++)
                        {
                        xarxa.Forward(new              double[]              {
testData.GetType1()[i].Item1, testData.GetType1()[i].Item2 });
                        Console.WriteLine("Resultat        :        "        +
xarxa.GetLayers().Last().GetNeurons()[0].GetAValue());
                        xarxa.Forward(new              double[]              {
testData.GetType2()[i].Item1, testData.GetType2()[i].Item2 });
                        Console.WriteLine("Resultat        :        "        +
xarxa.GetLayers().Last().GetNeurons()[0].GetAValue());
                        }
                        Console.WriteLine("Press any key to continue");
                        Console.ReadLine();
                        break;
                  case "8":
                        end = true;
                        break;

                  case "7":
                        topology = new List<int> { 2, 3, 2, 1 };
                        xarxa = new NeuralNet(topology, 0.005);
                        trainData = new DataSet(1, 2, 100000);
                        testData = new DataSet(1, 2, 20);
                        for (int i = 0; i < trainData.GetType1().Length; i++)
                        {
                        Console.WriteLine("mse: " + xarxa.Train(new double[] {
trainData.GetType1()[i].Item1, trainData.GetType1()[i].Item2 }, new double[] { 0
}));
                        Console.WriteLine("mse: " + xarxa.Train(new double[] {
trainData.GetType2()[i].Item1, trainData.GetType2()[i].Item2 }, new double[] { 1
}));
                        }
                        xarxa.Forward(new double[] { 2, -1.1 });
```

```
                        Console.WriteLine("Resultat        :        "        +
xarxa.GetLayers().Last().GetNeurons()[0].GetAValue());
                        xarxa.Forward(new double[] { -0.5, -0.2 });
                        Console.WriteLine("Resultat        :        "        +
xarxa.GetLayers().Last().GetNeurons()[0].GetAValue());
                        Console.WriteLine("Press any key to continue");
                        Console.ReadLine();
                        break;
                  default:
                        break;
            }
            }
            /*
            for (int i=0; i < 40000; i++)
            {
                  Console.WriteLine("mse:   "   +   xarxa.Train(new   double[]   {
trainData.GetType1()[i].Item1, trainData.GetType1()[i].Item2 }, new double[] {0}));
                  Console.WriteLine("mse:   "   +   xarxa.Train(new   double[]   {
trainData.GetType2()[i].Item1, trainData.GetType2()[i].Item2 }, new double[] {1}));
            }

            xarxa.Forward(new double[] { 1.2, -1.1 });
            Console.WriteLine("Resultat             :             "             +
xarxa.GetLayers().Last().GetNeurons()[0].GetAValue());
            xarxa.Forward(new double[] { -0.5, -0.2 });
            Console.WriteLine("Resultat             :             "             +
xarxa.GetLayers().Last().GetNeurons()[0].GetAValue());

            xarxa.PrintWeightsToFile("C:\\Users\\OuDret\\Desktop\\output.txt");
            xarxa.PrintTopology();
            */

      }

      }
}
```

8.1.2 Codi d'Implementació Aleatòria en Python

Codi en python per executar una xarxa neuronal inicialitzada amb pesos aleatoris i la mateixa topologia que es troba a la figura 27. Les dades novament seran punts d'un cercle de centre (0,0) i radi d'1u i un anell entre circumferències d'1.5u i 2.5u de radi, també centrades al centre de l'eix cartesià. Si per 10000 entrades diferents hi ha menys d'un 0,1 d'EQM de mitjana es guarden els pesos.

```
from argparse import _CountAction
from datetime import datetime
import math
import random
DIM = 10000
NUMRUNS = 1000000

def randomPoint(r1, r2):
	r = r2*math.sqrt(random.uniform(0,1) * (1 - math.pow((r1 / r2), 2)) +
math.pow((r1 / r2), 2))
	theta = random.uniform(0, 2*math.pi)
	x = r * math.cos(theta)
	y = r * math.sin(theta)
	return x,y

def initializeDataset(r1,r2):
	data=[]
	for i in range (DIM):
	data.append(randomPoint(0,r1))
	data.append(randomPoint(r1+0.5,r2+0.5))
	return data

def mse(teoric, output):
	return math.pow((teoric - output), 2)

def initRandomWeights():
	return random.uniform(-2,2), random.uniform(-2,2), random.uniform(-2,2),
random.uniform(-2,2),random.uniform(-2,2),random.uniform(-2,2),random.uniform(-
2,2),random.uniform(-2,2),random.uniform(-2,2),random.uniform(-
2,2),random.uniform(-2,2),random.uniform(-2,2),random.uniform(-
2,2),random.uniform(-2,2),random.uniform(-2,2),random.uniform(-
2,2),random.uniform(-2,2), random.uniform(-4,4),random.uniform(-
4,4),random.uniform(-4,4)

def hardcodedForward(x1,x2, weights):
	#forward first layer
	x3 = max(0,x1*weights[0]+x2*weights[3] + weights[14])
	x4 = max(0,x1*weights[1]+x2*weights[4] + weights[14])
	x5 = max(0,x1*weights[2]+x2*weights[5] + weights[14])

	#forward second layer
	x6 = max(0,x3*weights[6]+ x4*weights[8] + x5*weights[10] + weights[15])
	x7 = max(0,x3*weights[7]+ x4*weights[9] + x5*weights[11] + weights[15])
```

```
	#forward last (output layer)
	out = max(0,x6*weights[12]+x7*weights[13]+weights[16])

	return out

def main():
	try:
	data=initializeDataset(1,2)
	count=0
	while(True):

		with open("C:\\Users\\Joan\\Desktop\\log" + str(count)+ ".txt", "a")
as f:
		dt = datetime.now()
		goodOnesCounter = 0
		for i in range(NUMRUNS):
			weights = initRandomWeights()
			sum=0
			for j in range(len(data)):
				out = hardcodedForward(data[j][0], data[j][1], weights)
				error = mse(j%2, out)
				sum += error
			errorMean = sum/len(data)
			if (errorMean < 0.1):
				f.write("\nmean error= " + str(errorMean) +"\nweights =
" + str(weights) +  "\n")
				goodOnesCounter +=1
		#save log
		f.write("\n"+ str(goodOnesCounter) + " from " + str(NUMRUNS) + " have
decent weights")
		f.write("\nprogram have been running from " + str(dt) + " to "+
str(datetime.now()))
		count += 1
	except:
	return 1

main()
```

8.1.3 Codi Test Circuit

8.1.3.1 Codi Arduino

Codi per llegir 3 entrades analògiques amb un arduino i enviar els resultats pel serialport a un ordinador que les tracti.

```
float x0;
float x1;
float out;

void setup() {
  // put your setup code here, to run once:
  Serial.begin (9600);

}

void loop() {
  // put your main code here, to run repeatedly:
  x0  = analogRead(A0); //* (5 / 1023);
  x1  = analogRead(A1); //* (5 / 1023);
  out = analogRead(A2); //* (5 / 1023);
  Serial.print(x0*5/1024);
  Serial.print(",");
  Serial.print(x1*5/1024);
  Serial.print(",");
  Serial.print(out*5/1024);
  Serial.println(",");

  delay(1000);

}
```

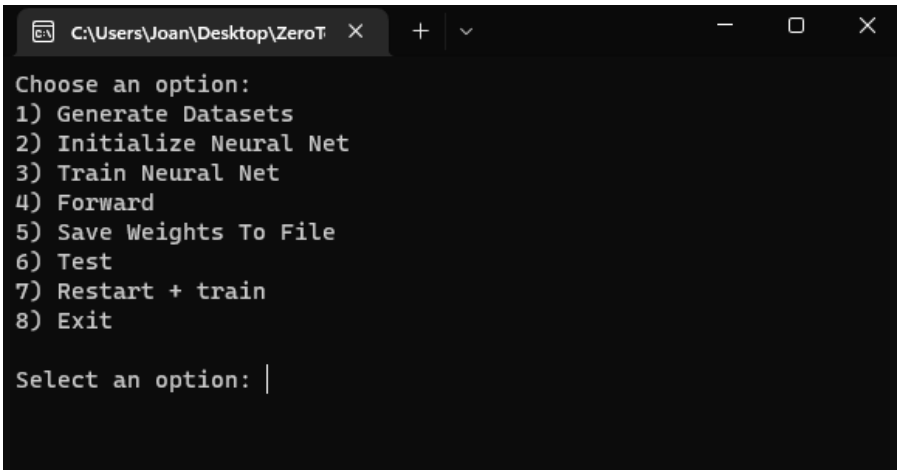
8.1.3.2 Codi Python

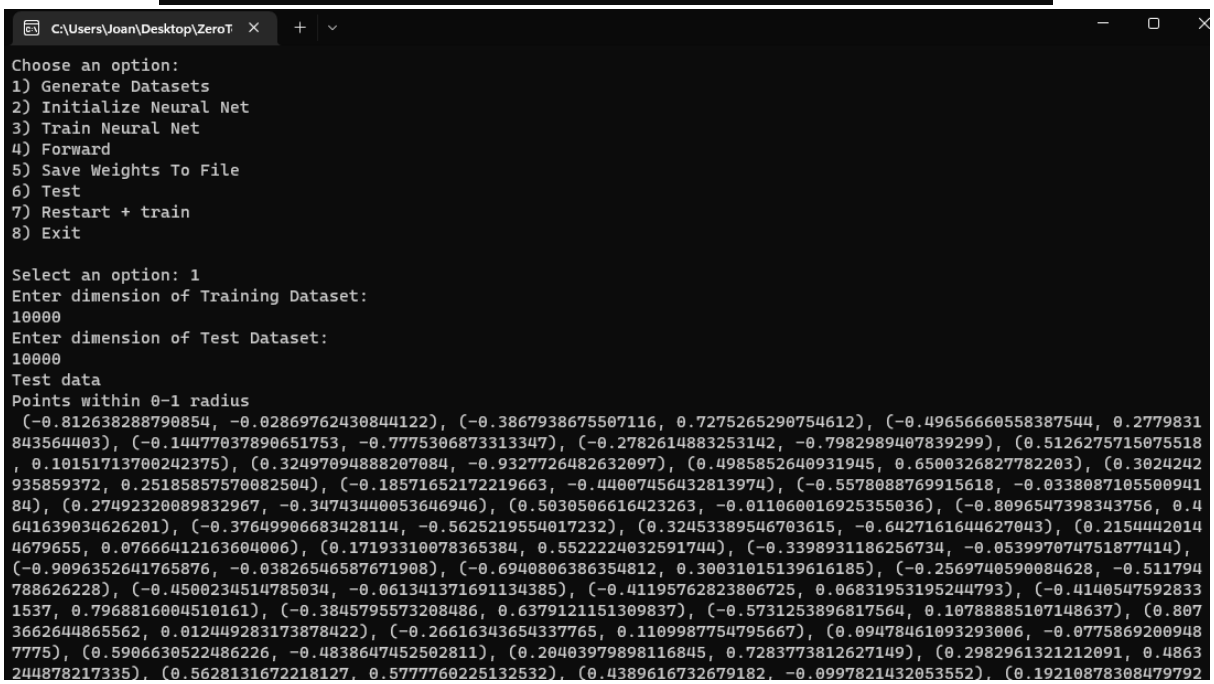
Codi python que llegeix del serial port que el connecta a l'arduino per recollir les dues entrades i la sortida del circuit, validar-ne el resultat i escriure-ho a un fitxer de log.

```
import serial
import time
import math

# make sure the 'COM#' is set according the Windows Device Manager
ser = serial.Serial('COM3', 9600, timeout=1)
time.sleep(2)

while True:
	line = ser.readline()   # read a byte
	if line:
	strin = line.decode()  # convert the byte string to a unicode string
	input= strin.split(',')
	print(input)
	x0=float(input[0])
	x1=float(input[1])
	out=float(input[2])
	print(x0, x1, out)
	radius = math.sqrt( math.pow(x0,2) + math.pow(x1,2) )
	if radius > 1.5:
		result = 1
	elif radius < 1:
		result = 0
	else:
		result = "-"

	f = open("C:/Users/jri2206/Desktop/TESTTFG/nn0.txt", "a")
	f.write(str(result)  +",  "+  str(x0*-1)+",  "+  str(x1*-1)+",  "+str(out)+",
"+str(radius) + "\n")
	f.close()

ser.close()
```

8.2 Imatges del la terminal del programa d'entrenament.

A continuació hi ha alguna captura de la terminal del programa fet que implementa l'entrenament d'una xarxa neuronal des de 0 fet.





```
1) Generate Datasets
2) Initialize Neural Net
3) Train Neural Net
4) Forward
5) Save Weights To File
6) Test
7) Restart + train
8) Exit

Select an option: 2
Enter Neural Net topology. ex:  2
 3
 2
 1
 0
  (0 to end)
2
```

```
Enter Neural Net topology. ex:  2
 3
 2
 1
 0
  (0 to end)
0
Enter learning Rate
0.111
```

```
mse: 0.025401634049152223
mse: 0.18717449175145 37
mse: 0.06612248735766565
mse: 0.01632509848287327
mse: 0.06309147706967526
mse: 0.00029028186716114224
mse: 0.006193410716502255
mse: 0.0026964013373136247
Resultat : 1.1160116064065484
Resultat : 0
```

8.4 Esquemes Circuits per Importar a Falstad.

Esquema AND

Les següents línies es poden importar al simulador CircuitJS de Falstad per tal de carregar el circuit de la figura 18.

```
$ 1 0.000005 10.20027730826997 50 5 43 5e-11
g 192 240 192 304 0 0
r 48 160 192 208 0 250
r 48 272 192 208 0 200
w 192 208 192 128 0
r 192 128 352 128 0 1000
w 352 128 352 224 0
O 432 224 528 224 0 0
w 352 224 432 224 0
a 192 224 352 224 8 15 -15 1000000 -0.00000999930004899657 0 100000
R 144 112 144 32 0 0 40 -6 0 0 0.5
r 192 208 144 112 0 1000
R 0 160 -48 160 0 0 40 1 0 0 0.5
R 0 272 -48 272 0 0 40 1 0 0 0.5
s 0 272 48 272 0 0 false
s 0 160 48 160 0 1 false
```

Esquema Xarxa Donut

Les següents línies es poden importar al simulador CircuitJS de Falstad per tal de carregar el circuit de la figura 28.

```
$ 1 0.000005 10.20027730826997 50 5 43 5e-11
R -368 176 -432 176 0 0 40 1.5 0 0 0.5
R -368 320 -432 320 0 0 40 -0.7 0 0 0.5
x -440 299 -415 302 4 24 x2
x -439 145 -414 148 4 24 x1
a 144 112 304 112 8 15 -15 1000000 0.00001474025802725030 4 0 100000
w 144 96 144 48 0
r 144 48 304 48 0 1000
w 304 48 304 112 0
w 304 112 368 112 0
g 144 128 144 144 0 0
w -368 176 -368 112 0
w -368 112 48 96 0
r 48 96 144 96 0 1215
w -368 320 -208 224 0
r -208 224 -128 224 0 1000
a -128 240 0 240 8 15 -15 1000000 -0.00000699986000279994 2 0 100000
g -128 256 -128 272 0 0
w -128 224 -128 192 0
r -128 192 0 192 0 1000
w 0 192 0 240 0
w 0 240 48 240 0
r 48 240 144 96 0 835
x -31 63 73 66 4 24 w\s\q\s0.823
x 25 232 137 235 4 24 w\s\q\s-1.203
R -64 -32 -160 -32 0 0 40 -5 0 0 0.5
x -96 -61 43 -58 4 24 bias\s\q\s-0.597
r -64 -32 48 -32 0 8350
w 48 -32 144 96 0
a 416 128 560 128 8 15 -15 1000000 -0.00001473996322798574 5 0 100000
g 416 144 416 192 0 0
r 368 112 416 112 0 1000
w 416 112 416 64 0
r 416 64 560 64 0 1000
w 560 64 560 128 0
x 435 13 465 16 4 24 *-1
d 592 128 640 128 2 1N5712
```

w 560 128 592 128 0
x 577 86 638 89 4 24 ReLU
x -80 323 59 326 4 24 bias\s\q\s-0.597
R -48 352 -144 352 0 0 40 -5 0 0 0.5
w -368 176 -192 464 0
r -192 464 -112 464 0 1000
w -112 464 -112 432 0
r -112 432 32 432 0 1000
a -112 480 32 480 8 15 -15 1000000 0.00001499970000599988 1 0 100000
w 32 432 32 480 0
g -112 496 -112 528 0 0
w 32 480 96 480 0
w -368 320 -192 624 0
w -192 624 96 624 0
a 208 496 416 496 8 15 -15 1000000 -0.00002214786572040084 0 100000
g 208 512 208 544 0 0
r 96 624 208 480 0 6450
w 208 480 208 416 0
r 208 416 416 416 0 1000
w 416 416 416 496 0
w 416 496 448 496 0
r 96 480 208 480 0 995
w 64 352 208 480 0
x 14 509 126 512 4 24 w\s\q\s-1.059
x 28 656 132 659 4 24 w\s\q\s0.157
x 657 470 718 473 4 24 ReLU
w 640 512 672 512 0
d 672 512 720 512 2 1N5712
x 515 397 545 400 4 24 *-1
w 640 448 640 512 0
r 496 448 640 448 0 1000
w 496 496 496 448 0
r 448 496 496 496 0 1000
g 496 528 496 576 0 0
a 496 512 640 512 8 15 -15 1000000 0.00002214742277194540 2 0 100000
w -32 720 32 928 0
R -128 720 -192 720 0 0 40 -5 0 0 0.5
x -174 694 -35 697 4 24 bias\s\q\s-0.597
a 304 960 448 960 8 15 -15 1000000 0.00000836572289817297 2 0 100000
g 304 976 304 1024 0 0
r 256 944 304 944 0 1000
w 304 944 304 896 0
r 304 896 448 896 0 1000
w 448 896 448 960 0

x 323 845 353 848 4 24 *-1
d 480 960 528 960 2 1N5712
w 448 960 480 960 0
x 465 918 526 921 4 24 ReLU
w -368 176 -192 864 0
w -368 320 -336 928 0
a 32 944 192 944 8 15 -15 1000000 -0.000008365890212630937 0 100000
r -192 864 32 928 0 1770
w 32 928 32 864 0
r 32 864 192 864 0 1000
w 192 864 192 944 0
g 32 960 32 992 0 0
w 192 944 256 944 0
x -149 866 -45 869 4 24 w\s\q\s0.564
r -336 928 32 928 0 645
w 528 960 592 960 0
w 720 512 768 512 0
w 640 128 672 128 0
x -679 -144 -420 -141 4 40 INPUT\sLAYER
x 105 -220 431 -217 4 40 HIDDEN\sLAYER\s1
x 1181 -251 1507 -248 4 40 HIDDEN\sLAYER\s2
x 1937 -146 2240 -143 4 40 OUTPUT\sLAYER
w 1024 -112 1152 224 0
x 880 -141 1011 -138 4 24 bias\s\q\s0.575
R 912 -112 816 -112 0 0 40 -5 0 0 0.5
a 1152 240 1360 240 8 15 -15 1000000 0.000003244769335884853 5 0 100000
w 1408 240 1440 240 0
x 1436 188 1497 191 4 24 ReLU
d 1360 240 1408 240 2 1N5712
w 1152 224 1152 144 0
r 1152 144 1360 144 0 1000
w 1360 144 1360 240 0
r 672 128 1152 224 0 1160
w 1440 240 1504 240 0
r 768 512 1152 224 0 610
r 592 960 1152 224 0 1290
g 1152 256 1152 320 0 0
x 862 147 981 150 4 24 w\s\s\q\s-0.861
x 913 325 1032 328 4 24 w\s\s\q\s-1.638
x 706 620 825 623 4 24 w\s\s\q\s-0.776
w 1440 704 1504 704 0
w 1360 608 1360 704 0
r 1152 608 1360 608 0 1000
d 1360 704 1408 704 2 1N5712

x 1436 652 1497 655 4 24 ReLU
w 1408 704 1440 704 0
a 1152 704 1360 704 8 15 -15 1000000 0.000014311051822603231 0 100000
g 1152 720 1152 784 0 0
w 1152 688 1152 608 0
R 1056 544 1040 448 0 0 40 -5 0 0 0.5
x 1092 489 1223 492 4 24 bias\s\q\s0.575
w 1088 592 1152 688 0
a 2048 464 2240 464 8 15 -15 1000000 -0.000008103647144302188 0 100000
g 2048 480 2048 512 0 0
w 2048 448 2048 384 0
r 2048 384 2240 384 0 1000
w 2240 384 2240 464 0
w 2240 464 2288 464 0
O 2288 464 2384 464 0 0
r 672 128 848 336 0 520
w 848 336 1152 688 0
x 662 267 781 270 4 24 w\s\s\q\s-1.928
r 768 512 1152 688 0 680
r 848 960 1152 688 0 2000
x 919 645 1038 648 4 24 w\s\s\q\s-1.472
w 592 960 848 960 0
x 896 801 1008 804 4 24 w\s\q\s-0.498
R 1760 -80 1664 -80 0 0 40 -5 0 0 0.5
x 1744 -141 1875 -138 4 24 bias\s\q\s0.809
r 1760 -80 1872 -80 0 6170
w 1872 -80 2048 448 0
r 912 -112 1024 -112 0 8700
r 1504 240 2048 448 0 575
r 1504 704 2048 448 0 655
x 1739 284 1851 287 4 24 w\s\q\s-1.753
x 1684 528 1783 531 4 24 w\s\q\s-1.52
r -48 352 64 352 0 8350
r -128 720 -32 720 0 8350
r 1056 544 1088 592 0 8700
o 138 64 0 4098 2.5 0.1 0 1

8.5 Resultats Execució aleatòria

Resultats després de 32 dies seguits d'executar l'script de l'apartat 8.1.2.

mean error= 0.0838198335509
weights = (-0.43476199203995014, -1.907148568377567, 1.6448340946261109, 0.9976090128117008, -1.77069951404419, -0.6371437921638186, 0.30979328895781366, 1.4128096586111694, -0.10222926090997975, 1.0799304490194355, 0.3308243287768642, 0.6770067250077783, 1.5550521520906155, 0.23582634447398654, 0.2740636681699069, -0.16216494131235315, -0.6497573792129767)

0 from 100000 have decent weights
program have been running from 2022-08-06 14:34:05.562403 to 2022-08-07 00:48:39.378033
0 from 100000 have decent weights
program have been running from 2022-08-07 00:48:39.378620 to 2022-08-07 11:03:14.929802
0 from 100000 have decent weights
program have been running from 2022-08-07 11:03:14.930361 to 2022-08-07 21:17:50.899366
0 from 100000 have decent weights
program have been running from 2022-08-07 21:17:50.899928 to 2022-08-08 07:32:25.932846
0 from 100000 have decent weights
program have been running from 2022-08-08 07:32:25.933404 to 2022-08-08 17:47:00.642918
0 from 100000 have decent weights
program have been running from 2022-08-08 17:47:00.643463 to 2022-08-09 04:01:32.762627
0 from 100000 have decent weights
program have been running from 2022-08-09 04:01:32.763186 to 2022-08-09 14:16:06.819637
0 from 100000 have decent weights
program have been running from 2022-08-09 14:16:06.820189 to 2022-08-10 00:30:42.761535
0 from 100000 have decent weights
program have been running from 2022-08-10 00:30:42.762095 to 2022-08-10 10:45:17.660746
0 from 100000 have decent weights
program have been running from 2022-08-10 10:45:17.661300 to 2022-08-10 20:59:50.255048
mean error= 0.09044550046010671
weights = (1.4077753782659355, -1.4185729955518447, -1.8327421534704675, -0.13360093238699378, -1.4854825879614615, 1.7813972102294917, -

1.9094267350872176, -1.0566855177032028, -1.2933845832258601, -1.0206741471871466, 0.796531689271883, -1.0325552360411177, 0.029543537235902306, -1.5603222322236587, -0.9931331478550636, 0.9285264938340694, 0.8091817450886198)

1 from 100000 have decent weights
program have been running from 2022-08-02 18:22:08.329274 to 2022-08-03 04:36:53.218925
mean error= 0.08564710294952838
weights = (1.3483951793865563, -0.813400561867156, -1.6788426729833108, 0.2651054789315124, -1.3551102715147585, 0.7194253701242106, -1.5906531311766168, 0.5084662448858994, -1.4190866490864233, -1.222902561544279, -1.8884610584824717, 1.0255981121804094, -1.0260130756476453, 0.07319915199061411, -0.12341936414417276, 3.009164932508588, 0.7720590905733014)

1 from 100000 have decent weights
program have been running from 2022-08-10 20:59:50.255638 to 2022-08-11 07:14:22.387276
0 from 100000 have decent weights
program have been running from 2022-08-11 07:14:22.387850 to 2022-08-11 17:28:58.039742
0 from 100000 have decent weights
program have been running from 2022-08-11 17:28:58.040292 to 2022-08-12 03:43:35.219565
0 from 100000 have decent weights
program have been running from 2022-08-12 03:43:35.220121 to 2022-08-12 13:58:07.668649
0 from 100000 have decent weights
program have been running from 2022-08-12 13:58:07.669219 to 2022-08-13 00:12:47.960814
0 from 100000 have decent weights
program have been running from 2022-08-13 00:12:47.961377 to 2022-08-13 10:27:33.098423
0 from 100000 have decent weights
program have been running from 2022-08-03 04:36:53.219463 to 2022-08-03 14:51:40.839391
0 from 100000 have decent weights
program have been running from 2022-08-03 14:51:40.839924 to 2022-08-04 01:06:25.690057
mean error= 0.09701922213174308
weights = (1.3912082708181601, -0.6350245375044485, 0.40879732569848715, 1.3008809201792055, 0.045531689218421434, -1.022894986249459, -1.0457232551216955, 0.8909222229106719, -1.626696976814657, 1.810240677466295, -1.2932709426662368, -0.6472672402645623, -0.8047014716804903, 0.14539937857607743, 0.5661184413891851, 3.965741600231304, 0.15555871880123373)

mean error= 0.09979802074932052
weights = (0.37583570167017255, -0.029573831173703713, -1.6050388847119272, -0.8990345659705823, 0.6508110472455431, 0.1480100938844764, 0.567135442287205, 0.42082173600226414, 1.2133157573253808, -1.0085419944161287, -0.01175041263104859, 0.502081907802252, 1.4174591368271523, 0.8375343933599191, 0.6421775518652213, 0.6299119670928119, -3.1222750726236628)

mean error= 0.08572463795413943
weights = (0.4503549596033998, 0.5937454608080865, -1.2507533007127498, 1.894329519128608, -0.8303305607349993, -1.223232061873544, -1.5399165966769939, -0.21550602380597095, -0.9799948842353996, -1.5392484909262638, -0.8640543557632898, -0.408192815200203, -1.7274443074905887, -0.31845384318338565, 0.0641123815815492, 2.1643128137260836, 1.0525916773230417)

mean error= 0.06702154280271072
weights = (0.8226900336526946, -1.0590087110039814, 0.5637171398028542, -1.2030527224082932, 0.15670935593959534, 1.5594744961315552, -0.8613013590306391, -1.9277614018175901, -1.6375566929269514, -1.472387559017435, -0.7756299992547087, -0.4981767429184294, -1.7529865328004242, -1.522515970944256, -0.5968592500894712, 0.5752907195321937, 0.8085978642491414)

4 from 100000 have decent weights
program have been running from 2022-08-04 01:06:25.690599 to 2022-08-04 11:21:13.622626
0 from 100000 have decent weights
program have been running from 2022-08-04 11:21:13.623212 to 2022-08-04 21:35:47.380790
0 from 100000 have decent weights
program have been running from 2022-08-04 21:35:47.381357 to 2022-08-05 07:50:23.609027
0 from 100000 have decent weights
program have been running from 2022-08-05 07:50:23.609587 to 2022-08-05 18:04:58.908815
0 from 100000 have decent weights
program have been running from 2022-08-05 18:04:58.909378 to 2022-08-06 04:19:32.809030
0 from 100000 have decent weights
program have been running from 2022-08-06 04:19:32.809589 to 2022-08-06 14:34:05.561852

0 from 100000 have decent weights
program have been running from 2022-08-17 15:44:31.924309 to 2022-08-18 01:57:49.004813

0 from 100000 have decent weights
program have been running from 2022-08-18 01:57:49.005371 to 2022-08-18 12:11:04.571266
0 from 100000 have decent weights
program have been running from 2022-08-18 12:11:04.571853 to 2022-08-18 22:24:19.091762
0 from 100000 have decent weights
program have been running from 2022-08-18 22:24:19.092315 to 2022-08-19 08:37:34.853619
0 from 100000 have decent weights
program have been running from 2022-08-19 08:37:34.854162 to 2022-08-19 18:50:47.590380
0 from 100000 have decent weights
program have been running from 2022-08-19 18:50:47.590896 to 2022-08-20 05:04:03.765190
0 from 100000 have decent weights
program have been running from 2022-08-13 19:46:49.023030 to 2022-08-14 05:59:35.050391
0 from 100000 have decent weights
program have been running from 2022-08-14 05:59:35.050936 to 2022-08-14 16:12:42.282923
0 from 100000 have decent weights
program have been running from 2022-08-14 16:12:42.283478 to 2022-08-15 02:25:51.414788
0 from 100000 have decent weights
program have been running from 2022-08-15 02:25:51.415361 to 2022-08-15 12:38:59.231073
0 from 100000 have decent weights
program have been running from 2022-08-15 12:38:59.231610 to 2022-08-15 22:52:14.837709
0 from 100000 have decent weights
program have been running from 2022-08-15 22:52:14.838271 to 2022-08-16 09:05:18.996037
0 from 100000 have decent weights
program have been running from 2022-08-16 09:05:18.996610 to 2022-08-16 19:18:02.686509
mean error= 0.09320546268432872
weights = (-1.814072689116951, -0.12076497001982833, 1.5138873888498519, -0.725247157030271, 1.844737939388954, 0.5580695926310852, -1.6545579156021128, -0.5163125183950381, -1.2321490649446636, 0.7058608028161606, -1.2285808175212187, -0.28671494598778846, -0.6010486967749755, -0.0040859805343460565, -0.6977617360409649, 2.26469581290831, 0.9940281988981354)

1 from 100000 have decent weights

program have been running from 2022-08-16 19:18:02.687070 to 2022-08-17 05:31:15.773238
0 from 100000 have decent weights
program have been running from 2022-08-17 05:31:15.773788 to 2022-08-17 15:44:31.923741
0 from 100000 have decent weights
program have been running from 2022-08-17 15:44:31.924309 to 2022-08-18 01:57:49.004813
0 from 100000 have decent weights
program have been running from 2022-08-18 01:57:49.005371 to 2022-08-18 12:11:04.571266
0 from 100000 have decent weights
program have been running from 2022-08-18 12:11:04.571853 to 2022-08-18 22:24:19.091762
0 from 100000 have decent weights
program have been running from 2022-08-18 22:24:19.092315 to 2022-08-19 08:37:34.853619
0 from 100000 have decent weights
program have been running from 2022-08-19 08:37:34.854162 to 2022-08-19 18:50:47.590380
0 from 100000 have decent weights
program have been running from 2022-08-19 18:50:47.590896 to 2022-08-20 05:04:03.765190
mean error= 0.06974335771122506
weights = (-0.18422706051141624, 1.9111237326444082, -1.2930979954965491, -1.9211157692481309, 0.3285878714710222, 1.6570715610229088, -0.880355496409865, 1.8422675535213089, -1.2873425810290406, -0.12960871817791908, -1.1426494342591802, 0.40655144089176876, -0.794320749409493, -0.026427684940557583, 0.14936361183679203, 3.884122354010084, 1.4508732976337644)

1 from 100000 have decent weights
program have been running from 2022-08-20 05:04:03.765748 to 2022-08-20 15:17:18.383426
0 from 100000 have decent weights
program have been running from 2022-08-20 15:17:18.384002 to 2022-08-21 01:28:19.112730
0 from 100000 have decent weights
program have been running from 2022-08-21 01:28:19.113265 to 2022-08-21 11:39:50.045318
0 from 100000 have decent weights
program have been running from 2022-08-21 11:39:50.045884 to 2022-08-21 21:50:30.131617
0 from 100000 have decent weights

program have been running from 2022-08-13 19:46:49.023030 to 2022-08-14 05:59:35.050391
0 from 100000 have decent weights
program have been running from 2022-08-21 21:50:30.132179 to 2022-08-22 08:01:06.533495
0 from 100000 have decent weights
program have been running from 2022-08-22 08:01:06.534062 to 2022-08-22 18:11:45.796568
0 from 100000 have decent weights
program have been running from 2022-08-22 18:11:45.797134 to 2022-08-23 04:22:30.675378
0 from 100000 have decent weights
program have been running from 2022-08-23 04:22:30.675948 to 2022-08-23 14:34:29.956756
0 from 100000 have decent weights
program have been running from 2022-08-23 14:34:29.957323 to 2022-08-24 00:47:36.207478
0 from 100000 have decent weights
program have been running from 2022-08-24 00:47:36.208045 to 2022-08-24 10:58:21.565475
0 from 100000 have decent weights
program have been running from 2022-08-24 10:58:21.566045 to 2022-08-24 21:09:09.028149
0 from 100000 have decent weights
program have been running from 2022-08-24 21:09:09.028676 to 2022-08-25 07:19:56.067383
mean error= 0.09412611580470703
weights = (1.1225597406157877, -0.16751971896314943, -1.6918250430287207, -0.0792422368432737, -1.2101033045955991, 1.9441070990219513, -1.5169490948241213, -1.9091762151608371, -0.6249617335190059, -1.7432009945366946, -1.2648891233351596, -0.522364536020357, -1.9240247267473305, -0.4112103984443003, -0.21237091506871142, 1.844945864573159, 0.9970215125168638)

1 from 100000 have decent weights
program have been running from 2022-08-25 07:19:56.067952 to 2022-08-25 17:32:24.013457
0 from 100000 have decent weights
program have been running from 2022-08-25 17:32:24.014038 to 2022-08-26 03:45:30.627880
0 from 100000 have decent weights
program have been running from 2022-08-14 05:59:35.050936 to 2022-08-14 16:12:42.282923
0 from 100000 have decent weights

program have been running from 2022-08-26 03:45:30.628447 to 2022-08-26 13:58:44.338223
0 from 100000 have decent weights
program have been running from 2022-08-26 13:58:44.338777 to 2022-08-27 00:12:02.961176
0 from 100000 have decent weights
program have been running from 2022-08-27 00:12:02.961738 to 2022-08-27 10:25:19.636142
0 from 100000 have decent weights
program have been running from 2022-08-27 10:25:19.636714 to 2022-08-27 20:38:36.737273
0 from 100000 have decent weights
program have been running from 2022-08-27 20:38:36.737838 to 2022-08-28 06:51:50.363264
0 from 100000 have decent weights
program have been running from 2022-08-28 06:51:50.363838 to 2022-08-28 17:05:07.604962
0 from 100000 have decent weights
program have been running from 2022-08-28 17:05:07.605490 to 2022-08-29 03:18:22.569682
0 from 100000 have decent weights
program have been running from 2022-08-29 03:18:22.570244 to 2022-08-29 13:31:37.135453
mean error= 0.09258143540620603
weights = (0.6061032013185739, -1.1806128664883775, 1.7201176979845711, -1.6459324711425527, 1.035485462771554, 1.6939895793661393, -1.3557624272204944, -1.5142840031475249, -0.8205934051704973, -1.4865478754364445, -0.5836863070389473, -0.35191955390783525, -1.3507298855061234, -0.09086781874496985, -1.404128374191222, 0.4541390409694799, 0.9300803290400408)

1 from 100000 have decent weights
program have been running from 2022-08-29 13:31:37.136016 to 2022-08-29 23:44:50.564224
0 from 100000 have decent weights
program have been running from 2022-08-29 23:44:50.564795 to 2022-08-30 09:58:05.794270
0 from 100000 have decent weights
program have been running from 2022-08-14 16:12:42.283478 to 2022-08-15 02:25:51.414788
0 from 100000 have decent weights
program have been running from 2022-08-30 09:58:05.794830 to 2022-08-30 20:11:17.673398
0 from 100000 have decent weights

program have been running from 2022-08-30 20:11:17.673964 to 2022-08-31 06:24:32.920786
0 from 100000 have decent weights
program have been running from 2022-08-31 06:24:32.921352 to 2022-08-31 16:37:49.289780
0 from 100000 have decent weights
program have been running from 2022-08-31 16:37:49.290369 to 2022-09-01 02:51:05.560847
0 from 100000 have decent weights
program have been running from 2022-09-01 02:51:05.561416 to 2022-09-01 13:04:22.276598
0 from 100000 have decent weights
program have been running from 2022-09-01 13:04:22.277167 to 2022-09-01 23:17:37.661713
mean error= 0.09939772946947287
weights = (0.07946875025278421, -0.294992197589965, 1.498939837643535, -1.488767692856578, -1.73363495384758, -0.23866579840007818, 1.209054942469037, -0.3969550182878492, -1.1040315204675055, -0.9475484719937084, -0.35481013984734977, -1.1177970697153548, -1.5655664306287118, 0.900128604736878, 1.5153656034369964, 1.8656438251378686, 2.4901157448849593)

1 from 100000 have decent weights
program have been running from 2022-09-01 23:17:37.662281 to 2022-09-02 09:30:55.697350
0 from 100000 have decent weights
program have been running from 2022-09-02 09:30:55.697946 to 2022-09-02 19:44:04.907958
0 from 100000 have decent weights
program have been running from 2022-09-02 19:44:04.908512 to 2022-09-03 05:57:18.983525
0 from 100000 have decent weights
program have been running from 2022-08-15 02:25:51.415361 to 2022-08-15 12:38:59.231073
0 from 100000 have decent weights
program have been running from 2022-08-15 12:38:59.231610 to 2022-08-15 22:52:14.837709
0 from 100000 have decent weights
program have been running from 2022-08-15 22:52:14.838271 to 2022-08-16 09:05:18.996037
0 from 100000 have decent weights
program have been running from 2022-08-16 09:05:18.996610 to 2022-08-16 19:18:02.686509
mean error= 0.09320546268432872

weights = (-1.814072689116951, -0.12076497001982833, 1.5138873888498519, -0.725247157030271, 1.844737939388954, 0.5580695926310852, -1.6545579156021128, -0.5163125183950381, -1.2321490649446636, 0.7058608028161606, -1.2285808175212187, -0.28671494598778846, -0.6010486967749755, -0.0040859805343460565, -0.6977617360409649, 2.26469581290831, 0.9940281988981354)

1 from 100000 have decent weights
program have been running from 2022-08-16 19:18:02.687070 to 2022-08-17 05:31:15.773238
0 from 100000 have decent weights
program have been running from 2022-08-17 05:31:15.773788 to 2022-08-17 15:44:31.923741