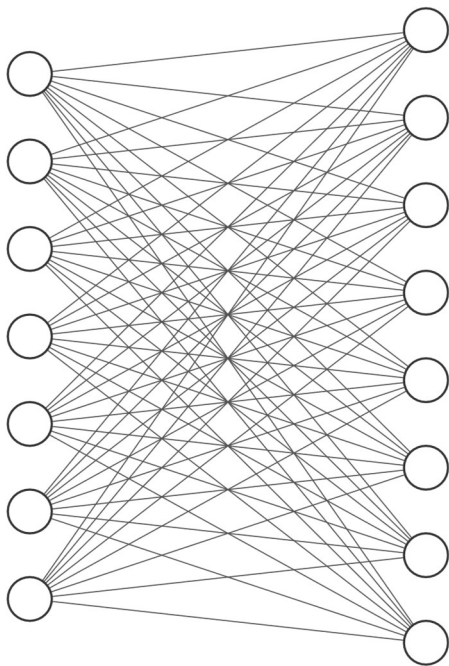


XARXES NEURONALS ARTIFICIALS



Joan Pau Condal Marco

Treball de Recerca

David Cambray Pla

2n Batxillerat J

Escola Pia Santa Anna Mataró, 2018

ÍNDEX

INTRODUCCIÓ	2
MOTIVACIONS.....	2
HIPÒTESIS.....	2
OBJECTIUS	3
METODOLOGIA.....	3
DIFERÈNCIES ENTRE XARXES NEURONALS I ALTRES PROGRAMES?.....	5
CONEIXEMENTS PREVIS A LES XARXES NEURONALS	8
MATRIUS.....	8
<i>Vector:</i>	8
<i>Matrïus:</i>	8
<i>Operacions bàsiques</i>	9
COLOR "RGB"	12
PORTES LÒGIQUES	14
<i>NOT</i>	14
<i>AND</i>	14
<i>OR</i>	15
FUNCIONAMENT DE LES XARXES NEURONALS ARTIFICIALS.....	16
APRENENTATGE SUPERVISAT	16
APRENENTATGE NO SUPERVISAT	16
ESTRUCTURA DE LES XARXES NEURONALS ARTIFICIALS.....	17
<i>Neurones</i>	17
<i>Les connexions</i>	17
EL PRINCIPI DE LES XARXES NEURONALS	18
EL PERCEPTRÓ.....	19
FEEDFORWARD.....	20
<i>Objectiu</i>	20
<i>Funcionament</i>	21
BACKPROPAGATION	22
<i>Aprenentatge supervisat</i>	22
USOS DEL PERCEPTRÓ.....	25
<i>Perceptró per prediccions</i>	25
<i>Perceptró per classificacions</i>	26
LIMITACIONS DEL PERCEPTRÓ	26
XARXES NEURONALS ARTIFICIALS (XNA)	28
ESTRUCTURA DE LES XNA	28
FUNCIONAMENT DE LES XNA.....	30
<i>Feedforward</i>	30
<i>Backpropagation</i>	37
ELEMENTS DE LA PROGRAMACIÓ	41
LES LLIBRERIES	41
<i>Les funcions</i>	41
PART PRÀCTICA	43
CONEIXEMENTS PREVIS.....	44
PRIMER INTENT AUTODIDACTA	45

PRIMERES IDEES.....	46
PRIMERS INTENTS DE PROGRAMACIÓ	46
LA SEGONA IDEA	47
DESENVOLUPAMENT DE LA SEGONA IDEA	47
PLANTEJAMENT DE LA PROGRAMACIÓ	48
FEEDFORWARD.....	48
BACKPROPAGATION	50
ELS PESOS	50
<i>FIREBASE</i>	51
DISSENY DE LA PÀGINA.....	53

Introducció

Motivacions

Ja fa uns mesos, navegant per YouTube, vaig veure un vídeo d'un divulgador científic en el que parlava del funcionament de YouTube i del seu algoritme (programa) de recomanació de vídeos. En aquest vídeo explicava que aquest algoritme era realment una "xarxa neuronal artificial" i et donava una petita idea de què eren.

En aquell moment jo no n'havia sentit mai a parlar, però em van cridar molt l'atenció. A partir d'aquí vaig començar a investigar per tenir una idea més clara de què eren, i vaig veure que era un tema molt interessant i profitós.

Així que a l'hora de triar el tema del treball de recerca, vaig decidir aventurar-me amb les xarxes neuronals artificials.

Hipòtesis

Un cop ja vaig tenir clar que aquest era el tema del meu treball de recerca, havia de crear una hipòtesi. Jo tenia molt clar que el meu objectiu final era aconseguir crear una xarxa neuronal artificial. El problema era que no sabia si amb tota la informació que hi havia per la web, als llibres i a altres fonts d'informació era suficient com per aconseguir el coneixement necessari per crear-ne una des de zero. Així que degut a això, la hipòtesi va ser: **"Avui en dia les intel·ligències artificials estan tant al nostre abast que un alumne de batxillerat pot crear i implementar una xarxa neuronal artificial des de zero pel seu compte"**.

Cal destacar que en la hipòtesi mateixa, dic "crear i implementar una xarxa neuronal artificial des de zero". I aquesta part és potser la més important de tota la hipòtesi, ja que en el món de la programació existeixen unes eines anomenades llibreries (explicació del que són a la part pràctica) que faciliten molt la feina. Jo tenia dubtes de si era possible crear-la des de, així que vaig formular la segona hipòtesi: **"És possible crear una xarxa neuronal artificial complexa sense l'ajuda d'una llibreria"**.

Objectius

En aquest treball de recerca, els objectius estan molt lligats a les hipòtesis. Si ens fixem en la primera hipòtesi, el primer objectiu és, sens dubte, el següent: “**Crear una xarxa neuronal**”, el principal objectiu i el més important. I si mirem la segona hipòtesi, el segon objectiu és: “**Crear una xarxa neuronal artificial sense l’ajuda de llibreries**”.

Metodologia

Ara que ja sabia que volia crear una xarxa neuronal artificial, només havia de pensar com ho faria. En aquest tema, tenia clar una cosa: al ser un tema tant recent i tant relacionat amb la tecnologia, a internet hi hauria molta informació.

El primer que vaig fer va ser visitar una pàgina web que em va recomanar un cop. La pàgina era de cursos *online*, anomenada *Coursera*. En ella, totes les universitats del món que volguessin poden fer cursos i pujar-los en forma de vídeo. Un punt a favor de la pàgina és que no has de pagar per fer els cursos, només per obtenir el títol.

Al cap d’uns dies d’anar-me informant de què eren les xarxes neuronals artificials, YouTube em va començar a recomanar vídeos sobre el tema, i vaig acabar en un canal fantàstic anomenat *The Coding Train*, creat per Daniel Shiffman, en el que hi havia un munt d’exemples i explicacions sobre xarxes neuronals artificials.

En aquest punt vaig tenir molt clar que per internet podia trobar tot el que volia. Però al cap d’un temps, vaig començar a sentir títols de llibres que tenien molt bona pinta i que també podien ser molt útils per al meu treball.

Així que finalment, la manera en que aconseguiria tota la informació seria per internet: *Coursera* i *YouTube* i alguns llibres. Això més per la part pràctica. Per la part teòrica,

també navegaria per internet. Però en aquest cas, YouTube i *Coursera*, no em serien tant útils. Així que vaig decidir informar-me a partir de *Wikipedia*, articles i llibres.

En el tema de la programació, ja tenia molt clar com ho faria: utilitzaria un llenguatge anomenat JavaScript, que et permet fer programes en pàgines web.

Diferències entre xarxes neuronals i altres programes?

Abans de saber què són exactament les xarxes neuronals artificials, és necessari saber perquè estan triomfant tant. En el moment en el que es van inventar, ja existien els programes ordinaris, però les xarxes neuronals artificials es van inventar igualment i van triomfar. Com és això?

Per entendre-ho millor, compararem dos programes: el primer serà un programa ordinari, i el segon serà una xarxa neuronal artificial:

- **Programa ordinari:** Sistema operatiu de telèfon mòbil
- **Xarxa neuronal artificial:** Programa de reconeixement d'imatges. En aquest programa, si li dones una fotografia amb un número, et diu de quin número es tracta.

Cal tenir en compte que on recau la principal diferència és en la programació. Per tant ens centrarem en aquest aspecte.

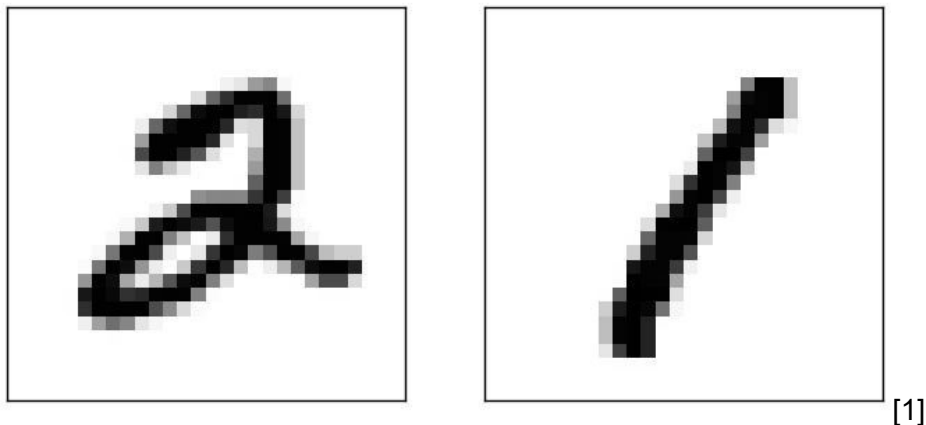
Per entendre com es programen els algoritmes que no són xarxes neuronals artificials, ens fixarem en l'exemple del sistema operatiu. Si tu fossis un programador que hagués de fer un sistema operatiu per telèfons mòbil, hauries de pensar totes les coses que pot fer un usuari sobre el mòbil, com podria ser:

- Mantenir premut el botó d'engegada
- Prémer el botó de baixar el volum
- Desplaçar el dit per la pantalla horitzontalment
- Fer clic a la icona d'una aplicació

I per cada una de les possibles accions de l'usuari hauries de indicar-li una possible resposta. Per tant acabaries amb un programa que, al rebre una acció de l'usuari, comprovaria si té una resposta escrita; en el cas que la tingués, l'executaria; i en el cas que no, no faria res o s'aturaria. És molt important tenir en compte que si tu, com a

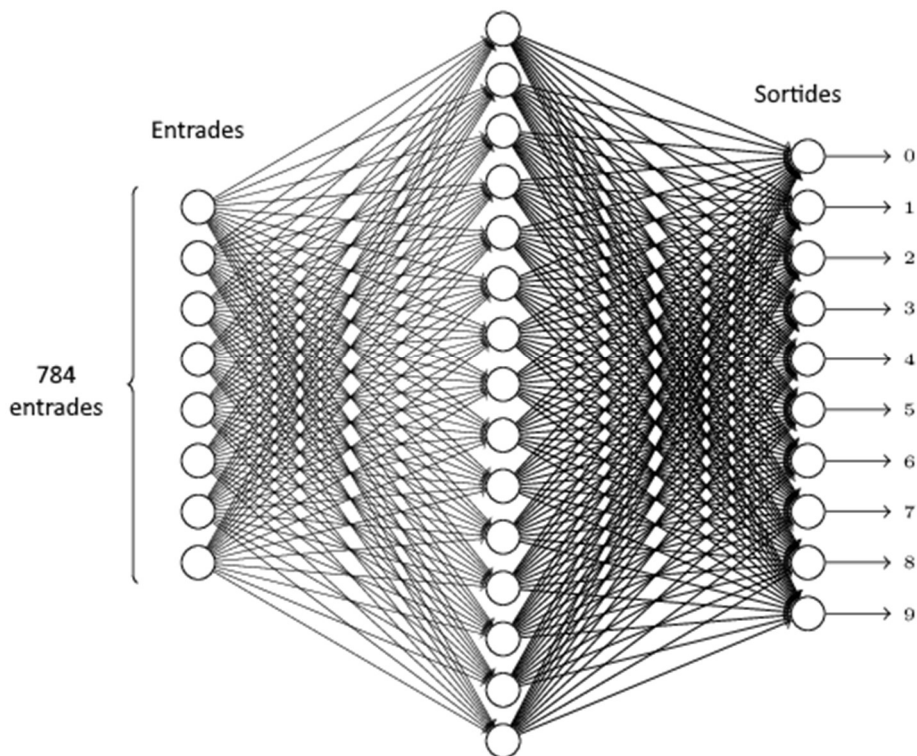
persona que sap de programació, miressis el codi, sabries exactament que fa aquest programa.

En canvi en les xarxes neuronals tu no programes una sèrie de normes ni unes funcions que el teu programa ha d'estar fent contínuament. Simplement programes una màquina que té un nombre concret d'entrades i un nombre concret de sortides. En el cas de la nostra màquina està preparada per agafar fotografies quadrades de 28x28 píxels i dir-te quin és el nombre que hi ha a la imatge.



Si féssim un programa ordinari que mirés els 784 píxels i et digués quin nombre és, hauríem de fer el mateix programa per cada número, i com que dues persones no dibuixen els números de la mateixa manera, hauríem de fer el programa per cada manera de dibuixar cada número. Acabaríem amb milions de línies de codi, el programa tardaria segles en funcionar, i no sempre encertaria el resultat.

En canvi podríem crear una xarxa neuronal, amb 784 neurones d'entrada, cada una corresponent a un píxel de la imatge i amb 10 neurones de sortida, cada una corresponent als números del 0-9.



[2]

En el nostre codi simplement crearem la màquina de la següent manera: indicarem el nombre de neurones a l'entrada, el nombre de neurones a la sortida, com interactuen les neurones entre elles i li afegirem una manera d'entrenar-la. En aquest punt ja tindriem la nostra xarxa neuronal artificial. I si ens miréssim el codi, no veuríem com funciona la xarxa neuronal, simplement veuríem com és. Al no estar regit per regles, si li donéssim una imatge que no és un nombre, simplement ens donaria un resultat, però no passaria com en el sistema operatiu que s'aturaria o no faria res.

És per això que les xarxes neuronals estan triomfant tant avui en dia. Poden fer tasques molt més complicades que qualsevol programa en molt menys temps.

Coneixements previs a les xarxes neuronals

Matrius

A l'hora de treballar amb les xarxes neuronals artificials, haurem de fer operacions amb molts números; i per això, per fer les coses més ordenades i més fàcils, treballarem amb matrius.

Les matrius són una eina matemàtica que serveix per emmagatzemar números i fer operacions. Els números es posen entre parèntesis o claudàtors, ordenats en files i columnes. En podem distingir dos tipus diferents:

Vector:

Els vectors són un tipus de matrius que consten només d'una columna. Depenent del nombre de files que tinguin, seran vectors de diferents dimensions, és a dir, un vector de tres files serà un vector tridimensional. És important destacar que no hi ha límit de dimensions. Per representar els vectors sempre ho farem amb lletres en minúscula.

$$a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Sempre que vulguem accedir a un dels nombres del vector en concret, ho podrem fer indicant la lletra que l'hi hem assignat al vector (en el nostre cas "a") i un subíndex: a_i , on el subíndex és un nombre real, més petit que el nombre total de files, referint-se a la fila. En el cas de la matriu anterior, podem veure que $a_2 = 2$.

Matrius:

Les matrius consten normalment de més d'una columna, i es denominen normalment "matriu $m \times n$ ", a on m representa el número de files i n representa el número de columnes.

$$A = \begin{bmatrix} -3 & 8 \\ 15 & 0.23 \end{bmatrix}$$

En el cas de les matrius, per accedir a un número en concret ho farem amb dos subíndex: i i j . En el cas de la matriu anterior seria a_{ij} , i referint-se a files i j a les columnes; per tant, en la matriu anterior, $a_{12} = 8$.

En el món de les matemàtiques, tant amb les files com amb les columnes, comencem a comptar des de 1; però a l'hora de programar, per fer les coses més senzilles, es comença a comptar des de zero.

Operacions bàsiques

Suma

En les matrius existeixen dos tipus de suma, la suma escalar i la suma de matrius.

En la suma escalar s'agafa una matriu (o vector) i es suma per un nombre. La matriu (o vector) resultant té el mateix nombre de files i de columnes que l'anterior, i a cada número de la matriu se l'hi ha sumat el nombre.

$$\begin{bmatrix} -6,3 & 4 \\ 21,5 & -12 \end{bmatrix} + 3 = \begin{bmatrix} -6,3 + 3 & 4 + 3 \\ 21,5 + 3 & -12 + 3 \end{bmatrix} = \begin{bmatrix} -3,3 & 7 \\ 24,5 & -9 \end{bmatrix}$$

En la suma de matrius (o de vectors), primer de tot hem de tenir en compte que només es pot fer amb matrius amb el mateix nombre de files i de columnes. En cas contrari, no podríem fer la operació. Per fer la suma de matrius, cada número es suma amb el número de l'altra matriu que ocupa la mateixa posició; i el resultat és una matriu de la mateixa mida que les altres dues.

$$\begin{bmatrix} 3,3 & -0,5 \\ -6 & 8,2 \end{bmatrix} + \begin{bmatrix} -4,6 & 13 \\ 2,7 & -0,8 \end{bmatrix} = \begin{bmatrix} 3,3 - 4,6 & -0,5 + 13 \\ -6 + 2,7 & 8,2 - 0,8 \end{bmatrix} = \begin{bmatrix} -1,3 & 12,5 \\ -3,3 & 7,4 \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e \\ f \\ g \end{bmatrix} = \nexists *$$

Multiplicació

De la mateixa manera que en la suma, en la multiplicació també tenim dos tipus: multiplicació escalar i multiplicació de matrius.

La multiplicació escalar funciona de la mateixa manera que la suma escalar: s'agafa una matriu qualsevol i es multiplica per un nombre qualsevol. El resultat final és una matriu de les mateixes dimensions que la matriu inicial i cada un dels seus números multiplicat per l'altre número.

$$\begin{bmatrix} 5 & -8 \\ 9 & 12 \\ -23 & 4 \end{bmatrix} \cdot 2 = \begin{bmatrix} 5 \cdot 2 & -8 \cdot 2 \\ 9 \cdot 2 & 12 \cdot 2 \\ -23 \cdot 2 & 4 \cdot 2 \end{bmatrix} = \begin{bmatrix} 10 & -16 \\ 18 & 24 \\ -46 & 8 \end{bmatrix}$$

Per multiplicar dues matrius, cal que la primera matriu tingui el mateix nombre de columnes que files de la segona matriu. Si no fos així, no podríem multiplicar les dues matrius.

Per començar, mirarem a la multiplicació d'una matriu horitzontal i un vector. Seguint la norma anterior, hem de tenir en compte que tant una matriu com l'altra han de tenir el mateix nombre de números dins, per tant multiplicarem una matriu $1 \times m$ i un vector $m \times 1$. La matriu final tindrà el mateix nombre de files que la primera matriu i el mateix nombre de columnes que la segona matriu, per tant quedarà una matriu resultant de 1×1 , un sol nombre.

Per calcular el resultat, es multiplica el primer nombre de la primera matriu pel primer nombre de la segona matriu, i es suma a la multiplicació del segon nombre de la primera matriu pel segon nombre de la segona matriu; i així fins a l'últim nombre.

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = 1 \cdot 3 + 2 \cdot 2 + 3 \cdot 1 = 10$$

A l'hora de multiplicar matrius bidimensionals, es pot simplificar a varies multiplicacions de files per columnes. Si tenim dues matrius 2×2 , la final serà de 2×2 . El primer nombre de la matriu final, a_{11} , serà el resultat de multiplicar la primera fila de la primera matriu amb la primera columna de la segona matriu. Per saber de quina fila i columna ve un nombre d'una matriu anteriorment multiplicada, cal mirar l'índex del nombre, per exemple: el nombre a_{23} , el primer nombre del subíndex fa relació al nombre de fila de la primera matriu, i el segon subíndex, al nombre de columna de la segona matriu que s'han multiplicat per aconseguir el nombre.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

A més dels dos tipus de multiplicació de matrius que hem vist abans, en tenim un tercer anomenat *Hadamard product*. Aquesta operació funciona de la mateixa manera que la suma de matrius, amb la diferència que és una multiplicació.

Per realitzar aquesta operació, necessitem dues matrius que siguin de la mateixa mida, amb el mateix nombre de files i de columnes. La matriu resultant tindrà la mateixa forma

que les dues anteriors. Per fer aquesta operació, cada número es multiplica amb el número de l'altra matriu que ocupa la mateixa posició:

$$\begin{bmatrix} 8 & 5 \\ -1 & 3 \end{bmatrix} \times \begin{bmatrix} 0.5 & 2 \\ 5 & 4 \end{bmatrix} = \begin{bmatrix} 8 \cdot 0.5 & 5 \cdot 2 \\ -1 \cdot 5 & 3 \cdot 4 \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ -5 & 12 \end{bmatrix}$$

Transposició

La transposició de matrius és una operació en la que només necessitem una matriu i la transformem. La matriu resultant té el mateix nombre de columnes que de files tenia la inicial i viceversa. En quant als valors de l'interior, el que hem de fer és invertir els subíndex. Per tant, el valor que hi havia a l'espai a_{ij} , ara estarà a l'espai a_{ji} . Per indicar que una matriu s'ha de transposar, es fa afegint la lletra "T" a la matriu.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \end{bmatrix}$$

Color “RGB”











Si volem utilitzar els colors en qualsevol tipus de programa, ja sigui per crear-los o perquè el programa els reconegui, els hi haurem d'assignar valors. La manera més comuna de fer-ho és amb el codi RGB.

Per crear colors en RGB, hem de tenir uns colors primaris. En el cas dels pigments utilitzats per la pintura, els colors primaris són el groc, el cian i el magenta. Els anomenem colors primaris ja que amb la combinació d'aquests tres colors podem arribar a qualsevol altre color, excepte el blanc.

En el cas dels ordinadors, els colors primaris no seran els mateixos, sinó que passaran a ser el vermell, el blau i el verd. La raó està en el funcionament de les pantalles. Cada píxel de la pantalla es pot il·luminar de tres colors, els tres anomenats abans, i amb això cada píxel agafa el color que necessita per ensenyar la imatge de la pantalla. Una avantatge d'aquests colors primaris és que poden crear tant el blanc com el negre, a diferència dels colors primaris dels pigments. Cada un dels tres colors primaris en cada píxel pot prendre diferents valors de quantitat. Si els tres colors del píxel estan a màxima quantitat, aquell píxel emetrà llum blanca, en canvi, si els tres colors del píxel estan a mínima quantitat, el píxel no emetrà llum, per tant, ho veurem negre.

El llenguatge de colors RGB, el que fa és recollir, en valors de 0 a 255, la quantitat dels colors dels píxels d'una zona concreta, ja sigui la pantalla sencera o una zona determinada. També es pot utilitzar per indicar-li a un programa que pinti una zona d'un cert color. Per tant, si el que es vol es pintar una zona d'un programa de color vermell, s'haurà de definir la zona perquè el programa sàpiga quins píxels ha de pintar, i s'haurà de dir, en RGB, que tots aquells píxels anteriorment escollits han de tenir una quantitat equivalent a 255 (o màxima) en el color vermell i 0 (o mínima) en els dos altres colors.

El terme RGB s'obté al agafar la primera lletra de cada un dels colors en anglès: *red*, *green*, *blue*. Tenint en compte que de cada un dels colors es poden prendre valors de fins a 255, podem veure que la quantitat de colors que es pot crear amb aquest codi és fins a 16.581.375 de colors diferents, tot i que variant un sol dígit en el codi de colors el color que es veu no canvia suficientment com per ser notable.

												
R	255	255	255	125	0	0	0	0	0	125	255	255
G	0	125	255	255	255	255	255	125	0	0	0	0
B	0	0	0	0	0	125	255	255	255	255	255	125

[3]

Portes lògiques

Les portes lògiques són les principals operacions de l'àlgebra booleana.

L'àlgebra booleana és una branca de l'àlgebra que treballa amb dos valors: cert o fals, normalment indicats amb 1 i 0 respectivament. Va ser introduïda per George Boole i va ser fonamental per la creació del codi binari, que més tard va permetre els llenguatges de programació.

Les portes lògiques, són petits processadors, ja siguin programats o circuits elèctrics, que reben senyals i tornen una resposta. El nombre de senyals pot variar des de una o dues, depenent de la porta lògica, a infinites. Cada senyal pot tenir dos valors: 1 o 0.

Les portes lògiques més conegudes són la *NOT*, la *AND*, la *OR*, la *XAND* i la *XOR*, però només ens centrarem en les tres primeres.

Cada porta lògica torna una senyal diferent depenent de les entrades, i aquí veurem les més típiques:

NOT

La porta lògica *NOT* funciona només amb una senyal d'entrada, i simplement torna el contrari del que entra. Per tant, si la senyal d'entrada és un 1, torna un zero i viceversa.

INPUT		OUTPUT
A		NOT A
0		1
1		0

[4]

AND

La porta lògica *AND*, requereix com a mínim dues senyals d'entrada, i en pot tenir tantes com sigui necessari. Aquesta porta lògica només tornarà 1 si totes les senyals d'entrada són 1. En cas contrari tornarà 0.

INPUT			OUTPUT
A	B		A AND B
0	0		0
0	1		0
1	0		0
1	1		1

[5]

OR

La porta lògica *OR*, també necessita dues senyals d'entrada, i en pot tenir tantes com calgui. Aquesta porta lògica és el contrari de la *AND*, ja que tornarà 1 sempre que només una de les senyals d'entrada sigui 1, i només tornarà 0 si totes les senyals d'entrada són 0.

INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

[6]

Funcionament de les xarxes neuronals artificials

La característica principal de les xarxes neuronals artificials, el que les fa destacar, és el fet de que poden aprendre dels seus errors. Quan diem que una xarxa neuronal pot aprendre ens referim a que canvia la manera en que interpreta la informació per donar cada cop respostes més precises. Tot això s'aconsegueix gràcies a anys de recerca de matemàtiques en aquest tema i, avui en dia, podem distingir dos grans tipus d'aprenentatges de les xarxes neuronals referint-nos a la manera en que s'entrenen: l'aprenentatge supervisat i l'aprenentatge no supervisat.

Aprenentatge supervisat

L'aprenentatge supervisat és utilitzat quan tens una gran base de dades amb respostes i vols que la teva xarxa busqui patrons entre les dades per poder predir resultats amb dades semblants de les quals no saps la resposta.

Per entrenar aquestes xarxes neuronals, primer els hi dones les dades, i aleshores, si la resposta que et dona és correcta, passes al següent grup de dades; i si la resposta que et torna és incorrecte, li dius quina era la resposta desitjada i ajustarà el seu funcionament intern per millorar el seu sistema de prediccions.

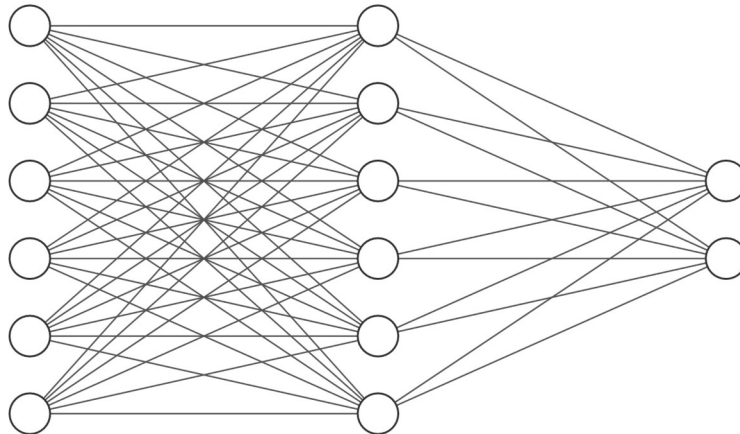
Aquest procés de comprovar la resposta i dir-li a la xarxa si s'ha equivocat o ha predit bé la resposta, es fa normalment amb algorismes externs a la xarxa. D'aquesta manera, és possible entrenar les xarxes neuronals amb bases de dades més grans (normalment sobrepasant els centenars de milers) i en un període més curt de temps.

Aprenentatge no supervisat

L'aprenentatge no supervisat s'utilitza amb dades de les quals tu no tens resposta ni relació. Aquestes xarxes utilitzen algorismes per maximitzar el seu funcionament i poder trobar patrons en les dades que se li donen. Aquestes xarxes neuronals es fan per intentar imitar el pensament lògic del ésser humà, buscant estructures amagades, patrons o característiques comunes.

Estructura de les xarxes neuronals artificials

Les xarxes neuronals artificials, com bé diu el seu nom, són xarxes de neurones. Al dir xarxes de neurones ens referim a moltes neurones connectades entre si per complir una funció. Normalment, aquestes neurones estan organitzades per columnes, i cada columna està connectada amb la següent. Un exemple de xarxa neuronal seria el següent:



[7]

Aquesta imatge és la representació d'una xarxa neuronal. En aquests diagrames s'hi poden veure representats les neurones (cercles) i les connexions (línies).

Neurones

En una xarxa neuronal artificial, les neurones són les unitats de processament, que reben unes dades, les processen i les envien. Totes aquestes neurones, processen la informació gràcies a fórmules matemàtiques especials per les xarxes neuronals que més tard veurem.

Les connexions

Les connexions de les xarxes neuronals, que transmeten la informació d'unes neurones a unes altres, són normalment anomenades *weights* o pesos. Aquestes connexions són la part de la xarxa neuronal que permet la seva modificació (entrenament) per tal de millorar resultats.

El principi de les xarxes neuronals

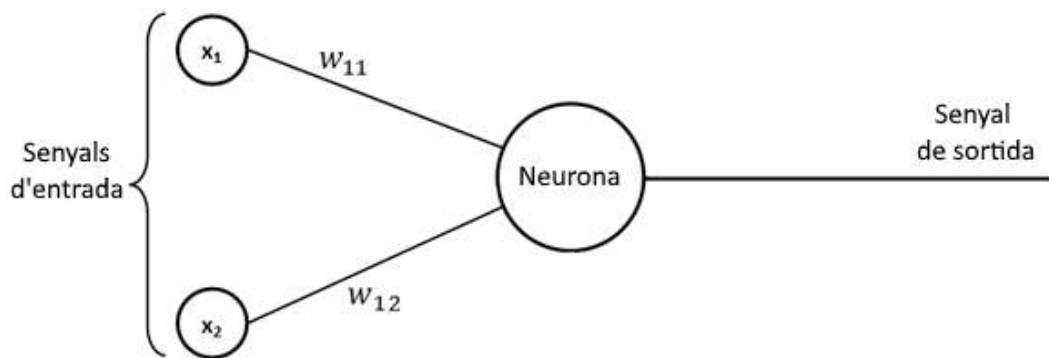
Les xarxes neuronals funcionen gràcies a observar la natura i imitar-la. Específicament, les xarxes neuronals artificials imiten el cervell animal, que és alhora una xarxa neuronal natural. Per fer funcionar les xarxes neuronals, es van crear aquestes dues parts: les neurones i les connexions. Com que les neurones són unitats de processament, sempre segueixen les mateixes fórmules, per tant, per unes dades d'entrada, sempre donaria el mateix resultat.

Aquí és on destaca la importància de les connexions. Al 1949, el neuròleg Donald Hebb va postular la seva teoria, dient que quan dues neurones tendien a excitar-se entre elles periòdicament, les seves connexions s'enfortien i viceversa.

Al desenvolupar les xarxes neuronals, van agafar aquesta teoria i la van aplicar. Per tant, en una xarxa neuronal artificial, les connexions entre cada neurona tenen diferents valors. Són aquests valors els que hem de ajustar per fer funcionar correctament la nostra xarxa neuronal artificial.

El perceptró

Per entendre com funcionen les xarxes neuronals artificials, primer de tot ens fixarem en l'exemple més senzill: el perceptró. El perceptró és la xarxa neuronal artificial més simple, el qual consta només d'una neurona. Aquesta neurona rebra unes senyals d'entrada, farà un càlcul i tornarà un resultat.



[8]

El nostre perceptró tindrà dues senyals d'entrada, anomenades x_1 i x_2 , i cada una d'aquestes senyals estarà connectada a la neurona mitjançant una connexió. Aquesta connexió pot ser més forta o més dèbil; i això ho indicarem amb el pes (w_{11} , w_{12}), un número per cada connexió. Per facilitar les operacions més endavant, tant els valors de la senyal d'entrada com els valors dels pesos, els emmagatzemarem en matrius. Per fer-ho, seguirem les següents normes:

Les senyals d'entrada s'emmagatzemaran en un vector

Els pesos tindran dos subíndex, el primer farà referència al nombre de neurona al que es dirigeix, i el segon farà referència al nombre de neurona del que ve.

Els subíndex dels pesos, fan referència a la posició en que han d'estar dins la matriu, el primer nombre referint-se a la fila i el segon nombre referint-se a la columna. D'aquesta

manera el pes w_{11} estarà a la primera fila i primera columna de la matriu, i el pes w_{12} estarà a la primera fila i la segona columna.

Entenent això, ja podem crear les matrius, que haurien de quedar de la següent manera:

Senyals d'entrada: $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$


Pesos: $\begin{bmatrix} w_{11} & w_{12} \end{bmatrix}$

El funcionament del perceptró es basa en dues fases molt diferenciades entre si: *feedforward* i *backpropagation*.

Feedforward

Objectiu

Ara que ja tenim l'estructura del perceptró, és hora de donar-li unes senyals d'entrada i fer que el nostre perceptró ens doni una resposta. Per poder veure si funciona o no, intentarem que el nostre perceptró resolgui portes lògiques, en concret la porta lògica AND.

		A	
		0	1
B	0	0	0
	1	0	1

[9]

Aquesta porta lògica només tornarà com a resposta un 1 si les dues senyals d'entrada són 1; en cas contrari tornarà 0.

Funcionament

Al iniciar el nostre perceptró, els pesos de les connexions es donaran de manera aleatòria, ja que no tenim manera de saber quins han de ser.

Fet això, li donem dues senyals d'entrada, per exemple: 0 i 1. Per calcular el resultat, el perceptró farà dues operacions: calcular un resultat inicial i convertir-lo mitjançant una **funció d'activació**.

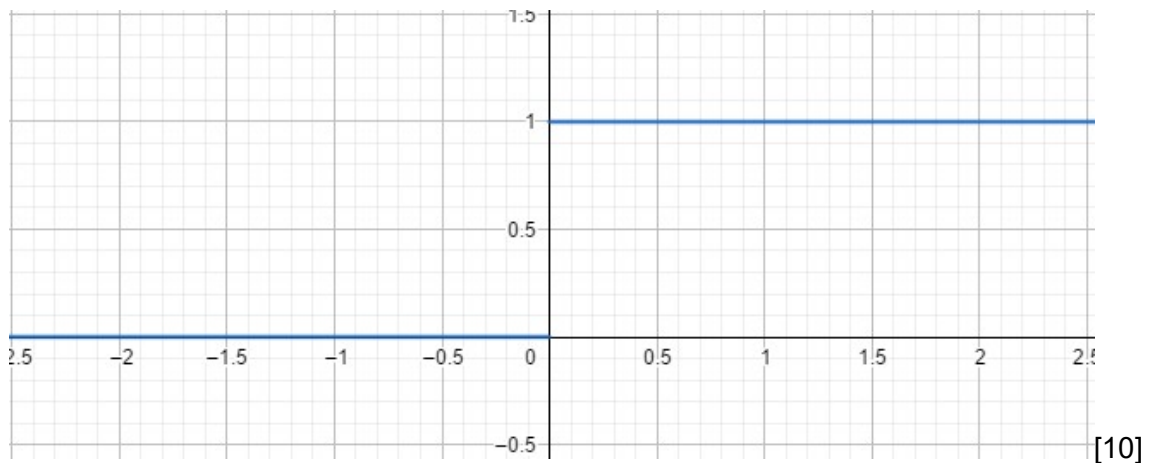
Per calcular el resultat inicial (al qual li assignarem la lletra z), el que el perceptró fa és agafar les senyals d'entrada i multiplicar-les pel pes corresponent. Després es farà la suma de tots els resultats. Com que estem emmagatzemant els valors en matrius, una multiplicació de matrius ens donarà el mateix resultat. Per tant podem dir que el resultat inicial és igual a la matriu dels pesos multiplicada per la matriu de les senyals d'entrada.

$$\mathbf{Z} = \mathbf{W} \cdot \mathbf{X}$$

$$\begin{bmatrix} 3 & 15 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 3 \cdot 0 + 15 \cdot 1 = 15$$

Al iniciar, el nostre perceptró ha assignat aleatòriament el valor 3 al pes 1 (w_{11}) i el valor 15 al pes 2 (w_{12}); per tant, el perceptró ens ha tornat -15 com a resultat inicial. Ara hem de normalitzar el resultat, és a dir, transformar el resultat a un rang de números que ens sigui útil. Com que estem treballant amb portes lògiques, el nostre rang de nombres adequat seria entre 0 i 1.

Per transformar aquest resultat al rang que necessitem ho farem amb una funció d'activació. La funció d'activació que nosaltres utilitzarem serà una anomenada *unit step*. Si nosaltres li introduïm qualsevol nombre a aquesta funció sempre ens en retornarà un entre 0 i 1. Aquesta funció, és una funció composta, de fórmula: $f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$.



Un cop el perceptró ja ens ha tornat un nombre entre 0 i 1, significa que ja ha fet els seus càlculs i ens ha donat una resposta per les dues senyals d'entrada que l'hi hem donat. Si recordem, les senyals d'entrada eren 0 i 1, i el resultat, ja passat per la funció d'activació, és 1. Com que nosaltres volíem que el perceptró resolgués la porta lògica AND, significa que aquest resultat està malament, ja que hauria de ser 0. Així que ara ens tocarà entrenar el perceptró perquè retorni el resultat que volem.

Backpropagation

El *backpropagation algorithm* o l'algoritme de propagació cap enrere, és una part del codi de la xarxa neuronal o del perceptró amb la tasca de modificar tots i cada un dels pesos de cada connexió per aconseguir la resposta desitjada. L'algoritme d'aprenentatge que utilitzarem pel perceptró s'anomena *supervised learning*.

Aprenentatge supervisat

L'algoritme d'aprenentatge supervisat, és un tipus d'algoritme que s'utilitza per entrenar el perceptró o la xarxa neuronal.

Abans de començar a canviar pesos, hem de veure com de malament ho ha fet el nostre perceptró, calculant l'error. Per calcular l'error direm que:

$$\text{error} = \text{resposta desitjada} - \text{resposta del perceptró}$$

Seguint aquesta fórmula, el nostre error és $e = 0 - 1 = -1$. En el cas que l'error fos 0, significaria que el perceptró hauria donat la resposta desitjada, per tant no caldria entrenar-lo; però com que en aquest cas no és 0, sí que cal entrenar-lo. La fórmula per variar el pes (o *weight*) és:

$$w_i = w_i + \Delta w_i$$

a on Δw_i és l'increment del pes, que pot ser tant positiu com negatiu. Per tant ara queda calcular aquest increment. La fórmula de l'increment és:

$$\Delta w_i = \text{error} * \text{input} * \text{learning rate}$$

a on *error* és l'error del perceptró i *input* és la senyal d'entrada d'aquell específic pes. També hi ha un terme nou anomenat *learning rate*, que és un nombre entre 0 i 1, i indica en quina proporció ha de variar el pes. Aquest terme serveix per poder canviar la velocitat en la que arribes al pes adequat, tenint en compte que quan més ràpid, més imprecís i viceversa.

És important saber que per l'aprenentatge supervisat cal tenir tres grups grans de dades:

- Dades d'entrenament: Són una sèrie de senyals d'entrada amb les seves respectives respostes. Aquestes dades són exclusives per entrenar el perceptró i corregir-lo si fes falta.
- Dades de prova: Són una sèrie de senyals d'entrada de les quals també sabem la resposta, però no s'usen per entrenar el perceptró, sinó per veure si l'entrenament està sent efectiu o no.

- Dades desconegudes: Són una sèrie de dades de les quals no saps la resposta que s'introduiran al perceptró un cop ja estigui entrenat per saber la resposta

És molt important tenir aquests tres grups de dades, sobretot el primer i el segon, ja que si només tinguéssim dades per entrenar el perceptró i no en tinguéssim per comprovar si l'entrenament va bé, podria passar que sobre entrenéssim el perceptró i només funcionés amb les dades d'entrenament i que no tornés una resposta correcta amb les dades desconegudes.

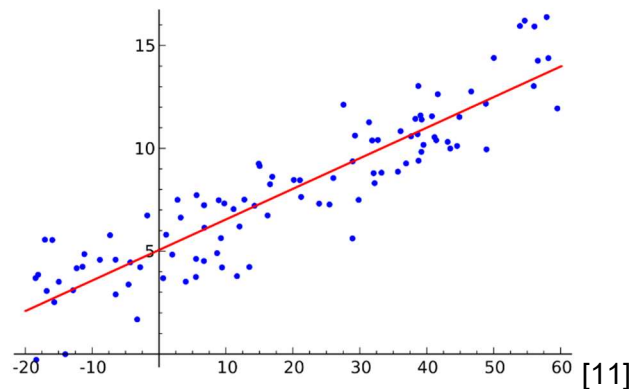
Usos del perceptró

Ara que ja hem vist com funciona un perceptró, des de que introduïm dades fins que l'entrenem i el millorem, cal veure les seves principals aplicacions, les quals són prediccions i classificacions.

Però abans, hem d'introduir un nou element: el *bias*. Si mirem a la fórmula $z = x * w$, podem arribar a la conclusió que si $x = 0$, la resposta sempre serà 0, no importa el valor del pes. Però hi haurà casos en els que amb $x = 0$ voldrem una resposta que no sigui 0, per això serveix el *bias*. Si implementem el *bias*, la fórmula queda $z = x * w + b$

Perceptró per prediccions

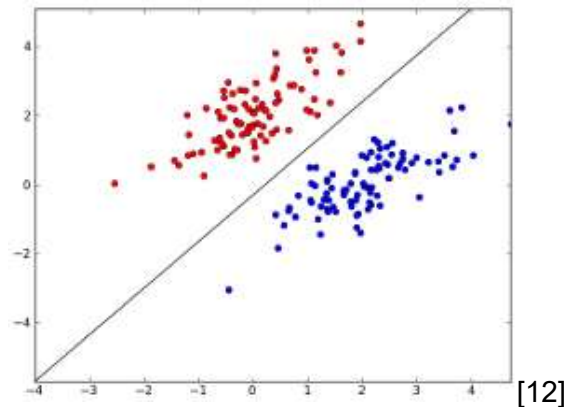
Imaginem que tenim un plànol cartesià a on a l'eix de les x tenim els metres quadrats d'una casa i a l'eix de les y tenim el seu preu. Si recol·lectem gran quantitat de dades, podrem veure que cada un dels punts sembla que estigui sobre una recta. Si entrenéssim un perceptró amb totes aquelles dades, ell mateix trobaria la fórmula de la recta, i per tant, podria fer prediccions.



El perceptró pot fer això gràcies a la similitud entre la seva fórmula $z = x \cdot w + b$ i la fórmula de la recta $y = mx + n$. Al entrenar el perceptró amb aquelles dades, el que fem és buscar el pes i el *bias*, que corresponen al pendent i al punt d'intersecció de la recta amb l'eix de les ordenades respectivament. Per tant, quan ja està ben entrenat, podem introduir-li metres quadrats d'una casa i que ens torni una predicció del seu preu.

Perceptró per classificacions

Una altra aplicació comú pel perceptró és crear una divisió per poder classificar. De la mateixa manera que en les prediccions, el perceptró crea una funció de recta, i quan li introduïm noves dades ens pot dir si pertany al grup de sobre la recta o al grup de sota.



Aquest tipus de perceptró és el que s'usa, per exemple, en l'exemple anterior de les portes lògiques, ja que si mirem a la taula de veritat, veurem que podem traçar una línia per dividir els resultats que són zero i els resultats que són 1.

		A	
		0	1
B	0	0	0
	1	0	1

[13]


		A	
		0	1
B	0	0	1
	1	1	1

[14]

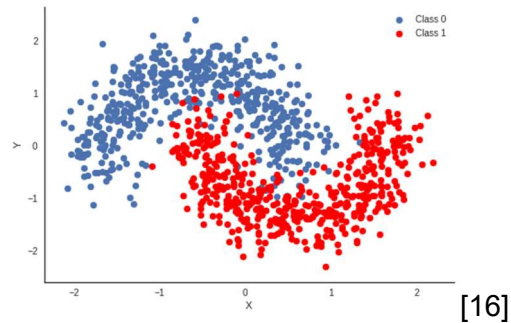
Limitacions del perceptró

Una de les grans limitacions del perceptró és que, com hem vist en els exemples anteriors, només funciona amb dades que són "linearment separables". Si tenim una taula de veritats d'una porta lògica en la qual necessitem dues línies per separar el

resultat o un model de prediccions que les dades no es poden predir amb una recta, un simple perceptró no podria fer cap de les dues tasques per molt que l'entrenéssim.

		A	
		0	1
B	0	0	1
	1	1	0

[15]



És per això que es va inventar el *multilayer perceptron* o el perceptró de múltiples capes; també anomenat xarxa neuronal.

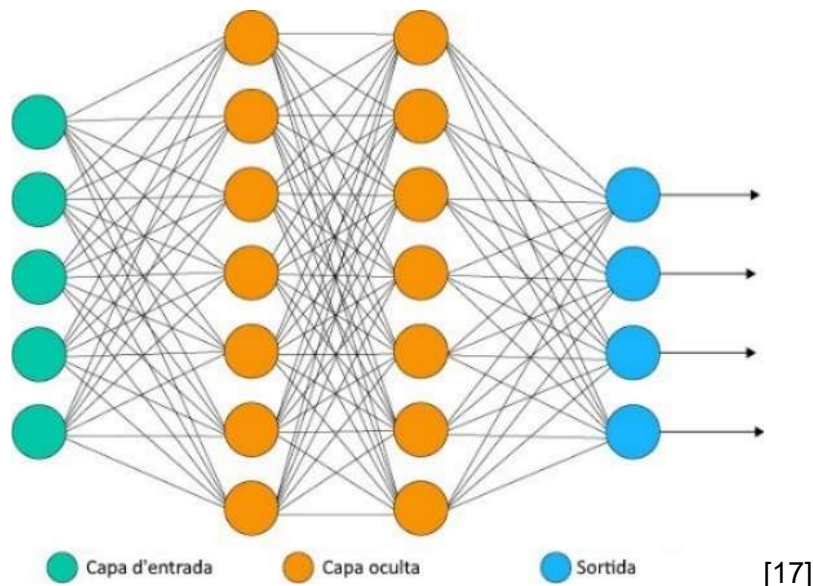
Xarxes neuronals artificials (XNA)

A causa de les limitacions del perceptró, l'any 1969, Marvin Lee Minsky i Seymour Aubrey Papert van publicar un llibre anomenat *Perceptrons: an introduction to computational geometry*, en el qual parlaven de les severes limitacions que tenia el perceptró. L'efecte del llibre va ser tant gran, que la recerca en el tema de les xarxes neuronals es va aturar completament durant casi 10 anys.

Després d'una sèrie de retallades de pressupost i poc avanç en el tema, vam arribar a principis del segle XXI, que amb grans bases de dades i ordinadors molt més potents, es van poder fer molts més avenços. Finalment, van sorgir les anomenades *Multilayer perceptron*, o Xarxes neuronals artificials.

Estructura de les XNA

Les xarxes neuronals artificials, estan formades per perceptrons connectats entre si, organitzats en tres capes: entrada, capa oculta i sortida.



La primera capa, la capa d'entrada, és la que s'encarrega de rebre les dades que haurà de processar; un cop les ha rebut, les envia a la capa amagada, que és l'encarregada

de processar aquestes dades i enviar-les a l'última capa: la sortida, que rep les dades de la capa amagada, i ens retorna un resultat.

El nombre de neurones de l'entrada i la sortida, dependrà de perquè volem utilitzar la xarxa neuronal. Per exemple: si la volem utilitzar per agafar dos nombres, sumar-los i que ens retorni el resultat, aquesta xarxa tindrà dues entrades (els dos nombres que volem sumar) i una sortida (el resultat); en canvi, si volem que sumi tres nombres, tindrà tres neurones d'entrada.

Funcionament de les XNA

Les xarxes neuronals artificials funcionen gràcies a dos algoritmes que tenen funcions específiques. El nom dels dos algoritmes són: *feedforward* i *backpropagation*. L'algoritme *feedforward* és l'algoritme que s'encarrega de donar-li les senyals d'entrada a la xarxa i calcular un resultat; mentre que el *backpropagation* és l'algoritme que s'encarrega d'entrenar la xarxa.

Feedforward

L'algoritme *feedforward* és l'algoritme que comença quan la xarxa rep les senyals d'entrada i acaba en quan aquesta retorna un resultat. Mentre dura aquest algoritme, cada una de les neurones està processant les dades per donar una resposta. L'algoritme és el que s'encarrega de que les dades flueixin perquè les neurones vagin calculant fins a obtenir una resposta.

Aquest algoritme comença donant les dades a les neurones de la capa d'entrada, una sola columna. Aquestes neurones són peculiars, ja que no fan els mateixos càlculs que tota la resta de neurones de la xarxa, de fet, no calculen res. La seva funció és normalitzar les dades.

Per optimitzar el funcionament de les xarxes neuronals, totes les dades que es processen es rebaixen de qualsevol rang de nombres normalment a dígit del 0 al 1. A aquest procés se l'anomena normalitzar. Existeixen moltes maneres de normalitzar les dades, i saber quina hauràs d'utilitzar, dependrà del context de la xarxa. L'exemple més comú per normalitzar és la divisió: si els nombres que entren a la xarxa saps que sempre aniran de 0 a un nombre concret, per exemple, 30; el que la neurona farà per normalitzar les dades, serà dividir cada una d'elles entre 30.

En quant les dades estan normalitzades, la columna o capa d'entrada ja ha fet tota la seva feina, i les dades són enviades a la següent columna: la capa oculta. Cada una de les columnes de la capa oculta treballa de la mateixa manera, els mateixos passos i les mateixes funcions. El que realment canvia, són les dades que reben. El funcionament

de les neurones de la capa oculta és més complicat que les neurones d'entrada, ja que requereix de càlculs.

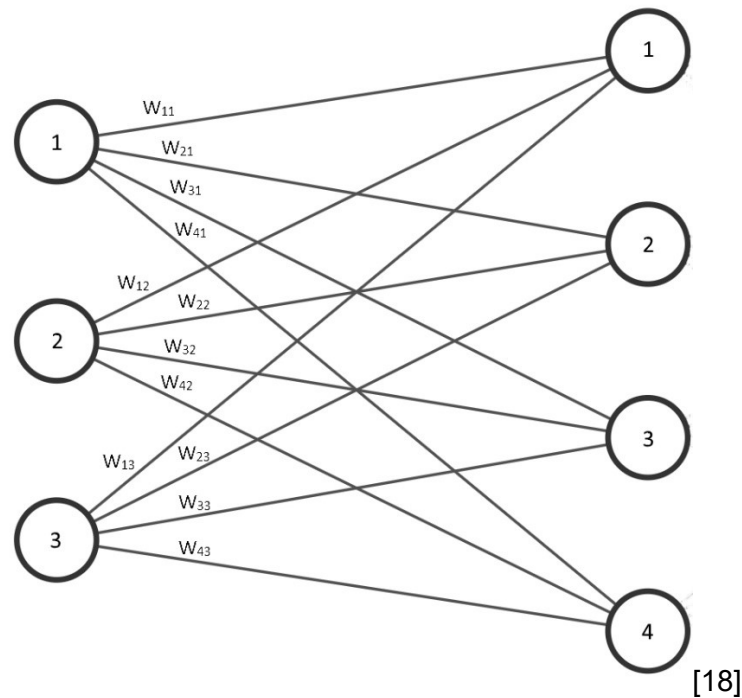
El funcionament de les neurones de la capa d'entrada es pot dividir en dues parts: el càlcul i la normalització.

Com hem vist en el perceptró, tot el càlcul que fa la neurona és agafar les dades d'entrada a les que està connectat i multiplicar cada dada pel pes que l'uneix a la neurona que està calculant. Finalment la neurona fa la suma de totes aquestes multiplicacions. El problema recau en que el perceptró està format només per una neurona, mentre que una sola columna de la xarxa pot estar formada per un nombre més gran de neurones. És per això que en les xarxes neuronals, a diferència dels perceptrons, les operacions es fan conjuntament en comptes de fer que cada una de les neurones fes aquella sèrie de càlculs. D'aquesta manera s'aconsegueix optimitzar el temps que tarda una columna en processar la informació. Per fer les operacions de manera col·lectiva, es fa ús de les matrius, que si s'organitzen de manera correcta, una sola operació seria necessària per aconseguir el mateix resultat que si cada una de les neurones fes els càlculs de manera individual.

Per fer que una sola operació de matrius causi el mateix resultat que totes les neurones fent les operacions individualment, cal organitzar-les d'una manera concreta. Ja que la operació no canvia respecte el perceptró, seguim requerint només de les senyals d'entrada i els pesos. Són aquest dos grups de dades que haurem d'organitzar de manera correcta per aconseguir el resultat final. La matriu de les senyals d'entrada, serà una matriu d'una sola columna, o vector, i tindrà el mateix nombre de files que neurones té la columna anterior, en aquest cas la columna d'entrada. A cada una de les files del vector anirà el nombre corresponent al resultat de la neurona. Per tant, a la primera fila del vector anirà el resultat de la primera neurona de la columna d'entrada, un cop ja ha estat normalitzat. Per representar els valors, els nombrarem. Per nombrar les senyals d'entrada ho farem amb una X , i li afegirem un subíndex que farà referència al nombre de neurona.

La segona matriu, la dels pesos, tindrà un nombre de files corresponent al nombre de neurones de la columna de sortida i un nombre de columnes corresponent al nombre de

neurones de la columna d'entrada. Per visualitzar-ho de la millor manera, ho farem amb un exemple. Contemplem la situació següent:



La columna d'entrada té tres neurones, i la de sortida, quatre. Cada una de les neurones de la primera columna està connectada a cada una de les quatre neurones de la següent columna. El nombre total de connexions serà el resultat de multiplicar el nombre de neurones de la primera columna amb el nombre de neurones de la segona columna; en aquest cas, 12. Sabem, ja que ho hem dit abans, que la matriu dels pesos tindrà quatre files i tres columnes. Per posar cada un dels pesos al lloc que li correspon dins la matriu, primer els nombrarem.

Per nombrar els pesos, ho farem amb la lletra “w” i dos subíndex. Si recordem, cada una de les connexions uneix dues neurones, i el valor de la connexió és el que anomenem pes. El primer dels dos subíndex fa referència al nombre de neurona al que arriba la connexió, i el segon subíndex fa referència al nombre de neurona en el que neix la connexió. D'aquesta manera, el pes de la connexió que uneix la segona neurona de la primera columna amb la tercera neurona de la segona columna, l'anomenarem w_{32} . Els subíndexs de cada un dels noms dels pesos, fan també referència al seu lloc dins la matriu dels pesos; el primer subíndex fent referència al nombre de fila en que es situarà el nombre i el segon subíndex fent referència al nombre de columna de la mateixa. Finalment, amb aquest exemple, ens quedarien les dues matrius de la següent manera:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}$$

En quan ja tenim les dues matrius creades de manera correcta, ja podem fer el càlcul. Si recordem aquest càlcul té el mateix efecte que si cada una de les neurones fes el càlcul individualment. Això significa que necessitarem més d'un resultat; en concret, quatre. Això és degut a que si les neurones calculessin per separat, cada una acabaria amb un resultat, i si aquesta operació que farem és equivalent, no ens pot donar un nombre diferent de resultats. Per tant, esperem que el resultat ens surti en forma de matriu de quatre files i una columna, un vector de quatre dimensions (quatre dimensions ja que a l'exemple, la columna de sortida tenia quatre neurones i esperem un resultat per neurona). La fórmula del càlcul, el resultat del qual anomenarem "z", és la següent:

$$z = w \cdot x$$

I el càlcul quedaria de la següent manera:

$$z = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 \cdot w_{11} + x_2 \cdot w_{12} + x_3 \cdot w_{13} \\ x_1 \cdot w_{21} + x_2 \cdot w_{22} + x_3 \cdot w_{23} \\ x_1 \cdot w_{31} + x_2 \cdot w_{32} + x_3 \cdot w_{33} \\ x_1 \cdot w_{41} + x_2 \cdot w_{42} + x_3 \cdot w_{43} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$$

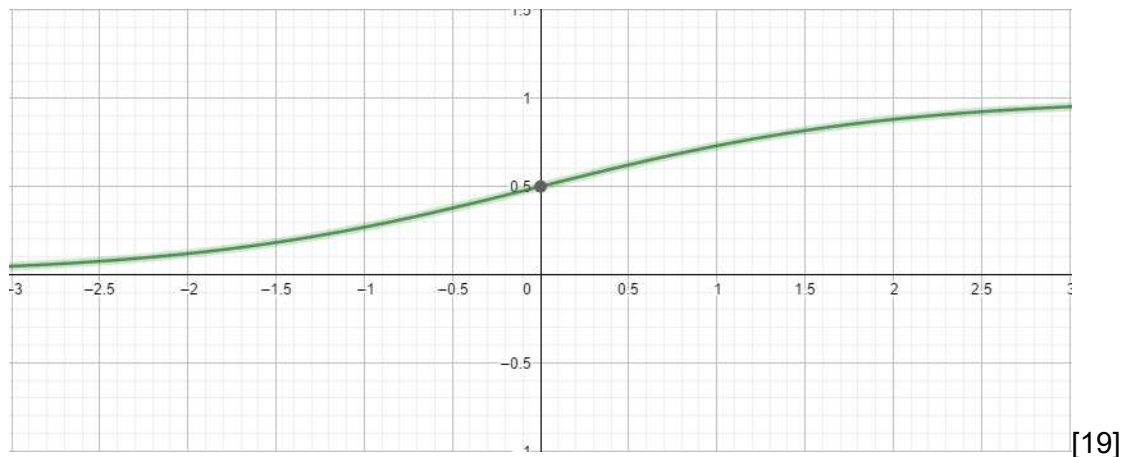
Si ens prenem un moment a comparar la operació amb el diagrama anterior al costat, podem veure que els resultats són el que esperàvem: la suma de cada una de les senyals d'entrada multiplicada pel seu pes; i el resultat és un vector de quatre dimensions, cada un dels resultats fent referència al resultat de cada una de les neurones de cada columna.

Hem de tenir en compte que aquests resultats poden no estar normalitzats, a causa de la multiplicació i la suma; i com hem dit abans, perquè la xarxa neuronal funcioni correctament, és millor que treballi amb valors normalitzats. Per normalitzar els valors, tenim una sèrie de funcions que ens estalvien el problema de no saber com fer-ho. Aquestes funcions s'anomenen *funcions d'activació*. Existeixen diverses funcions

d'activació, cada una rebaixa els nombres a un rang de nombres determinat i cada una s'aplica depenent de la funció de la xarxa neuronal. La funció d'activació més utilitzada és la anomenada *sigmoid function*. Aquesta funció redueix qualsevol valor a un entre 0 i 1. La fórmula de la funció és la següent:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Cal destacar un parell de coses de la funció següent. La primera és que per referir-se a la funció es fa servir la lletra grega sigma, en comptes d'utilitzar una lletra del nostre alfabet com la "f", que és la que s'utilitza normalment. La segona cosa a destacar és que està en funció de "z" i no en funció de "x", el que significa que el nombre que passarem per aquesta funció serà el que el calculat anteriorment, que hem anomenat "z". La funció té la següent forma:



[19]

L'últim pas per aconseguir el resultat de cada una de les neurones és normalitzar cada un dels resultats aconseguits anteriorment. Per aconseguir això, cal passar la matriu "z", obtinguda en el càlcul anterior per la funció d'activació, en aquest cas, la *sigmoid*:

$$\sigma(z) = \begin{bmatrix} \sigma(z_1) \\ \sigma(z_2) \\ \sigma(z_3) \\ \sigma(z_4) \end{bmatrix}$$

Fet això, considerarem la matriu obtinguda el resultat de la operació, i per tant, significa que en aquest punt la columna ja ha acabat de fer càlculs i pot passar aquestes dades

a la següent columna perquè repeteixi el procés amb aquests resultats com a senyals d'entrada fins que arribi a l'última columna.

Tot el procés explicat abans, des de la creació de matrius fins a la normalització dels resultats és el que es repetirà seguidament des de la primera de totes les columnes fins a la columna de sortida inclosa.

A la columna de sortida, hi ha un aspecte important a destacar, i és la funció d'activació que s'utilitzarà. En molts casos, es pot fer ús de la funció *sigmoid*, i per tant no canviaria res; però moltes vegades les xarxes neuronals utilitzen les neurones de sortida per expressar probabilitat. Si la nostra xarxa vol saber quina probabilitat hi ha de que un resultat sigui 1 o sigui 2, tindrà dues neurones de sortida, una per cada probabilitat. La principal diferència recau en que la suma de totes les probabilitats en tant per ú, ha de donar 1 i no pot donar ni més ni menys. En el cas que volguéssim crear una xarxa similar, ja sigui amb dues neurones de sortida o més, no podem utilitzar la funció *sigmoid* ja que no ens assegura que la suma de resultats doni sempre ú. És per això que, en aquest cas, s'utilitza una altra funció d'activació.

La funció d'activació que s'utilitza en el cas anterior s'anomena *softmax function*, i és la que s'encarrega de que la suma de tots els valors de sortida sigui igual a u. Una diferència important entre aquesta funció d'activació i l'anterior és que l'anterior podia retornar un resultat sense saber quins eren els altres, mentre que aquesta ha de saber tots i cada un dels resultats per poder funcionar. Aquesta és una de les raons de perquè aquesta funció d'activació no es pot representar. Tot i això, la seva fórmula és la següent:

$$S(z)_i = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}$$

Per entendre el que fa la funció, explicarem pas per pas el que fa per calcular el primer resultat de quatre que tenim. El primer pas és la divisió, com a numerador tenim el nombre e elevat al primer resultat, i com a denominador tenim el sumatori del nombre e elevat a cada un dels quatre resultats.

D'aquesta manera s'aconsegueix que la suma de cada un dels resultats sigui igual a 1, fent que la xarxa pugui donar probabilitats com a resultat.

Backpropagation

En aquest punt hem aconseguit que la nostra xarxa ens doni un resultat gràcies a l'algoritme anterior. Però les xarxes neuronals destaquen per la seva capacitat d'aprendre. Per aconseguir que la nostra xarxa sigui capaç de corregir els seus errors, hem d'implementar aquest segon algoritme, que anirà corregint els valors dels pesos fins a aconseguir que l'error sigui mínim. Aquest algoritme consta de dues parts: càlcul d'errors i càlcul dels nous pesos.

Càlcul d'errors

La primera part per entrenar a una xarxa neuronal és mirar si ho ha fet malament, i en cas que sí, mirar si ho ha fet molt malament o poc malament. És per això que el primer pas és calcular l'error.

De la mateixa manera que en l'algoritme anterior, aquest també va columna per columna, però com diu el seu nom, va endarrere; des de l'última columna fins a la primera.

La funció que calcula l'error de cada una de les neurones de cada columna, depèn de la funció d'activació que s'ha utilitzat anteriorment. Cada funció d'activació suposarà una funció o una altra. En els exemples anteriors hem utilitzat dues funcions d'activació: la *sigmoid* i la *softmax*; per tant, mirarem el càlcul d'error per cada una d'aquestes funcions.

Abans d'això, però, ens hem de fixar en certs aspectes importants. El primer de tot és que el càlcul d'error s'utilitza per l'aprenentatge supervisat, per tant la xarxa ha de saber quins resultats eren els esperats abans d'entrenar. El segon aspecte important és que hem de tenir les dades de sortida de cada una de les columnes per calcular l'error de cada una de les neurones de la xarxa neuronal.

Per calcular l'error en la funció *sigmoid* utilitzarem la següent fórmula:

$$\text{error} = \text{resposta desitjada} - \text{resposta de la XNA}$$

I per calcular l'error amb la funció *softmax* farem ús de la següent fórmula:

$$e = -(y \cdot \log(p) + (1-y) \cdot \log(1-p))$$

a on “y” és la resposta desitjada i “p” és la resposta de la xarxa neuronal.

Aquestes dues fórmules serveixen per calcular l'error de la columna de sortida, ja que tenim manera de saber quin havia de ser el resultat esperat, però per les columnes de la capa oculta, no tenim manera de saber què hauria de ser el resultat correcte. És per això que fem ús de càlculs per determinar directament l'error de les columnes.

D'aquesta manera determinem que l'error de les columnes de la capa oculta, és la multiplicació de la matriu dels pesos que uneixen aquella columna amb la següent multiplicat per la matriu de l'error de la columna següent.

A causa de que cada una de les neurones comet una part de l'error total, és a dir, cada neurona comet error, l'error ve determinat per matrius d'una sola columna, vectors, i al final existirà una matriu d'errors per cada una de les columnes de la xarxa neuronal artificial.

En quant ja tenim tots els errors calculats, podem passar a calcular els nous pesos.

Càlcul de pesos

Per entrenar realment la xarxa neuronal, el que cal fer és variar els pesos fins que aconseguim els valors que minimitzen l'error de la xarxa neuronal. Com que no podem retocar els pesos a cegues, degut a la quantitat de valors que hi ha, per aconseguir els valors òptims, fem ús de l'error anteriorment calculat per tenir una idea de cap a on i amb quanta diferència hem de variar els pesos.

La fórmula per variar els pesos és el seu valor anterior més l'increment d'aquests:

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

El que de veritat ens interessa calcular és aquest increment de pesos, la quantitat en que variarem aquests, el que hem anomenat “increment de pesos”. La fórmula per calcular aquest increment de pesos és la següent:

$$\Delta w_{ij} = lr \cdot E \cdot \sigma'(O) \cdot I^T$$

El primer element que veiem a la fórmula és “lr” que significa *learning rate*. Aquest terme és un nombre que determina la velocitat a la que volem canviar els pesos. Aquest terme és com un percentatge que serveix perquè, encara que sapiguem cap a on em de variar els pesos, no els variem massa de cop i ens passem, sinó que els variem a poc a poc i així aconseguim arribar amb més precisió, tot i que més lentament. Aquest valor normalment és més petit que 1 agafant valors com 0,1 o 0,00001 entre altres. Hem de tenir en compte que si aquest valor és massa gran, anirem ràpids entrenant, però aquest entrenament no serà precís i mai arribarem a un punt a on els pesos anul·lin completament l’error; mentre que si el valor és massa petit, el resultat seria exacte però tardaríem massa temps en arribar al resultat que la precisió no sortiria a compte.

El segon terme que trobem a la fórmula és “E”, referint-se a l’error de la columna mateixa. Cal tenir en compte que la variació de pesos que estem calculant equival pels pesos que van a la columna de la que agafem els valors. Per tant, si agafem l’error de la columna de sortida, la variació de pesos que calcularem serà la dels pesos que arriben a aquella columna.

El tercer terme que trobem s’anomena *gradient*, que és la derivada de la funció *sigmoid*. Les dades que passarem per aquesta funció seran les dades de sortida de la columna. Si calculem els pesos que val a la columna de sortida, aquestes dades de sortida serien la resposta de la xarxa neuronal. La funció derivada té la forma següent:

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

El que trobem dins la funció a la fórmula anterior és “O”, que es refereix a “output”, senyal de sortida.

L'últim terme de la fórmula és I^T . La lletra “I” fa referència a “input”, que és el resultat que ens ha donat la columna anterior. Aquest terme, una matriu, està transposat.

L'ordre en que es fan les operacions és d'esquerra a dreta, important ja que sinó el resultat no seria correcte; tot i que primer es fan totes les multiplicacions de matrius i finalment es multiplica el resultat per el *learning rate*. Cal destacar que la multiplicació de matrius entre l'error i el *gradient* no és una multiplicació de matrius normals, sinó que és *Hadamard product*.

Un cop calculat aquest increment de pesos per cada grup de pesos, li sumem el resultat al valor dels pesos que teníem anteriorment; i repetint aquest procés múltiples vegades, aconseguirem apropar la xarxa en un punt en que l'error total és 0.

Elements de la programació

Ara que ja hem parlat de com funcionen les xarxes neuronals, hem de veure alguns dels elements més importants i característics de la programació. En aquest apartat veurem que són les llibreries i parlarem del llenguatge de programació JavaScript.

Les llibreries

Les funcions

Tots els codis i programes funcionen gràcies a una sèrie de funcions que li permeten certa funcionalitat a l'algoritme. Aquestes funcions són blocs de programa als que se li assignen un nom i contenen una sèrie de passos a seguir.

Hem d'entendre que quan un programa és executat, l'ordinador que l'està executant llegeix el codi, sense aturar-se, des de la primera línia fins a la última. Si no volem que una part del programa s'executi a l'inici, podem posar-ho dins d'una funció. Al fer això, l'ordinador que està executant el programa es guardarà la funció a la memòria en quan la llegeixi però no l'executarà.

Per tant, tenim que les funcions són trossos de programa que no s'executen a l'inici, sinó que es guarden a la memòria, i que tenen un nom concret. La gràcia de les funcions, el que fa que siguin tant útils, és que es poden executar des de qualsevol moment específic durant un programa. Un exemple per veure-ho molt clar és una funció que crea un punt en un lloc en el que hem clicat amb el ratolí. A dins de la funció, li posarem el codi necessari perquè agafi les coordenades del ratolí i faci un punt allà, i al programa li direm que cada cop que l'usuari faci clic amb el ratolí executi aquella funció.

Qualsevol llenguatge de programació ve amb unes funcions ja creades, que poden anar des de crear un text o les operacions matemàtiques més bàsiques fins a detectar quan l'usuari prem una tecla o fa clic amb el ratolí. I moltes vegades, fent ús de funcions creades per un mateix, es pot arribar a crear un programa complex. Tot i així hi haurà vegades que creant funcions no serà suficient per un projecte que es vol fer, sinó que necessites una eina que redueixi la teva feina aportant-te funcions que simplifiquen el

codi o que et permeten crear objectes que, amb codi normal seria molt complicat. És aquí a on apareixen les llibreries.

Les llibreries són codis que pots importar als teus projectes i t'ofereixen tota una sèrie de funcions que reduiran la llargària del teu codi o t'aportaran noves eines que podràs utilitzar per fer el teu programa més eficient. Un exemple d'aquestes noves eines que et poden aportar les llibreries seria matrius. No hi ha cap llenguatge de programació que de base pugui fer càlculs de matrius perquè ja te tona una sèrie de funcions escrites, però es pot descarregar una llibreria d'internet que et permeti fer operacions de matrius en el teu projecte.

PART PRÀCTICA

En aquesta part pràctica, us explicaré els passos que vaig seguir per aprendre a fer les xarxes neuronals des de 0 i les dificultats amb les que em vaig trobar. Al final del treball, a l'annex, podreu trobar una còpia del codi de la xarxa neuronal.

Els principals objectius d'aquest treball de recerca eren crear una xarxa neuronal i, si fos possible, crear-la sense cap ajuda de llibreries. Per aconseguir aquests objectius, la millor manera de fer-ho seria buscar una idea senzilla per la xarxa neuronal, ja que si la idea fos massa complexa, seria molt difícil aprendre tota la teoria de xarxes neuronals i de programació per aconseguir una cosa tant complicada.

Una de les hipòtesis a demostrar, era que un alumne de segon de batxillerat pot crear una xarxa neuronal des de zero; i això també ho discutirem aquí.

Al principi del treball, tenia una metodologia molt clara: aprendre gràcies a les plataformes de YouTube i Coursera i fer la xarxa neuronal amb el llenguatge de programació JavaScript.

La plataforma Coursera, és una pàgina web de cursos online. En aquesta pàgina, universitats de tot el món publiquen els seus cursos sobre el tema que vulguin. Per publicar i monetitzar els cursos, les universitats ho poden fer de tres maneres: publicar un curs completament gratuït, publicar un curs que només puguis veure o publicar el curs i que només hi pugui accedir la gent que el compra. Al acabar aquests cursos, l'estudiant rep un diploma de la universitat conforme ha fet el curs. En el curs gratuït, no has de pagar per rebre el diploma; en el curs en que només pots veure les classes, no pots fer exàmens ni rebre el diploma si no pagues; i en el curs en el que pagues per accedir-hi, reps el diploma al acabar-lo.

Coneixements previs

Abans de començar a explicar la metodologia de la creació de la xarxa neuronal, cal destacar que no partia d'uns coneixements nuls en quant a programació. A mitjans de 4rt d'ESO (fa 2 anys), vaig descobrir els llenguatges de programació, i el meu interès en ells van fer que comencés a aprendre de forma autodidacta. Durant tot un any, vaig estar navegant internet en busca d'exemples i tutorials per aprendre tant com pogués dels llenguatges de programació.

El primer llenguatge de tots que vaig descobrir va ser un anomenat “Processing”. Aquest llenguatge era molt visual i fàcil d’entendre. Però al ser un llenguatge de programació seguia uns bàsics que segueixen tots els llenguatges. Per tant ja tenia una bona base per enfrontar-me a altres llenguatges més sofisticats. El següent llenguatge que més em va interessar, va ser C++, un llenguatge molt utilitzat avui en dia. No vaig aprofundir molt, però sí que vaig aprendre suficient com per utilitzar els coneixements més endavant.

Per últim, em vaig interessar en JavaScript. Vaig buscar i fer diversos cursos on-line d’aquest llenguatge. Vaig aconseguir un bon nivell.

En resum, al començar el treball de recerca, tenia una base sòlida de llenguatges de programació i havia aprofundit en C++ i JavaScript; però de xarxes neuronals, no n’havia sentit a parlar, encara.

Primer intent autodidacta

Per començar el treball de recerca, tant la part teòrica com la part pràctica, havia de saber que eren les xarxes neuronals i com funcionaven. Per començar a aprendre, vaig decidir buscar cursos on-line, ja que en podia trobar que estiguessin bé i fossin gratuïts. Finalment, vaig començar un curs sobre xarxes neuronals a una pàgina anomenada “Coursera”.

El curs estava pensat per fer durant onze setmanes, dedicant-se tres hores a la setmana. Les primeres setmanes del curs eren introducció, i parlaven d’on venien les xarxes neuronals i hi havia classes d’àlgebra bàsica i matrius per qui ho necessités. D’aquestes primeres setmanes, vaig acabar fent només les classes de matrius, que eren tres vídeos de 20 min cada un. Cal destacar que un concepte de les matrius estava explicat de manera errònia, cosa que va aportar confusió més endavant.

La resta de setmanes del curs, les vaig fer per sobre, ja que vaig veure que aquest curs no anava enlloc. Vaig acabar aprenent que era *feedforward* i en total, vaig extreure dotze pàgines d’apunts a mà amb molts diagrames i dibuixos.

Primeres idees

Al “acabar” l’anterior curs, ja sabia què eren les xarxes neuronals, tot i que tenia poques idees de que podien fer. Tot i això, era moment de triar una idea per la meva futura xarxa neuronal. Directament, vaig descartar qualsevol idea que tingués a veure amb reconeixement d’imatges o de veu, ja que era conscient de que eren coses de nivell avançat i requerien de màquines potents per treballar, cosa de la que no disposava. Idees de nivell avançat les vaig descartar principalment per temps: tenia un estiu per aprendre la programació i la teoria necessària, i sabia que no era possible. I exemples que requerissin de màquines potents els vaig descartar, ja que treballava des del meu portàtil, que era poc potent, i fer projectes ambiciosos en ell significava empenyar-se per aplicacions que no responen o per temps de càrrega massa grans.

Al final vaig tenir la primera idea: fer una xarxa neuronal a la que li diguessis un parell de pel·lícules que t’agraden, i a través de factors comuns (director, durada, tema...), et proposés una o dues que et poguessin agradar.

Encara no sabia com es podien programar les xarxes neuronals artificials, però sí que sabia que, per començar el projecte, necessitaria una base de dades gran amb pel·lícules i les seves característiques principals. Com que omplir bases de dades és complicat i tediós, vaig dissenyar i crear una pàgina web amb un petit qüestionari que cada persona a qui li compartís ompliria amb la seva pel·lícula preferida.

Començant a investigar com podia fer aquest programa, vaig descobrir que no ho podria fer amb una xarxa neuronal artificial, sinó que es feia amb un altre tipus de programa. En aquest punt, vaig abandonar la idea.

Primers intents de programació

En aquell moment, m’havia quedat sense idea, per tant, vaig començar a fer la part teòrica. Avançant, va arribar un moment en el que vaig acabar l’apartat del perceptró, i vaig pensar que seria una bona idea fer-ne un; tot i que no contava completament com a xarxa neuronal. Com a exemple, vaig utilitzar el mateix que a l’apartat abans mencionat: un perceptró que resolgués portes lògiques. Com que la intenció era fer un programa senzill i no perdre molt temps creant-lo, vaig decidir fer-ho en C++, sent la primera raó el fe de que feia molt que no l’utilitzava i era un dels meus preferits. També va ser perquè era un llenguatge eficient i no necessitava cap aspecte visual per fer el perceptró.

La creació del programa del perceptró no va comportar molts problemes, ja que era un programa senzill i vaig poder evitar treballar amb matrius gràcies a la poca quantitat de dades que havia de processar. A més a més, ja que tots els programes de programació tenen un sistema per operar amb portes lògiques, vaig poder automatitzar el procés d'aprenentatge del perceptró amb poques línies de codi més.

Tot i que el programa, teòricament, havia de funcionar, no ho feia; i vaig estar dies canviant trossos de programa i revisant una i altre vegada per detectar algun error, però no vaig poder descobrir perquè el perceptró no funcionava, ja que tornava resultats erronis després d'haver estat entrenat.

La segona idea

En quan em vaig informar més sobre les xarxes neuronals més senzilles vaig veure que la gran majoria classificaven. Això em va ser de gran ajuda, ja que ara podia tornar a pensar sobre la idea de la meua xarxa neuronal senzilla, ja que hi havia moltes coses que es podien classificar. Tenia clar que volia que fos molt visual i que el que fos que classifiqués la xarxa, havia de dependre de la percepció humana, ja que així m'assegurava que el que feia la xarxa no ho podria fer qualsevol altre programa. Pensant en aquesta idea, vaig acabar pensant en colors, ja que es podien classificar de moltes maneres.

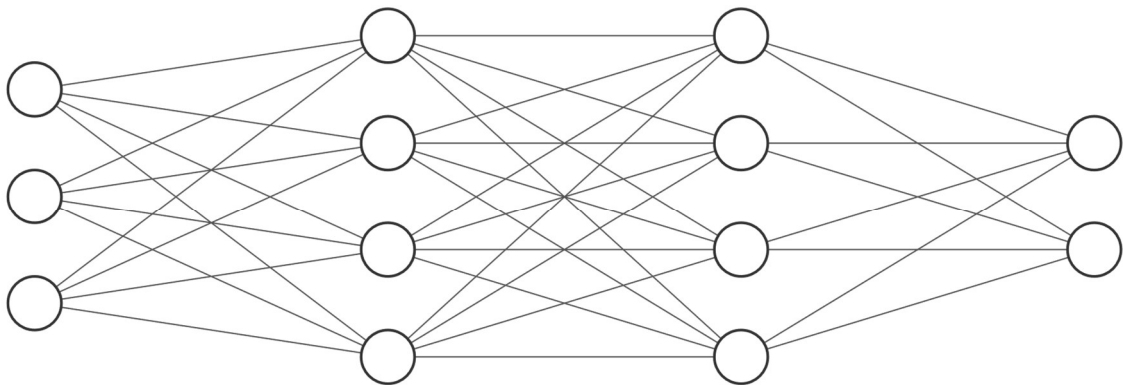
Finament, vaig arribar a la meua idea final: la xarxa neuronal classificaria els colors segons si eren clars o foscos.

Desenvolupament de la segona idea

Un cop ja tenia la idea triada, havia de passar al desenvolupament de la xarxa neuronal. A l'hora de crear una xarxa neuronal, cal tenir dos factors en compte: la seva estructura i la seva presentació.

Per la estructura de la xarxa, havia de triar el nombre de columnes i el nombre de neurones de cada columna. Per la columna d'entrada, triar el nombre de neurones era fàcil: tres; ja que la xarxa llegiria el color en rgb, per tant, rebria tres dades diferents. I per triar les neurones de sortida tampoc era molt difícil: dues; una per indicar el percentatge de probabilitat de que el color fos clar i l'altre per fosc.

El fet de triar el nombre de columnes i neurones de la capa oculta ja era més difícil. Els factors a tenir en compte a l'hora de triar és que quantes més columnes, més complicat seria crear-la i més temps tardaria, però els resultats serien molt més precisos; mentre que si posava poques columnes la xarxa seria més imprecisa però aniria més ràpid i seria més fàcil de fer. Al final em vaig quedar en un punt intermedi: dues columnes de quatre neurones cada una.



Plantejament de la programació

Ara que ja tenia la idea i que estava segur de que es podia dur a terme amb una xarxa neuronal, m'havia de plantejar tots els aspectes de com ho faria: des de com funcionaria la xarxa neuronal a com interactuaria l'usuari amb ella. Per començar per algun lloc, vaig decidir començar per la part tècnica abans que la visual.

Com hem vist a la teoria, la xarxa havia de constar de dues parts o processos, el *feedforward*, en el que donava la resposta que creia; i el *backpropagation*, a on es corregia gràcies a l'usuari si era necessari.

Feedforward

El primer pas que havia de prendre per començar a fer la xarxa neuronal era pensar en com fer l'algoritme amb el que la xarxa tornaria una resposta. Per fer això, la xarxa

necessitava unes senyals d'entrada, i com hem quedat abans, aquestes serien un color en rgb. En aquest punt, ja sabia per on començar: havia de fer un sistema que em generés un color de manera aleatòria i que l'usuari pogués interactuar per activar la xarxa neuronal.

Per fer això vaig decidir crear un requadre que seria del color que es triés aleatòriament, i un botó que, cada cop que fos clicat, canviés el color. Fer això no va ser complicat, ja que qualsevol llenguatge de programació té una funció que permet escollir un número aleatori entre dos nombres determinats; i per pintar un requadre d'un cert color, no era més complicat que dues línies de codi.

En el moment en que vaig acabar de fer el sistema que escollia els colors, ja podia començar a programar la xarxa neuronal, ja que la senyal d'entrada era tot el que de moment necessitava.

Com hem vist a la part teòrica, el que fa la xarxa neuronal per donar una resposta és fer varies multiplicacions de matrius entre les dades d'entrada i els pesos; el que significava que necessitaria una manera d'operar amb matrius. Tots els llenguatges de programació tenen unes matemàtiques bàsiques incorporades, com podria ser: suma, resta, multiplicació, divisió, funcions trigonomètriques... però no n'hi ha cap que tingui incorporat un sistema d'operacions amb matrius.

Per aconseguir-ho, normalment faria ús d'una llibreria externa, una sèrie de funcions que algú va programar que em permetrien operar amb matrius. Utilitzar això tenia dos majors inconvenients: el primer es que estaria directament negant la meua hipòtesis, que afirmava poder crear la xarxa neuronal artificial sense ajuda de cap llibreria, i el segon inconvenient era que hauria d'aprendre les funcions de la llibreria, saber com utilitzar-les i haver de tractar amb errors per causa del funcionament de la llibreria. És per això que vaig decidir crear el meu propi codi que em permetria crear i operar amb matrius. En aquell moment no podia programar-ho sense ajuda, ja que no tenia els coneixements necessaris; així que vaig decidir seguir un tutorial de YouTube, fet per Daniel Shiffman.

Al acabar aquest tutorial tenia un programa que em permetia operar i crear matrius i que sabia com funcionava i exactament quines funcions tenia i com funcionava. Aquesta petita avantatge em va ser molt útil ja que em va fer més fàcil la detecció d'errors en el futur i la implementació de noves funcions en cas de necessitat.

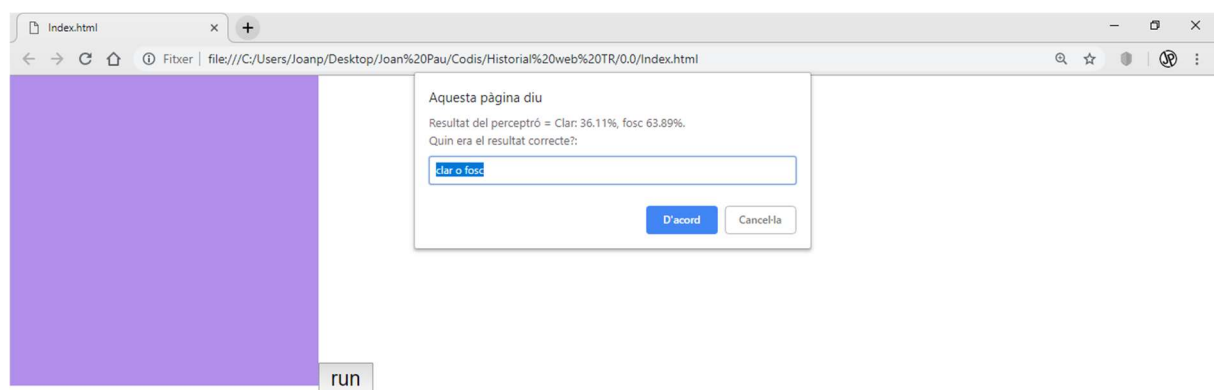
En aquest moment ja vaig començar a programar la xarxa neuronal amb el seu primer algoritme: el *feedforward*. En acabar, ja tenia un programa que agafava un color aleatori

en rgb i em deia si era clar o si era fosc. Una cosa a tenir en compte del programa en aquest punt és que els pesos, que són l'element que fan que la xarxa funcioni millor o pitjor, es triaven també de manera aleatoria cada cop que l'usuari entrava a la web de la xarxa neuronal. En aquest punt no era un problema, però més endavant sí que ho serà.

Backpropagation

Ara que a tenia el primer dels dos algoritmes de la xarxa, podia continuar amb el següent, el que li permetria a la xarxa aprendre i funcionar. Com que estava treballant amb l'aprenentatge supervisat, el primer que havia de fer era preguntar a l'usuari quina era la resposta correcta. Per fer això, feia que quan la xarxa ja tingués la resposta, apareixes una notificació que preguntés per una resposta a l'usuari.

En quan vaig tenir allò fet, vaig donar per feta la primera versió de la xarxa, i vaig decidir aparcar el tema de l'algoritme de *backpropagation* ja que era molt complicat i necessitava informar-me més. Mentrestant, aniria retocant i afegint aspectes visuals per millorar l'experiència de l'usuari a la web de la xarxa. En aquest punt, la web tenia la forma següent:



Els pesos

Com s'ha mencionat abans, cada cop que l'usuari entrava a la pàgina web, els pesos es generaven de manera aleatòria. Si volia continuar amb el desenvolupament de la xarxa, havia de canviar això, ja que sinó, la xarxa mai podria aprendre i fer l'algoritme que entrenava la xarxa seria inútil ja que tot l'entrenament fet es perdria. Això significava que, abans de continuar fent la xarxa, havia de trobar una solució a aquest problema i aplicar-la.

La solució era relativament senzilla: cada cop que l'usuari entrenés la xarxa, els pesos canviarien de valor, i en aquell moment hauria de copiar el valor dels pesos en un arxiu per després, el següent cop extreure les dades del fitxer i donar aquells valors als pesos. D'aquesta manera, els pesos es guardarien cada cop que es tornés a iniciar la pàgina web.

Per guardar els valors dels pesos, es pot fer de dues maneres: localment o al núvol. La avantatge de fer-ho localment, és que no requereix d'accés a internet i el fitxer es guardaria en un lloc segur: el disc dur del meu ordinador. La principal desavantatge de fer-ho d'aquesta manera és que la pàgina web no podria ser publicada, ja que després, per agafar els valors dels pesos des de qualsevol ordinador que no fos el meu, no ho podria fer perquè el fitxer no existiria.

És per això que vaig decidir fer-ho a núvol. D'aquesta manera, els valors estarien en una base de dades o en un fitxer al núvol i podrien ser accedits per qualsevol persona que entrés a la web des de un dispositiu que no fos el meu ordinador. Per guardar els pesos en una base de dades a núvol, vaig utilitzar la aplicació de *firebase*.

Firestore

Firestore és una aplicació gratuïta de *Google* que et permet crear una base de dades al núvol i escriure i llegir dades en ella. Aquesta base de dades està preparada per poder ser utilitzada des de aplicacions de mòbil, pàgines web, videojocs, i moltes plataformes més.

La meva intenció amb aquesta plataforma era que en el moment en que l'usuari es connectés a la pàgina web, la xarxa copiés les dades de la base de dades per actualitzar el valor dels pesos i que cada cop que fos entrenada i els pesos canviessin de valor, s'enviessin els nous valors a la base de dades.

Era la primera vegada que treballava amb la aplicació del *Firestore*, per tant, no sabia com funcionava. Però, al ser una eina de *Google*, tenia a l'abast una explicació molt

extensa per part dels desenvolupadors a on hi havia tota una guia d'inici; i a més, hi havia molta gent a fòrums d'internet que havien penjat la seva explicació de com utilitzar aquesta aplicació. Gràcies a aquestes ajudes, vaig ser capaç d'entendre el funcionament bàsic; i al cap de pocs dies d'estar informant-me, vaig aconseguir aprendre a escriure informació a la base de dades.

En quan ho vaig saber fer, vaig incorporar al codi de la xarxa una funció que escrivia els valors dels pesos a la base de dades cada cop que s'executava la xarxa neuronal. En aquell moment la xarxa encara no llegia els valors de la base de dades, només escrivia. En aquest punt em vaig trobar amb un problema que, en aquell moment no era gran cosa però sabia que s'havia de resoldre ja que sinó en el futur seria més tediós i complicat.

El problema era que cada cop que la xarxa escrivia informació a la base de dades, creava una carpeta amb els valors dels pesos a dins, per tant, cada cop que la xarxa es feia funcionar, creava una nova carpeta a la base de dades, i si volia eliminar les que no em servien, ho havia de fer a mà. Això era un problema ja que les xarxes neuronals estan pensades per ser entrenades com a mínim milers de vegades. Al ser la meua xarxa una més senzilla, no esperava que el nombre d'entrenaments arribés al miler, però eliminar mil carpetes d'una base de dades seria una feina tediosa.

Per tant, vaig estar buscant la manera per fer que, cada cop que la xarxa era entrenada, actualitzés les dades de la base de dades enlloc de crear una nova carpeta. Trobar informació per solucionar aquesta classe de problemes, més concrets, sempre es difícil. La meua primera idea per solucionar el problema va ser intentar trobar la manera de que cada cop que s'enviessin dades, totes les carpetes que no eren la que s'acabava d'enviar fossin eliminades. Més tard, vaig ser conscient que aquest enfocament era incorrecte. Després d'estar informant-me més sobre com funcionava la aplicació de *Firebase*, vaig descobrir que hi havia dues maneres diferents d'enviar dades:

- **Set:** Una de les funcions que es podien utilitzar per enviar dades a *Firebase* era *set*. Aquesta funció creava una nova carpeta amb les dades que s'enviaven a la base de dades i és la que estava utilitzant fins a aquest moment.

- **Update:** La funció *Update*, de la mateixa manera que la *set*, envia dades a la base de dades, però enlloc de crear una nova carpeta, canviava els valors de les dades que ja estaven a la base de dades.

En saber que existien aquestes dues funcions vaig tenir clar que, per solucionar el problema, havia d'utilitzar la funció *Update* per penjar les dades enlloc de la *Set* que és la que havia estat utilitzant. Al canviar el programa perquè funcionés amb la funció *Update*, vaig aconseguir solucionar el problema, i quan les dades eren enviades, s'actualitzaven els valors de la base de dades enlloc de crear una nova carpeta, cosa que feia que ja no hagués d'eliminar totes les carpetes que s'anaven creant.

Disseny de la pàgina

Un cop ja van estar solucionats tots els problemes més importants que hi havia ja podia passar a fer l'algoritme per entrenar la xarxa neuronal, però encara no havia fet suficient recerca com per aconseguir fer l'algoritme i fer-lo funcionar, així que vaig decidir fer la pàgina web de la xarxa neuronal més acollidor.

El primer pas va ser idear un esbós del que volia que fos la pàgina web. Un dels aspectes més importants a tenir en compte per dissenyar la pàgina era que la xarxa havia de ser l'element principal, no podia haver-hi res que es veiés abans que la xarxa.