**Portada**

**Índex**

1. Introducció ..................................................................................................................... 3
   1. Motivacions ....................................................................................................... 3
   2. Hipòtesis ............................................................................................................ 3
   3. Objectius ............................................................................................................ 4
   4. Metodologia .................................................................................................... 4
2. Xarxes neuronals i programes ordinaris ..........................................................................
3. Abans de començar…
   1. Matrius
      1. Vectors
      2. Matrius
      3. Operacions bàsiques
         1. Suma
         2. Multiplicació
         3. Transposició
   2. Colors “RGB”
   3. Portes lògiques
4. Com funciona una xarxa neuronal
   1. El perceptró
      1. *Feedforward*
      2. *Backpropagation*
         1. Aprenentatge supervisat
      3. Usos del perceptró
         1. Perceptró per prediccions
         2. Perceptró per classificacions
      4. Limitacions del perceptró
   2. Xarxes neuronals artificials (XNA)
      1. Estructura de les XNA
      2. Funcionament de les XNA
         1. *Feedforward*
            1. *Softmax function*
         2. *Backpropagation*
            1. Aprenentatge supervisat

Càlcul de l’error

Càlcul del pes

1. Conceptes bàsics de programació
   1. En què consisteix la programació?
   2. HTML i JavaScript
   3. Python
   4. TensorFlow
2. Les xarxes neuronals i el futur
3. Creació d’una xarxa neuronal
   1. Metodologia utilitzada
   2. Principals problemes.....Programació de la xarxa

**Introducció**

1. **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 del que 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 del que eren, i vaig veure que era un tema molt interessant i del qual li podria treure molt profit.

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

1. **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 jo 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 que crear-la des de zero fos possible així que vaig formular la segona hipòtesi: “**És possible crear una xarxa neuronal artificial complexa sense l’ajuda d’una llibreria**”.

1. **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”.

1. **Metodologia**

Ara que ja sabia que volia crear una xarxa neuronal artificial, només havia de pensar com ho faria. En aquest tema, només 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.

**Xarxes neuronals i programes ordinaris**

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 menys 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 serie 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 pixels 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, hauriem de fer el mateix programa per cada número, i com que dues persones no dibuixen els números de la mateixa manera, hauriem de fer el programa per cada manera de dibuixar cada número. Acabariem amb milions de línies de codi, el programa tardaria segles en funcionar, i no sempre encertaria el resultat.

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



En el nostre codi simplement creariem la màquina de la següent manera: indicariem el nombre de neurones a l’entrada, el nombre de neurones a la sortida, com interactuen les neurones entre elles i li afegiriem una manera d’entrenar-la. En aquest punt ja tindriem la nostra xarxa neuronal artificial. I si ens miressim el codi, no veuriem com funciona la xarxa neuronal, simplement veuriem 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.

**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:

1. **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 =

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: ai, on el subíndex és un nombre real referint-se a la fila. En el cas de la matriu anterior, podem veure que a2 = 2.

1. **Matrius:**

Les matrius consten normalment de més d’una columna, i es denominen normalment “matriu m x n”, a on *m* representa el número de files i *n*representa el número de columnes.

A =

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 *aij*, *i* referint-se a files i  *j* a les columnes; per tant, en la matriu anterior, a12 = 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 conter des de zero.

**Operacions bàsiques**

1. **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.

+ 3 = =

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.

\*

1. **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.

Per multiplicar dues matrius, cal que la primera matriu tingui el mateix nombre de columnes que files te 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 x m i un vector m x 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 x 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.

x = 1 · 3 + 2 · 2 + 3 · 1 = 10

A l’hora de multiplicar matrius bidimensionals, es pot simplificar a varies multiplicacions de files per columnes. Si tenim dues matrius 2 x 2, la final serà de 2 x 2. El primer nombre de la matriu final, a11, 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 a23, 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.

· = =

**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 aij, ara estarà a l’espai aji. Per indicar que una matriu s’ha de transposar, es fa afegint la lletra “T” a la matriu.

=

**Color “RGB”**

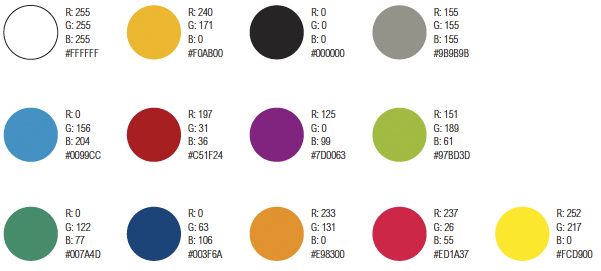
Si volem utilitzar els colors en qualsevol tipus de programa, ja sigui per crear-los o perquè el programa els reconegui, els hi haurem d’assignar valors. La manera més comuna de fer-ho és amb el codi rgb.

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 predeterminada. També es pot utilitzar per tu, indicar-li a un programa que pinti una zona d’un cert color. Per tant, si tu vols pintar una zona del teu programa de color vermell, primer hauràs de definir la zona perquè el programa sàpiga quins píxels ha de pintar, i després diràs, en rgb, que tots aquells píxels anteriorment escollits hauran de tenir una quantitat equivalent a 255 (o màxima) en el color vermell i 0 (o mínima) en els dos altres colors.

Per fer això simplement hauràs de posar entre parèntesis tres nombres d’un rang concret en un ordre concret. El rang, serà la quantitat, de 0 a 255; i l’ordre serà primer la quantitat de la llum vermella, després la quantitat de la llum verda i per últim la quantitat de la llum blava. Si fem això, el color vermell pur hauria de quedar de la següent manera: (255, 0, 0). Si agafem l’ordre dels colors (vermell, verd, blau) i els traduïm a l’anglés, podrem veure que rgb prové de la primera lletra de cada un dels colors: *red, green, blue*.



**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:

1. ***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.



1. ***AND***

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



1. ***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.



**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 es 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 correcte, 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 algoritmes externs a la xarxa. D’aquesta manera, és possible entrenar les xarxes neuronals amb bases de dades més grans (normalment sobrepassant 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 algoritmes 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 esta connectada amb la següent. Un exemple de xarxa neuronal seria el següent:



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 rebrà unes senyals d’entrada, farà una càlculs i tornarà un resultat.



El nostre perceptró tindrà dues senyals d’entrada, anomenades x1 i x2, 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 (w11, w12), 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 w11 estarà a la primera fila i primera columna de la matriu, i el pes w12 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:

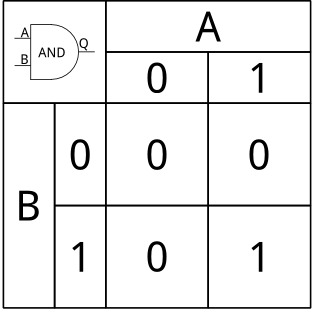
Pesos:

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.



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.

***z = w · x***

x = 3 · 0 + 15 · 1 = 15

Al iniciar, el nostre perceptró ha assignat aleatòriament el valor 3 al pes 1 (w11) i el valor 15 al pes 2 (w12); 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: .



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*.

1. **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:

**error = resposta desitjada – 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:

**wi = wi + wi**

a on wi 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:

**wi = error \* *input* \* *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:

1. 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.
2. 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.
3. 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**

1. **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.



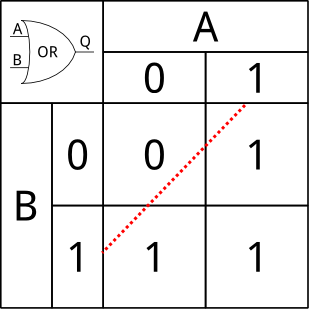
El perceptró pot fer això gràcies a la similitud entre la seva fórmula z = x·w +b i la fórmula de la recta y = mx + n. Al entrenar el perceptró amb aquelles dades, el que fem és buscar el pes i el *bias*, que corresponen al pendent i al punt d’intersecció de la recta amb l’eix de les ordenades respectivament.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

1. **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.

**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.

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

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 *Multylayer perceptron*, o Xarxes neuronals artificials.

**Xarxes neuronals artificials (XNA)**

1. **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.

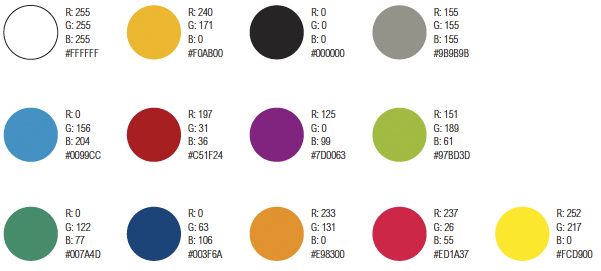
El nombre de neurones de l’entrada i la sortida, dependrà de perquè volem utilitzar la xarxa neuronal. Per exemple: si la volem utilitzar per agafar dos nombres, sumar-los i que ens retorni el resultat, aquesta xarxa tindrà dues entrades (els dos nombres que volem sumar) i una sortida (el resultat);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

1. **Funcionament de les XNA**

***Feedforward***

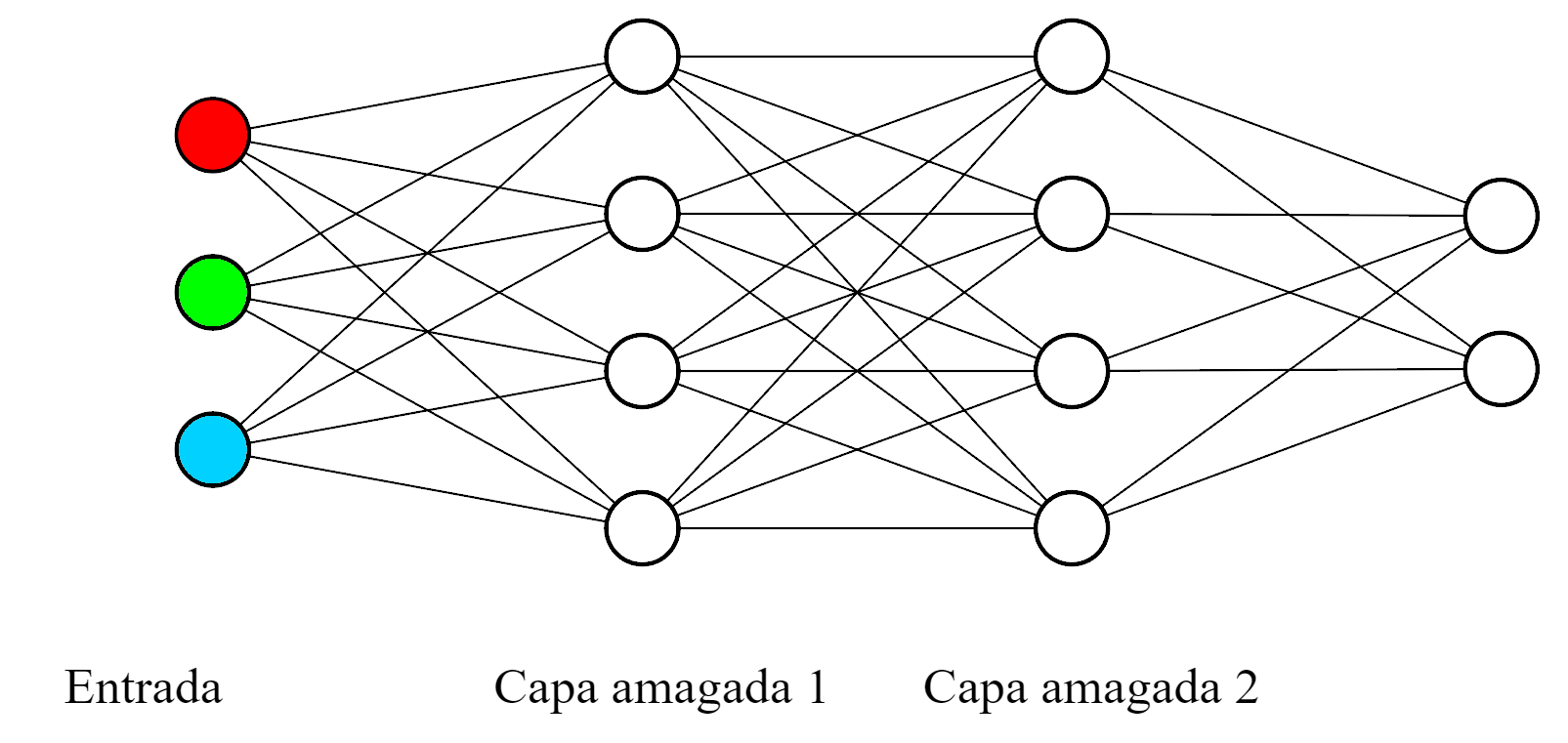
*Feedforward* és l’algoritme que utilitzarem perquè la nostra xarxa neuronal ens retorni un resultat amb unes dades que l’hi introduïm.

A aquesta xarxa neuronal artificial, l’hi introduirem un color en valor *rgb*, i ens dirà quina és la possibilitat de que aquell color sigui clar o sigui fosc. Per tant tindrà tres neurones d’entrada, la primera per la quantitat de vermell del color, la segona per la quantitat de verd i la tercera per la quantitat de blau.



*En aquesta imatge es poden apreciar diferents colors amb el seu valor rgb, cada un dels valors r, g o b té un número d’entre 0 i 255; i els tres combinats poden formar tots els colors de l’espectre visible.*

De neurones de sortida, en tindrà dues; la primera representarà la possibilitat de 0 a 1 de que el color sigui clar, i la segona representarà la possibilitat de que el color sigui fosc. La suma de les dues possibilitats sempre donarà 1. En la capa amagada, tindrà dues columnes de quatre neurones cada una. Sabent ja el nombre de columnes i de neurones a cada una d’elles, podem fer el diagrama, que ens ajudarà a entendre millor el seu funcionament.

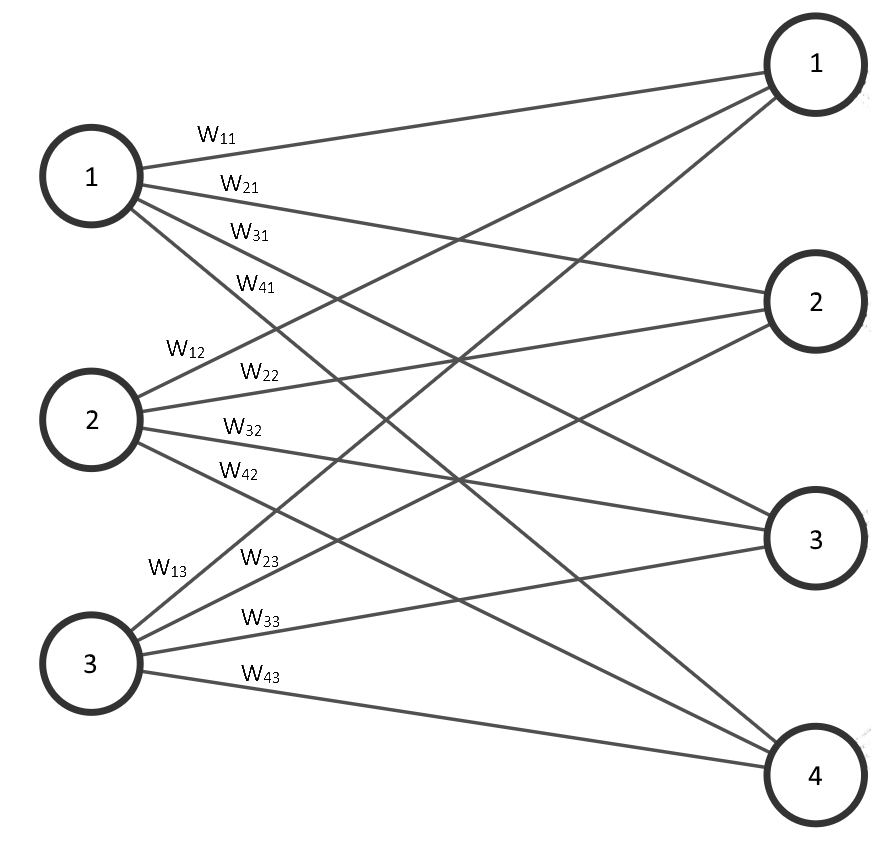


Ara que ja tenim definida l’estructura de la nostra xarxa neuronal, hem de tenir clar com emmagatzemarem les dades, tant les d’entrada com els valors de cada un dels pesos, per facilitar i optimitzar el funcionament.

Per emmagatzemar les dades, utilitzarem matrius. En total, per aquesta xarxa neuronal, utilitzarem 7 matrius. De moment ens fixarem en les dues primeres. La primera emmagatzemarà el valor de les senyals d’entrada, i la segona emmagatzemarà el valor de les connexions de la primera columna a la segona. Per organitzar-ho millor, donarem números a les neurones de cada columna i als pesos de la següent manera:

* Les neurones s’enumeraran per columnes, la primera neurona de cada columna serà la nº 1, la de sota serà la nº 2 i així fins a arribar al final de la columna.
* Cada un dels pesos tindrà dos subíndex: wab, el primer farà referència al nombre de neurona al que va i el segon farà referència al nombre de neurona del que ve.

Seguint aquestes normes d’enumeració, les dues primeres columnes de la xarxa neuronal haurien de quedar numerades d’aquesta manera:



I ara que ja ho tenim tot enumerat, podem crear les matrius. La primera, la de les senyals d’entrada, quedaria de la següent manera: . I la matriu dels pesos de la següent manera: , seguint la norma de que el primer número del subíndex es refereix a la fila de la matriu i el segon nombre a la columna.

Ara que ja tenim les matrius preparades, ja podem introduir-li dades a la xarxa i fer que ens torni un resultat. Per provar la nostra xarxa neuronal, farem que ens digui si un color groguenc, de valors r: 252, g:217 i b: 0 (veure a la imatge nº jo), és clar o fosc. Aleshores, tenim que les tres dades d’entrada són: 252, 217 i 0. Perquè la nostra xarxa neuronal treballi bé, primer de tot farem que normalitzi els valors, és a dir, que els passi a un rang d’entre 0 i 1; i això ho farem dividint els tres valors d’entrada entre 255. En aquest punt, ja tenim la matriu de les senyals d’entrada omplerta, amb els següents valors: .

A la matriu dels pesos, definirem els valors de manera aleatòria. I d’aquesta manera, ens queda així: .

Ara ja podem fer el mateix que vam fer amb el perceptró: primer de tot, hem de calcular un resultat inicial, i després el passarem per una funció d’activació. Per calcular el resultat inicial, multiplicarem la matriu dels pesos per la matriu dels nombres d’entrada. Si ho hem fet tot bé, la matriu resultant, hauria de ser un vector de quatre nombres, que correspondran als resultats per cada una de les neurones de la segona columna.

x = = .

Ara que ja tenim un resultat inicial, l’haurem de passar per una funció d’activació. Aquesta funció d’activació, anomenada *sigmoid*, ens retornarà un valor entre 0 i 1 per qualsevol nombre que l’hi introduïm. La fórmula de la funció *sigmoid* és: , tenint en compte que z és el resultat inicial, i la seva gràfica és:



Ara hem de repetir la mateixa operació dos cops més: un per arribar a la segona columna de la capa oculta i un segon per arribar a les senyals de sortida. Cal tenir en compte que en les dues operacions, la matriu dels pesos varia respecte la primera i sempre té valors aleatoris.

Si recordem, teníem dues neurones a la capa de sortida, i volíem que cada neurona ens retornés una possibilitat del 0 al 1, tenint en compte que les dues possibilitats sumades havien de donar 1. Això ho farem canviant la funció d’activació, però només per aquesta operació. La funció d’activació que utilitzarem s’anomena *softmax.*

***Softmax function***

En la nostra xarxa neuronal, la capa de senyal de sortida ha calculat un resultat inicial dels valors que ha rebut de la capa anterior, i aquests resultats inicils són: . Ara, la funció d’activació *softmax* transformarà aquests resultats a probabilitats, fent que la suma dels dos sigui 0. Si mirem a la funció veiem que . El que aquesta funció fa, per cada un dels resultats, és elevar el nombre *e* al resultat i dividir-ho entre la suma del nombre *e* elevat a cada un dels resultats. Per tant, en el nostre cas quedaria: resultat 1 = = 0,96, resultat 2 = = 0,04.

***Backpropagation***

Per entrenar la nostra xarxa neuronal, utilitzarem un mètode anomenat: “Aprenentatge supervisat”.

1. **Aprenentatge supervisat**

El procés d’entrenament de l’aprenentatge supervisat, es divideix en dues parts: el càlcul d’error i la rectificació dels pesos.

1. **Càlcul d’error**

El càlcul d’error de les XNA és més complicat al del perceptró, ja que aquestes tenen moltes més connexions i neurones. Per entendre ben bé que està passant al entrenar la XNA, ho farem començant per l’última columna i acabant a la primera, ja que és la manera de treballar de l’algoritme.

Per tant, començarem per la columna de la sortida que, ja que té dues neurones, haurem de calcular l’error per cada una d’elles. Per fer-ho utilitzarem la mateixa fórmula que amb el perceptró:

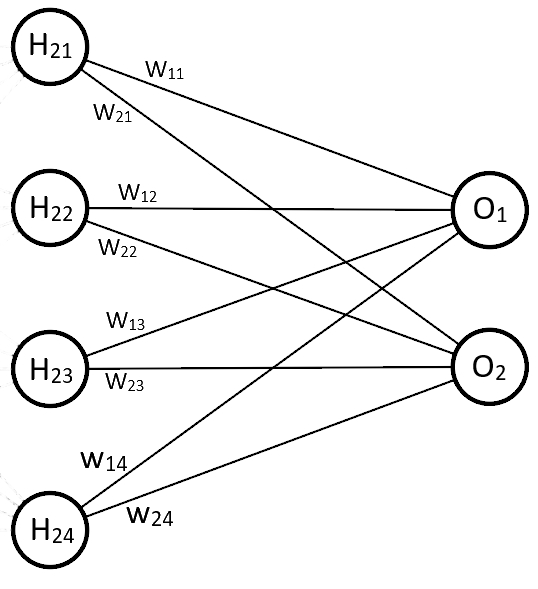
**error = resposta desitjada – resposta de la XNA**

Tenint això en compte, ho hem de calcular. Tant la resposta desitjada (que l’hi introduirem nosaltres) com la resposta de la XNA (que ha calculat anteriorment), són vectors amb dues dades. La resposta que ens ha calculat la XNA (a la qual anomenarem “*out*”) és *out* = i com que l’hi hem dit que calculi un color groguenc, el resultat hauria de ser , ja que es tracta d’un color clar. Per tant, per calcular l’error de les dues neurones de la capa de sortida, farem una simple resta de matrius:

eo = - = =

Hem anomenat aquest vector d’errors “eo” *e* referint-se a error i *o* referint-se a *output*, les senyals de sortida.

Ara que ja hem calculat l’error de les neurones de la capa de sortida, hem de calcular l’error de la columna anterior, la segona columna de la capa amagada:



El principal inconvenient de calcular l’error d’aquesta capa, és que cada una de les neurones té dues connexions, i cada connexió va a una neurona amb un error diferent; per tant, no podem utilitzar la mateixa fórmula que per la senyal de sortida.

En canvi, la fórmula que utilitzarem per calcular l’error de cada una de les neurones “H”, serà fer la suma dels pesos d’aquella neurona multiplicats per l’error de la neurona a la que van. Per tant l’error de la neurona H21, serà el pes w11 multiplicat per l’error e1 i el pes w21 multiplicat per l’error e2. Si ho mirem des d’un punt de vista matemàtic, podrem veure que s’assembla a una multiplicació de matrius, i les matrius que multipliquem són la dels pesos, transposada, i la de l’error de sortida.

w = , wT =

*eH = wT · eo = ·* =

La matriu resultant té quatre nombres, i cada un d’ells equival a l’error d’una de les neurones de la tercera columna. Per anomenar aquells errors, hem utilitzat la següent nomenclatura: eH2x, a on *e* es refereix a error, ja que són els valors de l’error; *H* es refereix a *hidden*, la capa oculta; el número 2 es refereix a que és l’error de la segona columna de la capa oculta i l’últim nombre, fa referència a la neurona.

Ara, hem de repetir aquest procés dos cops més: un per la primera columna de la capa oculta i una altre per la columna d’entrada. Quan haguem calculat cada un dels errors de cada neurona, haurem acabat amb quatre matrius d’errors, una per cada columna; i només quedarà variar els pesos.

(Altres tipus de xarxes neuronals artificials)

(Conceptes bàsics de programació)

(Aplicacions de les xarxes neuronals artificials)