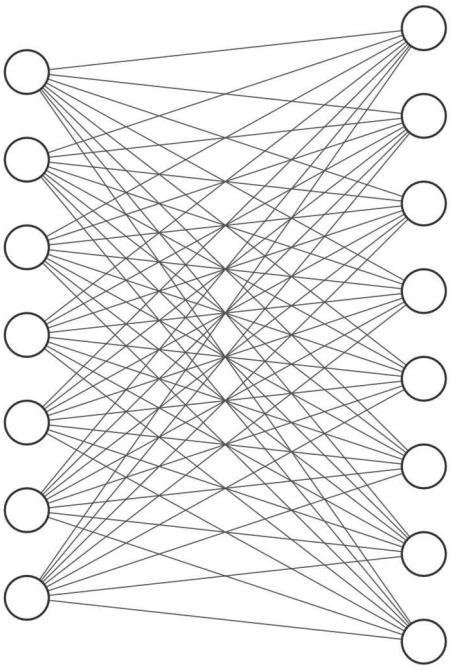
XARXES NEURONALS ARTIFICIALS



Joan Pau Condal Marco

Treball de Recerca

David Cambray Pla

2n Batxillerat J

Escola Pia Santa Anna Mataró, 2019

2.2. 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 aquest cas.

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. Un 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 indicar-li a un programa que pinti una zona d’un cert color. Per tant, si el que es vol és 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.

2.3. 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 un senyal diferent depenent de les entrades, i aquí veurem les més típiques:

2.3.1. NOT

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

INPUT	OUTPUT
A	NOT A
0	1
1	0

Il·lustració 3
Taula de veritat de la porta lògica NOT

2.3.2. AND

La porta lògica *AND*, requereix com a mínim dos 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

Il·lustració 4
Taula de veritat de la porta lògica AND

2.3.3. OR

La porta lògica *OR,* també necessita dos senyals d'entrada, i en pot tenir tants com calgui. Aquesta porta lògica tornarà 1 sempre que només un dels senyals d'entrada sigui 1, i només tornarà 0 si tots els senyals d'entrada són 0.

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

Il·lustració 5
Taula de veritat de la porta lògica OR

3. 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. 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.

3.1. 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 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 temps més curt.

3.2. 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.

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.

4.3. Usos del perceptró

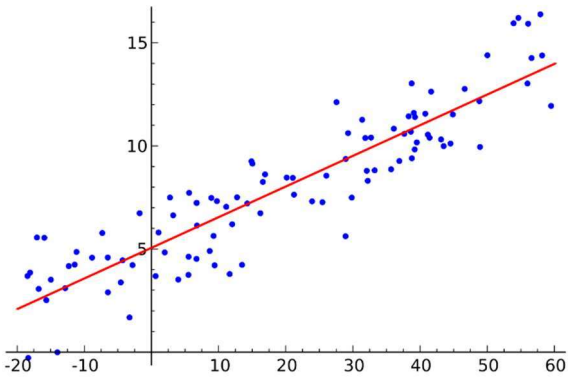
Ara que ja hem vist com funciona un perceptró, des de que introduïm dades fins que l'entrenem, cal veure les seves principals aplicacions, les quals són prediccions i classificacions.

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**

4.3.1. 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.



Il·lustració 9
Exemple gràfic de un model de 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. A l'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.

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

En quant 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:

$$\boldsymbol{z = w \cdot x}$$

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 (il·lustració 15) 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 la columna de sortida.

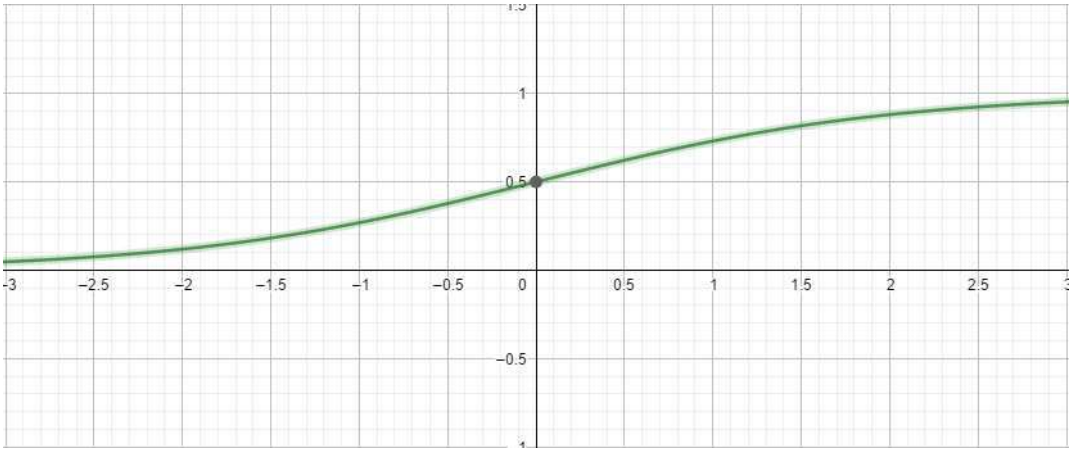
Hem de tenir en compte que aquests resultats poden no estar normalitzats, a causa de la multiplicació i la suma; i com hem dit abans, perquè la xarxa neuronal funcioni correctament, és millor que treballi amb valors normalitzats. Per normalitzar els valors, tenim una sèrie de funcions que ens estalvien 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 l'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 d'aspectes de la funció anterior. El primer 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. El segon aspecte 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, representada, té la següent forma:



Il·lustració 16
Representació de la fòrmula

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

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

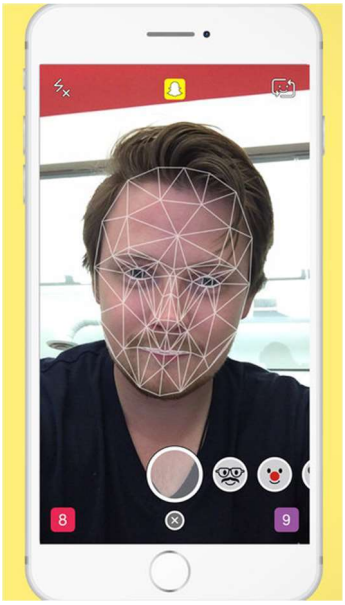
Fet això, considerarem la matriu obtinguda el resultat de la operació, i per tant, significa que en aquest punt la columna ja ha acabat de fer càlculs i pot passar aquestes dades

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 serien les matrius. No hi ha cap llenguatge de programació que de base pugui fer càlculs de matrius, però es pot descarregar una llibreria d'internet que et permeti fer operacions de matrius en el teu projecte.

7. Usos de les xarxes neuronals

Actualment, les xarxes neuronals són molt utilitzades a tot arreu, però a on més destaquen són en aquelles feines que requereixen d'interpretació com reconeixement d'imatges o de veu. En aquest apartat veurem on ens podem trobar les xarxes neuronals durant el dia.

Aplicacions com poden ser *Instagram* o *Snapchat*, i també el *Face ID* d'*Apple*, utilitzen xarxes neuronals per detectar cares a través de la càmera frontal del mòbil. Les xarxes socials utilitzen l'eina de detecció de cares per posar unes màscares en moviment, mentre que *Apple* detecta la cara per comprovar si és el propietari del telèfon i desbloquejar-lo.



Il·lustració 17
Reconeixement de cares mitjançant una xarxa neuronal a la aplicació Snapchat

Il·lustració 18
Apple utilitza el reconeixement de cares

En el mateix àmbit del reconeixement d'imatges, un ús a destacar és el del pilot automàtic. L'empresa de cotxes elèctrics *Tesla* fa ús de xarxes neuronals per guiar el pilot automàtic gràcies a la informació de sensors de ràdio i càmeres situats entorn el cotxe.

Després del reconeixement d'imatges, un àmbit en què les xarxes són molt utilitzades és el reconeixement de veu. Assistents personals com *Google Assistant* o la *Siri* fan ús de les xarxes neuronals per:

- Transcriure la veu de l'usuari a text
- Entendre la petició de l'usuari
- Imitar la veu humana per donar una resposta

També són usades per funcionar en borsa, veient què fan les empreses i calculant si surt a compte o no comprar les seves accions; en l'industria aeronàutica, per calcular la trajectòria que seguiran els coets i reaccionar si es detecta cap problema; o en la navegació per internet, al recomanar publicitat específica per un usuari, entre d'altres.

8. 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 amb el 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 premut, 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 què 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: 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 és que estaria directament negant la meva hipòtesi, 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 a 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 exemple de YouTube, fet per Daniel Shiffman.

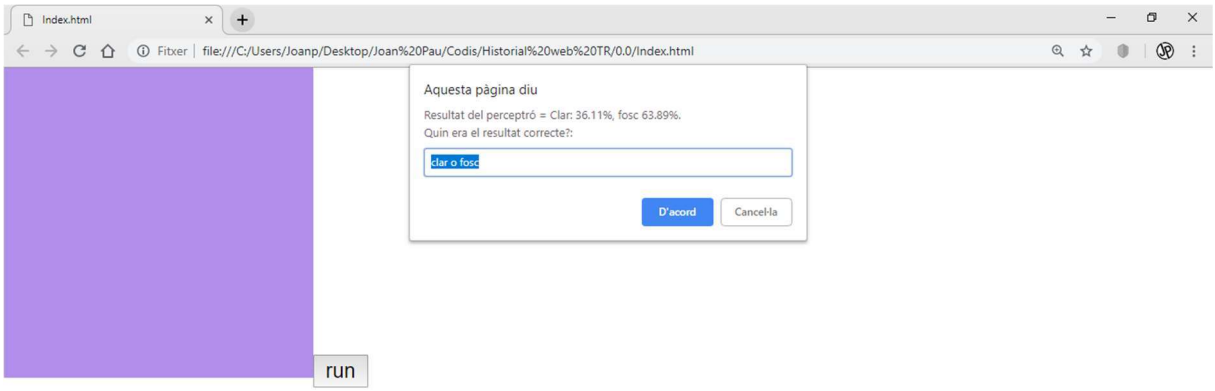
A l'acabar aquest exemple tenia un programa que em permetia operar i crear matrius i que sabia com funcionava exactament i quines funcions tenia. Aquest petit avantatge em va ser molt útil ja que em va fer més fàcil la detecció d'errors en el futur i la implementació de noves funcions en cas de necessitat.

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. Un aspecte 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 aleatòria 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.

9. Backpropagation

Ara que ja 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, aparegués una notificació que preguntés per una resposta a l'usuari.

En quant 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:



Il·lustració 20
Disseny inicial de la pàgina web de la xarxa

Aquest algoritme, el vaig acabar més endavant, quan ja havia fet suficient recerca com per poder programar-lo.

BIBLIOGRAFIA

1. Imatges:

- **Il·lustració 1:** Michael Nielsen (2018), Xarxa neuronal de reconeixement d'imatges amb dígits: http://neuralnetworksanddeeplearning.com/chap1.html
- **Il·lustració 2:** Michael Nielsen (2018), Estructura de la xarxa neuronal (editada): http://neuralnetworksanddeeplearning.com/chap1.html
- **Il·lustració 3, 4, 5:** *Wikipedia, Logic gates*: https://en.wikipedia.org/wiki/Logic_gate#Symbols
- **Il·lustració 6, 19:** Joan Pau Condal Marco (2018), NN SVG (creat): http://alexlenail.me/NN-SVG/index.html
- **Il·lustració 7:** Conner DiPaolo (2016), *Perceptron* (editada): https://www.google.com/search?q=perceptron&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiQtejbw6ffAhXIsaQKHV5pCggQ_AUIECgD&biw=1366&bih=626#imgrc=fODsmeFK93WHRM:, https://github.com/cdipaolo/goml/tree/master/perceptron
- **Il·lustració 8, 16:** Joan Pau Condal Marco (2018), Captura de pantalla (Geogebra): https://www.geogebra.org/graphing?lang=es
- **Il·lustració 9:** *Wikipèdia* User: Sewaqu, *Linear regression*: https://en.wikipedia.org/wiki/Linear_regression#/media/File:Linear_regression.svg
- **Il·lustració 10:** *mlpy Developers*: http://mlpy.sourceforge.net/docs/3.2/lin_class.html
- **Il·lustració 11, 12:** Joan Pau Condal Marco, creada amb *Paint.net*: https://www.getpaint.net/
- **Il·lustració 13:** Cerca de *Google*, *bayesian neural network*: https://www.google.com/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjXwqqajtXfAhVO0RoKHWuODEcQjRx6BAgBEAU&url=https%3A%2F%2Fpymc3-testing.readthedocs.io%2Fen%2Frtd-docs%2Fnotebooks%2Fbayesian_neural_network_opvi-advi.html&psig=AOvVaw1y030Hkf37Rg0tMW7io8Po&ust=1546725112523085
- **Il·lustració 14:** Diagrama d'una xarxa neuronal, *Research Gate*: https://www.researchgate.net/figure/Diagram-of-an-artificial-neural-network_fig1_329096802
- **Il·lustració 15:** Joan Pau Condal Marco (2018), NN SVG (creat i editat): http://alexlenail.me/NN-SVG/index.html
- **Il·lustració 17:** Cerca de *Google*, reconeixement de cares *Snapchat*: https://www.google.com/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiktuav0tTfAhXSyIUKHeMRAQkQjRx6BAgBEAU&url=https%3A%2F%2Fwww.thrillist.com%2Ftech%2Fnation%2Fsnapchat-facial-recognition-conspiracy-theory&psig=AOvVaw0Z1TMqhlVBvNe159qr1Neg&ust=1546709052524468
- **Il·lustració 18:** Cerca de *Google, Face ID*: https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwj496XK09TfAhUH1xoKHSpbCTMQjRx6BAgBEAU&url=https%3A%2F%2Fwww.scribd.com%2Farticle%2F372083126%2FQ-A-How-Apple-S-Face-Id-Facial-Recognition-Works&psig=AOvVaw113Kv-dcbBAI473Ey0hdrc&ust=1546709325130987

- **Il·lustració 20, 21, 22, 23:** Treball de recerca, Joan Pau Condal Marco (captura de pantalla): https://joanp131.github.io/Llibreria-xarxes-neuronals/

2. Webgrafia:

2.1.1 *Coursera*:

- *Neural networks and deep learning*, deeplearning.ai: https://www.coursera.org/learn/neural-networks-deep-learning
- *Machine Learning, Stanford university*: https://www.coursera.org/learn/machine-learning

2.1.2 YouTube:

- El algoritmo de YouTube YA NO EXISTE | Redes Neuronales, QuantumFracture: https://youtu.be/JBZx03342eM
- *My first machine learning game*, Jabrils: https://www.youtube.com/playlist?list=PL0nQ4vmdWaA0mzW4zPffYnaRzzO7ZqDZ0
- *Neural Networks*, 3blue1brown: https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
- *Essence of calculus*, 3blue1brown: https://www.youtube.com/playlist?list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr
- *Neural networks - The nature of code*, Daniel Shiffman (*The coding train*): https://www.youtube.com/playlist?list=PLRqwX-V7Uu6aCibgK1PTWWu9by6XFdCfh
- TensorFlow.js - *Intelligence and Learning*, Daniel Shiffman (*The Coding Train*): https://www.youtube.com/playlist?list=PLRqwX-V7Uu6YIeVA3dNxbR9PYj4wV31oQ
- TensorFlow.js - *Color classifier*, Daniel Shiffman (*The Coding Train*): https://www.youtube.com/playlist?list=PLRqwX-V7Uu6bmMRCIoTi72aNWHo7epX4L

2.1.3 *Wikipèdia*:

- Artificial Neural Network, *Wikipedia*, consultat el mes de juliol: https://en.wikipedia.org/wiki/Artificial_neural_network
- Linear classifier, *Wikipèdia*, consultat el mes de juliol: https://en.wikipedia.org/wiki/Linear_classifier
- Linear regression, *wikipèdia*, consultat el mes de juliol: https://en.wikipedia.org/wiki/Linear_regression
- Perceptron, *Wikipèdia*, consultat el mes de juliol: https://en.wikipedia.org/wiki/Perceptron
- High Threshold logic, *Wikipèdia*, consultat el mes de juliol: https://en.wikipedia.org/wiki/High_Threshold_Logic

- *Types of artificial neural network*, *Wikipèdia*, consultat el mes de juliol: https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks
- *Hebbian Theory*, *Wikipedia*, consultat el mes de juliol: https://en.wikipedia.org/wiki/Hebbian_theory
- *Brain*, *Wikipedia*, consultat el mes de juliol: https://en.wikipedia.org/wiki/Brain
- *Sigmoid function*, *Wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/Sigmoid_function
- *Matrix (mathematics*), *Wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/Matrix_(mathematics)
- *Heaviside step function*, *Wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/Heaviside_step_function
- *Supervised learning*, *wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/Supervised_learning
- *Backpropagation*, *Wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/Backpropagation
- *Multilayer perceptron*, *Wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/Multilayer_perceptron
- *History of artificial intelligence*, *Wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/History_of_artificial_intelligence
- *AI winter*, *Wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/AI_winter
- *Unsupervised learning*, *Wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/Unsupervised_learning
- *Deep learning*, *Wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/Deep_learning
- *Google Assistant, Wikipèdia*, consultat el mes d'agost: https://en.wikipedia.org/wiki/Google_Assistant

2.1.4 W3schools:

- *How to create a collapsible*, consultat el mes d'octubre: https://www.w3schools.com/howto/howto_js_collapsible.asp
- *On Scroll header*, consultat el mes d'octubre: https://www.w3schools.com/howto/howto_js_sticky_header.asp

2.1.5 Altres:

- TensorFlow.js, *Google*, consultat el mes de juliol: https://js.tensorflow.org/
- *Neural networks and deep learning*, consultat el mes de juliol: http://neuralnetworksanddeeplearning.com/
- *NN.svg*, consultat el mes d'agost: http://alexlenail.me/NN-SVG/index.html
- GitHub, consultat el mes de juliol: https://github.com/
- *StackOverflow*, consultat el mes de juliol: https://stackoverflow.com
- *Nvidia*, consultat el mes d'agost: https://www.nvidia.com/en-us/research/machine-learning-artificial-intelligence/
- Siri, google search, consultat el mes d'agost: https://www.google.es/search?q=siri&oq=siri&aqs=chrome..69i57j0j69i59j0l2j69i60.620j0j4&sourceid=chrome&ie=UTF-8

- *Google Assistant*, Google, consultat el mes d'agost: https://assistant.google.com/
- Neural Networks, consultat el mes d'agost: http://ml4a.github.io/ml4a/neural_networks/
- *Loss function*, consultat el mes d'octubre: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html