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**

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 dígits 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 . 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:

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:

I el càlcul quedaria de la següent manera:

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:

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:



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*:

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:

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.

1. **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:

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

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

*e* = −(y · log(p) + (1−y) · 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.

1. **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:

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:

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:

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

L’últim terme de la fórmula és IT. 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 finament 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.