UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

L'ALGORISME DE RETROPROPAGACIÓ D'ERRORS DES D'UN PUNT DE VISTA MATEMÀTIC

Autor: Marina Mallol Blay

Director: Dr. Miquel Bosch Gual
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 20 de juny de 2021

Abstract

In this project, we are going to study the backpropagation algorithm from a mathematical perspective and analize why this algorithm, which was forgotten in the past, is now used in supervised learning in order to train artificial neural networks.

First of all we will talk about how artificial neural networks are organized and how they work. Next, we will study the gradient descend method, the Newton's method, and other methods, used in learning. Finally, we will see how the backpropagation algorithm works and the calculations derived from it.

Resum

En aquest treball, estudiarem l'algorisme de retropropagació d'errors des d'un punt de vista matemàtic i analitzarem per què aquest algorisme, que en un passat va caure en l'oblit, s'usa ara en l'aprenentatge supervisat per entrenar xarxes neuronals artificials.

Primer de tot parlarem de com s'organitzen i com funcionen les xarxes neuronals artificials. A continuació, farem un estudi del mètode del gradient descendent, del mètode de Newton, i altres mètodes usats en l'aprenentatge de les xarxes. Finalment, veurem com funciona l'algorisme de retropropagació i els càlculs que se'n deriven.

2020 Mathematics Subject Classification. 68T07, 68T20

Agraïments

Vull agrair al Dr. Miquel Bosch, professor titular al Departament de Matemàtiques i Informàtica l'orientació, seguiment i supervisió d'aquest treball, així com per tenir sempre disponibilitat per qualsevol qüestió, tot i aquests difícils i estranys temps de pandèmia.

També vull donar el meu agraïment etern als meus pares i germans, per donar-me suport al llarg d'aquests quatre anys de grau, per l'empenta en els moments més durs i celebracions en les victòries. Mai us ho podré agrair prou.

Índex

1 Introducció

Si Alan Turing pogués veure l'enorme revolució que hi ha hagut al món després de la màquina Turing que ell mateix va idear, no sabem si es sorprendria, doncs ja al 1950 defensava que la computació podria imitar el pensament humà. Tampoc sabem si es sorprendrien John McCarthy, Marvin Minsky i Claude Shannon, tres científics destacats de l'època que al 1956 van donar la definició d'intel·ligència artificial: *la ciència i l'enginy de fer màquines intel·ligents, especialment programes de càlcul intel·ligents*, i que van predir que la societat no tardaria en viure rodejada de màquines.

No obstant, a la seva època es van topar amb diverses barreres per poder desenvolupar la intel·ligència artificial. Per començar, els ordinadors no eren prou eficients; tot i que semblin molt les 40 mil operacions per segon que eren capaces de fer els ordinadors als anys 50, en l'actualitat en poden fer 100 milions per segon. D'altra banda, la capacitat d'emmagatzament de dades era pràcticament nul·la, tot en paper, mentre que ara en dos centímetres cúbics tenim un USB de 128 Gigabytes. A més, la tecnologia no era gens econòmica.

Així doncs, el creixement de la intel·ligència artificial ha sigut gradual. De fet, els avenços en aquesta disciplina sempre han anat de la mà dels avenços en les tecnologies TIC (Tecnologia de la Informació i de les Comunicacions). Un dels esdeveniments més destacats i importants va ser al 1997 quan l'ordinador Deep Blue de IBM va aconseguir vèncer al millor jugador d'escacs del moment. Aquesta "simple"partida va fer veure a les empreses tecnològiques que la IA podria analitzar i processar una enorme quantitat de dades en un temps raonable.

Les quantitats d'aplicacions de la IA són immenses, tant en el món de l'oci i comercial, com en temes de salut, esport i publicitat. I, tot i l'eficàcia en aquesta disciplina, les exigències de la societat també s'han accelerat: un banquer vol estar segur, en menys d'un segon, que l'empremta dactilar d'un nou client no pertany ja al d'un altre client, de manera que suposaria un frau. Es vol comprar per internet a l'instant amb la màxima qualitat i major preu comparat.

Quan parlem d'intel·ligència artificial, solem associar aquest concepte a les xarxes neuronals artificials. Aquestes xarxes neuronals, a la pràctica, estan implementades a tot tipus d'aparells a la societat, com ara en assistents de veu com Siri d'Apple o Alexa d'Amazon. També en programes de reconeixement facial o classificadors d'objectes. O bé en aplicacions financeres, traducció automàtica, i un llarguíssim etcètera.

En aquest treball ens centrarem en fer un estudi d'aquestes xarxes. Veurem què són, com funcionen, quins paràmetres es veuen involucrats, com aprenen i s'entrenen i un dels seus algorismes més populars: la retropropagació d'errors.

1.1 Estructura de la Memòria

En aquest estudi de l'algorisme de retropropagació d'errors des d'un punt de vista matemàtic, començarem explicant què són les xarxes neuronals artificials i introduirem tots els conceptes que necessitarem per entendre-les. Dedicarem un apartat especial als nodes de les xarxes neuronals i finalment posarem d'exemple una xarxa capaç de reconèixer dígits.

A continuació parlarem de l'aprenentatge de les xares neuronals usant optimització basada en el gradient, però també comentarem altres mètodes que s'usen per tal que

aquest aprenentatge sigui el més ràpid possible.

Finalment, veurem com funciona l'algorisme de retropropagació d'errors. Explicarem el funcionament, entrant amb detall en totes les operacions involucrades i finalment, parlarem de les diverses funcions d'activació que s'utilitzen.

2 Xarxes neuronals artificials

En aquest apartat introduirem les xarxes neuronals artificials i alguns conceptes bàsics relacionats. També posarem l'exemple d'una xarxa neuronal capaç de reconèixer dígits on veurem la gran quantitat de paràmetres que s'involucren i on entendrem la magnitud del problema. Aquesta secció s'explica d'una manera senzilla, de manera que els conceptes més tècnics es deixaran per més tard.

El material que he consultat per redactar aquest apartat són [1], [4], [5] i [7] i es poden veure adjunts a la bibliografia d'aquest treball.

2.1 Què són les xarxes neuronals artificials?

Una **xarxa neuronal artificial** és un conjunt de neurones artificials que s'agrupen en capes connectades entre si i que es transmeten senyals. S'usen molt en camps on buscar solucions o característiques usant programació convencional resulta molt difícil, com ara visió per computador, reconeixement de veu, etc., doncs són sistemes que aprenen i es formen per si sols, enlloc de ser programats explícitament. Aquestes xarxes neuronals estan inspirades en les xarxes neuronals biològiques que constitueixen els cervells animals.

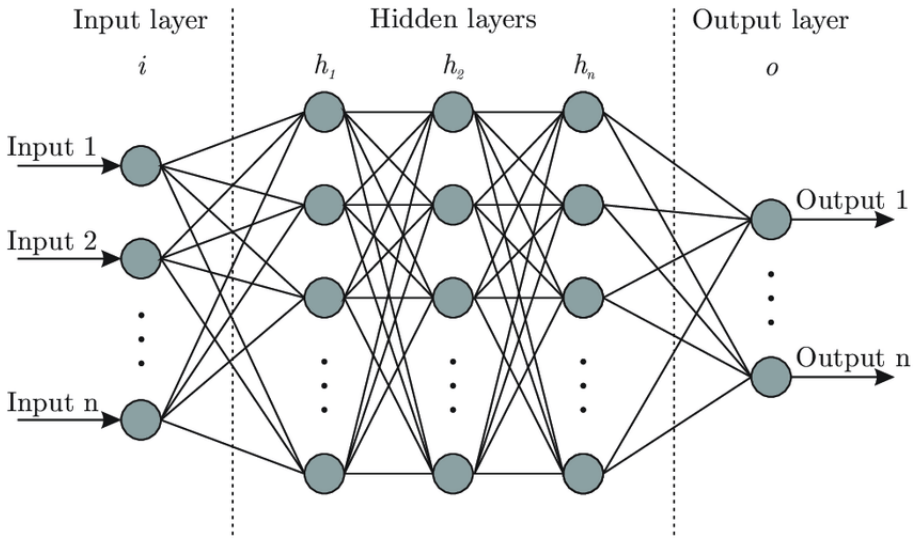

Figura 1: Estructura d'una xarxa neuronal artificial

Es distingeixen tres tipus de capes: d'entrada, ocultes i de sortida. Una **capa d'entrada** està formada per neurones que reben dades o senyals procedents de l'entorn. La **capa de sortida** es compon de neurones que proporcionen la resposta de la xarxa neuronal. I la **capa oculta** no té una connexió directa amb l'entorn, és a dir, està formada per neurones que tenen entrades que venen de capes anteriors i sortides que passen a les neurones de capes posteriors.

Existeixen diverses varietats de xarxes neuronals segons la seva arquitectura. Considerant l'estructura podem parlar de **xarxes monocapa**, compostes per una única capa de neurones, o **xarxes multicapa**, compostes per varies capes. Tenint en compte el flux de dades, podem distingir entre **xarxes unidireccionals** (feedforward en anglès) i **xarxes**

recurrents o realimentades (feedback en anglès). Mentre que a les unidireccionals la informació circula en un únic sentit, en les xarxes recurrents o realimentades, la informació pot circular entre les diverses capes de neurones en qualsevol sentit, inclús del revés.

També distingim les capes **completament connectades** (fully-connected layer en anglès), on les neurones d'una mateixa capa no estan connectades entre elles però si amb totes les neurones de les capes anteriors i posteriors.

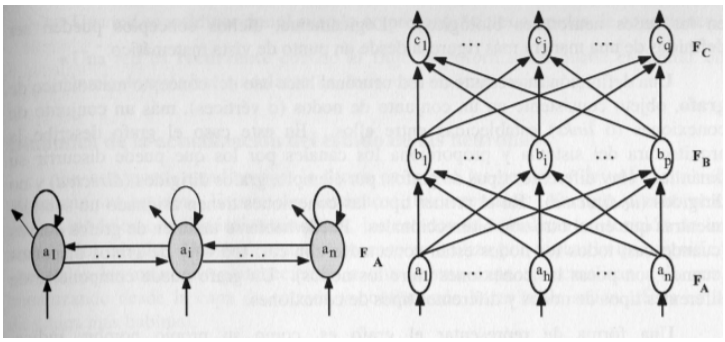

Figura 2: La figura de l'esquerra correspon a una xarxa neuronal monocapa i realimentada, mentre que la figura de la dreta és una xarxa neuronal multicapa i unidireccional.

En aquest treball farem un estudi de les xarxes neuronals multicapa i unidireccionals, amb capes completament connectades, que utilitzen l'algorisme de retropropagació d'errors (backpropagation en anglès) per entrenar.

2.2 Els nodes

Cada neurona de la primera capa oculta estarà connectada amb totes les neurones de la capa d'entrada, i cada connexió tindrà un **pes** associat, w_{ji}^k (pes del node i a la capa k per l'entrada del node j de la capa $k-1$), que és senzillament un número positiu o negatiu. Aquestes connexions són direccionals, és a dir, la informació només es pot propagar en un únic sentit. Els valors de les neurones d'entrada multiplicats pels pesos associats determinen l'impuls nerviós que rep la neurona següent. Aquest valor, es processa a l'interior de la neurona mitjançant una funció anomenada **activació**, a_i^k (activació de la neurona i a la capa k), que retorna un valor que s'envia com una sortida. Queda doncs:

$$a_i^k = \sum_{j=1}^{r_{k-1}} w_{ji}^k o_j^{k-1}$$

on o_j^{k-1} és el valor de la sortida del node j a la capa $k-1$, és a dir, el valor d'activació provinent de la neurona j.

L'activació de la neurona és bàsicament una mesura de com de positiva és la suma ponderada. Cal tenir en compte però que a vegades volem que la neurona s'encengui només quan la suma ponderada és major que un cert nombre, és a dir, volem que hi hagi cert marge perquè resti inactiva. Per tant, afegirem un altre valor a aquesta suma

ponderada. Aquest número addicional s'anomena **biaix**, b_i^k (biaix del node i a la capa k).

$$a_i^k = b_i^k + \sum_{j=1}^{r_{k-1}} w_{ji}^k o_j^{k-1}.$$

Les activacions d'una capa determinen les activacions de la següent capa, és a dir, certs grups de neurones que s'activen fan que altres s'activin. A més, s'acostuma a voler valors d'activació que estiguin en el l'interval $[0, 1]$, per tant, haurem de comprimir la línia real $\mathbb{R}$ per posar-la en aquest rang. Per aquest pas, podem usar la funció sigmoide $\sigma(t) = \frac{1}{1+e^{-t}}$, de manera que números molt negatius passen a ser propers a 0, mentre que números molt positius queden propers a 1,

$$a_i^k = \sigma(b_i^k + \sum_{j=1}^{r_{k-1}} w_{ji}^k o_j^{k-1}).$$

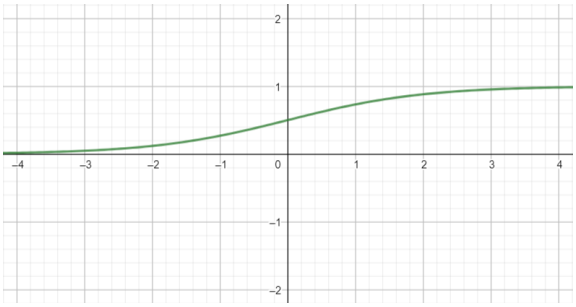

Figura 3: Funció sigmoide

Suposem que a la capa $k = 1$ hi ha n nodes, mentre que a la capa $k - 1$ n'hi ha m, en forma matricial queda:

$$\begin{pmatrix} a_1^1 \\ a_2^1 \\ \vdots \\ a_n^1 \end{pmatrix} = \sigma\Big(\begin{pmatrix} b_1^1 \\ b_2^1 \\ \vdots \\ b_n^1 \end{pmatrix} + \begin{pmatrix} w_{1,1}^1 & w_{1,2}^1 & \cdots & w_{1,n}^1 \\ w_{2,1}^1 & w_{2,2}^1 & \cdots & w_{2,n}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1}^1 & w_{m,2}^1 & \cdots & w_{m,n}^1 \end{pmatrix} \begin{pmatrix} o_1^0 \\ o_2^0 \\ \vdots \\ o_m^0 \end{pmatrix} \Big) \tag{2.1}$$

Cal mencionar també que les primeres xarxes neuronals utilitzaven aquesta funció sigmoide per reduir la suma al rang $[0, 1]$, però actualment es fa més ús de la funció $ReLu = max(0, a)$. Amb aquesta funció els entrenaments són més senzills i les xarxes neuronals profundes funcionen millor. Tot i així, al llarg d'aquest treball continuarem usant les sigmoide per simplificar el procés.

En resum, les neurones prendran valors d'entrada de totes les neurones de les capes anteriors i cada una traurà un valor entre 0 i 1. A més, tota connexió tindrà un pes i un biaix associat.

D'altra banda, quan parlem d'aprenentatge, ens referim a aconseguir que la xarxa neuronal trobi els valors adequats per a tots aquests pesos i biaixos, de manera que minimitzi

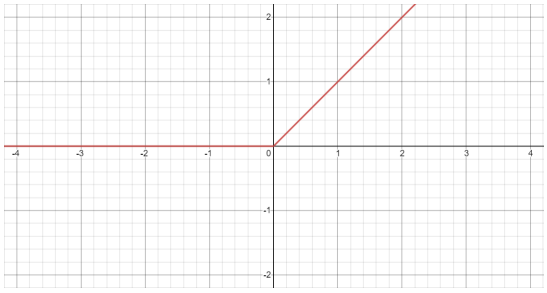

Figura 4: Funció ReLu

la funció. En els capítols que es presenten a continuació farem un estudi matemàtic de l'ús de la retropropagació d'errors a les xarxes neuronals com a algorisme d'aprenentatge.

Aquestes xarxes neuronals artificials multicapa són capaces de calcular una gamma més àmplia de funcions que les xarxes compostes per una única capa d'elements. Però l'esforç computacional necessari per trobar la combinació correcta de pesos incrementa quan es consideren més paràmetres i topologies més complexes.

2.3 Una xarxa neuronal capaç de reconèixer dígits

Abans de començar aquesta secció fem un petit aclariment. És molt important que anem en compte quan idealitzem quelcom, per no eliminar-ne aspectes bàsics. A vegades resulta útil entendre conceptes amb detalls falsos, encara que sapiguem que no és la realitat. Per exemple, pensar que hi ha neurones que es passen nombres naturals en comptes de pics d'energia.

Per entendre el funcionament de les xarxes neuronals artificials posem per exemple una xarxa capaç de reconèixer dígits.

Imaginem que tenim una imatge en blanc i negre d'un número, entre el 0 i el 9, de 28×28 píxels, que són 784 en total. Cada un d'aquests píxels té un número que representa el valor de l'escala de grisos del píxel corresponent, que oscil·la entre 0 per als píxels negres i 1 per als blancs. Ara, alimentem la capa d'entrada de la xarxa amb aquesta imatge, on el número de cada píxel anirà dins d'una neurona de la xarxa i en serà l'activació.

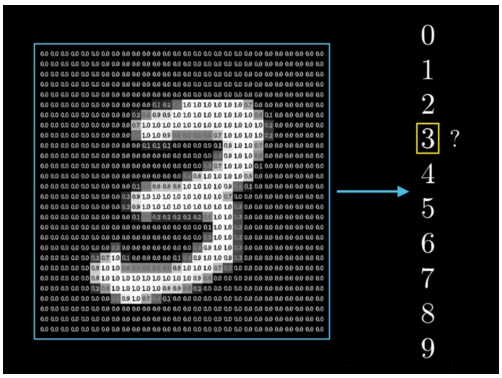

Figura 5: Imatge formada per 784 píxels que correspon al dígit 3

Així doncs, tenim 784 neurones que formen la capa d'entrada de la nostra xarxa. En canvi, la capa de sortida té 10 neurones, cadascuna de les quals representa un dels dígits del 0 al 9. L'activació en aquestes neurones, de nou, és un nombre que oscil·la entre el 0 i l'1, i representa amb quina seguretat la xarxa pensa que la imatge donada és tal dígit. També tenim un parell de capes ocultes entre la capa d'entrada i la capa de sortida. Recordem que les activacions en una capa determinen les activacions de la següent capa.

En resum, si alimentem la xarxa amb una imatge que il·lumina totes les neurones de la capa d'entrada d'acord amb la brillantor de cada píxel de la imatge, aquest patró d'activació provoca algun patró molt específic a la següent capa, que provoca algun patró a la següent i això, finalment, dona algun patró a la capa de sortida. La neurona més brillant d'aquesta capa de sortida és el resultat de la xarxa davant el dígit que representa la imatge.

Entrant en més detall, quan veiem un número, el nostre ull reconeix components diferents. Per exemple, al número 8 veiem dos bucles units, el superior i l'inferior. Per tant, podem pensar la nostra última capa oculta com neurones que representen subcomponents, per exemple, la neurona superior pot ser un bucle. És clar que aquesta afirmació no funciona així a la pràctica, i que explicant-ho d'aquesta manera resulta molt més aclaridor, però caldria esperar un comportament equivalent.

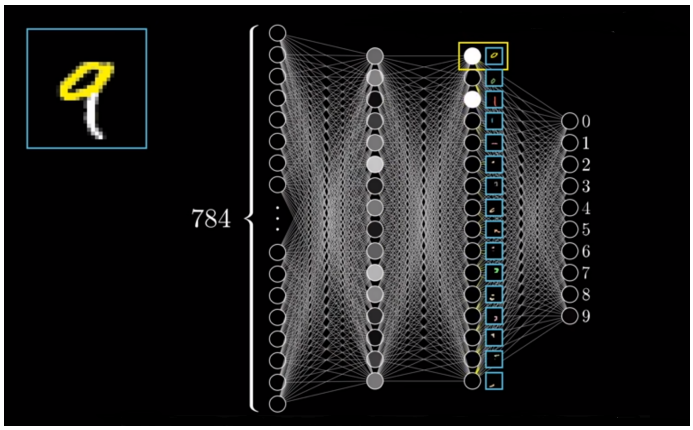

Figura 6: Bucle com a subcomponent a la tercera capa

Cada vegada que alimentem una imatge amb un bucle a la part superior (com un 8, un 9, ...) hi ha alguna neurona específica que tindrà l'activació propera a 1. D'aquesta manera, passar de la tercera capa a l'última requereix aprendre quina combinació de subcomponents correspon a cada un dels diferents dígits. Però aquesta tasca és igual de difícil.

Reconèixer un bucle també es pot dividir en subproblemes. Una manera raonable de fer-ho seria reconèixer primer diverses vores petites que formen aquest bucle.

De manera similar, si tenim una línia recta vertical (que es pot veure en dígits com 1, 4, 7) ho podem entendre com una vora llarga o com un patró format per diverses vores més petites, així podem pensar que cada neurona de la segona capa correspongui a diverses vores petites rellevants.

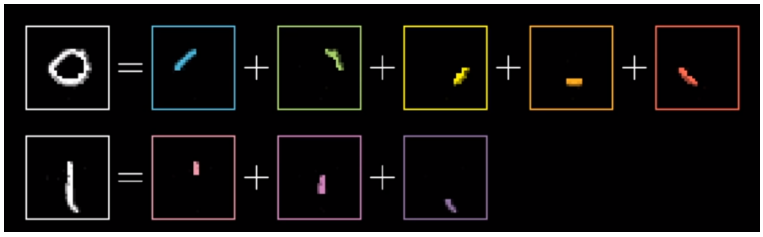

Figura 7: Representació de com es parteix en subcomponents

Així doncs, l'objectiu és tenir algun mecanisme que pugui assignar píxels a vores o vores a patrons o patrons a dígits. Per tant, quan parlem d'aprenentatge, ens referim a que l'ordinador trobi els valors dels paràmetres per a tots aquests nombres i així resolgui el problema.

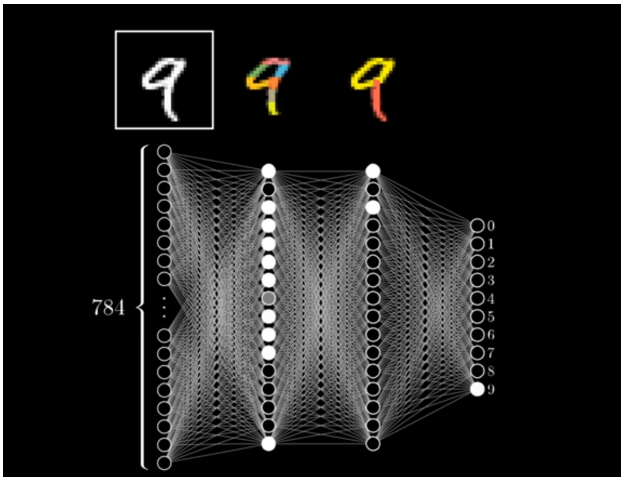

Figura 8: Representació total dels passos que segueix la xarxa per reconèixer els dígits, partint el número en subcomponents a cada capa

Així doncs, esperem que les neurones de la segona capa siguin capaces de captar si la imatge té una vora o no en una certa regió.

La pregunta que ens plantegem ara és quins paràmetres hauríem de tenir, quins nombres s'haurien de triar per tal que la vora sigui prou clara i així captar el patró, com ara el patró de vores que conjuntament formen un bucle. El que fem doncs és assignar un pes a cadascuna de les connexions entre la nostra neurona i les neurones de la primera capa. Després, prenem totes aquestes activacions de la primera capa i en fem la suma ponderada segons aquests pesos.

Recordem que podem tenir pesos negatius i positius, de manera que si volem il·luminar una vora, podem pensar en tenir una subcomponent positiva i les seves vores negatives, tal i com es mostra a la Figura 9.

Per tant, aquesta activació de la neurona acaba sent la mesura de com de positiva és la suma ponderada. Però, potser no volem que la neurona s'encengui quan la suma ponderada és més gran que zero, sinó més gran que un altre nombre. Aquí és doncs on

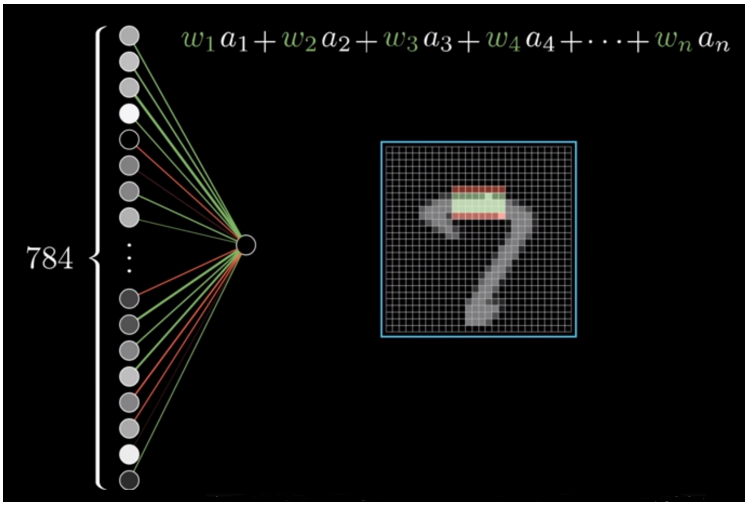

Figura 9: Els píxels verds correspondran a pesos positius, mentre que els vermells a negatius.

farem ús del biaix.

És clar doncs que els píxels centrals (on hi ha la vora, en la imatge anterior) són clars i els que els envolten són més foscos. Volem que els números d'activació calculats estiguin entre 0 i 1, per tant, ajustem la línia real usant la funció sigmoide.

Per entendre la quantitat de valors implicats, recordem que tenim les 784 neurones a la capa d'entrada i posem 16 neurones a la primera capa oculta. Tindrem 784×16 pesos i 16 biaixos, i seran només les connexions entre aquestes dues primeres capes. Doncs la resta de capes també tenen connexions amb pesos i biaixos associats. Si tenim una segona capa oculta amb 16 neurones i una capa de sortida de 10 neurones, acabarem tenint $784 \times 16 + 16 \times 16 + 16 \times 10$ pesos i $16 + 16 + 10$ biaixos. En total, 13002 pesos i biaixos, que poden ser tunejats i activats per tal que aquesta xarxa es comporti com esperem.

Realment, podem entendre tota la xarxa actuant com una funció. Pren 784 valors d'entrada i en treu 10 de sortida. És una funció molt complicada, que implica molts paràmetres com a pesos i biaixos, operacions i una funció sigmoide. I en certa manera és bo que sembli complicat ja que si fos senzill, quin tipus d'esperança tindríem que pogués assumir el repte de reconèixer dígits?

3 Aprenentage

L'aprenentatge d'una xarxa neuronal artificial podria interpretar-se com la recerca d'un mínim d'una funció f, anomenada **funció d'error** o **funció de cost**, en l'espai dels pesos. Cada combinació de pesos produeix una funció d'error que anem modificant i corregint iterativament, de manera que l'objectiu és minimitzar-la.

Esmentem breument la forma que prendrà la nostra funció d'error:

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2,$$

on θ és el conjunt de paràmetres, pesos i biaixos, X el conjunt de tuples d'entrada-sortida (x_i, y_i), $\hat{y}_i$ la sortida calculada per la xarxa i y_i la sortida desitjada. Aquesta funció s'explica amb més detall en l'apartat de l'algorisme de retropropagació d'errors, però és important mencionar-la d'entrada, ja que l'utilitzarem per veure com funcionen alguns mètodes d'aprenentatge.

L'algoritme d'aprenentatge d'una xarxa neuronal unidireccional és el cas general més difícil de treballar, ja que podem topar amb el cas d'optimitzar funcions d'error discontínues o que apareixin múltiples mínims locals en les superfícies. Però existeixen procediments numèrics eficaços per buscar mínims en funcions derivables.

Abans d'entrar en detall, anem a comentar un altre aspecte interessant en l'aprenentatge, que és l'existència de dos tipus: **l'aprenentatge supervisat** i **l'aprenentatge no supervisat**.

- En l'aprenentatge supervisat, la xarxa s'entrena amb dades d'entrada i amb dades de sortida associades. Per a realitzar aquest aprenentatge és necessari un conjunt de parelles de dades d'entrada-sortida per a ensenyar la xarxa. En primer lloc hi ha un període d'entrenament, on s'adapten els pesos de la xarxa en funció de les dades aportades minimitzant una funció d'error. Comparant la resposta de la xarxa amb la resposta desitjada es modifiquen els pesos de manera que minimitzin aquest error (per exemple mitjançant retropropagació). L'aprenentatge continua fins que l'error és menor a un valor establert o fins que és mínim.

- Per contrari, en l'aprenentatge no supervisat, no es proporcionen exemples del comportament desitjat, és a dir, dades, sinó que és la xarxa mateixa que adapta els pesos de connexió en funció de la correlació existent entre les activacions de les neurones involucrades en les connexions. Aquest tipus d'aprenentatge no s'estudiarà en aquest treball.

En aquest apartat he usat el material de la bibliografia corresponent a [1], [2], [5], [10], [11], [12], [13] i [17].

3.1 Optimització basada en el gradient

Així doncs, l'optimització consistirà en minimitzar (o maximitzar) una funció d'error o funció de cost $f(x)$ variant sistemàticament els valors d'entrada x d'un conjunt i calculant el valor de sortida de la funció. Normalment, buscarem optimitzar $f(x)$ en termes de minimitzar-la, però la maximització es pot aconseguir minimitzant $-f(x)$.

Fem doncs un breu repàs de càlcul elemental. Per introduir conceptes considerarem $f : \mathbb{R}^n \to \mathbb{R}$ i ens centrarem primer en el cas $n = 1$.

Suposem que tenim una funció $y = f(x)$, on x i y són nombres reals. La derivada d'aquesta funció $y = f(x)$ es pot escriure com $f\prime(x)$ o $\frac{dy}{dx}$ i té una interpretació geomètrica clara, que posa de manifest la relació amb el creixement de la funció. Dona el pendent de la recta tangent a la gràfica $f(x)$ a qualsevol punt $(a, f(a))$ on $a \in \mathbb{R}^n$. També ho podem interpretar com que especifica com escalar un petit canvi ξ, positiu o negatiu, en l'entrada per tal d'obtenir el canvi corresponent en la sortida:

$$f(x + \xi) \approx f(x) + \xi f'(x)$$

Per tant, la derivada és útil per minimitzar una funció perquè ens indica com canviar x per fer una petita millora en y. Podem així reduir $f(x)$ variant x iterativament amb signe oposat de la derivada. Aquesta tècnica s'anomena **gradient descendent** (Cauchy 1847) i l'utilitzem per minimitzar la funció d'error. Així doncs, podem entendre el gradient descendent com una forma d'anar fent petites passes cap a la direcció que indica el gradient.

A vegades ens trobarem el cas on $f'(x) = 0$. Aquests punts es coneixen amb el nom de **punts crítics** o **punts estacionaris** i la derivada no dona cap informació de quina direcció agafar. Tenim tres tipus de punts crítics o estacionaris: mínim, màxim o d'inflexió. I aquests dos primers poden ser locals o globals.

Definim ara els conceptes de mínim i màxim local:

Definició 3.1. *Una funció $f(x)$ té un **mínim local o relatiu** al punt a si existeix un entorn $I(a, \xi)$ tal que*

$$f(a) \leq f(x) \qquad \forall x \in I(a, \xi)$$

*Anàlogament es defineix **màxim local o relatiu**.*

Per tant, en un mínim local no és possible decrementar $f(x)$ fent infinitèsims passos. Igual que en un màxim local no és possible incrementar $f(x)$ fent infinitèssims passos.

Per mínim i màxim global:

Definició 3.2. *Sigui A un subconjunt dels reals. Una funció $f(x)$ té un **mínim global** al punt a si*

$$f(a) \leq f(x) \qquad \forall x \neq a, x \in A$$

*Anàlogament es defineix **màxim global**.*

Els punts que no són màxims ni mínims s'anomenen **punts de sella**, i són els punts sobre els quals l'elevació es màxima en una direcció, i mínima en la direcció perpendicular.

Figura 10: Punts crítics o estacionaris

L'optimització en el context de l'aprenentatge profund pot resultar molt difícil, ja que és possible trobar mínims locals que no siguin globalment òptims. De fet, s'optimitzen funcions que poden tenir diferents mínims locals no òptims, i varis punts de sella envoltats de zones molt planes. Així doncs, serà suficient trobar un valor de f molt petit, però no necessàriament mínim en el sentit formal.

També, ens podem trobar que l'entrada de la funció f sigui multidimensional. Per tant, minimitzarem funcions $f : \mathbb{R}^n \to \mathbb{R}$, amb múltiples entrades ($n > 1$) i un únic valor de sortida.

Per funcions amb múltiples entrades, farem ús de les derivades parcials.

Observació 3.3. Algunes consideracions:

- En lloc de $\frac{\partial f}{\partial x_j}(a)$ també escriurem $f_{x_j}(a)$, o bé $D_j f(a)$.
- Si una funció té derivades parcials en tot punt $x \in A$, utilitzarem la notació $\frac{\partial f}{\partial x_j}(x)$ per a designar la funció que envia cada $x \in A$ a la seva j-èssima derivada parcial en aquest punt.

Podem interpretar les derivades parcials $\frac{\partial}{\partial x_i} f(x)$ com que mesuren quant varia f quan només la component x_i varia en x. A més:

Definició 3.4. *Si existeixen totes les derivades parcials de f en un punt a, definim el* ***gradient*** *de f de la forma*

$$\bigtriangledown f(a) = Grad(f)(a) = \big(f_{x_1}(a), \ldots, f_{x_n}(a)\big).$$

De manera que el gradient generalitza la noció de derivada en el cas en què la derivada sigui respecte a un vector $(x_1, \ldots, x_n)$: el gradient de f és el vector que conté totes les derivades parcials de f, i on l'element j del gradient és la derivada parcial de f respecte x_j.

També usarem el concepte de derivada direccional en la direcció del vector unitari u, que és el pendent de la funció f en la direcció u:

Definició 3.5. *Sigui $f : A \subset \mathbb{R}^n \to \mathbb{R}$, a un punt de A, i u un vector unitari. Direm que f té* ***derivada direccional*** *respecte de u en el punt a, si existeix*

$$\lim_{t \to 0} \frac{f(a + tu) - f(a)}{t} = D_u f(a) = df_a(u).$$

Observem que per a tot $j = 1, \ldots, n$, es compleix $f_{x_j}(a) = (D_{e^j} f)(a)$.

En altres paraules, la derivada direccional és la derivada de la funció $f(x+tu)$ respecte t i avaluada a $t = 0$. Usant la regla de la cadena:

$$\frac{\partial}{\partial t} f(x + tu) = \bigtriangledown_x f(x) u = \partial_u f(x)$$

i usant derivades direccionals, trobarem la direcció en la qual f decreix més ràpidament, i per tant, ens permet minimitzar f :

$$min_{u,u^T u=1} \bigtriangledown_x f(x) u = min_{u,u^T u=1} \| \bigtriangledown_x f(x) \|_2 \|u\|_2 cos\theta$$

on θ és l'angle entre u i el vector gradient i $\|\cdot\|_2$ és la **norma usual** d'un vector $x = (x_1, \ldots, x_n)$ de $\mathbb{R}^n$, que es defineix mitjançant

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

Com que u és un vector unitari, tenim $\|u\|_2 = 1$ (que és equivalent a escriure $u^T u = 1$) i si ignorem factors que no depenen de u, queda $min_u cos\theta$. Això es minimitza quan u apunta cap a la direcció oposada del gradient. El gradient negatiu apunta directament cap al mínim, de manera que podem decrementar f. Així doncs, de nou, estem aplicant el mètode del **gradient descendent**.

El gradient descendent proposa un nou punt:

$$x* = x - \xi \bigtriangledown_x f(x)$$

on ξ s'anomena taxa d'aprenentatge (learning rate en anglès).

La **taxa d'aprenentatge** és un paràmetre molt comunament usat en les xarxes neuronals, i determina la mida del pas a fer a cada iteració mentre es mou cap al mínim de funció f. Té un valor positiu, sovint entre 0 i 1. Com més baix sigui el valor, més lent anirem pel pendent descendent, i tot i que pot semblar una bona idea per assegurar que no esquivem cap mínim local, també podria significar que triguem molt a convergir, sobretot si ens quedem atrapats a una regió plana. Per contrari, una taxa d'aprenentatge massa elevada faria que saltés per sobre dels punts mínims.

Podem escollir ξ de diverses maneres, i de vegades fins i tot, podem trobar la mida del pas que fa que la derivada direccional desapareixi. En altres casos, podem evitar iterar aquest algoritme i simplement saltar directament al punt crític resolent l'equació $\bigtriangledown_x f(x) = 0$ per x.

En general, no és suficient triar un pas tal que $f(x*) < f(x)$; si no ho triem bé, podem fer que l'algorisme convergeixi a un punt no estacionari. Afortunadament, hi ha diverses regles relativament senzilles per escollir el pas de gradient que, a més, tenen associada una garantia de de convergència.

- **Regla de la minimització limitada**: a cada iteració $\xi*$ és tal que la funció f es minimitza al llarg de la direcció $\bigtriangledown_x f(x)$, això és, $\xi*$ satisfà:

  $$\xi* = argmin_{\xi \in (0,1)} f(x - \xi \bigtriangledown_x f(x)).$$

  Tot i que aquesta opció és molt intuïtiva, requereix resoldre un problema d'optimització unidimensional a cada pas.

- **Regla d'Armijo**: Donats els paràmetres $\alpha \in (0, 0.5)$ i $\gamma \in (0, 1)$ i un pas inicial $\xi = 1$, realitza una reducció $\xi \longleftarrow \xi\gamma$ fins que la condició de descens

  $$f(x - \xi \bigtriangledown_x f(x)) \leq f(x) + \alpha\xi\langle \bigtriangledown_x f(x), - \bigtriangledown_x f(x)\rangle$$

  s'assoleix. A la pràctica, se solen triar $\alpha = 0.5$ i $\gamma = 0.8$. La condició es pot interpretar com que acceptem una fracció α de descens en f. Com a exemple tenim la figura següent. Busquem el mínim de la funció f representada en blau i anem reduint ξ mitjançant γ fins que la línia discontínua quedi per sobre de la funció f.

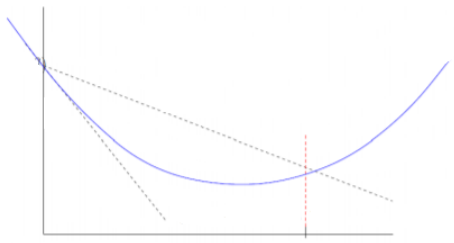

Figura 11: Regla d'Armijo

El gradient descendent convergeix quan cada element del gradient és zero (o, a la pràctica, molt a prop de zero), i quan aquesta situació es compleix podem dir que hem trobat un punt crític.

Tot i que el descens del gradient es limita a l'optimització en espais continus, el concepte de fer petits passos cap a millors configuracions es pot generalitzar a espais discrets.

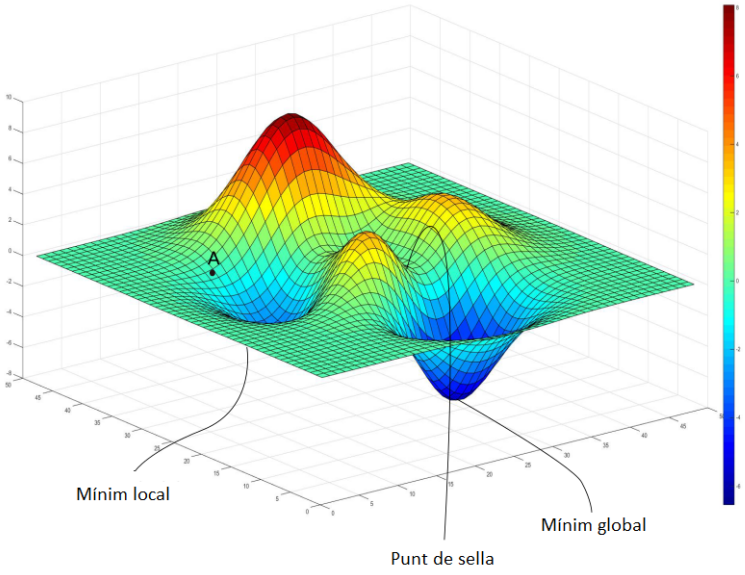

Figura 12: Suposem que estem al punt A i volem arribar al punt més baix de la superfície el més ràpid possible fent passes de mida ξ. A cada pas calcularem la nova direcció en què decrementa més ràpidament.

3.2 Més enllà del gradient descendent

De vegades voldrem trobar totes les derivades parcials d'una funció $f = (f_1, \ldots, f_n)$ tal que $f_i : \mathbb{R}^m \to \mathbb{R}$ amb un vector d'entrada i un escalar de sortida. La matriu que conté totes aquestes derivades parcials és la **matriu Jacobiana** $J \in \mathbb{R}^{nxm}$ de f i es defineix com

$$J_{i,j} = \frac{\partial}{\partial x_j} f_i(x).$$

Per tant, s'escriu de la forma:

$$J(f)_x = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \cdots & \frac{\partial f_1}{\partial x_m}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \cdots & \frac{\partial f_n}{\partial x_m}(x) \end{pmatrix}. \tag{3.1}$$

També ens interessarà la derivada d'una derivada, és a dir, la **segona derivada**. Per exemple, per a una funció $f : \mathbb{R} \to \mathbb{R}$ d'una única dimensió d'entrada i sortida, denotem $\frac{d^2}{dx^2}f = \frac{d}{dx}(\frac{df}{dx})$ o $f''(x)$. Si en canvi, f és una funció $f : \mathbb{R}^n \to \mathbb{R}$, la derivada respecte a x_i de la derivada de f respecte a x_j es denota com $\frac{\partial^2}{\partial x_i \partial x_j}f$.

La segona derivada ens diu com varia la primera derivada quan variem l'entrada. I, la interpretació geomètrica que en podem fer és que mesura la curvatura de la gràfica f.

Teorema 3.6. *Sigui f una funció derivable a un interval I i sigui $a \in I$, pel que existeix $f''(a)$. Aleshores:*

1. *Si $f''(a) > 0$ la funció és estrictament convexa al punt a.*
2. *Si $f''(a) < 0$ la funció és estrictament còncava al punt a.*
3. *Si a és punt d'inflexió aleshores $f''(a) = 0$.*

Pot passar que la primera i segona derivades siguin totes nul·les, i només certes eines matemàtiques, que no comentarem en aquest treball, ens aclaririen quin tipus de convexitat tenim, però si la funció es pot anar derivant podrem resoldre aquestes qüestions mirant les derivades successives.

Per exemplificar les explicacions, imaginem ara que tenim una funció quadràtica:

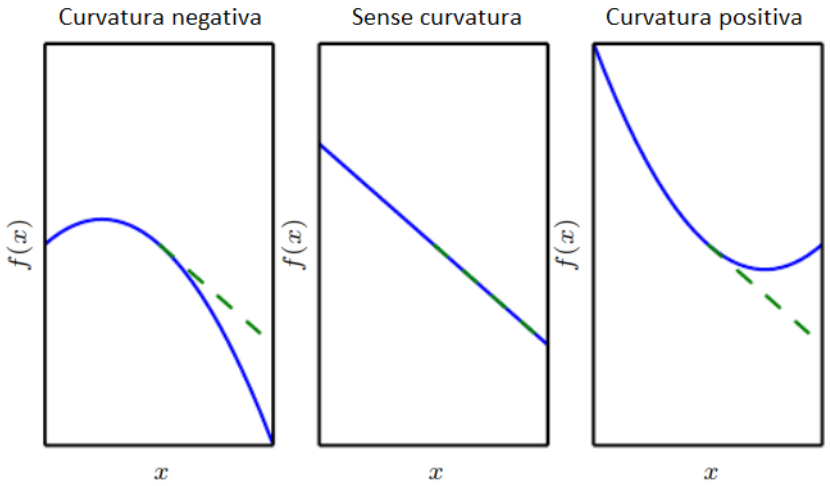

Figura 13: La línia verda discontínua indica el valor de la funció d'error que s'espera obtenir basant-se en la informació del gradient a mesura que fem passos seguint el gradient negatiu.

Si la segona derivada d'aquesta funció és igual a zero, $f''(x) = 0$, llavors no hi ha curvatura, i és una perfecta línia plana el valor de la qual es pot predir usant el gradient. Si el gradient és 1, podem fer un pas de mida ξ al llarg del gradient negatiu i la funció d'error disminuirà en ξ.

Si la segona derivada és negativa, $f''(x) < 0$, la funció es corba cap avall, de manera que la funció d'error en realitat disminueix en més de ξ.

Finalment, si la segona derivada és positiva, $f''(x) > 0$, la funció es corba cap amunt, de manera que la funció d'error pot disminuir en menys de ξ.

A més, també podem usar la segona derivada per determinar si un punt crític és un màxim local, un mínim local o un punt de sella. Farem ús del criteri de la segona derivada:

Criteri de la segona derivada

Sigui $f : I \to \mathbb{R}$ una funció dues vegades derivable en un entorn obert $I \subset \mathbb{R}$ que conté a un punt a i tal que $f'(a) = 0$, és a dir, a és punt crític de $f(x)$. Direm que:

1. Si $f''(a) > 0$, aleshores f té un mínim local o relatiu a $(a, f(a))$. Això significa que $f'(x)$ augmenta a mida que ens movem a la dreta de a i $f'(x)$ decrementa a mida que ens movem a l'esquerra a. A més, $f'(a - \alpha) < 0$ i $f'(a + \alpha) > 0$ per a α prou petit.

2. Si $f''(a) < 0$, aleshores f té un màxim local o relatiu a $(a, f(a))$. I les consideracions que hem comentat anteriorment, s'apliquen de manera respectiva.

3. Si $f''(a) = 0$, aleshores el criteri no decideix. És a dir, és possible que $f(x)$ tingui un màxim o un mínim relatiu en a, que sigui un punt de sella o part d'una regió plana. En aquests casos, caldria estudiar derivades superiors.

Per a una funció $f : \mathbb{R}^n \to \mathbb{R}$, amb múltiples dimensions d'entrada, hi ha moltes segones derivades parcials. Aquests derivades es poden agrupar juntes en l'anomenada **matriu Hessiana**. La matriu Hessiana es defineix com:

$$H(f)(x)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}(x),$$

Per tant, s'escriu de la forma:

$$H(f)_x = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(x) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_2 \partial x_2}(x) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \frac{\partial^2 f}{\partial x_n \partial x_2}(x) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n}(x) \end{pmatrix} \tag{3.2}$$

i ens serà molt útil per trobar punts crítics i per estudiar la concavitat i convexitat d'una funció de diverses variables.

A més, sempre que les segones derivades parcials siguin contínues seran commutatives, és a dir, l'ordre de derivació per obtenir-les no importa:

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) = \frac{\partial^2 f}{\partial x_j \partial x_i}(x),$$

de manera que la matriu hessiana és simètrica. I com que la matriu Hessiana és real i simètrica, llavors tindrà un conjunt de valors propis reals i una base ortogonal de vectors propis.

Tornant a la derivada direccional, recordem que és la magnitud del canvi de f per a un canvi en la direcció de u. Ara, per a la segona derivada direccional, tindrem el canvi de magnitud de la primera derivada direccional. Així doncs, la segona derivada en una direcció específica representada per un vector unitari u, ve donada per

$$\partial^2_{uu} f(x) = \partial_u(\partial_u f) = \lim_{t\to 0} \frac{\partial_u f(x+tu) - \partial_u f(x)}{t} =$$
$$= \lim_{t\to 0} \frac{\bigtriangledown f(x+tu)u - \bigtriangledown f(x)u}{t} = \lim_{t\to 0} \frac{u_i \partial_{x_i} f(x+tu) - u_i \partial_{x_i} f(x)}{t} =$$
$$= u_i \partial_{x_i x_j} f(x) u_j = u^T H u$$

Quan u és un vector propi de H, la segona derivada en aquesta direcció ve donada pel valor propi corresponent, és a dir, per

$$Hu = \alpha u$$

tenim que

$$u^T H u = \alpha u^T u = \alpha$$

per ser u unitari. Per a altres direccions de u, la segona derivada direccional és una mitjana ponderada de tots els valors propis, amb pesos entre 0 i 1, doncs $\|u\| = 1$ per la norma usual de $\mathbb{R}^n$. Concretament, si

$$u = \sum_{i=1}^{n} \alpha_i e_i,$$

on e_i és una base ortonormal donada pels vectors propis de H, tenim que, pel teorema de Pitàgores,

$$1 = \|u\|_2 = \sum_{i=1}^{n} \alpha_i^2$$

a partir de la qual cosa podem concloure que α_i^2 estan entre 0 i 1, i vectors propis que tenen un angle més petit amb u reben més pes. El valor propi màxim determina la segona derivada màxima i el valor propi mínim determina la segona derivada mínima.

Així doncs, la segona derivada direccional ens diu com de bo podem esperar que el gradient descendent actui. Podem fer Taylor de segon ordre a la funció $f(x)$ al voltant del punt actual $x^{(0)}$:

$$f(x) \approx f(x^{(0)}) + (x - x^{(0)})^T g + \frac{1}{2}(x - x^{(0)})^T H (x - x^{(0)})$$

on g és el gradient i H la Hessiana a $x^{(0)}$. Ara, si usem un valor d'aprenentatge de ξ, tindrem un nou punt $x = x^{(0)} - \xi g$. Ho substituïm a la nostra aproximació i tenim:

$$f(x^{(0)} - \xi g) \approx f(x^{(0)}) - \xi g^T g + \frac{1}{2}\xi^2 g^T H g.$$

De manera que ara tenim: el valor original de la funció, $f(x^{(0)})$, l'esperada millora deguda al pendent de la funció, $\xi g^T g$ i la correcció que hem d'aplicar per tenir en compte la curvatura de la funció, $\frac{1}{2}\xi^2 g^T H g$.

Hem de tenir present que els valors que pugui prendre aquest darrer terme, poden afectar significativament al resultat. Si és massa gran, el gradient descendent podria moure's cap amunt. Quan $g^T Hg$ és zero o negatiu, l'aproximació de la sèrie de Taylor prediu que augmentar ξ infinitament disminuirà f infinitament. Quan $g^T Hg$ és positiu, resoldre per la mida de pas òptima que més disminueix l'aproximació de la sèrie de Taylor de la funció resulta

$$\xi* = \frac{g^T g}{g^T Hg}.$$

I, si en el pitjor dels casos g s'alinea amb el vector propi de H corresponent al valor propi màxim λ_{max}, llavors aquesta mida òptima de pas vindria donada per

$$\xi* = \frac{1}{\lambda_{max}}.$$

En el cas que ens envolta, per determinar els punts màxims, mínims o punts de sella, podem tornar a fer ús del criteri de la segona derivada, però ho generalitzarem a dimensions múltiples per l'entrada. Per tant, haurem d'examinar totes les segones derivades de la funció. Per fer-ho, ens servirem de la descomposició pròpia de la matriu Hessiana.

Teorema 3.7. *Suposem que $f : A \to \mathbb{R}$ és una funció de classe $\mathcal{C}^2$ en un obert $A \subset R^n$. Si $\bigtriangledown_x f(p) = 0$ en un punt $p \in A$, llavors:*

1. *Si $H(f)_p$ és definida positiva, és a dir, si $H(f)_p(x-p, x-p) > 0$ per a tot $x \neq p$ o equivalentment tots els valors propis són positius, llavors f té un mínim local en p.*
2. *Si $H(f)_p$ és definida negativa, és a dir, si $H(f)_p(x-p, x-p) < 0$ per a tot $x \neq p$, llavors f té un màxim local en p.*
3. *Si f té un mínim local en el punt p, llavors $H(f)_p$ és semidefinida positiva, és a dir, si $H(f)_p(x-p, x-p) \geq 0$.*
4. *Si f té un màxim local en el punt p, llavors $H(f)_p$ és semidefinida negativa, és a dir, si $H(f)_p(x-p, x-p) \leq 0$.*
5. *Si $H(f)_p$ no és semidefinida (positiva o negativa), llavors f no té un extrem local en p.*

Observem a més, que si f té un extrem local en p, llavors cal que $H(f)_p$ sigui semidefinida (positiva o negativa). Els punts crítics de f que no són extrems locals els anomenem punts de sella.

També podem usar el següent criteri algebraic:

Teorema 3.8. *Suposem que $f : A \to \mathbb{R}$ és una funció de classe $\mathcal{C}^2$ en un obert $A \subset R^n$. Si $\bigtriangledown_x f(p) = 0$ en un punt $p \in A$, llavors:*

1. *Si $H(f)_p$ és definida positiva, és a dir, tots els valors propis són positius, llavors f té un mínim local en p. A més, la segona derivada en qualsevol direcció és positiva.*
2. *Si $H(f)_p$ és definida negativa, és a dir, tots els valors propis són negatius, llavors f té un màxim local en p. A més, la segona derivada en qualsevol direcció és negativa.*

3. *Quan almenys un valor propi és positiu i almenys un valor propi es negatiu, sabem que p és un màxim local en una secció transversal de f i un mínim local en l'altre.*

Hem de tenir en compte també, que en múltiples dimensions pot haver-hi una àmplia varietat de segones derivades diferents en un sol punt, perquè n'hi ha una diferent per a cada direcció.

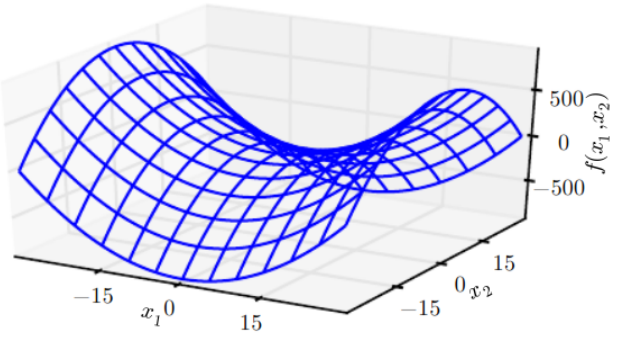

Figura 14: Per a la funció $f(x_1, x_2) = x_1^2 - x_2^2$ tenim un punt de sella que conté curvatura positiva i negativa. Al llarg de l'eix x_1 la funció es corba cap amunt. Aquest eix és un vector propi de la Hessiana i té un valor propi positiu. Per contrari, al llarg de l'eix x_2 la funció es corba cap avall i té un valor propi negatiu.

Abans d'acabar aquest apartat introduirem un parell de conceptes més que ens faran falta: la norma matricial i el número de condició.

Definició 3.9. *Siguin $\|\cdot\|_a$ i $\|\cdot\|_b$ dues normes en $\mathbb{C}^m$ i $\mathbb{C}^n$ respectivament. S'anomena* ***norma matricial*** *$\|\cdot\|$ en $\mathbb{C}^{m\times n}$ induïda per tals normes vectorials a:*

$$\begin{array}{rccl} \|\cdot\|: & \mathbb{C}^{m\times n} & \longrightarrow & \mathbb{R} \\ & A & \longmapsto & \|A\| = sup_{x\neq 0}\frac{\|Ax\|_a}{\|x\|_b} \end{array}$$

I la definició anterior verifica les propietats de norma. A més, en el cas que $m = n$ tindrem que $\|\cdot\|_a = \|\cdot\|_b = \|\cdot\|$.

Definició 3.10. *Sigui $A = (a_{i,j})$ una matriu $n \times n$. S'anomena* ***número de condició de A*** *associat a la norma $\|\cdot\|$ a $Cond(A)$ tal que*

$$Cond(A) = \|A\|\|A^{-1}\|.$$

Direm que la matriu A està **ben condicionada** si el seu número de condició és proper a 1 (≥ 1) i es diu que està mal condicionada si es significativament major que 1. Aquest últim cas implica que petites variacions en les dades poden produir grans variacions en els resultats. Les funcions que varien molt ràpidament quan els valors d'entrada són lleugerament pertorbats poden ser problemàtiques ja que els errors d'arrodoniment en aquests podria resultar en grans canvis en els valors de sortida.

Així doncs, el número de condició de la Hessiana mesura quant varien aquestes segones derivades. Quan és un nombre de condició deficient, es realitza un descens del gradient

molt pobre. Això es deu al fet que en una direcció, la derivada augmenta ràpidament, mentre que en una altra, augmenta lentament. El descens del gradient desconeix aquest canvi en la derivada, i no sap que és millor explorar en la direcció on la derivada es manté negativa durant més temps.

També és difícil triar una bona mida de pas, doncs ha de ser prou petit per evitar saltar per sobre el mínim i anar cap amunt en direccions amb curvatura molt positiva. Això significaria que el pas és massa petit per fer progressos significatius en altres direccions amb menys curvatura.

Aquest problema es pot resoldre usant informació de la matriu Hessiana. El mètode més senzill per fer-ho es coneix amb el nom de **mètode de Newton**, que és un algorisme per trobar aproximacions del zero d'una funció amb valors reals.

El mètode de Newton parteix d'una aproximació inicial $x^{(0)}$ i obté una millor aproximació $x*$, donada per la fórmula:

$$x* = x^{(0)} - H(f)(x^{(0)})^{-1} \bigtriangledown f(x^{(0)}).$$

L'expressió anterior es pot derivar d'utilitzar una expansió de segon ordre de la sèrie de Taylor aproximant $f(x)$ a prop d'algun punt $x^{(0)}$:

$$f(x) \approx f(x^{(0)}) + (x - x^{(0)})^T \bigtriangledown_x f(x^{(0)}) + \frac{1}{2}(x - x^{(0)})^T H(f)(x^{(0)})(x - x^{(0)}).$$

Quan f és una funció quadràtica definida positiva, apliquem una vegada l'equació anterior de $x*$ del mètode de Newton per saltar al mínim de la funció directament. I, quan f no és quadràtica, però es pot aproximar localment com una funció quadràtica definida positiva, apliquem diverses vegades l'equació de $x*$ del mètode de Newton. Actualitzant iterativament l'aproximació i saltant al mínim de l'aproximació es pot arribar al punt crític molt més ràpid que usant el gradient descendent.

Aquesta mètode és útil si estem prop d'un mínim local, però és dolent si es tracta d'un punt de sella. En canvi, el gradient descendent no s'atrau cap als punts de sella a menys que el gradient apunti cap a ells.

Els algorismes d'optimització com el gradient descendent que només utilitzen el gradient s'anomenen **algoritmes d'optimització de primer ordre**, mentre que els algorismes d'optimització com el mètode de Newton que també fan servir la matriu Hessiana s'anomenen **algorismes optimització de segon ordre**.

3.3 Gradient descendent estocàstic

Com ja hem comentat abans, la funció d'error a minimitzar és

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2,$$

que és un mitjana aritmètica. Per tant, si volem calcular les derivades de la funció d'error respecte als pesos, haurem de derivar cada sumand. A la pràctica, resulta molt costós fer-ho en cada iteració del gradient descendent ja que cada iteració de l'algorisme només millora la nostra funció d'error en petits passos.

Per resoldre aquest problema, existeix l'anomenat **gradient descendent estocàstic**, i consisteix en aproximar el valor de la derivada de $E(X, \theta)$ respecte als pesos calculant-la

només per subconjunt, triats a l'atzar. Fent la mitjana aritmètica sobre aquesta petita mostra podrem troba una aproximació ràpida i fiable del gradient real.

Cada vegada s'utilitzen algoritmes més complexos. La majoria d'ells es basen en mètodes del gradient aplicant lleugeres modificacions, com el que comentarem a continuació.

3.3.1 Gradient descendent estocàstic i el mètode del moment

Aquest mètode, en general, és més òptim que el gradient descendent estocàstic sol ja que ajuda a accelerar els gradients en les direccions correctes, donant lloc a una convergència més ràpida. Comentem-lo.

Per introduir el **mètode del moment** podem pensar el procés de la següent manera: imaginem una bola sobre la superfície d'error. Inicialment la bola és estàtica i, per tant, començarà a moure's en la direcció de més desnivell, és a dir, seguirà el gradient. Però tan bon punt agafi velocitat, deixarà de seguir aquesta direcció i el seu moment d'inèrcia farà que continuï.

El moment doncs, és essencialment un petit canvi a l'actualització dels paràmetres del gradient descendent estocàstic de manera que el moviment en l'espai de paràmetres és una mitjana dels diversos passos en el temps. Això es fa introduint una component de velocitat v.

Aquest mètode agilita el moviment en direccions de forta millora i també ajuda la xarxa a evitar els mínims locals.

A cada iteració determina la següent actualització com una combinació lineal del gradient i de l'actualització anterior. Després, actualitzem el pes de la xarxa. Tenim:

$$v_t = \beta v_{t-1} - \alpha \frac{\partial E}{\partial w}(t)$$

i

$$w = w - v_t$$

on v_t és el vector velocitat a temps t, que representa cada actualització dels pesos. La velocitat decau per un factor β, tal que $0 < \beta < 1$, i que determina la contribució relativa del gradient actual i dels gradients anteriors al canvi de pes,

$$\triangle w_t = v_t = \beta v_{t-1} - \alpha \frac{\partial E}{\partial w}(t) = \beta \triangle w_{t-1} - \alpha \frac{\partial E}{\partial w}(t)$$

Per tant, acabem de veure que la variació del pes es pot expressar en termes de la variació del pes previ i del gradient actual.

Cal tenir molta cura en establir el moment. A l'inici de l'aprenentatge, si generem els pesos de forma aleatòria, pot ser que tinguem gradients amb valors molt elevats. És a dir, tindrem una col·lecció de pesos que no ens ajuden per la tasca que volem desenvolupar. Per tant, és important que a l'inici tinguem un moment petit, per exemple $\beta = 0.5$. Una vegada la xarxa s'ha desfet dels gradients elevats, ens interessa augmentar lleugerament el moment fins arribar a un valor final de $\beta = 0.9$ o fins i tot $\beta = 0.99$.

3.3.2 Una versió millorada: el mètode de Nesterov

El **mètode de Nesterov** és una versió millorada del gradient descendent estocàstic usant moments i proposa:

- Primer fer un salt en la direcció del gradient anterior
- Després, avaluar el gradient del punt on hem ens trobem i corregir.

És a dir, es basa en la idea que és millor corregir una errada un cop s'ha fet.

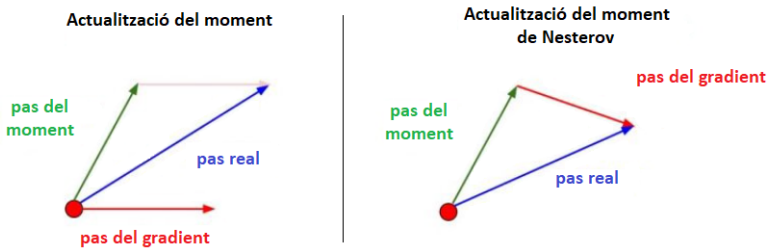

Figura 15: Actualització del moment v.s. actualització del moment pel mètode de Nesterov

3.4 Altres mètodes

La varietat de mètodes d'aprenentatge accelerats per a xarxes neuronals multicapa és enorme. No s'ha volgut fer una anàlisi exhaustiva de tots aquests algorismes, sinó que simplement comentarem aquells que tenen més importància a la pràctica, com ara la regla delta-bar-delta, o l'algorisme de Levenberg-Marquardt.

3.4.1 Regla delta-bar-delta

Un mètode d'aprenentatge accelerat habitualment usat és la **regla delta-bar-delta**, que es basa en l'existència d'una constant d'aprenentatge diferent per a cada pes de la xarxa.

S'aplica la següent regla a cada pes: si la direcció en què l'error decreix és la mateixa en què ha decrescut recentment, s'augmenta la mida del pas. En canvi, si la direcció és oposada, decrementa la mida del pas.

La direcció en la qual l'error decreix es determina mitjançant el signe de la derivada de l'error respecte el pes: si és positiu, l'error creix quan s'augmenta el pes i, si és negatiu, l'error decreix en augmentar el pes.

Per definir "recentment", si d_e és la derivada de l'error respecte el pes i f és el promig de la derivada actual i de les passades, podem definir $f_{e+1} = \gamma f_e + (1-\gamma)d_e$ on $\gamma \in (0,1)$. Quan més gran és el valor de γ més pesen les derivades passades a la mitjana. El signe de d ens dona una indicació de la tendència del canvi actual del pes, i el signe de f ens dona una indicació de les tendències passades. Si ambdós tenen igual signe podem augmentar el tamany del pes. Si per contrari, tenen signe diferent, la direcció haurà canviat i haurem de reduir el pes.

3.4.2 Algorisme de Levenberg-Marquardt

L'**algoritme de Levenberg-Marquardt**, també conegut com el **mètode de mínims quadrats esmorteïts**, s'utilitza per a resoldre problemes de mínims quadrats no lineals.

Aquests problemes de minimització sorgeixen especialment en l'ajust de corbes de mínims quadrats.

No obstant això, com passa amb molts algoritmes d'ajustament, el l'algorisme només troba un mínim local, que no és necessàriament el mínim global. Interpola entre l'algoritme de Gauss-Newton i el mètode de descens de gradient, i tot i ser més robust que Gauss-Newton, és a dir, que en molts casos troba una solució fins i tot si comença molt lluny del mínim final, tendeix a ser una mica més lent que G-N.

Usant aquest algorisme ens estalviem haver de calcular una matriu Hessiana. Quan la funció d'error és un sumatori de quadrats, la matriu Hessiana es pot aproximar com $H = J^T J$, i el pendent es pot calcular com $g = J^T e$, on J és la matriu Jacobiana que conté les derivades primeres dels errors de la xarxa respecte els pesos, i e és un vector d'errors de la xarxa.

Així doncs, usa aquesta aproximació de la matriu Hessiana a la següent adaptació de tipus Newton:

$$x_{k+1} = x_k - (J^T J + \mu I)^{-1} J^T e$$

Quan l'escalar $\mu = 0$, és el mètode de Newton, usant la matriu Hessiana aproximada. Quan μ és gran, tenim el gradient descendent del pendent amb un tamany de pas petit. Com que el mètode de Newton és ràpid i exacte, l'objectiu serà canviar tant ràpidament com sigui possible cap al mètode de Newton. Així, si μ disminueix després de cada pas, és a dir, hi ha reducció en la funció d'error, i augmenta quan el pas següent suposi que l'error augmenta. D'aquesta manera la funció d'error es reduirà sempre en cada iteració de l'algorisme.

4 Retropropagació d'errors

Backpropagation en anglès, que ve de "backward propagation of errors"significa **retropropagació d'errors** i és un algorisme de càlcul d'aprenentatge supervisat que utilitza el gradient descendent per entrenar xarxes neuronals artificials. D'aquesta manera, donada una xarxa neuronal artificial i una funció d'error, el mètode calcula el gradient de la funció respecte els pesos de la xarxa neuronal, usant un cicle retropropagatiu.

Recordem que el funcionament general d'una xarxa neuronal comença quan apliquem un patró a l'entrada de la xarxa com a estímul, que es propaga des de la capa d'entrada, passant per les capes ocultes, fins l'última capa, que genera una sortida. Aquest senyal de sortida es compara amb la sortida desitjada i es calcula un senyal d'error per a cada una de les sortides.

En la retropropagació, després, aquestes sortides d'error es propaguen endarrere, fent el procés en el sentit invers. És a dir, partint de la capa de sortida es calcula el gradient cap a les capes ocultes, deixant per últim la primera capa de pesos. A més, els càlculs parcials del gradient d'una capa es reutilitzen en el càlcul del gradient de la capa anterior. Aquest flux invers de la informació d'error permet un càlcul eficient del gradient de cada capa enlloc de calcular-lo per cada capa per separat. Malgrat tot, les neurones de les capes ocultes només reben una fracció del senyal total de l'error, basant-se aproximadament en la contribució relativa que hagi aportat cada neurona a la sortida original. Així doncs totes les neurones de la xarxa han rebut un senyal d'error que descriu la seva contribució relativa a l'error total.

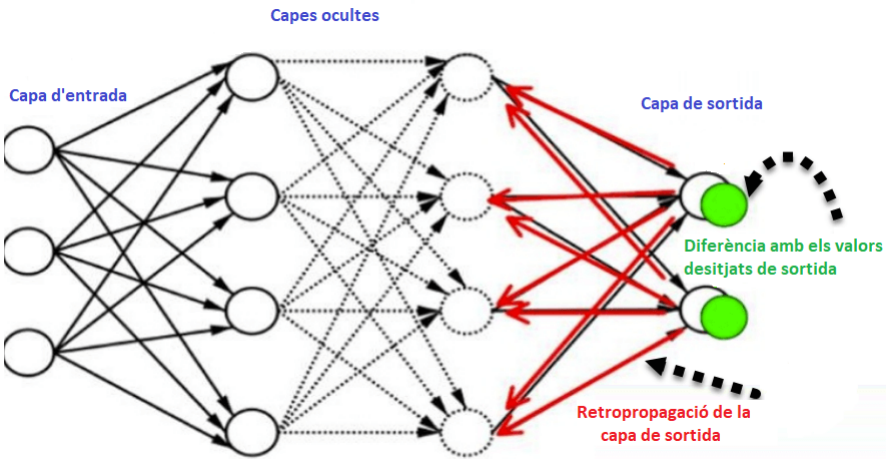

Figura 16: Esquema del procés de retropropagació d'errors (ref. [16])

En resum, podem distingir-ne dues fases:

- Propagació cap endavant, on calculem les sortides de la xarxa pels valors d'entrada.
- Propagació cap enrere, on anem calculant els corresponents valors de l'error i actualitzant els pesos, des de la última capa fins a la primera.

A mesura que s'entrena la xarxa, les neurones de les capes ocultes s'organitzen a si mateixes de tal manera que aprenen a reconèixer diferents característiques. Després de

l'entrenament, quan se'ls presenti un patró arbitrari a l'entrada, respondran amb una sortida activa si la nova entrada conté un patró que s'assembli a aquella característica que de manera individual hauran après a reconèixer durant el seu entrenament.

Tot i que la retropropagació d'errors és un algorisme ràpid, senzill, fàcil de programar i que en general, funciona bé, no ha estat fins fa poc que tornat a guanyar popularitat gràcies a l'ús de les xarxes neuronals profundes (deep learning, en anglès) que s'usen en xarxes neuronals propenses a errors, com ara les usades pel reconeixement d'imatges, el reconeixement de veu o el processament del llenguatge natural i que podem veure aplicades a la vida quotidiana en controls d'accessos d'aeroports, dictats automàtics, sistemes portàtils o assistents intel·ligents, d'entre un llarg etcètera.

Pels continguts que es desenvolupen en aquesta secció he consultat el material referent a [1], [3], [6], [8] i [15] de la bibliografia adjunta.

4.1 Història

El terme retropropagació d'errors i el seu ús general a les xarxes neuronals es va popularitzar al 1986 gràcies a D.E. Rumelhart, G.E. Hinton i R.J. Williams amb la publicació d'un article titulat *Representacions d'aprenentatge per errors de retropropagació*, on van demostrar experimentalment que aquest mètode pot generar representacions internes de dades útils en capes ocultes de xarxes neuronals. Però, la tècnica ja tenia antecedents com a mètode general d'optimització:

La regla de la cadena, usada en l'algorisme, va ser inventada al segle XVII per Leibniz i l'Hôpital. El càlcul i àlgebra s'han utilitzat durant molt de temps per resoldre problemes d'optimització, però el mètode del gradient descendent no es va introduir com a tècnica per aproximar iterativament la solució a problemes d'optimització fins al segle XIX gràcies a Cauchy.

A partir de la dècada de 1940, s'utilitzaven aquestes tècniques d'aproximació de funcions per motivar models d'aprenentatge automàtic. No obstant això, els primers models es basaven en models lineals. L'aprenentatge de funcions no lineals requeria del desenvolupament d'una xarxa multicapa i d'un mitjà per calcular el gradient.

La minimització d'errors a través del gradient descendent (Hadamard, 1908) en l'espai de paràmetres de sistemes complexos, no lineals, diferenciables, multi-etapa i relacionats amb les xarxes neuronals, data almenys de principis dels anys seixanta. Al 1961, el concepte bàsic de retropropagació contínua d'errors va ser derivat en el context de la teoria del control per J.K.H. Arthur i E. Bryson. També s'havia discutit que es podia realitzar un descens més pronunciat en l'espai de pesos d'aquests sistemes, quan al 1969, Bryson i Ho van donar un mètode d'optimització del sistema dinàmic en diverses etapes.

Els sistemes dels anys seixanta ja eren eficients en el sentit de la programació dinàmica. En aquell aleshores es va aconseguir reutilitzar els càlculs de derivades usant matrius jacobianes estàndard d'una "capa" a l'anterior, sense abordar explícitament ni els enllaços directes entre "neurones"de diverses capes ni els possibles guanys d'eficiència addicionals a causa de l'escassetat de la xarxa.

La retropropagació d'errors aviat es va utilitzar explícitament per minimitzar les funcions d'error mitjançant l'adaptació dels pesos. Al 1974, P. Werbos va esmentar la possibilitat d'aplicar aquest principi a les xarxes neuronals artificials, i un programa informàtic per obtenir i implementar automàticament l'algorisme per a qualsevol sistema diferenciable.

Al 1982, Hopfield va aportar la seva idea d'una xarxa neuronal i més tard, al 1993, Wan va ser la primera persona a guanyar un concurs internacional de reconeixement de patrons amb l'ajut del mètode de retropropagació d'errors.

Després de l'èxit d'aquest algorisme, la investigació de xarxes neuronals va guanyar popularitat i va arribar a un màxim a principis dels anys noranta. Durant la dècada de 2000 va caure desafavorit, però va tornar a principi de l'any 2010 gràcies als sistemes de càlcul, molt més barats i potents. Aquest fet ha permès usar-ho en molts camps, com ara en la visió automàtica, el processament del llenguatge natural, etc.

Les idees bàsiques darrere de les xarxes modernes no han canviat substancialment des dels anys vuitanta. És a dir, encara s'usa el mateix algorisme de retropropagació i el mateix enfocament del gradient descendent. No obstant això, un petit nombre de canvis algorítmics han millorat el rendiment de les xarxes neuronals notablement.

En resum, la retropropagació va ser un dels primers mètodes capaços de demostrar que les xarxes neuronals artificials podien aprendre bones representacions internes, és a dir, quins eren els valors que calculaven les capes ocultes de manera que aprenien trets no trivials. Els experts que examinaven xarxes multicapa entrenades amb retropropagació van trobar que molts nodes van aprendre característiques similars a les dissenyades per experts humans i les trobades per neurocientífics que investigaven xarxes neuronals biològiques en cervells de mamífers.

Més important encara, a causa de l'eficiència de l'algorisme i del fet que els experts ja no havien de descobrir les característiques adequades, la retropropagació va permetre aplicar xarxes neuronals artificials a un camp de problemes molt més ampli que fins aleshores havia quedat fora de límit a causa del temps i del cost.

4.2 Definició formal

Per definir formalment la retropropagació d'errors, partirem d'un conjunt de dades, un xarxa neuronal prealimentada i una funció d'error. Això és:

1. Un conjunt de dades, $X = \left\{ \left(\vec{x_1}, \vec{y_1}\right), \ldots, \left(\vec{x_N}, \vec{y_N}\right) \right\}$ de tamany N, format per tuples d'entrada-sortida $\left(\vec{x_i}, \vec{y_i}\right)$, on $\vec{y_i}$ és la sortida desitjada de la xarxa per l'entrada $\vec{x_i}$.

2. Una xarxa neuronal prealimentada, on denotem per θ el conjunt de tots paràmetres i on els més importants són els pesos i els biaixos. Denotem per w_{ji}^k el pes entre el node j a la capa $k-1$ i el node i a la capa k, i per b_i^k el biaix del node i a la capa k. Els pesos estan inicialitzats de manera aleatòria.

3. Una funció d'error, $E(X, \theta)$, que és una mitjana aritmètica entre les sortides desitjades $\vec{y_i}$ i les sortides calculades per la xarxa neuronal $\hat{\vec{y_i}}$ per les entrades $\vec{x_i}$, i valors particulars dels paràmetres θ.

En aquest treball ens centrarem en una xarxa neuronal amb una única sortida, però l'algorisme es pot aplicar a una xarxa amb qualsevol nombre de sortides mitjançant l'aplicació de la regla de la cadena i la regla de la potència. Així, els nostres parells d'entrada-sortida seran de la forma $(\vec{x}, y)$, és a dir, el valor objectiu y no serà un vector.

Recordem també les variables d'una xarxa i que usarem en l'algorisme de retropropagació d'errors:

w_{ji}^k: pes del node i a la capa k per l'entrada del node j de la capa $k-1$
b_i^k: biaix del node i a la capa k
a_i^k: sumatori de productes més el biaix (activació) per el node i de la capa k
o_i^k: sortida del node i en la capa k
r_k: nombre de nodes a la capa k
g: funció d'activació per als nodes de les capes ocultes
g_o: funció d'activació per als nodes de la capa de sortida

El nostre objectiu serà calcular el gradient de la funció d'error $E(X, \theta)$ respecte als pesos w_{ji}^k i biaixos b_i^k per entrenar la xarxa neuronal amb el gradient descendent. Després, segons la taxa d'aprenentatge ξ, cada iteració del descens del gradient actualitza els pesos i els biaixos (denotats θ) segons l'equació:

$$\theta^{t+1} = \theta^t - \xi \frac{\partial E(X, \theta^t)}{\partial \theta},$$

on θ^t denota els paràmetres de la xarxa neuronal a la iteració t en el gradient descendent, i on la funció d'error a minimitzar, que és l'error quadrat mitjà és:

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2,$$

on y_i és el valor objectiu en la tupla d'entrada-sortida $(\vec{x_i}, y_i)$, $\hat{y_i}$ és la sortida calculada per la xarxa per l'entrada $\vec{x_i}$ i N el tamany del conjunt de dades.

Recordem que quan alimentem una xarxa neuronal amb un patró d'estímul, aquesta ens retorna una sortida, que compararem amb el valor de sortida desitjat. El problema però, apareix amb les xarxes neuronals multicapa realimentades, ja que les capes ocultes no tenen una sortida objectiu. Doncs simplement s'utilitzen com a passos intermedis en el càlcul i no en podem definir una funció d'error específica. Així doncs, hem de decidir com aprendre bones representacions internes, és a dir, quins haurien de ser els pesos i els biaixos per als nodes de les capes ocultes.

També hem de tenir en compte que qualsevol valor de la funció d'error d'aquests nodes ocults dependrà dels valors dels paràmetres de les capes anteriors i de les capes següents, degut a la naturalesa de l'algorisme de retropropagació, doncs la sortida d'aquest node afectarà el càlcul de la funció d'error $E(X, \theta)$.

Aquesta lligam entre paràmetres de diferents capes necessita una bona organització per no ralentir el càlcul del gradient del descens final. La retropropagació tracta aquests dos problemes: simplifica les matemàtiques del descens del gradient i alhora facilita un càlcul eficient.

4.3 Derivació dels gradients

Pel càlcul de les derivades en l'algorisme de retropropagació d'errors farem ús especial de la regla de la cadena.

4.3.1 La regla de la cadena

La regla de la cadena s'usa per calcular les derivades de funcions que són composició d'altres funcions, i les derivades de les quals es coneixen. La retropropagació d'errors usa

aquesta regla en un ordre d'operacions específic que resulta molt eficient.

Sigui x un nombre real, i siguin f i g funcions tals que $f, g : \mathbb{R} \to \mathbb{R}$. Suposem que $y = g(x)$ i $z = f(g(x)) = f(y)$. Llavors la regla de la cadena diu que:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}.$$

Si ho generalitzem més enllà del cas escalar, suposem que $x = (x_1, \ldots, _m) \in \mathbb{R}^m$, $y = (y_1, \ldots, _n) \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$, i $f : \mathbb{R}^n \to \mathbb{R}$. Si $y = g(x)$ i $z = f(y)$, llavors:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \qquad \forall i = 1, 2, \ldots, m.$$

En notació vectorial, és equivalent a escriure:

$$\bigtriangledown_x z = \left(\frac{\partial y}{\partial x}\right)^T \bigtriangledown_y z,$$

on $\frac{\partial y}{\partial x}$ és la matriu Jacobiana $n \times m$ de g.

Així doncs, veiem que el gradient de z respecte d'una variable x es pot obtenir multiplicant una matriu jacobiana $(\frac{\partial y}{\partial x})^T$ per un gradient $\bigtriangledown_y z$. L'algoritme de retropropagació consisteix en realitzar múltiples productes de jacobianes-gradients.

Més en general, aplicarem l'algorisme a tensors de dimensionalitat arbitrària, i no només a vectors, tot i que conceptualment, és el mateix. L'única diferència és com els nombres es disposen en una quadrícula per formar un tensor. Podríem imaginar que aplanem cada tensor en un vector abans d'executar la retropropagació, calculant un gradient de valor vectorial, i després remodelant el gradient en un tensor. Així doncs, la retropropagació continua sent multiplicar els jacobians per gradients.

De la mateixa manera que amb els vectors, per denotar el gradient d'un valor z respecte d'un tensor X, escrivim $\bigtriangledown_X z$, però ara els seus índex tenen diverses coordenades. Ho podem abstreure mitjançant l'ús d'una única variable i per representar la tupla completa d'índexs. Per a totes les possibles tuples d'índex i, $\left(\bigtriangledown_X z\right)_i$ dona $\frac{\partial z}{\partial X_i}$. Això és exactament el mateix que per a tots els possibles índexs enters en un vector, $\left(\bigtriangledown_X z\right)_i$ dona $\frac{\partial z}{\partial X_i}$. Utilitzant aquesta notació, podem escriure la regla de la cadena usant tensors com: si $Y = g(X)$ i $z = f(Y)$, llavors

$$\bigtriangledown_X z = \sum_j \left(\bigtriangledown_X Y_j\right)\frac{\partial z}{\partial Y_j}.$$

Abans d'avaluar aquesta expressió en un ordinador hem de fer algunes consideracions prèvies. En primer lloc, moltes subexpressions es podrien repetir varis cops durant el gradient. Així doncs, s'haurà de triar si emmagatzemar aquestes subexpressions o recalcular-les varies vegades, corrent el risc de fer inviable la implementació de la regla de la cadena. Però en altres casos, calcular la mateixa subexpressió varies vegades podria ser una forma vàlida de reduir el consum de memòria a costa d'un temps d'execució més alt.

És possible que altres algoritmes puguin evitar més subexpressions realitzant simplificacions a la xarxa neuronal artificial, o conservant la memòria tornant a calcular enlloc d'emmagatzemar algunes subexpressions.

Reprenent la derivació de la funció d'error, anem a fer unes quantes consideracions prèvies.

Per simplificar la notació: posem r_{k-1} el nombre de nodes de la capa $k-1$ i considerarem que el biaix b_i^k del node i a la capa k s'incorporà als pesos com w_{0i}^k amb una sortida fixa de $o_0^{k-1} = 1$ per al node 0 de la capa $k-1$. Per tant,

$$w_{0i}^k = b_i^k.$$

I queda:

$$a_i^k = b_i^k + \sum_{j=1}^{r_{k-1}} w_{ji}^k o_j^{k-1} = \sum_{j=0}^{r_{k-1}} w_{ji}^k o_j^{k-1},$$

on el costat esquerre és la formulació original i el costat dret és la nova formulació. Recordem doncs que l'activació està definida com una suma ponderada de pesos de totes les activacions de les capes anteriors, més el biaix. Després, es passa a través de la funció d'activació g i queda,

$$o_i^k = g(a_i^k).$$

En aquesta suma ponderada, hi haurà neurones de capes anteriors que tindran més efecte en la suma ja que alguns pesos es multiplicaran per valors d'activació més grans. Hem de tenir en compte però, que no podem influir directament en l'activació, només podrem controlar els pesos i biaixos.

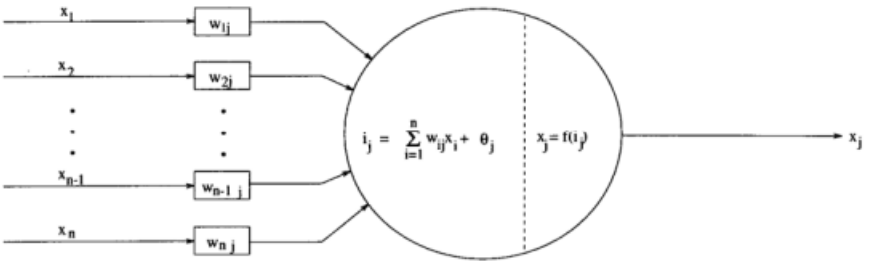

Figura 17: Esquema (usant notació diferent a la nostra) d'una neurona artificial (ref. [17])

Recordem doncs que la retropropagació d'errors intenta minimitzar la funció d'error següent respecte als pesos de la xarxa neuronal:

$$E(X, \theta) = \frac{1}{2N} \sum_{d=1}^{N} (\hat{y_d} - y_d)^2$$

calculant, per a cada pes w_{ji}^k, el valor de $\frac{\partial E}{\partial w_{ji}^k}$. Com que la funció d'error es pot descompondre en una suma de termes d'errors individuals per a cada parell d'entrada-sortida, la derivada també es pot calcular respecte a cada parell d'entrada-sortida individualment i després combinar-la al final (ja que la derivada d'una suma de funcions és la suma de les derivades de cada funció). Així doncs:

$$\frac{\partial E(X, \theta)}{\partial w_{ji}^k} = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial}{\partial w_{ji}^k} \left(\frac{1}{2} (\hat{y_d} - y_d)^2 \right) = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial E_d}{\partial w_{ji}^k}.$$

on escriurem E_d per a cada sumand $E_d = \frac{1}{2}\left(\hat{y_d} - y_d\right)^2$.

Per tant, pel que fa a la derivació, l'algorisme de retropropagació d'errors només es refereix a un parell d'entrada-sortida. Un cop derivat, es pot generar la forma general de tots els parells de X combinant els gradients individuals. Per tant, la funció d'error en qüestió per a la derivació és:

$$E = \frac{1}{2}\left(\hat{y} - y\right)^2,$$

on ometrem el subíndex d a E_d, $\hat{y_d}$, i y_d per simplificar.

4.3.2 Derivades de la funció error

La derivació de l'algorisme de retropropagació comença aplicant la regla de la cadena a la derivada parcial de la funció d'error

$$\frac{\partial E}{\partial w_{ji}^k} = \frac{\partial E}{\partial a_i^k}\frac{\partial a_i^k}{\partial w_{ji}^k},$$

on a_i^k és l'activació del node i a capa k abans de passar a la funció d'activació per generar la sortida. Aquesta descomposició de la derivada parcial diu bàsicament que el canvi en la funció d'error a causa d'un pes és producte del canvi en la funció d'error E a causa de l'activació a_i^k vegades el canvi en l'activació a_i^k a causa del pes w_{ji}^k. El primer terme l'anomenarem error, i es denota com

$$\delta_i^k \equiv \frac{\partial E}{\partial a_i^k}.$$

El segon terme es pot calcular a partir de l'equació de a_i^k anterior:

$$\frac{\partial a_i^k}{\partial w_{ji}^k} = \frac{\partial}{\partial w_{ji}^k}\left(\sum_{l=0}^{r_{k-1}} w_{li}^k o_l^{k-1}\right) = o_j^{k-1}.$$

Per tant, la derivada parcial de la funció d'error E respecte d'un pes w_{ji}^k és

$$\frac{\partial E}{\partial w_{ji}^k} = \delta_i^k o_j^{k-1}.$$

Així doncs, la derivada parcial d'un pes és producte del terme d'error δ_i^k al node i de la capa k, i la sortida o_j^{k-1} del node j de la capa $k-1$. Això té un sentit intuïtiu ja que el pes w_{ji}^k connecta la sortida del node j de la capa $k-1$ a l'entrada del node i de la capa k.

És important tenir en compte que les derivades parcials anteriors s'han calculat totes sense tenir en compte una funció d'error o una funció d'activació particular. Com hem esmentar anteriorment, la retropropagació clàssica utilitza la funció d'error donada per la mitjana dels quadrats i la funció d'activació sigmoide.

El càlcul de l'error δ_i^k es mostrarà depenent dels valors dels termes d'error de la següent capa. Per tant, el càlcul dels termes d'error procedirà cap enrere des de la capa de sortida fins a la capa d'entrada. Aquí és on rep el nom la **retropropagació** o **propagació d'errors cap enrere**.

4.3.3 Càlcul de l'error en la capa de sortida i les capes ocultes

Començant per la capa final, la retropropagació intenta definir el valor δ_1^m, on m és la capa final (posem el subíndex 1 i no i perquè aquesta derivació es refereix a la xarxa neuronal d'una sola sortida que estem considerant, de manera que només hi ha un node de sortida, $i = 1$). Per exemple, una xarxa neuronal de quatre capes tindrà $m = 3$ per a la capa final, $m = 2$ per a la segona, etc. Expressant la funció d'error E en termes del valor a_1^m (ja que δ_1^m és una derivada parcial respecte a_1^m) dona

$$E = \frac{1}{2}\left(\hat{y} - y\right)^2 = \frac{1}{2}\big(g_o(a_1^m) - y\big)^2,$$

on $g_o(x)$ és la funció d'activació de la capa de sortida. Per tant, aplicant la derivada parcial i utilitzant la regla de la cadena dona

$$\delta_1^m = \left(g_0(a_1^m) - y\right) g_o\prime(a_1^m) = \left(\hat{y} - y\right) g_o\prime(a_1^m).$$

Si ara ho unim tot, la derivada parcial de la funció d'error E respecte un pes de la capa final w_{j1}^m és

$$\frac{\partial E}{\partial w_{j1}^m} = \delta_1^m o_j^{m-1} = \left(\hat{y} - y\right) g_o\prime(a_1^m) o_j^{m-1}.$$

Reprenem ara la pregunta de com calcular les derivades parcials de les capes ocultes i d'entrada, tenint en compte que no tenim una sortida amb la què podem comparar, com en la capa de sortida. Per sort, la regla de cadena ens torna a ser útil. Considerem la següent equació per al terme d'error δ_i^k a la capa $1 \leq k < m$:

$$\delta_i^k = \frac{\partial E}{\partial a_i^k} = \sum_{l=1}^{r_{k+1}} \frac{\partial E}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_i^k},$$

on l oscil·la entre 1 i r_{k+1}. Notem que, com que el biaix l'entrada, o_0^k, corresponent a w_{0j}^{k+1} és fix, el seu valor no depèn de les sortides de capes anteriors i, per tant, l no comença amb valor 0.

Connectant el terme d'error δ_l^{k+1} dona la següent equació:

$$\delta_i^k = \sum_{l=1}^{r_{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_i^k}.$$

I recordant la definició de a_l^{k+1}

$$a_l^{k+1} = \sum_{j=1}^{r_k} w_{jl}^{k+1} g\big(a_j^k\big),$$

on $g(x)$ és la funció d'activació de les capes ocultes,

$$\frac{\partial a_l^{k+1}}{\partial a_j^k} = w_{jl}^{k+1} g'\big(a_j^k\big).$$

En connectar-ho a l'equació anterior es produeix una equació final per al terme d'error δ_i^k a les capes ocultes:

$$\delta_i^k = \sum_{l=1}^{r_{k+1}} \delta_l^{k+1} w_{jl}^{k+1} g'\left(a_j^k\right) = g'\left(a_j^k\right) \sum_{l=1}^{r_{k+1}} w_{jl}^{k+1} \delta_l^{k+1}.$$

Finalment, si ho unim tot, la derivada parcial de la funció d'error E respecte un pes de les capes ocultes w_{ji}^k per a $1 \leq k < m$ és

$$\frac{\partial E}{\partial w_{ji}^k} = \delta_i^k o_j^{k-1} = g'\left(a_j^k\right) o_j^{k-1} \sum_{l=1}^{r_{k+1}} w_{jl}^{k+1} \delta_l^{k+1}.$$

En resum, si considerem la derivada de la funció d'error respecte un pes de la capa de sortida tindrem:

$$\frac{\partial E}{\partial w_{j1}^m} = \delta_1^m o_j^{m-1} = (\hat{y} - y)\, g_o\prime(a_1^m) o_j^{m-1}.$$

Mentre que si és la derivada parcial de la funció d'error respecte un pes de les capes ocultes serà:

$$\frac{\partial E}{\partial w_{ji}^k} = \delta_i^k o_j^{k-1} = g'\left(a_j^k\right) o_j^{k-1} \sum_{l=1}^{r_{k+1}} w_{jl}^{k+1} \delta_l^{k+1}.$$

4.3.4 La fase endavant i la fase enrere

Així doncs i com ja sabem, els errors flueixen cap enrere, des de l'última capa fins a la primera, de manera que l'error δ_i^k a la capa k depèn dels errors δ_k^{k+1} de la capa $k+1$.

Per tant, necessitem calcular els primers termes d'error a partir de la sortida calculada $\hat{y} = g_o(a_1^m)$ i la sortida objectiu y. Després, els termes d'error de la capa anterior es calculen realitzant una suma de productes (ponderada per w_{jl}^{k+1}) dels termes d'error de la capa següent i escalant-los per $g'\left(a_j^k\right)$. L'operació es repeteix fins que s'arriba a la capa d'entrada.

Aquesta retropropagació d'errors és molt similar al càlcul directe que calcula la sortida de la xarxa neuronal. Per tant, el càlcul de la sortida sovint s'anomena **fase endavant** (forward phase en anglès) mentre que el càlcul dels termes d'error i les derivades se sol anomenar **fase enrere** (backward phase en anglès).

Mentre es va en la direcció cap endavant, les entrades es recombinen repetidament des de la primera capa fins a l'última mitjançant sumes de productes dependents dels pesos w_{ji}^k i transformacions per funcions d'activació no lineals $g(x)$ i $g_o(x)$. En el sentit invers en canvi, les entrades són els termes d'error de la capa final, que es recombinen repetidament des de l'última capa fins a la primera mitjançant sumes de productes dependents dels pesos w_{jl}^{k+1} i transformacions per factors escalars no lineals $g_o'\left(a_l^m\right)$ i $g'\left(a_l^k\right)$.

A més, com que els càlculs per a la fase enrere depenen de les activacions a_j^k i sortides o_j^k dels nodes de la capa anterior i de la següent, tots aquests valors s'han de calcular abans que pugui començar la fase enrere. Per tant, la fase endavant precedeix la fase enrere per a cada iteració del gradient descendent. Així doncs, les activacions a_j^k i sortides o_j^k seran recordats per utilitzar-lo en la fase següent. Un cop finalitzada la fase cap enrere

i conegudes les derivades parcials, els pesos (i els biaixos associats $b_j^k = w_{0j}^k$) es poden actualitzar per gradient descendent. Aquest procés es repeteix fins que es troba un mínim local o es compleix el criteri de convergència.

4.4 Un resum general de la retropropagació d'errors

Utilitzant els termes i equacions de les seccions anteriors, l'algorisme de retropropagació es pot resumir en les cinc fórmules següents:

- Per a les derivades parcials,
$$\frac{\partial E_d}{\partial w_{ji}^k} = \delta_i^k o_j^{k-1}$$
- Per al terme d'error de la capa final,
$$\delta_1^m = g_o'(a_1^m)\,(\hat{y_d} - y_d)$$
- Per als termes d'error de les capes ocultes,
$$\delta_j^k = g'\big(a_j^k\big) \sum_{l=1}^{r^{k+1}} w_{lj}^{k+1} \delta_l^{k+1}$$
- Per combinar les derivades parcials de cada parell d'entrada-sortida,
$$\frac{\partial E(X,\theta)}{\partial w_{ji}^k} = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial}{\partial w_{ji}^k} \left(\frac{1}{2} \left(\hat{y_d} - y_d\right)^2 \right) = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial E_d}{\partial w_{ji}^k}$$
- Finalment, per actualitzar els pesos,
$$\Delta w_{ji}^k = -\xi \frac{\partial E(X,\theta)}{\partial w_{ji}^k}.$$

I procedeix de la següent manera. Assumim una taxa d'aprenentatge adequada ξ i inicialitzem aleatòriament els paràmetres w_{ji}^k:

1. Calcular la fase endavant de cada parell d'entrada-sortida $(\vec{x_d}, y_d)$ i emmagatzemar els resultats $\hat{y_d}$, a_i^k, i o_i^k per a cada node i de la capa k, on k va des de la capa 0 d'entrada, fins la capa m de sortida.
2. Calcular la fase enrere per a cada parell d'entrada-sortida $(\vec{x_d}, y_d)$ i emmagatzemar els resultats $\frac{\partial E_d}{\partial w_{ji}^k}$ per a cada pes w_{ji}^k que connecta el node j de la capa $k-1$ amb el node i de la capa k procedint de la capa m de sortida, fins la capa 1 d'entrada. Els valors que arriben als nodes d'entrada són les derivades parcials $\frac{\partial E}{\partial x_i}$ respecte a cada entrada.
    - Avaluar el terme d'error de la capa final δ_1^m mitjançant la segona equació.
    - Propagar de nou els termes d'error de les capes ocultes δ_i^k, calculant cap enrere des de la capa oculta final $k = m-1$, i utilitzant repetidament la tercera equació.

- Avaluar les derivades parcials de l'error individual E_d respecte w_{ji}^k de la primera equació.

3. Combinar els gradients individuals per a cada parell d'entrada-sortida $\frac{\partial E_d}{\partial w_{ji}^k}$ per obtenir el gradient total $\frac{\partial E(X,\theta)}{\partial w_{ji}^k}$ per a tot el conjunt de parells d'entrada-sortida $X = \{(\vec{x_1}, y_1), \ldots, (\vec{x_N}, y_N)\}$ mitjançant la quarta equació (per exemple, amb una mitjana simple dels gradients individuals).

4. Actualitzar els pesos segons la taxa d'aprenentatge ξ i el gradient total $\frac{\partial E(X,\theta)}{\partial w_{ji}^k}$ utilitzant la cinquena equació (movent-se en la direcció del gradient negatiu).

Per acabar aquest capítol 4, recordem que fins ara hem considerat una funció d'activació sense especificar. Així doncs, a la següent secció parlarem de l'ús de funció d'activació sigmoide.

4.5 Ús de la funció d'activació sigmoide

La funció sigmoide $\sigma(x) = \frac{1}{1+e^{-x}}$ té propietats matemàtiques que, combinades amb una funció d'activació de sortida adequada, simplifiquen molt el funcionament de l'algorisme. Així, a la formulació clàssica, la funció d'activació per a nodes ocults és la sigmoide, $g(x) = \sigma(x)$, i la funció d'activació de sortida és la funció identitat, $g_o(x) = x$ (la sortida de xarxa és només una suma ponderada de la seva capa oculta, és a dir, l'activació).

Els valors de $\sigma(x)$ pertanyen a $[0, 1]$, i s'entén que 0 representa que no passa flux pel node de sortida, mentre que 1 significa que en surt la màxima freqüència.

La retropropagació d'errors és un factor motivador important en l'ús històric de les funcions d'activació sigmoides degut a propietat de la derivada:

$$g'(x) = \frac{\partial \sigma(x)}{\partial x} = \sigma(x)\big(1 - \sigma(x)\big).$$

Per tant, calcular la derivada de la funció sigmoide només requereix recordar la sortida $\sigma(x)$ i connectar-la a l'equació anterior. A més, la derivada de la funció d'activació de sortida també és molt senzilla:

$$g_o'(x) = \frac{\partial g_o(x)}{\partial x} = \frac{\partial x}{\partial x} = 1.$$

Per tant, l'ús d'aquestes dues funcions d'activació elimina la necessitat de recordar els valors d'activació a_1^m i a_i^k a més dels valors de sortida o_1^m i o_j^k, reduint considerablement la memòria de l'algorisme. Això es deu al fet que la derivada de la funció d'activació sigmoide, en la fase enrere, només necessita recordar la sortida de la fase anterior i no depèn del valor d'activació real (com el cas de la formulació general on $g'\big(a_j^k\big)$). De la mateixa manera, la derivada per a la funció d'activació d'identitat no depèn de res, ja que és una constant.

Per tant, per a una xarxa neuronal amb unitats sigmoidals ocultes i una unitat de sortida identitat, les equacions del terme d'error són les següents:

- Per al terme d'error de la capa final,

$$\delta_1^m = \hat{y_d} - y_d$$

- Per als termes d'error de les capes ocultes,

$$\delta_i^k = o_i^k(1 - o_i^k) \sum_{l=1}^{r^{k+1}} w_{li}^{k+1} \delta_l^{k+1}$$

Una propietat poc desitjada de la neurona amb funció d'activació sigmoide és que quan pren valors extrems, ja siguin 0 o 1, el gradient en aquestes regions pràcticament s'anul·la. Així doncs, a la pràctica actual, rarament la trobarem implementada.

Així doncs, i com ja hem comentat en ocasions anteriors, s'usen funcions d'activació com les funcions lineals rectificades $ReLU$ (rectified linear units en anglès), $ReLU = max(0, z)$ que milloren notablement el rendiment de les xarxes.

4.6 Altres funcions d'activació

En aquest apartat farem una breu introducció a altres funcions d'activació, sense entrar en molt detall.

4.6.1 Funció lineal

És un model simple, però computacionalment molt limitat. La funció d'activació en aquest cas és la identitat $\sigma(x) = x$.

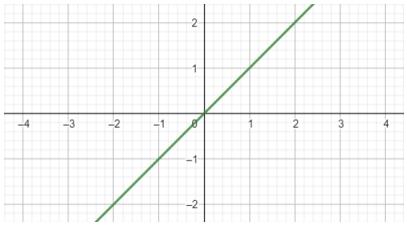

Figura 18: Funció lineal

4.6.2 Funció de llindar binari

Consisteix bàsicament en calcular la suma dels pesos dels inputs i enviar un pic d'activitat si la suma supera el llindar ω. L'equació del model es pot escriure de dues formes equivalents. Si $\omega = -b_i$:

$$z = \sum_{j=1}^{N} w_{ji}^k o_j^{k-1}$$

$$Y = \begin{cases} 1 \; si \; z \; \geq \; \omega \\ 0 \; altrament \end{cases}$$

O bé,

$$z = \sum_{j=0}^{N} w_{ji}^k o_j^{k-1}$$

$$Y = \begin{cases} 1 \; si \; z \; \geq \; 0 \\ 0 \; altrament \end{cases}$$

4.6.3 Funció tangent hiperbòlica

La funció tangent hiperbòlica és $\sigma(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Aquesta funció pren un nombre real i li assigna un valor de l'interval $[-1, 1]$.

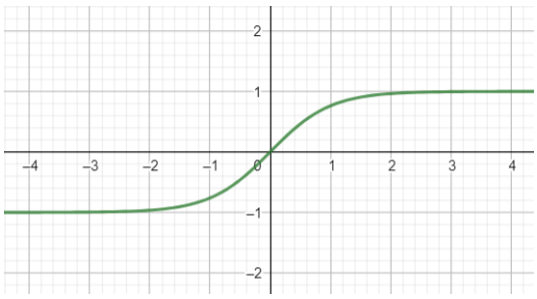

Figura 19: Funció tangent hiperbòlica

Com en la funció sigmoide, les activacions se saturen i anul·len el gradient. La gran diferència és que, en aquest cas, es tracta d'una funció centrada a l'origen i per això, a la pràctica, la funció tangent hiperbòlica és preferible a la sigmoide.

4.6.4 Funció lineal rectificada, ReLU

Aquesta funció ha esdevingut molt popular en els últims anys. Recordem que $ReLU = max(0, z)$ i pren la gràfica:

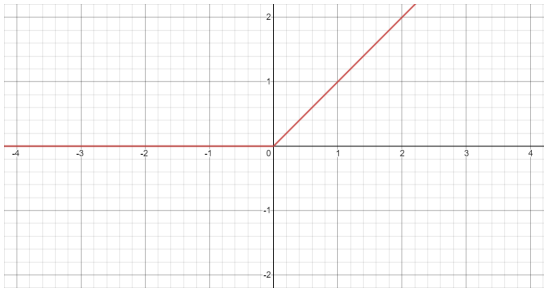

Figura 20: Funció ReLU

En comparació amb la funció sigmoide i la tangent hiperbòlica, que requereixen d'operacions cares com ara l'exponencial, en el cas de la *ReLU* només cal que apliquem una funció de màxim en els valors respecte a 0. S'ha comprovat també que aquest tipus de funcions acceleren la convergència del gradient descendent comparat amb les funcions sigmoide i tangent hiperbòlica.

Desafortunadament no és una solució perfecta. Les unitats *ReLU* són fràgils durant l'entrenament ja que poden anul·lar-se. Per exemple, un gradient molt gran pot causar

una actualització dels pesos tal que la neurona no torni a activar-se per cap altre valor. Si això succeeix, el gradient que passa per aquesta neurona serà nul a partir d'aquest punt.

4.6.5 Funció binària estocàstica

$$z = \sum_{j=0}^{N} w_{ji}^{k} o_{j}^{k-1}$$

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

on $P(s=1)$ és la probabilitat que es creï un pic de sortida.

La diferència ara és que es fa una decisió probabilística. És intrínsecament aleatòria. En aquest cas, si tenim un valor d'entrada gran i positiu, gairebé sempre crearà 1. Mentre que si el valor d'entrada és gran i negatiu, molt probablement crearà 0.

Finalment, les xarxes cap endavant continuen tenint molt potencial per treballar. En el futur, s'espera que s'apliquin a moltes més tasques, i que els avenços en algorismes d'optimització i disseny de models milloraran encara més el seu rendiment.

5 Conclusions

Els avantatges més importants d'usar l'algorisme de retropropagació d'errors són que és un mètode ràpid, senzill i fàcil de programar. A més, no té paràmetres per ajustar a part del nombre d'entrada. És un mètode flexible ja que no requereix coneixements previs sobre la xarxa i en general, com a mètode estàndard, funciona bé. Per últim, no necessita cap menció especial de les característiques de la funció per tal d'aprendre.

Els desavantatges que podem comentar es refereixen al rendiment real de la retropropagació ja que és un problema que depèn de les dades d'entrada i pot ser força sensible a segons quines.

Per altra banda, referit als mètodes d'aprenentatge que hem vist, podem comentar que existeixen moltes tàctiques i millores, de manera que aquests algorismes cada cop són més ràpids i eficaços. En aquest treball només s'han comentat els més interessants i més usats a la pràctica. Els estudis duts a terme actualment i en un futur, permetran que aquests algorismes funcionin molt més ràpid i amb menys càlculs, de manera que es redueixin els costos.

Finalment, l'estudi que he realitzat per explicar aquest algorisme, m'ha permès adquirir coneixements, no només referits a la retropropagació, sinó a les xarxes neuronals i al seu funcionament, així com conèixer tàctiques i mètodes d'optimització molt diversos i interessants.

Molts dels mètodes que comentats en aquest treball no han estat analitzats amb molta profunditat per qüestions diverses, però seria una bona manera de continuar aquest estudi i fer-lo més extens.

Referències

[1] I. J. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, The MIT Press, 2016.

[2] F. Berzal: On the Goldbach problem, *Redes neuronales y Deep learning*, 978-1731265388, Edición independiente, 2018.

[3] D. Rumelhart, G. Hinton and R. Williams, *Learning representations by backpropagating errors*, Nature 323, 533-536, https://doi.org/10.1038/323533a0, 1986.

[4] P. Larrañaga, I. Inza, A. Moujahid, *Redes neuronales, T8*, Universidad del País Vasco, 2017.

[5] 3BLUE1BROWN SERIES, *Neural Networks*, https://www.youtube.com/watch?v=Ilg3gGewQ5U, 2018.

[6] Andrew Ng, *Machine Learning*, https://www.coursera.org/learn/machine-learning/, 2021.

[7] Wikipedia, *Redes neuronales artificiales*, 2021, https://es.wikipedia.org/wiki/Redneuronalartificial.

[8] Wikipedia, *Propagación de errores*, 2021, https://es.wikipedia.org/wiki/PropagaciC3B3ndeerrores.

[9] A. Navarro, *Estudi sobre les xarxes neuronals artificials*, Universitat de Barcelona, 2016

[10] S. Miquel, *Métodos de aproximación, completado y separación de matrices de inteligencia artificial*, Universidad de Valladolid, 2020.

[11] Benemérita Universidad Autónoma de Prueba, https://www.cs.buap.mx/ sandoval/-MetodosNumericos/NormaMatrices.pdf, Álgebra II, Tema 8: Normas de matrices y número de condición.

[12] V. Bushaev, *How do we train neural newtworks?*, Towards Data Science, 2017, https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73

[13] V. Bushaev, *Stochastic gradient descent with momentum*, Towards Data Science, 2017, https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

[14] Wikipedia, *Función de activación*, 2021, https://es.wikipedia.org/wiki/FunciC3B3ndeactivaciC3B3n

[15] J. McGonagle, G. Shaikouski, C. Williams, *Backpropagation*, 2021, https://brilliant.org/wiki/backpropagation/deriving-the-gradients

[16] N. Kumar, *Backpropagation, Lecture Note*, 2021, https://www.guru99.com/images/1/0308190937BackPropaga1.png

[17] UPC, *Introducció a les xarxes neuronals artificials, capítol 3*, 2021, https://upcommons.upc.edu/bitstream/handle/2099.1/6483/05.pdf?sequence=6isAllowed=y