

Treball final de grau
GRAU DE MATEMÀTIQUES
Facultat de Matemàtiques
Universitat de Barcelona

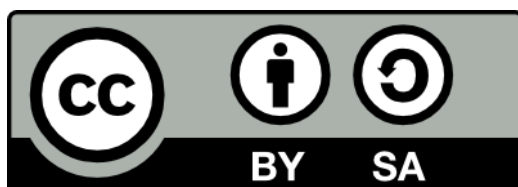
ESTUDI SOBRE LES XARXES NEURONALS ARTIFICIALS

AUTORA: ANDREA NAVARRO I LÓPEZ

DIRECTOR: DR. ORIOL PUJOL

DEPARTAMENT DE MATEMÀTICA APLICADA I ANÀLISI

Barcelona, 22 de gener de 2016



Aquesta obra està subjecta a la Llicència Reconeixement-CompartirIgual (CC BY-SA) International Creative Commons. Per veure una còpia de la llicència, visitar:

<http://creativecommons.org/licenses/by-sa/4.0/>

Abstract

One of the main features of the living beings, in special the human, is the capacity of learning. And the fact is that the great dream of the artificial intelligence is to achieve the creation of machines as intelligent as any one of us. Even still being away from succeeding, many researchers in the matter think that the best way to achieve it is from algorithms of learning, that test to imitate how the human brain works. The artificial neural networks simulate the behaviour of the biological nervous system. It treats of a system of models of artificial neurons, interconnected, arranged in a network with the purpose to collaborate to attain their common aim: the learning. In spite of the differences of structure between the biological and the artificial system, in both cases we find a massive group of simple units of process, the neurons. And it is in their connections where the intelligence of the network is placed. We plan to tackle the backpropagation algorithm both in theoretical and practical approaches so we can build an object recognition system.

Resum

Una de les característiques principals dels éssers vius, en especial els humans, és la capacitat d'aprenentatge. I el fet és que, el gran somni de la intel·ligència artificial és aconseguir crear màquines tan intel·ligents com qualsevol de nosaltres. Tot i encara estar lluny d'assolir-ho, molts investigadors en la matèria opinen que la millor forma d'aconseguir-ho és a partir d'algorismes d'aprenentatge, que proven d'imitar com funciona el cervell humà. Les xarxes neuronals artificials simulen el comportament del sistema nerviós biològic. Es tracta d'un sistema de models de neurones artificials, interconnectades, disposades en una xarxa amb la finalitat de col·laborar per assolir l'objectiu comú: l'aprenentatge. Malgrat les diferències d'estructura entre el sistema biològic i l'artificial, en tots dos casos trobem un conjunt massiu d'unitats de procés molt simples, les neurones. I és en les seves connexions on es troba la intel·ligència de la xarxa. Ens plantejem estudiar-ne l'algorisme d'aprenentatge cap enrere així com poder crear un sistema de reconeixement d'objectes.

Agraïments

En primer lloc, vull agrair de manera molt especial al Dr. Oriol Pujol, professor titular al Departament de Matemàtica Aplicada i Anàlisi, per la orientació inicial a l'hora d'acotar el tema del projecte, pel seguiment i la supervisió acurada d'aquest treball, així com la confiança dipositada i per tenir la porta del despatx sempre oberta per qualsevol qüestió.

A les empreses que ofereixen als estudiants la possibilitat de tenir un primer contacte amb el món professional. A Bismart per haver confiat en mi i permetre'm conèixer el món de l'aprenentatge automàtic.

Als de l'esplai per les hores de feina compartides i l'expectació a l'obtenir els resultats dels experiments. Navarro Armengol's per l'energia i confiança. I a l'Adrià per recolzar-me dia a dia, incansablement.

I, en especial a tu, per l'empenta que m'has transmès i deixat de per vida.

Índex

1	Introducció	1
1.1	Motivació	1
1.2	Ojectius	1
1.3	Què s’ha fet	1
1.4	Estructura de la memòria	2
2	Breu història	3
2.1	1940’s: L’inici de les xarxes neuronals	3
2.2	Primera etapa d’or	3
2.3	Anys tranquils	4
2.4	Entusiasme renovat	4
3	Aprenentatge automàtic	5
3.1	Tipus d’aprenentatge automàtic	6
3.1.1	Aprenentatge supervisat	6
3.1.2	Aprenentatge no-Supervisat:	9
3.1.3	Aprenentatge per reforçament:	9
4	Perceptró	10
4.1	Base biològica	10
4.2	Model matemàtic	11
4.3	Models de neurona	17
4.3.1	Neurona lineal	17
4.3.2	Neurona de llindar binari	17
4.3.3	Neurona sigmoide	18
4.3.4	Neurona tangent hiperbòlica	19
4.3.5	Neurona lineal rectificada, ReLU	19
4.3.6	Neurona binària estocàstica	20
4.4	Aprenentatge del perceptró	20
4.5	La superfície d’error per una neurona lineal	21

4.6	Visió geomètrica dels perceptrons	22
5	Xarxes neuronals	24
5.1	Arquitectura de les xarxes neuronals	24
5.1.1	Xarxa neuronal completament connectada.	24
5.1.2	Xarxes recurrents	26
5.1.3	Xarxes connectades simètricament	26
5.2	Aprenentatge	26
5.2.1	L'algorisme de propagació cap enrere	30
5.2.2	El mètode del moment	34
5.2.3	Dropout: una forma eficient de combinar xarxes neuronals	36
6	Classificació d'imatges	38
7	Prova de concepte	39
7.1	Descripció del problema i les dades	39
7.2	Arquitectura	40
7.3	Paràmetres experimentals i resultats	42
7.4	Conclusions	46

1 Introducció

1.1 Motivació

Arrel d'unes pràctiques en empresa, realitzades durant la primavera 2015 a través d'un conveni entre la Facultat de Matemàtiques de la Universitat de Barcelona i l'empresa en qüestió, vaig tenir l'oportunitat de conèixer el món de l'aprenentatge automàtic, un àmbit desconegut per mi fins aleshores.

L'empresa és especialitzada en Business Intelligence (BI), i necessitava una persona que comencés a treballar en aprenentatge automàtic per poder aplicar-ho a projectes reals en un futur proper. Al ser socis estratègics de Microsoft en l'àmbit de BI, estan especialment interessats en treballar amb la plataforma Azure del gegant tecnològic. Això suposa una ràpida introducció a l'aprenentatge automàtic i una relativament fàcil implementació dels problemes de negoci mitjançant experiments. L'Estudi d'Aprenentatge Automàtic d'Azure, és una eina que disposa de conjunts de dades d'exemple, algorismes ja implementats i eines matemàtiques i estadístiques necessàries per al correcte tractament de les dades. Tots aquests recursos s'apliquen als experiments de la forma *drag and drop*, és a dir, arrossegant i deixant anar. A més, són mòduls dels que podem fer una petita modificació dels paràmetres, però en general no pode'm conèixer-ne exactament què fan internament. És per aquest motiu que, tot i buscar solucions a problemes de negoci usant aprenentatge automàtic, no coneixes el veritable funcionament dels diferents algorismes d'aprenentatge. Així doncs, va néixer la idea de realitzar el Treball Final de Grau sobre què són i com funcionen les xarxes neuronals artificials.

1.2 Objectius

En aquest treball em plantejo entendre què són les xarxes neuronals artificials, quan s'usen i com funcionen. També vull aplicar-ho a un cas concret per resoldre el problema del reconeixement d'objectes, en concret dígit escrit a mà.

1.3 Què s'ha fet

He desenvolupat la teoria necessària per tal d'entendre què són i com funcionen els experiments d'aprenentatge automàtic supervisat, així com la documentació referent a altres tipus d'aprenentatge. M'he documentat sobre els perceptrons, els seus avantatges i inconvenients. He desenvolupat la teoria suficient per entendre les xarxes neuronals multicapa i el seu aprenentatge. He desenvolupat la teoria i pràctica necessària per entendre a fons el funcionament de l'algorisme de propagació cap enrere.

1.4 Estructura de la memòria

En aquest estudi sobre les xarxes neuronals artificials presentarem l'evolució de la matèria com a primer apartat. Seguirem descobrint què és l'aprenentatge automàtic i les diferents formes que podem trobar. A la secció següent presentarem el model biològic de neurona, farem la seva aproximació matemàtica, veurem quines varietats hi ha i com poden aprendre. Seguirem, un apartat més endavant, construint xarxes neuronals, descobrint la seva arquitectura i també el seu aprenentatge. Analitzarem l'algorisme d'aprenentatge principal, l'algorisme de propagació cap enrere. Seguirem un apartat més i passarem per sobre el reconeixement d'objectes i coneixerem el problema de la classificació d'imatges. Per acabar, tenim l'apartat de Prova de concepte on busquem aplicar els coneixements assolits per resoldre un cas pràctic.

2 Breu història

El camp de les xarxes neuronals, com gairebé qualsevol altre camp científic, té una llarga història de desenvolupament amb alts i baixos com veurem a continuació. En concret, podem distingir quatre etapes en la història de les xarxes neuronals: inici de les xarxes neuronals, primera etapa d'or, anys tranquils i entusiasme renovat.

2.1 1940's: L'inici de les xarxes neuronals

Sovint es considera que la neurocomputació va tenir els seus inicis en la publicació de l'article de recerca de McCulloch and Pitts [3], publicat el 1943, que mostrava que fins i tot xarxes neuronals senzilles podien calcular qualsevol funció lògica o aritmètica, i que va ser àmpliament llegit i influent. Altres investigadors, principalment Norbert Wiener i von Neumann, van escriure un llibre i un article, respectivament on es suggeria que la recerca en el disseny de sistemes que simulessin el cervell humà podria resultar interessant.

Però va ser al 1949, quan Hebb va publicar el seu llibre *The Organization of Behaviour* i per primera vegada es proposa una llei d'aprenentatge concreta a les sinapsis de les neurones. A partir d'això, Hebb usa aquesta llei per construir una explicació qualitativa d'alguns resultats experimentals de la psicologia.

2.2 Primera etapa d'or

Al 1957 Frank Rosenblatt, Charles Wightmann i d'altres col·laboradors, investigadors del MIT, van desenvolupar el primer neurocomputador *Mark I perceptron*. El mateix Rosenblatt al 1959 descriu diferents versions de perceptró, formula i verifica el Teorema de Convergència de Perceptrons.

No és fins una dècada més tard, al 1969, que Marvin Minsky i Seymour Papert publiquen un anàlisi matemàtic precís del perceptró per mostrar que no és capaç de representar molts dels problemes importants, com aprendre una funció tipus XOR. Així doncs es desmitificava el fet que les xarxes neuronals eren una metodologia genèrica capaç de resoldre tot tipus de problemes. Aquest fet va marcar un punt i a part a la popularitat i fons d'investigació. La idea que altres models més potents poguessin arribar a mostrar els mateixos problemes va fer preveure que el camp de les xarxes neuronals seria recerca sense sortida i va donar com a resultat una falta de fons per la investigació pel següents 15 anys.

2.3 Anys tranquils

A causa de la falta de finançament per la recerca, tampoc hi havia congressos ni xerrades i, amb prou feines hi havia publicacions al respecte. Tot i això, al 1973 Christoph von der Malsburg usa un model de neurona no lineal i biològicament més motivada. L'any següent, Paul Werbos desenvolupa un procediment d'aprenentatge anomenat *backpropagation* per la seva tesi a Harvard, tot i que no va ser fins una dècada més tard que el procediment obté la importància que té avui. Entre 1976 i 1980 Stephen Grossberg presenta articles on analitza matemàticament diversos models neuronals. A més, es dedica a mantenir xarxes neuronals capaces d'aprendre per elles mateixes sense desfer/destruir els aspectes apresos abans. L'any 1983 Fukushima, Miyake i Ito introdueixen el model neuronal de *neocognition* capaç de reconèixer caràcters escrits a mà.

2.4 Entusiasme renovat

A partir de la influència de John Hopfield que va convèncer personalment investigadors de la importància del camp i l'àmplia publicació sobre *backpropagation* de Rumelhart, Hinton i Williams, el camp de les xarxes neuronals mica en mica semblava refer-se.

Al 1986, amb múltiples capes de xarxes neuronals com a novetat, el problema era com estendre models de neurones introduïts abans a múltiples capes. Tres grups independents d'investigadors, un dels quals van incloure David Rumelhart, un ex membre del departament de psicologia de Stanford, es va acostar amb idees similars que ara es diuen xarxes de propagació cap enrere, ja que distribueix els errors de reconeixement de patrons per tota la xarxa. Aquestes noves xarxes usen múltiples capes, per tant ja s'ha introduït el concepte d'aprenentatge profund o *deep learning*. El resultat és que les xarxes de propagació cap enrere necessiten possiblement milers d'iteracions per aprendre i resulten ser d'aprenentatge lent.

Actualment, les xarxes neuronals artificials tenen moltes aplicacions. La idea fonamental és que si funciona a la natura, hem de trobar la forma de fer-ho funcionar de forma artificial.

El futur de les xarxes neuronals, però, rau en el desenvolupament de maquinari. A l'igual que les màquines avançades que juguen a escacs com Deep Blue, les xarxes neuronals ràpides i eficients depenen del maquinari que s'especifica per al seu ús.

Ens trobem també que la recerca que es concentra en el desenvolupament de les xarxes neuronals és relativament lenta. A causa de les limitacions dels processadors, les xarxes neuronals poden trigar setmanes en aprendre.

3 Aprenentatge automàtic

Per resoldre un problema usant un ordinador necessitem un algorisme. Segons l'enciclopèdia catalana [1], un algorisme és *un procediment de càlcul que consisteix a acomplir un seguit ordenat i finit d'instruccions que condueix, un cop especificades les dades, a la solució que el problema genèric en qüestió té per a les dades considerades*. És a dir, a partir d'unes dades d'entrada calcular o generar unes dades de sortida segons les necessitats del problema. Per un mateix problema podem trobar diferents algorismes i estarem interessats en el més eficient, és a dir, el que requereixi el menor nombre d'instruccions o memòria, o ambdós alhora.

Ens trobem que per algunes tasques no tenim tal algorisme. Aquest cas pot ser, per exemple, per distingir correus electrònics legítims de correu brossa. Sabem que d'entrada tenim un correu electrònic, que en el cas més senzill és un simple document de caràcters. Sabem que la sortida hauria de ser un Sí/No segons com classifiquem l'entrada. Però ens falta com relacionar entrada i sortida. El que podem considerar correu brossa canvia en el temps i entre individus.

Tot i això, el que ens falta de coneixement ho podem complementar amb les dades. Fàcilment podem recopilar milers de missatges d'exemple, alguns dels quals sabem que són correu brossa. El que volem aprendre és què fa que un correu sigui etiquetat com a correu brossa a partir del missatge. En d'altres mots, volem que la màquina aprengui a extreure automàticament l'algorisme per aquesta tasca. No cal fer aprendre una màquina a ordenar nombres, per això ja en tenim algorismes eficients. Ens cal que ho faci per problemes que no sabem com els resol el nostre cervell, o sabem com ho fa però són massa complexes per ser programats.

Hi haurà casos en els que no serem capaços d'identificar el procés que transforma unes dades d'entrada en unes de sortida per complet, però podrem construir-ne una aproximació bona i útil que detecti certs patrons, regularitats i tendències. Aquests patrons poden ajudar-nos a entendre el procés, o podem utilitzar-los per fer prediccions: suposant que el futur, almenys un futur pròxim, no serà molt diferent del passat (moment de recollir les dades de la mostra), les prediccions futures també es pot esperar que siguin correctes.

Per ser intel·ligent, un sistema que està en un entorn variant ha de tenir la capacitat d'aprendre'n. Si el sistema és capaç d'adaptar-se als canvis, no caldrà que el programador o dissenyador del sistema proveeixi solucions per tots els casos possibles.

L'aprenentatge automàtic és usat per trobar solucions a diferents problemes de les àrees de visió per ordinador, robòtica, reconeixement de la parla, entre molts d'altres.

3.1 Tipus d'aprenentatge automàtic

Hi ha dos tipus principals d'aprenentatge automàtic: el supervisat i el no supervisat. D'entrada pot semblar que la diferència entre un i altre sigui l'intervenció humana, però el cert és que aquests conceptes tenen més a veure amb el format en el que trobem les dades. Des del punt de vista pràctic, la forma d'encarar els dos escenaris és completament diferent. En poques paraules, l'aprenentatge supervisat és guiat per exemples proporcionats per experts i, per tant, classificats segons el seu criteri. Així doncs es podria proporcionar una classificació subjectiva. Mentre que l'aprenentatge no supervisat podria trobar una classificació objectiva, al no disposar d'exemples pre-classificats.

Com hem vist, l'aprenentatge automàtic es basa en aprendre propietats d'un conjunt de dades i aplicar-ho a unes dades noves. Per això en els experiments d'aprenentatge automàtic fem una partició de les dades que disposem. La partició sovint és 75% - 25%, i correspon al conjunt de dades d'entrenament (o *training set*) i conjunt de dades de prova (o *test set*). El procés d'aprenentatge disposa doncs de dues etapes clau. La primera és la d'entrenament del model on introduïm les dades d'entrenament i obtenim uns valors de sortida. A mida que les dades passen pel sistema, aquest va refinant l'aprenentatge sobre els patrons que segueixen les dades. La part de les dades que encara no hem usat, el conjunt de prova, l'introduïm en l'etapa de comprovació de l'eficàcia del sistema. Així aconseguim que el model treballi amb dades noves, que no ha vist fins aquesta última fase.

3.1.1 Aprenentatge supervisat

En l'aprenentatge supervisat volem aprendre a predir una sortida donat un vector d'entrada. Es tracta d'una tècnica per deduir una funció a partir de dades d'entrenament. L'objectiu de l'aprenentatge supervisat és crear una funció capaç de preveure el valor corresponent a qualsevol objecte d'entrada vàlid després d'haver vist una sèrie d'exemples, les dades d'entrenament. Per això, s'ha de generalitzar a partir de les dades presentades a situacions no vistes anteriorment. L'algorisme genera una funció que relaciona cada entrada amb la sortida desitjada. Cada cos d'entrenament té un vector d'entrada x i un vector objectiu t (target). És a dir, es proporciona un conjunt de dades al sistema que han estat etiquetats de forma manual amb els valors esperats. Distingim dos grans tipus: la regressió, que té per objectiu un nombre real o vector de reals i la classificació, que hi té una etiqueta de classe.

El model és de la forma

$$y = f(x, W),$$

on x , W són els paràmetres que ajustarem perquè ens quadri el model amb les dades.

Aprenentatge implica ajustar els paràmetres per reduir la discrepància entre

la sortida desitjada, t , en cada cas d'entrenament i la sortida actual, y , produïda pel model. Això vol dir que minimitzarem la diferència entre l'objectiu i les dades d'entrada (training set).

Per regressió, s'usa la diferència de quadrats, $\frac{1}{2}(y - t)^2$.

Per classificació es pot usar la diferència de quadrats, però hi ha altres mesures que són generalment més senzilles.

En aquest tipus d'aprenentatge, disposem de les dades d'entrenament etiquetades, és a dir, ja disposem del valor que hem de predir. Una vegada hem entrenat el model, volem saber-ne el seu rendiment. Per això disposem d'una eina anomenada matriu de confusió, on cada columna representa el nombre de prediccions de cada classe, mentre que cada fila representa els valors en la classe real. Resulta especialment útil per veure si el sistema està confonent classes diferents.

El en cas de tests d'hipòtesis tenim:

		Valor real		
		H_0	H_1	total
valor predit	H'_0	Cert Positiu	Fals Negatiu	P'
	H'_1	Fals Positiu	Cert Negatiu	N'
total		P	N	

A l'avaluar el model, desitgem que tingui el millor rendiment possible i això equival a tenir la matriu de confusió el màxim diagonal possible.

Amb aquestes notacions podem definir altres conceptes útils per a l'avaluació dels models.

Accuracy: Ràtio d'observacions correctament predides.

$$\frac{\text{Cert Positiu} + \text{Cert Negatiu}}{\text{TOTAL}}$$

Precision: Ràtio d'observacions correctes positives.

$$\frac{\text{Cert Positiu}}{\text{Cert Positiu} + \text{Fals Positiu}}$$

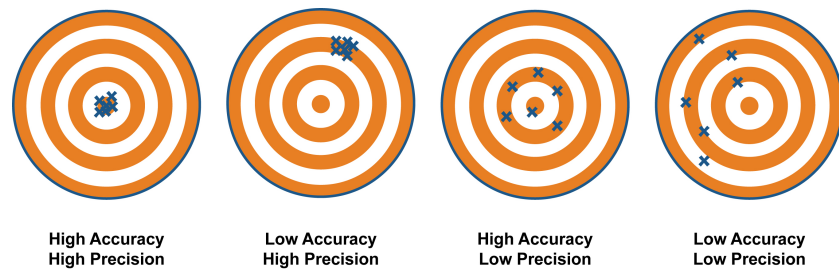


Figura 1: Diferència gràfica entre *accuracy* i *precision*. Imatge extreta de <http://kaffee.50webs.com/Science/labs/Lab-Precision.vs.Accuracy.html>

Recall: Ràtio d'observacions positives correctament predides.

$$\frac{\text{Cert Positiu}}{\text{Cert Positiu} + \text{Fals Negatiu}}$$

Definim els noms en anglès perquè la traducció d'*accuracy* i *precision* és precisió, però corresponen a conceptes diferents com podem comprovar per les seves definicions i per la *Figura 1*.

En el cas de problemes de classificació la matriu de confusió és de la forma:

Resultat de la predicció					
		classe 1	...	classe n	total
valor real	classe 1'	classe 1' classe 1	classe 1' ...	classe 1' classe n	A'
	...	...	...	...	
	classe n'	classe n' classe 1	classe n' ...	classe n' classe n	C'
total		A	B	C	

3.1.2 Apenentatge no-Supervisat:

Volem descobrir una bona representació interna de l'input. En aquest tipus d'aprenentatge segmentem les dades per buscar les variables a predir. És a dir, les dades no estan etiquetades. Podríem dir que és un sistema que s'entrena sense mestre i ha d'extreure per sí mateix la informació rellevant de les dades.

L'exemple més comú d'aquest tipus d'aprenentatge és el *clustering* on definim quantes classes de sortida volem, sense necessitat d'especificar sobre quina variable, i el sistema ha de trobar com separar les dades per tal de poder classificar-les. Un exemple pot ser el reconeixement de vehicles, sense saber de quins disposem. Aleshores el sistema pot generar-nos les classes per: quantitat de rodes, pes, mides..

També té altres objectius ja que proporciona una representació de l'input compacta i amb poques dimensions. I a més, aquesta representació és econòmica.

3.1.3 Aprenentatge per reforçament:

En aquest cas, la sortida és una acció o seqüència d'accions. Un cop realitzada l'acció que pren el sistema s'obté un retorn per part de l'entorn (una gratificació o una penalització). L'objectiu al triar cada acció és maximitzar la suma esperada de les futures recompenses.

Resulta un tipus d'entrenament difícil perquè s'aprèn mitjançant prova-error i requereix un elevat nombre de repeticions.

4 Perceptró

L'àrea de les xarxes neuronals va néixer amb l'objectiu de modelar sistemes neuronals biològics, però des de llavors ha anat evolucionant i s'ha convertit en una qüestió d'enginyeria donats els bons resultats aconseguits en tasques d'aprenentatge automàtic. No obstant, començarem amb un anàlisi descriptiu molt breu del sistema biològic en que les xarxes neuronals estan inspirades.

4.1 Base biològica

La raó més important per la que estudiem computació neuronal és per resoldre problemes pràctics usant nous algorismes d'aprenentatge inspirats en el cervell. Els algorismes d'aprenentatge poden ser molt útils encara que no sigui exactament com funciona el cervell.

El cervell humà és el millor sistema que coneixem, i la seva unitat de càlcul és la neurona. En el sistema nerviós humà trobem aproximadament 86 milions de neurones que estan connectades per aproximadament 10^{14} sinapsis.

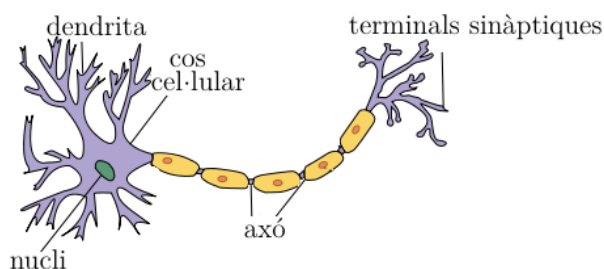


Figura 2: Neurona esquemàtica.

Cada neurona rep senyals d'entrada de les seves dendrites i produeix senyals de sortida al llarg del seu (únic) axó. L'axó, finalment, es ramifica i es connecta a través de les sinapsis en les dendrites d'altres neurones. Les sinapsis són les estructures que es creen quan un arbre dendrític es connecta amb un axó. O sigui entrades i sortides de diferents neurones es connecten.

Com podem veure, una neurona del cortex típica té: cos(cèl·lula), axó i arbre dendrític.

Cada neurona rep inputs d'altres neurones. Algunes neurones estan connectades als receptors. Les neurones es comuniquen entre elles enviant repunts d'activitat. L'efecte de cada línia d'entrada d'una neurona és controlat pel pes sinàptic que pot ser positiu o negatiu. Els pesos sinàptics s'adapten i d'aquesta forma la xarxa apren a realitzar càlculs útils (reconeixement d'objectes, entendre el llenguatge, fer plans, controlar el cos...) Un nombre enorme de pesos pot afec-

tar la computació en molt poc temps. Es tracta d'una autèntica banda ampla que funciona millor que qualsevol lloc de treball.

L'efectivitat de les sinapsis pot canviar variant el nombre de vesícules de transmissió i/o canviant el nombre de molècules receptores. Comparades a la memòria del PC, les sinapsis són lentes però tenen ventatge respecte RAM, són molt petites i de baix consum, s'adapten, cosa que és molt important. De fet, usen senyals locals per variar la potència/força i així és com aprenem a fer càlculs complicats.

Cada tros de cortex fa coses diferents tot i que visualment sembla el mateix a tot arreu. El cortex és capaç d'adaptar-se. Està fet per motius generals, però sap aprendre tasques específiques basant-se en l'experiència, en cas que sigui necessari. Això proporciona computació en paral·lel i flexibilitat.

4.2 Model matemàtic

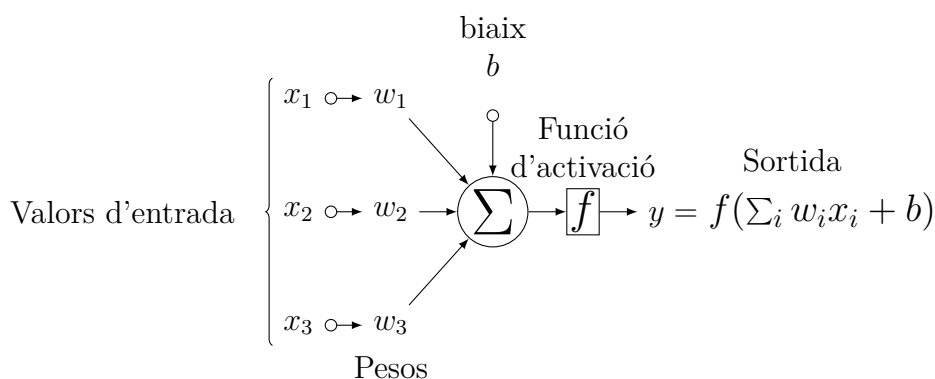


Figura 3: Model matemàtic d'una neurona.

En el model computacional d'una neurona, els senyals que viatgen al llarg dels axons (per exemple, x_0 interactuen multiplicativament (per exemple $w_0 x_0$) amb les dendrites de l'altra neurona basades en la força sinàptica. La idea és que els punts forts sinàptics (els pesos w) es poden aprendre i controlen la força d'influència (i la seva direcció: excitant (pes positiu) o inhibidor (pes negatiu)) d'una neurona a una altra. En el model bàsic, les dendrites porten el senyal al cos de la cèl·lula on tots queden acumulats.

El cervell humà és bastant diferent a un ordinador. Per començar, l'ordinador generalment té només un processador mentre el cervell està format per moltíssimes unitats de processament, les neurones, que operen en paral·lel. Tot i que els detalls no es coneixen, es considera que les unitats de processament del cervell són més simples i lentes que la de l'ordinador. Al cervell, les neurones tenen al voltant de 10^{14} connexions, anomenades sinapsis, totes elles funcionant en paral·lel. En un ordinador, el processador està actiu i la memòria està separada i passiva. Es creu que al cervell, tant el processament com la memòria es

distribueixen junts a través de la xarxa; el processament es realitza a través de les neurones, i la memòria està a les sinapsis entre les neurones.

Segons Marr (1982), entendre un sistema de processament d'informació té tres nivells, anomenats els nivells d'anàlisi:

1. La teoria computacional correspon a l'objectiu del càlcul i a una definició abstracta de la tasca a realitzar.
2. La representació i algorisme tracta com l'entrada i la sortida són representats i quines especificacions té l'algorisme per la transformació de l'entrada a la sortida.
3. Implementació del hardware és la realització física real del sistema.

Quedarà clar amb l'exemple d'ordenació:

La teoria computacional és ordenar un conjunt d'elements. La representació pot usar enters i d'algorisme podem usar Quicksort. Un cop compilat, el codi executable per un processador en particular que ordena enters representats en binari és la implementació del hardware.

La idea és que per la mateixa teoria computacional pot haver-hi diferents representacions, algorismes i implementacions de hardware.

El cervell és una implementació del hardware per l'aprenentatge o el reconeixement de patrons.

Per la mateixa raó que els primers intents de crear objectes o màquines voladores tenien dissenys semblants als ocells, es considera que els intents de construir estructures amb les capacitats del cervell s'hi assemblen, amb xarxes que contenen més unitats de processament. Però a l'igual que va passar amb els avions, que no tenen ales abatibles, es creu que aquesta similitud de dissenys seguirà fins que es descobreixi la teoria computacional de la intel·ligència. Per tant podem dir que per entendre el cervell, quan treballem amb xarxes neuronals artificials, estem al nivell d'anàlisi de representació i algorisme.

El perceptró és una unitat bàsica de processament artificial. Té valors d'entrada que poden venir de l'entorn o de sortides d'altres perceptrons. Associat amb cada valor d'entrada, $x_j \in \mathfrak{R}, j = 1, \dots, d$ tenim el pes de la connexió o pes sinàptic $w_j \in \mathfrak{R}$ i el valor de sortida, y , que en el cas més simple és la suma dels valors d'entrada:

$$y = \sum_{j=1}^d w_j x_j + w_0,$$

on w_0 és el valor d'intersecció per fer el model més general. En general és modelat com el valor del pes que prové d'una unitat extra, la del biaix x_0 , que sempre té valor +1.

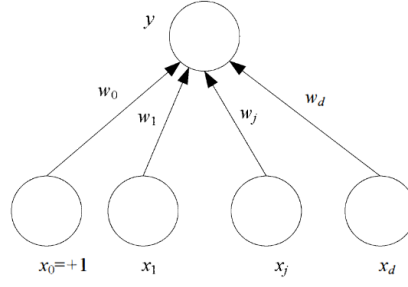


Figura 4: Perceptró simple. $x_j, j = 1, \dots, d$ són les unitats d'entrada. x_0 és la unitat del biaix. y és la unitat de sortida. w_j és el pes per la connexió de l'entrada x_j a la sortida.

Podem escriure la sortida d'un perceptró usant el producte matricial $y = w^T x$, on $w = [w_0, w_1, \dots, w_d]^T$ i $x = [1, x_1, \dots, x_d]^T$ són vectors augmentats per incloure-hi el pes i entrada del biaix.

Durant la fase de comprovació, amb pesos donats, w , per entrada x , calculem la sortida y . Per implementar una tasca donada necessitem aprendre els valors dels pesos w , els paràmetres del sistema. Aquests valors de sortida correctes són generats donats els valors d'entrada.

Quan $d = 1$ i x és alimentada per l'entorn a partir d'una unitat d'entrada, tenim $y = wx + w_0$, que és l'equació d'una recta que té pendent w i w_0 com a punt d'intersecció amb l'eix vertical. Per tant, aquest perceptró de només una unitat d'entrada i una de sortida pot ser usat per implementar un ajust lineal. Amb més inputs, la recta es torna un (hiper)pla, i el perceptró amb més d'una entrada es pot usar com ajust lineal multivariant.

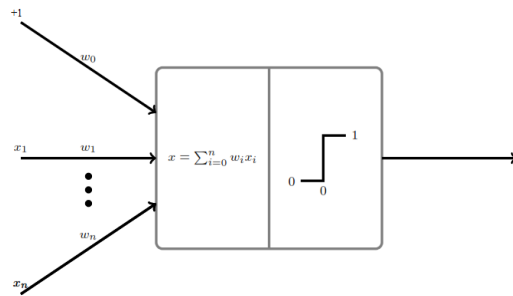


Figura 5: Perceptró simple. $x_j, j = 1, \dots, d$ són les unitats d'entrada. $x_0 = +1$ és la unitat del biaix. y és la unitat de sortida. w_j és el pes per la connexió de l'entrada x_j a la sortida. La funció d'activació és la funció llindar $s(\cdot)$

El perceptró defineix un hiperpla i, per tant, es pot fer servir per dividir l'espai de sortida en dos: la part on sigui estrictament positiu, i la restant. Si definim $s(\cdot)$ com una funció de llindar

$$s(a) = \begin{cases} 1 & \text{si } a > 0 \\ 0 & \text{altrament} \end{cases}$$

aleshores podem

$$triar = \begin{cases} C_1 & \text{si } s(w^T x) > 0 \\ C_2 & \text{altrament} \end{cases}$$

Cal recordar que si usem un discriminant lineal estem assumint que les classes són linealment separables.

A la *Figura 5* representem un perceptró amb funció d'activació la funció de llindar $s(\cdot)$ definida unes línies més amunt.

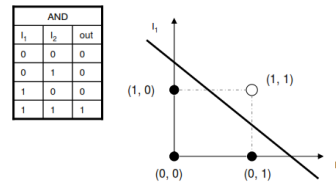


Figura 6: Límit de decisió de la porta lògica AND.

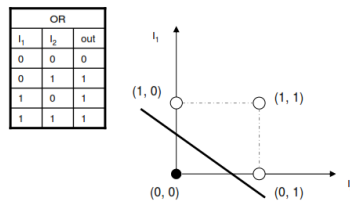


Figura 7: Límit de decisió de la porta lògica OR.

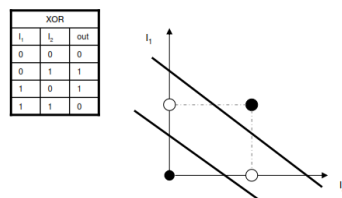


Figura 8: Límit de decisió de la porta lògica XOR.

L'hiperplà queda doncs separats per la superfície de decisió. Aquesta és la superfície on la sortida de la unitat és precisament el valor de llindar, és a dir,

$\sum_i w_i x_i = \theta$. Així doncs, resulta fàcil dibuixar els límits de decisió per les portes lògiques AND i OR, com podem observar a les *Figures 6 i 7*.

La dificultat de representar la funció lògica XOR resulta evident veient que no es pot separar per una sola línia de decisió. Vegem-ne una explicació senzilla de les limitacions dels perceptrons al reconèixer bits iguals, que equival al problema de la funció lògica XOR:

Donades les dades, per exemple:

Casos positius (són iguals): $(1, 1) \rightarrow 1$ $(0, 0) \rightarrow 1$

Casos negatius (són diferents): $(1, 0) \rightarrow 0$ $(0, 1) \rightarrow 0$.

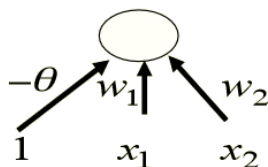


Figura 9: Perceptró considerat per l'exemple de limitació. Imatge extreta de [11].

Estem demanant a la unitat si els dos valors són iguals, i no pot aprendre-ho. El motiu és que les 4 parelles input-output donen 4 desigualtats que no es poden satisfer. Si considerem un perceptró com el de la *Figura 9* obtenim:

$$\begin{array}{ll} (1, 1) \rightarrow 1 & (0, 0) \rightarrow 1 \\ (1, 0) \rightarrow 0 & (0, 1) \rightarrow 0. \end{array} \implies \begin{array}{ll} w_1 + w_2 \geq \theta & 0 \geq \theta \\ w_1 < \theta & w_2 < \theta. \end{array}$$

Si sumem les dues primeres: $w_1 + w_2 \geq 2\theta$

Si sumem les dues segones: $w_1 + w_2 < 2\theta$.

Fet que ens porta a una contradicció.

Imaginem un espai de dades on els eixos corresponen a les components dels vectors d'entrada. Cada vector d'entrada és un punt de l'espai. El vector de pesos defineix un pla en aquest espai que és perpendicular al vector de pesos i la seva distància a l'origen és el valor del llindar θ . Això correspon doncs a la *Figura 10*.

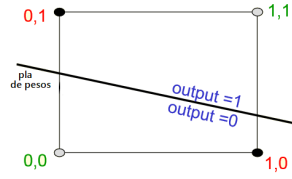


Figura 10: Visió geomètrica de l'exemple. Imatge extreta de [11].

Per solucionar aquest tipus de problemes podem canviar la funció d'activació per tal que tingui més d'un límit de decisió, o bé usar una xarxa neuronal més complexa capaç de generar límits de decisió més complexes.

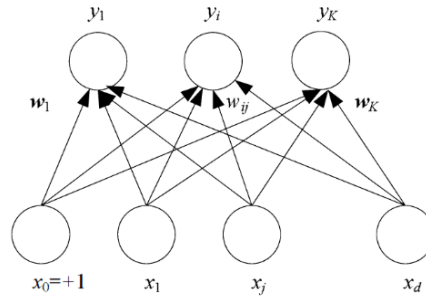


Figura 11: K perceptrons. $x_j, j = 1, \dots, d$ són les unitats d'entrada. x_0 és la unitat del biaix. y_i són les unitats de sortida. w_{ij} són els pesos per la connexió de l'entrada x_j a la sortida y_i .

Quan hi ha $K > 2$ sortides, tenim K perceptrons, cadascun d'ells amb un vector de pesos w_i .

$$y_i = \sum_{j=1}^d w_{ij}x_j + w_{i0} = w_i^T x$$

$$y = Wx$$

on w_{ij} és el pes de la connexió entre el valor d'entrada x_j i el de sortida y_i . W és la matriu de pesos de w_{ij} , de dimensió $K \times (d+1)$, que té per files els vectors de pesos dels K perceptrons.

Així doncs hem vist que un perceptró que tingui una sola capa de pesos pot aproximar només funcions lineals dels valors d'entrada i no pot resoldre problemes tipus XOR. Si els usem per classificació, els perceptrons multicapa (MLP, multilayer perceptrons) poden implementar discriminants no lineals i, si s'usen per regressió, poden aproximar funcions no lineals dels valors d'entrada.

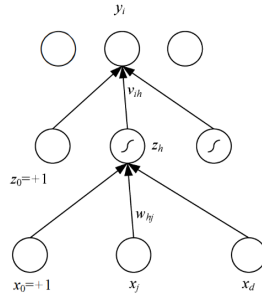


Figura 12: Estructura d'un perceptró multicapa.

4.3 Models de neurona

Per poder modelar coses hem d'idealitzar-les. És a dir, entendre-ho com quelcom més simple del que ja en coneixem el funcionament o en tenim una idea. L'idealització elimina detalls complicats que no són importants per entendre el concepte. Un cop hem entès la base, és fàcil afegir-hi complexitat. 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.

Tornem a la funció d'activació, que és definida segons les necessitats del problema que tractem. Cada funció d'activació pren un valor d'entrada i hi aplica una operació matemàtica fixa. A la pràctica podem trobar diferents neurones, segons les funcions d'activació que hi tenen implementades.

4.3.1 Neurona lineal

És el model més simple, però computacionalment limitat. La funció d'activació en aquest cas és la identitat. Tenim:

$$y = b + \sum_i x_i w_i$$

on y és la sortida, b és el biaix, x_i l'input i -èsim i w_i el pes de l'input i .

4.3.2 Neurona de llindar binari

Consisteix bàsicament en:

- 1) Calcular la suma dels pesos dels inputs.
- 2) Enviar un pic d'activitat si la suma supera el llindar.

L'equació del model es pot escriure de dues formes equivalents, si $\theta = -b$:

$$z = \sum_i x_i w_i \quad (4.1) \quad z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 & \text{si } z \geq \theta \\ 0 & \text{altrament} \end{cases} \quad (4.2) \quad y = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{altrament} \end{cases} \quad (4.3)$$

4.3.3 Neurona sigmoide

La funció sigmoide és $\sigma(x) = \frac{1}{1+e^{-x}}$. A grans trets, pren un nombre real i li assigna un valor entre 0 i 1. En particular, per a valors negatius molt grans és 0, i pels positius molt grans valdrà 1.

L'output és un valor real que surt d'una funció suau i acotada del total de l'entrada. Típicament s'usa la funció logística.

$$z = b + \sum_i x_i w_i$$

$$y = \frac{1}{1 + e^{-z}} \quad (4.4)$$

on z és l'input total i y és l'output.

Aquest tipus de neurones es comporten bé (funció suau - derivada contínua) i faciliten la feina per crear aprenentatge tal i com es veurà més endavant.

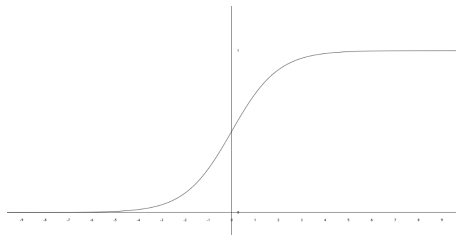


Figura 13: Gràfica de la funció sigmoide.

La funció sigmoide s'ha usat molt històricament degut a la simplicitat de la seva gràfica i té una bona representació de l'activació de la neurona, ja que té recorregut $[0, 1]$. S'entén que 0 representa que no passa flux pel node de sortida, mentre que 1 significa que en surt la màxima freqüència. A la pràctica actual, però, rarament la trobarem implementada ja que la funció es satura i elimina els gradients. Una propietat molt 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.

4.3.4 Neurona 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, similarment a l'anterior, pren un nombre real i li assigna un valor entre -1 i 1 , amb aquests inclosos.

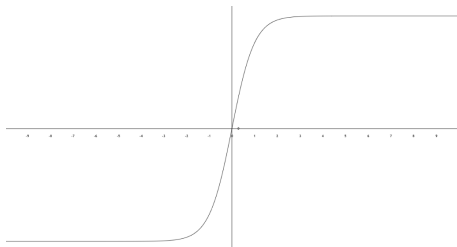


Figura 14: Gràfica de la funció $\tanh(x)$.

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.

4.3.5 Neurona lineal rectificada, ReLU

És un altre tipus de neurona que convina propietats de neurones lineals i neurones de llindar binari. Calcula la suma lineal per pesos dels inputs. L'output és una funció no lineal de l'input total.

$$z = b + \sum_i x_i w_i$$
$$y = \begin{cases} z & \text{si } z \geq 0 \\ 0 & \text{altrament} \end{cases} \quad (4.5)$$

on l'output és y .

La neurona lineal rectificada (ReLU) ha esdevingut molt popular en els últims anys. Calcula la funció $f(x) = \max(0, x)$. En altres paraules, correspon a una neurona de llindar binari amb llindar $\theta = 0$.

En comparació amb la neurona 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 d'una matriu respecte a 0. S'ha descobert també que aquest tipus de neurones accelera la convergència del descens del gradient estocàstic en comparació amb les funcions sigmoide i tangent hiperbòlica.

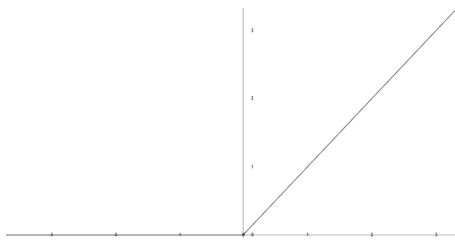


Figura 15: Gràfica de la funció $f(x) = \max(0, x)$.

Desafortunadament no és una solució perfecta. Les unitats ReLU són fràgils durant l'entrenament i poden morir, és a dir, anul·lar-se. Per exemple, un gradient molt gran que flueixi per una neurona ReLU 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.

4.3.6 Neurona binària estocàstica

$$z = b + \sum_i x_i w_i$$

$$P(s = 1) = \frac{1}{1 + e^{-z}} \quad (4.6)$$

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

La diferència amb l'anterior és que en comptes de prendre el real com a sortida, fan una decisió probabilística. És intrínsecament aleatòria.

En aquest cas, si tenim un valor d'entrada gran i positiu, gairebé sempre crearà 1. Mentres que si el valor d'entrada és gran i negatiu, molt probablement crearà 0.

4.4 Aprenentatge del perceptró

La regla d'aprenentatge del perceptró és:

1. Inicialitzar els pesos amb valors triats a l'atzar.
2. Per cada parella d'entrenament $x, y_{objectiu}$
 - Calcular la sortida y .
 - Calcular l'error, $\delta = (y_{objectiu} - y)$

- Usar l’error per actualitzar els pesos com segueix:

$$\Delta w = \eta \delta x \quad \text{o} \quad w_{vell} = w_{nou} + \eta \delta x$$

on η és la taxa d’aprenentatge i determina com de suau s’executa el procés.

3. Repetir **2** fins que convergeixi, és a dir, fins que δ sigui 0.

La regla d’aprenentatge del perceptró sovint ve donada per:

$$w_{vell} = w_{nou} + \eta \delta x$$

on

$$\delta = y_{objectiu} - y$$

4.5 La superfície d’error per una neurona lineal

Si entenem la forma d’aquestes superfícies, entendrem millor què passa quan una neurona lineal està aprenent.

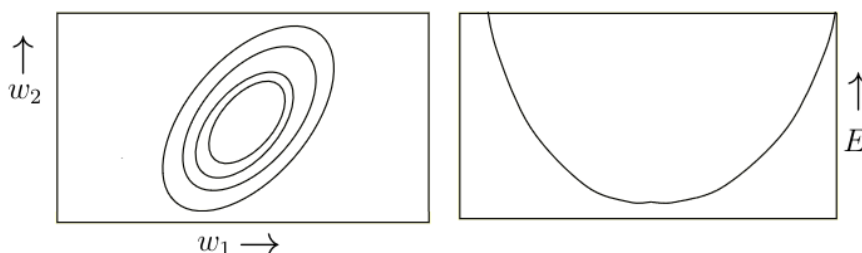


Figura 16: Superfície d’error en el cas de neurones lineals amb error quadràtic. Imatge extreta de [11]

Imaginem un espai de dimensió n on cadascuna de les primeres $n - 1$ dimensions corresponen a cada pes i la dimensió restant correspon a l’error. Així doncs, l’altura dels punts d’aquest espai correspondrà a l’error que hem produït. Per neurones lineals amb un error quadràtic, la representació és un paraboloid el·líptic de revolució.

Usant aquesta superfície d’errors podem obtenir una imatge del que està passant a mida que fem aprenentatge.

Sense entrar en detalls, visualment podem percebre la diferència entre el tipus d’entrenament que apliquem. Estem parlant d’aprenentatge per lots/series de dades, *batch learning*, o aprenentatge connectat, *online learning*.

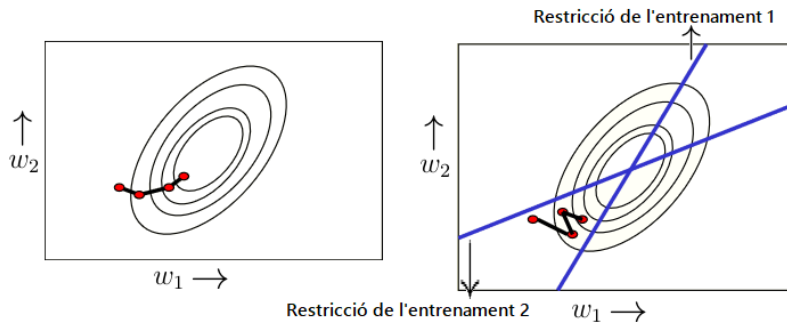


Figura 17: Evolució de l'error segons el tipus d'alimentació de les dades al sistema. Esquerra: *Batch learning*. Dreta: *Online learning*. Imatge extreta de [11]

Si canviem els pesos en les derivades, equival a fer *Steepest descent* a la superfície d'errors. És a dir, va avançant en perpendicular a les línies de contorn de la superfície d'error.

Però també podem fer aprenentatge *online*, on després de cada entrenament canviem els pesos en proporció al gradient per aquell cas concret d'entrenament. En aquest procediment, per a cada cas d'entrenament es crea la restricció: hem de quedar-nos entre una i altra restricció. Cada línia que seguim per avançar és perpendicular a la línia de restricció que correspongui.

4.6 Visió geomètrica dels perceptrons

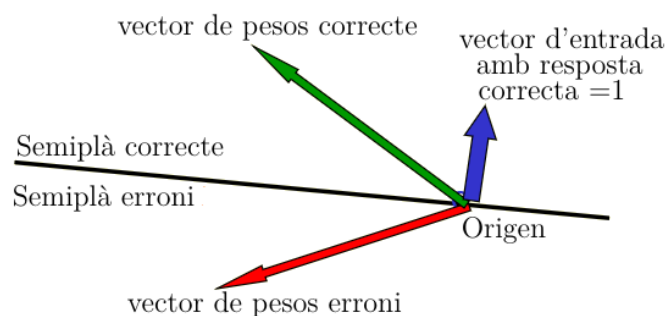


Figura 18: Espai de pesos. Imatge extreta de [11]

Definim un espai de pesos n -dimensional, amb n el nombre de pesos. Un punt de l'espai representa un ajust concret de tots els pesos. Si assumim que hem eliminat el llindar θ , cada cas d'entrenament es pot representar com un hiperpla que passa per l'origen.

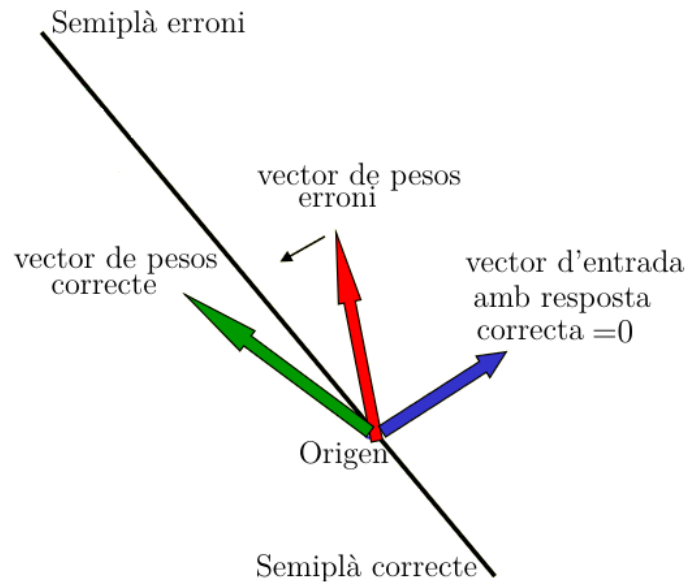


Figura 19: Espai de pesos. Imatge extreta de [11]

Cada cas d'entrenament defineix un pla (línia negra a la *Figura 18*). El pla passa per l'origen i és perpendicular al vector d'entrada. A un costat del pla, la sortida és errònia perquè el producte escalar del vector de pesos amb el vector d'entrada té valor negatiu.

Per obtenir tots els casos d'entrenament correctes hem de trobar un punt al semipla correcte de tots els plans. Cal tenir en compte que l'existència d'aquest punt no està garantida.

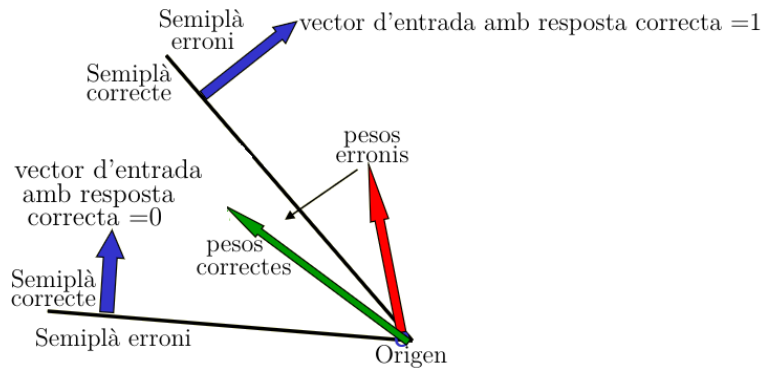


Figura 20: Espai de pesos. Imatge extreta de [11]

Si hi ha vectors de pesos que són correctes per a tots els casos, vol dir que es troben a l'*hipercon* que té el vèrtex a l'origen. Veure *Figura 20*. És a dir, el promig de dos vectors de pesos correctes és un vector de pesos correcte. Així doncs el problema és convex.

5 Xarxes neuronals

5.1 Arquitectura de les xarxes neuronals

Les xarxes neuronals es modelen com col·leccions de neurones connectades en un graf acíclic. Com hem explicat en el model biològic i matemàtic d'una neurona, les sortides d'unes neurones poden ser les entrades d'altres. No pot haver-hi cicles perquè això implicaria un bucle infinit en la circulació d'informació per la xarxa. Les neurones de la xarxa estan organitzades en diferents capes de la xarxa.

5.1.1 Xarxa neuronal completament connectada.

El tipus de capa més habitual és la connectada completament (fully-connected layer), on les neurones d'una mateixa capa no estan connectades entre elles però les neurones de dues capes adjacents estan connectades totes les unes amb les altres.

Aquestes xarxes calculen/computen una sèrie de transformacions entre entrada i sortida, i a cada capa obtenim una nova representació de l'entrada en la que s'han canviat alguns aspectes.

L'activitat de les neurones a cada capa és una funció no lineal de les activitats de la capa anterior.

Notem un detall important de notació, i és que quan parlem de xarxes neuronals N -capa, no estem comptant la capa d'entrada. Tota xarxa neuronal té capa d'entrada i de sortida, tot i que normalment l'estructura de les xarxes consta de més capes intermitges, anomenades **capes ocultes**. Per tant, una xarxa neuronal uni-cap, serà una xarxa neuronal que no té capes ocultes i, per tant, té l'entrada directament connectada a la sortida.

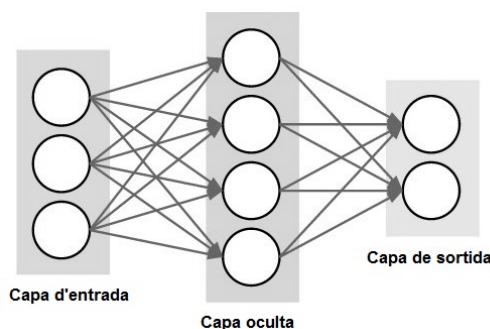


Figura 21: Xarxa completament connectada amb una capa oculta.

A diferència de la resta de capes de les xarxes neuronals, en general la capa de sortida no té una funció d'activació. També podem pensar-ho com que la seva

funció d'activació és lineal; la funció identitat. Això passa perquè normalment la última capa representa els valors de les puntuacions per a cada classe en casos de classificació o algun valor real objectiu pels casos de regressió.

Ens interessa saber quina paràmetres podem usar per descriure una xarxa neuronal. Els més habituals són el nombre de capes, evidentment, i també el nombre de neurones i la quantitat de paràmetres per aprendre. Ho veurem millor mitjançant un exemple basant-nos en la figura següent:

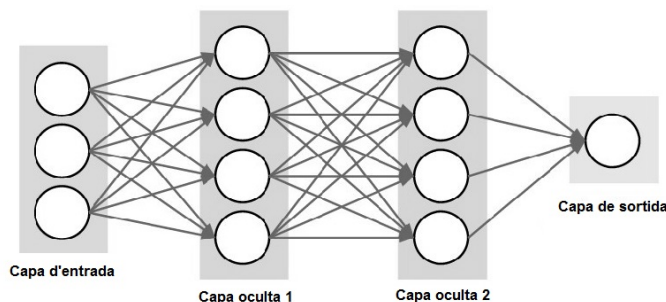


Figura 22: Xarxa completament connectada amb dues capes ocultes.

En aquesta xarxa tenim dues capes ocultes, $4 + 4 + 1 = 9$ neurones. També trobem $(3 \times 4) + (4 \times 4) + (4 \times 1) = 12 + 16 + 4 = 32$ pesos i $4 + 4 + 1 = 9$ biaixos, que fan un total de 41 paràmetres per aprendre.

Per posar-nos en context, les xarxes neuronals convolucionals modernes tenen de l'ordre de 100 milions de paràmetres i consten d'entre deu i vint capes ocultes. D'aquí ve el terme *deep learning*.

Com explica Michael Nielsen a [4], les xarxes neuronals amb, com a mínim una capa oculta, són aproximadors universals. És a dir, donada tota funció contínua $f(x)$ i un $\epsilon > 0$, existeix una xarxa neuronal $g(x)$ amb una capa oculta, i la seva corresponent funció d'activació, tal que $\forall x, |f(x) - g(x)| < \epsilon$. És a dir, la xarxa neuronal pot aproximar tota funció contínua.

I la gran pregunta que sorgeix un cop llegit aquest resultat és: Si amb una capa oculta n'hi ha prou per aproximar qualsevol funció contínua, per què pot servir afegir-ne d'altres? Doncs tot i ser un resultat matemàticament correcte i interessant, a l'hora d'aplicar-ho resulta poc pràctic. El fet que xarxes neuronals més profundes funcionen millor que les que disposen d'una sola capa oculta és una observació empírica, tot i que el seu poder d'aproximació i representació de funcions sigui igual.

En general i a la pràctica, les xarxes neuronals 3-capa funcionen millor que les 2-capa, però agafar més profunditat no té perquè ajudar. Això contrasta amb les xarxes neuronals convolucionals, on la profunditat resulta un factor molt important per un bon sistema de reconeixement, principalment perquè les imatges contenen estructures jeràrquiques. Per tant, múltiples capes de processament prenen sentit per aquest tipus d'informació.

En general, és millor tenir una xarxa amb massa capes ocultes que no amb massa poques. Si ens quedem curts, el model pot no tenir prou flexibilitat per captar les no-linealitats de les dades, però si ens passem de llarg, els pesos de més poden quedar reduïts a 0. Normalment el nombre de capes ocultes és entre 5 i 100. Aquest nombre augmenta a mida que tenim més dades d'entrada i casos d'entrenament. Tot i que sembli estrany, l'elecció de la quantitat de capes ocultes sovint ve donada per l'experiència i coneixements previs. És a dir, no tenim un procediment clar per definir-ne el nombre concret. [7]

Hi ha molta matèria sobre la profunditat de les xarxes neuronals i la seva eficiència. Recomano lectures [5] i capítol 6.4 de [6].

5.1.2 Xarxes recurrents

Tenen grafs dirigits en el seu graf de connexions. Això significa que puntualment es pot tornar enrere i visitar una neurona on ja havíem estat. Són molt més potents que Feed-Forward NN. Però poden tenir dinàmiques molt complicades i això les fa unes xarxes difícils d'entrenar.

Són biològicament realistes i una forma molt natural de modelar dades seqüencials. Les unitats amagades actuen com una xarxa molt profunda. En aquest tipus d'estructures arrosseguem tota l'estona els mateixos pesos.

Tenen l'habilitat de recordar informació en el seu estat amagat per un llarg període de temps, tot i que és difícil d'entrenar-les per obtenir aquesta habilitat.

5.1.3 Xarxes connectades simètricament

Són com xarxes recurrents però les connexions entre unitats són simètriques, tenen el mateix pes en ambdues direccions. Són molt més simples d'analitzar que les recurrents. Les seves capacitats queden també molt restringides.

Les xarxes simètricament connectades sense unitats amagades s'anomenen "*xarxes Hopfield*".

5.2 Aprenentatge

Per tal d'obtenir un aprenentatge, el primer que necessitem és un conjunt de dades a partir del qual aprendre'n. Aquest conjunt l'anomenem conjunt de dades d'entrenament, o *training data set*. Per descomptat, quan comprovem l'efectivitat del nostre aprenentatge, no ho farem sobre aquestes mateixes dades d'entrenament. Necessitem un altre conjunt de dades, que la màquina no vegi quan l'entrenem, que anomenem conjunt de dades de prova, *test data set*.

El procés de convergència dels perceptrons funciona assegurant que cada cop que els pesos canvien, s'acosten al conjunt de pesos factibles. Això, però, no

pot ser extès a altes xarxes més complicades on ens podem trobar que passos intermitjos de bones solucions donen una mala solució. Per tant, per xarxes neuronals multi-capa no usarem el procediment d'aprenentatge del perceptró. En aquests casos necessitarem una altra forma de comprovar que el procés està progessant. En comptes de veure que els pesos s'acosten a un bon conjunt de pesos, veurem que els valors de l'output s'acosten a uns valors objectiu.

Per assolir un bon aprenentatge, volem un algorisme que ens permeti trobar pesos tals pels que les dades de sortida de la xarxa s'aproximin a $y(x)$ per a tots els valors d'entrenament d'entrada, x .

L'exemple més senzill és una neurona lineal amb mesura d'error quadràtic. Aquesta neurona té per output un valor real que és la suma dels inputs amb pesos.

$$y = \sum_i w_i x_i = w^T x$$

On y és l'estimació de la neurona per l'output desitjat, w és l'output de pesos i x és el vector d'entrada.

L'objectiu de l'aprenentatge és minimitzar l'error sumat en tots els casos d'entrenament. Anomenem E el conjunt de dades d'entrenament. Per quantificar l'assoliment de l'objectiu necessitem una funció de cost:

$$C(w) = \frac{1}{2} \sum_{x \in E} (t_x - y_x)^2$$

on w denota el conjunt de tots els pesos de la xarxa, N és el nombre total d'exemples d'entrenament, t_x denota el valor objectiu per les dades d'entrenament x i y_x és el valor obtingut a la sortida de la xarxa per les dades x .

Anomenem C la funció de cost quadràtica, també anomenada *mean squared error* o simplement *MSE*.

$C(w)$ és sempre no negativa, ja que tots els termes de la suma són no negatius. A més, $C(w) \approx 0$ precisament quan t_x és aproximadament igual als valors de sortida y_x per a totes les dades d'entrenament. Per tant el nostre algorisme d'entrenament treballa bé si és capaç de trobar pesos w tals que $C(w) \approx 0$. Per contra, no ho fa tant bé si obtenim $C(w)$ gran. Això significa que y_x no és prou a prop de l'objectiu t_j per gran part de les dades d'entrada.

Així doncs, podem afirmar que l'objectiu del nostre algorisme d'aprenentatge és minimitzar la funció de cost $C(w)$, és a dir, volem trobar valors per w que produeixin el mínim cost possible.

Així doncs ens trobem amb un problema d'optimització i l'abordarem mitjançant un algorisme anomenat *stochastic gradient descent*.

Farem una aproximació a diferents estratègies d'aprenentatge.

Donada la simplicitat de la comprovació de l'efectivitat d'un conjunt de pesos W , la primera idea que ens pot venir al cap és simplement provar molts pesos aleatoris diferents i anar-nos quedant amb el que funcioni millor. No és difícil imaginar que hi ha altres mètodes més rigorosos i efectius per assolir tal objectiu que no pas esperar que l'atzar ens otorgui els valors que minimitzin la funció de cost.

D'aquesta primera idea ens quedarem només amb el fet de generar el primer conjunt de pesos de forma aleatòria i, a partir d'aquí, anar refinant-ne els valors.

Introduïm ara una analogia molt usada: l'excursionista dels ulls tapats. Suposem que ens trobem en un terreny amb pendent, tenim els ulls tapats i sabem que hem d'arribar al punt més baix.

La següent idea que ens pot venir al cap és, suposant que som l'excursionista dels ulls tapats, anar provant amb el peu tots els punts del voltant on som per decidir cap a quina direcció el terreny fa baixada. Fer una passa en aquella direcció i repetir el procediment. Això podem expressar-ho com: partim d'uns pesos W aleatoris, hi generem una pertorbació δW i si la funció de cost en aquest punt $W + \delta W$ és menor, actualitzem els pesos. Tot i arribar al resultat desitjat, no deixa de ser computacionalment car.

Però la veritat és que no cal tamptejar el terreny amb el peu per trobar la direcció de baixada. Podem calcular-la directament ja que aquesta direcció és el gradient de la funció de cost. En el cas de l'excursionista, aquest cas correspondria en parar els peus, sentir cap a on fa baixada el terreny (sense provar per tot el seu voltant) i caminar cap allà.

Una forma d'atacar el problema és usant eines de càlcul per provar de trobar el mínim analíticament. En aquest cas podríem calcular les derivades i usar-les per trobar els valors on C és un extrem. Aquesta aproximació pot funcionar si tenim poques variables, però treballant amb xarxes neuronals sovint tenim moltes més variables de les que podríem assumir per fer els càlculs directament. Per tant descartem l'opció de l'anàlisi per resoldre el problema.

Definim el gradient de $C(w)$ com el vector de les derivades parcials $(\frac{\partial C}{\partial w_1}, \dots, \frac{\partial C}{\partial w_m})^T$

Suposem que C és una funció de m variables. Aleshores la variació ΔC en C produïda per la variació $\Delta v = (\Delta v_1, \Delta v_2, \dots, \Delta v_m)^T$ és:

$$\Delta C \approx \nabla C \cdot \Delta v, \quad (5.1)$$

on el gradient ∇C és el vector $(\frac{\partial C}{\partial w_1}, \dots, \frac{\partial C}{\partial w_m})^T$.

En particular, podem triar $\Delta v \equiv -\eta \nabla C$, on η és un paràmetre petit i positiu, que anomenem taxa d'aprenentatge (o *learning rate*). Així, garantim que l'apro-

ximació a l'expressió (5.1) serà negativa, com volíem. Ara tenim una forma de seguir el gradient cap al mínim, tot i tenir una funció C de diverses variables, que serà aplicar repetidament la regla d'actualització:

$$w \rightarrow w' = w - \eta \nabla C \quad (5.2)$$

Podem pensar en aquesta regla com la forma de definir l'algorisme de descens de gradient (*gradient descent*), ja que ens dona una forma repetida d'anar modificant w amb l'objectiu de trobar el mínim de la funció C . Cal destacar, però, que tot i que aquesta regla no sempre funciona correctament, a la pràctica el descens de gradient, en general funciona molt bé. Aplicada a xarxes neuronals, és una eina molt potent per minimitzar la funció de cost i, per tant, per ajudar la xarxa a aprendre.

Per tant, el descens de gradient es pot veure com una forma d'anar fent petites passes cap a la direcció que indica el gradient.

Notem que la funció de cost és de la forma $C = \frac{1}{N} \sum_x C_x$, és a dir, la mitjana sobre les funcions de cost $C_x \equiv \frac{(t_x - y_x)^2}{2}$ per cada exemple de dades d'entrenament. A la pràctica, per calcular el gradient ∇C hem de calcular cadascun dels ∇C_x per separat per cada entrada de dades d'entrenament, x , i després fer-ne la mitja sobre la quantitat d'entrenaments fets, $\nabla C = \frac{1}{N} \sum_x \nabla C_x$. Desafortunadament, quan tenim una gran quantitat de dades d'entrenament això pot tenir un cost temporal molt elevat, i l'aprenentatge es desenvolupa molt lentament.

Però hi ha una idea que pot ser usada per fer avançar aquest aprenentatge, l'anomenat descens del gradient estocàstic. Consisteix en aproximar el valor de ∇C calculant ∇C_x només per una petita part, triada a l'atzar, dels entrenaments realitzats. Fent la mitjana aritmètica sobre aquesta petita mostra podem trobar una aproximació ràpida i fiable del gradient real ∇C .

És a dir, etiquetem els entrenaments triats a l'atzar com $X_1, X_2, \dots, X_m$ i els anomenem mini-batch. Donada una mida de la mostra m prou gran, esperem que la mitjana aritmètica de ∇C_{x_j} aproximarà bé el valor ∇C_x , per tant:

$$\frac{\sum_{j=1}^m \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C,$$

on la segona suma és sobre tot el conjunt de dades d'entrenament.

Així doncs, tenim:

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^m \nabla C_{X_j},$$

fet que confirma que podem aproximar el gradient total a partir d'una part, prou representativa i triada a l'atzar, de mini-batch. Això, representat sobre xarxes neuronals és:

$$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}, \quad (5.3)$$

on le sumes són sobre tots els exemples d'entrenament X_j del mini-batch. Després prenem un altre mini-batch triat a l'atzar i entrenem la xarxa amb aquests nous valors. Així fins a gastar les dades del conjunt d'entrenament, quan es diu que s'ha completat una fase d'entrenament. En aquest punt, comencem amb una altra fase.

Fins ara hem vist que les xarxes neuronals poden aprendre els seus pesos usant l'algorisme de descens de gradient. El que no hem vist encara és precisament com fer el càlcul del gradient de la funció de cost. Ara veurem un algorisme ràpid per fer obtenir-lo: l'algorisme de propagació cap enrere (o *backpropagation*).

5.2.1 L'algorisme de propagació cap enrere

Van ser *Rumelhart*, *Hinton* i *Williams* que, a mitjans dels anys 80, van donar a conèixer l'algorisme conegut com algorisme de retropropagació o propagació enrere, en anglès *back-propagation algorithm*. Segons [9] *és un algorisme d'aprenentatge supervisat utilitzat pels perceptrons multicapa que ajusta els pesos de les connexions de totes les capes de la xarxa neuronal artificial de manera proporcional a l'error en les neurones de sortida*. I, com s'explica a [10], té com a objectiu trobar un conjunt de pesos que minimitzi l'error comès al classificar patrons.

A grans trets, podem distingir-ne dues parts:

- 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.

Comencem per deixar clara la notació. Usarem w_{jk}^l per denotar el pes de la connexió que uneix la k -èsima neurona de la capa $l - 1$ amb la j -èsima neurona de la capa l .

Usem a_j^l per representar l'activació de la neurona j -èsima a la capa l -èsima.

Amb aquesta notació, l'activació a_j^l de la neurona j -èsima de la capa l -èsima està relacionada amb les activacions de la capa $l - 1$ per l'equació:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1}\right), \quad (5.4)$$

on la suma és sobre totes les neurones k de la capa $l - 1$ -èsima.

Definim la matriu de pesos w^l per cada capa l . Els valors d'aquesta matriu són els pesos que arriben a les neurones de la capa l -èsima. És a dir, el valor

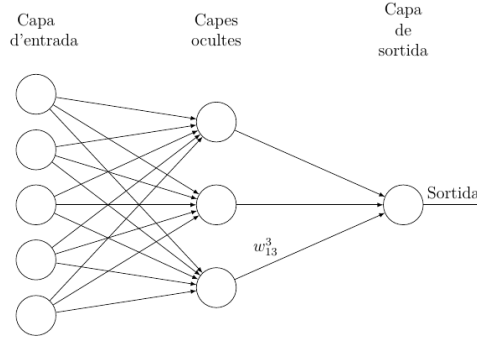


Figura 23: w_{13}^3 indica el pes que uneix la 3a neurona de la capa (3-1)=2, amb la 1a neurona de la capa 3.

a la fila j i la columna k és w_{jk}^l . Definim també el vector d'activació a^l que té per components els valors a_j^l . Sigui v un vector qualsevol, les components de $\sigma(v)$ són $\sigma(v)_j = \sigma(v_j)$ i denoten l'aplicació de la funció σ sobre v , element per element. Així doncs ja tenim prou eines per reescriure l'equació 5.4 en la forma vectorial compacta:

$$a^l = \sigma(w^l a^{l-1}), \quad (5.5)$$

Aquesta expressió ens dóna una idea global de com les activacions d'una capa estan relacionades amb les de la capa anterior: apliquem la matriu de pesos i hi afegim la funció d'activació que correspongui σ , sempre i quan no estiguem a la capa de sortida.

De fet, a 5.5, per calcular a^l , primer fem el càlcul intermedi $z^l = w^l a^{l-1}$. Aquesta quantitat correspon a l'entrada ponderada a les neurones de la capa l . Serà útil més endavant, per tant val la pena tenir-la en compte.

Per tant,

$$\begin{aligned} a^l &= \sigma(z^l) \\ z_j^l &= \sum_k w_{jk}^l a_k^{l-1} \end{aligned} \quad (5.6)$$

L'algorisme de propagació cap enrere requereix suma de vectors i producte matricial, però també necessitem una operació especial i menys coneguda. Suposem que tenim dos vectors, s i t , de la mateixa dimensió, aleshores usem $s \odot t$ per denotar el producte element per element, a l'igual que fa amb matrius de la mateixa mida. Aquest producte element per element s'anomena *producte de Hadamard* o de *Schur*.

$$\text{A mode d'exemple entre vectors: } \begin{pmatrix} a \\ b \end{pmatrix} \odot \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a*c \\ b*d \end{pmatrix}$$

$$\text{i entre matrius: } \begin{pmatrix} a & b \\ c & d \end{pmatrix} \odot \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} a*e & b*f \\ c*g & d*h \end{pmatrix}$$

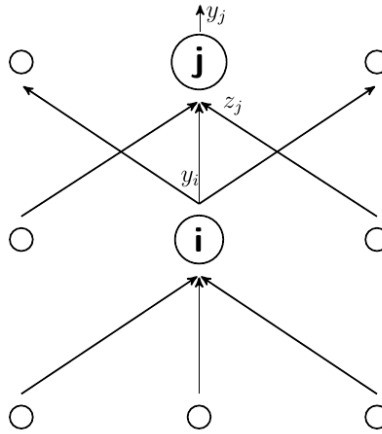


Figura 24: Ampliació d'una xarxa neuronal per posar-nos en situació.

Anem a veure com funciona l'algorisme de propagació cap enrere per un únic cas d'entrenament:

- Primer fem el pas de les dades cap endavant, *forward*.
- Definim la funció usada per calcular l'error. En aquest cas serà la funció de cost.

$$E = \frac{1}{2} \sum_{j \in \text{output}} (t_j - y_j)^2$$

- Convertim la discrepància entre el valor de sortida i el valor desitjat en la derivada de l'error.

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$

- La base de l'algorisme de propagació cap enrere és calcular les derivades de l'error d'una capa, començant per la de sortida i, a partir d'aquesta, calcular les derivades de l'error a la capa que la precedeix. Per tant ara volem calcular les derivades de l'error per cada unitat de les capes ocultes a partir de les derivades de la capa superior. És a dir, un cop tenim la derivada respecte la sortida d'una de les neurones de la capa de sortida, volem usar totes aquestes derivades de les neurones de sortida per calcular les derivades de les neurones de la capa adjacent inferior.

Fem pas a pas com es propaga la derivada error respecte la sortida d'una unitat/neurona.

Considerem una unitat de sortida j , i una unitat i que pertany a una capa oculta, com a la *Figura 24*. La sortida de la unitat oculta és y_i , mentre que la

sortida de la unitat j és y_j . El total d'informació rebuda per la unitat j és $z_j = \sum_i w_{ij} y_i$.

El primer que hem de fer és passar de la derivada respecte l'error respecte a y_j a una derivada de l'error respecte a z_j . Per fer-ho usarem la regla de la cadena.

La derivada de l'error respecte el total d'informació d'entrada a y_j és:

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{dy_j}{dz_j} = \frac{\partial E}{\partial y_j} \mathbb{1}_{y_j=z_j}$$

recordem que $y_j = f(z_j) = \text{relu}(z_j) = \begin{cases} z_j & \text{si } z_j > 0 \\ 0 & \text{altrament} \end{cases}$

per tant, $\frac{dy_j}{dz_j} = f'(z_j) = \text{relu}'(z_j) = \begin{cases} 1 & \text{si } z_j > 0 \\ 0 & \text{altrament} \end{cases}$

Ara podem calcular la derivada de l'error respecte la sortida de la unitat i . Serà la suma de les derivades dels errors de les sortides de la unitat i :

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j} \frac{dy_j}{dy_i} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

Ara ja tenim les eines necessàries per calcular la derivada de l'error respecte tots els pesos que arriben a la unitat j .

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$

Aquestes equacions ens proporcionen una forma de calcular el gradient de la funció de cost. Anem a escriure-ho explícitament:

1. **Entrada x :** Establir l'activació a^1 per la capa d'entrada.
2. **Alimentar la xarxa endavant:** Per a cada $l = 2, 3, \dots, L$ calcular $z^l = w^l a^{l-1}$ i $a^l = \sigma(z^l)$.
3. **Error de sortida:** Calculem l'error generat a la capa de sortida. $E = \frac{1}{2} \sum_{j \in \text{output}} (t_j - y_j)^2$
4. **Propagar l'error cap enrere:** Per a cada $l = L - 1, L - 2, \dots, 2$ calcular

$$\begin{aligned} \frac{\partial E}{\partial z_j} &= \frac{\partial E}{\partial y_j} \mathbb{1}_{y_j=z_j} \\ \frac{\partial E}{\partial y_i} &= \sum_j w_{ij} \frac{\partial E}{\partial z_j} \\ \frac{\partial E}{\partial w_{ij}} &= y_i \frac{\partial E}{\partial z_j} \end{aligned}$$

5. **Sortida:** El gradient de la funció de cost ve donat per l'últim $\frac{\partial E}{\partial w_{ij}}$.

L'algorisme de propagació cap enrere calcula el gradient de la funció de cost per un sol exemple d'entrenament, $C = C_x$. A la pràctica és comú combinar la propagació cap enrere amb algun algorisme d'aprenentatge com descens de gradient estocàstic, a on calculem el gradient per diversos exemples d'entrenament. En particular, donat un conjunt de m exemples d'entrenament, l'algorisme següent realitza l'aprenentatge de descens de gradient basat en mini-batch.

1. **Fer-hi entrar un conjunt d'exemples.**

2. **Per a cada bloc de dades d'entrenament:**

- **Entrada x :** Establir l'activació a^1 per la capa d'entrada.
- **Alimentar la xarxa endavant:** Per a cada $l = 2, 3, \dots, L$ calcular $z^{x,l} = w^l a^{x,l-1}$ i $a^{x,l} = \sigma(z^{x,l})$.
- **Error de sortida:** Calculem l'error generat a la capa de sortida. $E = \frac{1}{2} \sum_{j \in \text{output}} (t_j - y_j)^2$
- **Propagar l'error cap enrere:** Per a cada $l = L - 1, L - 2, \dots, 2$ calcular

$$\begin{aligned}\frac{\partial E}{\partial z_j} &= \frac{\partial E}{\partial y_j} 1_{y_j=z_j} \\ \frac{\partial E}{\partial y_i} &= \sum_j w_{ij} \frac{\partial E}{\partial z_j} \\ \frac{\partial E}{\partial w_{ij}} &= y_i \frac{\partial E}{\partial z_j}\end{aligned}$$

- **Sortida:** El gradient de la funció de cost ve donat per l'últim $\frac{\partial E}{\partial w_{ij}}$.

3. **Descens de gradient:** Per a cada capa $l = L, L - 1, \dots, 2$ actualitzar els pesos d'acord amb la regla

$$w^l = w^l - \frac{\eta}{m} \sum_x \frac{\partial E}{\partial w_{ij}}$$

5.2.2 El mètode del moment

Imaginem una bala de vidre sobre la superfície d'error. La posició de la bala en el pla horitzontal representa el vector de pesos. Inicialment la bala és estàtica i, per tant, començarà a moure's en la direcció de més desnivell; seguirà el gradient. Però tan bon punt agafi velocitat, ja no seguirà la direcció del gradient. El seu moment farà que segueixi en la direcció anterior.

El moment ajuda a esmorteir les oscil·lacions en les direccions de gran curvatura combinant el gradient amb signes contraris. S'acumula velocitat en direccions amb un pendent suau però constant.

$$v(t) = \alpha v(t-1) - \epsilon \frac{\partial E}{\partial w}(t) \quad (5.7)$$

on $v(t)$ és el vector velocitat a temps t . El temps t representa cada actualització dels pesos. L'efecte del gradient és incrementar la velocitat prèvia. La velocitat decau per un factor α que serà lleugerament inferior a 1.

$$\begin{aligned} \Delta w(t) &= v(t) \\ &= \alpha v(t-1) - \epsilon \frac{\partial E}{\partial w}(t) \\ &= \alpha \Delta w(t-1) - \epsilon \frac{\partial E}{\partial w}(t) \end{aligned} \quad (5.8)$$

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

Cal tenir molta cura a l'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. Val la pena que a l'inici tinguem un moment petit, per exemple $\alpha = 0.5$. Una vegada la xarxa s'ha desfet dels gradients elevats i els pesos estan atrapats en un *barranc* (és a dir, hem assolit una fase normal d'aprenentatge), ens interessa augmentar lleugerament el moment fins arribar a un valor final de $\alpha = 0.9$ o fins i tot $\alpha = 0.99$.

La combinació d'una taxa d'aprenentatge petita i un moment elevat permet arribar més lluny que no pas si treballem amb una taxa d'aprenentatge elevada sense moment. Usant una taxa d'aprenentatge elevada arribarem a oscil·lacions divergents a través del barranc.

El moment de Nesterov (1983), una versió millorada.

El mètode estàndard del moment primer calcula el gradient a la posició actual i després fa el salt cap a la direcció del gradient actualitzat. Però Ilya Sutskever ha descobert recentment que hi ha una nova forma de moment que en general ofereix millors resultats. Aquest nou mètode es basa en el mètode de Nesterov per optimitzar funcions convexes. La proposta és la següent:

- Primer fem el salt en la direcció del gradient anterior.
- Després mesurem el gradient del punt on hem acabat i corregim.

Com podem veure, es basa en la idea que és millor corregir una errada un cop s'ha fet.

Resulta difícil percebre les diferències entre els dos mètodes del moment. Una bona forma és veure-les representades.

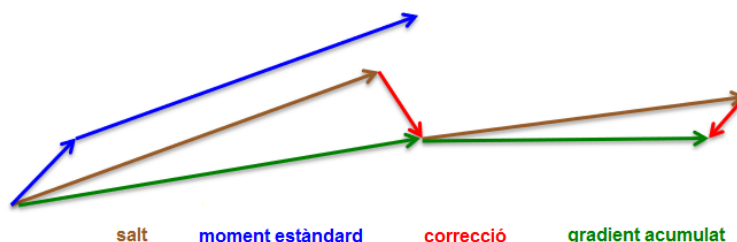


Figura 25: Comparació del moment estàndard i el de Nesterov

5.2.3 Dropout: una forma eficient de combinar xarxes neuronals

És una forma de combinar models de xarxes neuronals molt grans sense necessitat d'entrenar separatament un gran nombre de models i ha esdevingut molt exitós en competicions recents.

Per cada cas d'entrenament ometem a l'atzar una part de les unitats de les capes ocultes, acabant així amb una arquitectura de xarxa diferent per a cada cas. Podem entendre-ho com si tinguéssim un model diferent per a cada entrenament.

Però, si ometem part dels nodes, com podem entrenar el model amb un sol cas d'entrenament? La resposta ve donada pel compartiment de pesos.

Comencem veient dues formes diferents de combinar les sortides de mòduls diferents.

Barreja fent la mitjana

Podem combinar models fent la mitjana de les seves probabilitats de sortida.

Model A: .3.2.5

Model B: .1.8.1

Combinació.2.5.3

Producte

Podem combinar models prenent la mitjana geomètrica de les seves probabilitats de sortida.

Model A: .3 .2 .5

Model B: .1 .8 .1

Combinació $\sqrt{.3}/sum \sqrt{.16}/sum \sqrt{.05}/sum$

Multipliquem cada parell de probabilitats i després en fem l'arrel.

Com podem fusionar tots aquests models eficientment a temps de prova?

Considerem una xarxa neuronal amb una sola capa oculta. Cada cop que prenem unes dades d'entrenament, ometem cada neurona de la xarxa amb una probabilitat de 0.5. I fem l'entrenament de la xarxa amb l'absència d'aquestes unitats. Això vol dir que prenem una mostra a l'atzar de 2^H arquitectures diferents, on H és el nombre d'unitats ocultes. És una quantitat molt gran d'arquitectures, i totes elles comparteixen pesos.

Per tant, podem entendre el dropout com una equiparació de models.

Prenem una mostra d'entre 2^H models, d'on la majoria mai seran triats. Per tant, molt pocs són entrenats i només una vegada. La compartició de pesos entre models significa que cada model és fortament regularitzat per la resta.

I què fem al moment de prova? Podríem prendre com a mostra moltes arquitectures diferents i fer la mitjana geomètrica de les distribucions de sortida. Però això seria molta feina.

Obtenim millors resultats si usem totes les capes ocultes però reduïm a la meitat els seus pesos de sortida. Així obtenim el mateix efecte que prenent la mostra. Això és perquè usant totes les unitats ocultes i prenent la meitat de pesos de sortida és equivalent al càlcul de la mitjana aritmètica de les prediccions que tots els 2^H models haurien realitzat.

I si tenim més capes ocultes, podem simplement usar un dropout de 0.5 a cada capa.

La capa d'entrada pot ser tractada com qualsevol altra però amb una petita variació, ja que també hi podem aplicar dropout però amb una probabilitat més elevada de quedar-nos amb cada unitat.

El sistema de reconeixement d'objectes desenvolupat per Alex Krizhevsky [8], a part d'haver batut rècords, usa dropout.

Si tenim una xarxa neuronal profunda que sobreajusta els valors, la tècnica del dropout en general ens farà reduir dràsticament l'error produït. De fet, tota xarxa que requereixi que el seu entrenament sigui aturat a la força abans d'acabar, funcionarà millor si hi apliquem dropout, tot i que pot costar més temps per entrenar i pot requerir més unitats ocultes.

Però si la nostra xarxa neuronal no està sobreestimant els valors, segurament hem de provar d'usar-ne una de més gran i aplicar-hi dropout. Això ho podem fer suposant, per descomptat, que tenim prou potència computacional.

6 Classificació d'imatges

El problema de la classificació d'imatges consisteix en assignar a una imatge d'entrada una etiqueta d'entre una col·lecció fixa de categories. Aquest és un dels problemes centrals de la Visió per Ordinador que, tot i semblar simple, té una gran varietat d'aplicacions útils. A més, altres tasques importants de la Visió per Ordinador poden ser reduïdes o enteses com classificació d'imatges.

El sistema de visió humana és una gran meravella natural. A cada hemisferi del cervell tenim una escorça visual primària, també anomenada V_1 , que conté 140 milions de neurones, amb deu bilions d'interconnexions. Però la visió humana no només involucra V_1 , sino tota una sèrie d'escorces visuals, V_1, V_2, V_3, V_4, V_5 , que executen processament d'imatges progressivament més complexes. Això vol dir, que carreguem permanentment al nostre cap un superordinador, magníficament adaptat per entendre el món visual i sintonitzat per l'evolució al llarg de centenars de milions d'anys.

El reconeixement d'objectes resulta una tasca senzilla pels humans, per això d'entrada pot semblar un problema trivial per ser resolt per una màquina. Cal tenir en compte que hi ha molts aspectes que influeixen en com es representen els objectes com ara la variació del punt de vista, les condicions d'il·luminació, la variació d'escala i molts d'altres.

La dificultat del problema de reconeixement de patrons visuals esdevé aparent si provem de programar el nostre propi programa per reconèixer dígit escrit a mà. D'entrada podríem pensar el reconeixement de les formes com quan ho expliquem a infants que aprenent els nombres: el 5 té una panxa i un barret, el 9 té una volta a dalt i un pal a baix a la dreta. Però no és simple ni pràctic per expressar-li a l'ordinador. Si volem programar-ho, ràpid obtindrem gran quantitat d'excepcions i casos especials.

Les xarxes neuronals encaren el problema d'una forma diferent. La idea és agafar una gran quantitat de dades d'entrada per desenvolupar-ne l'entrenament. Aquestes dades d'entrada, en el cas del reconeixement de dígit seran mostres de dígit escrit a mà.

Però l'arquitectura de les xarxes neuronals artificials no té en compte l'estructura espacial de les imatges. Per exemple, tracta els píxels d'entrada que estan separats i els que estan ben a prop com exactament al mateix nivell. Aquests conceptes d'estructura espacial s'han de deduir de les dades d'entrenament. En aquest moment és quan entren en joc les xarxes convolucionales profundes, ja que tenen una arquitectura especial que està particularment ben adaptada per a la classificació d'imatges. Aquest nou tipus de xarxes donen per un altre treball final de grau, així que no entrarem en detall. Simplement saber que les xarxes convolucionales profundes són ràpides d'entrenar, són molt bones classificant imatges i, per això, són el tipus de xarxa que s'utilitza per a la majoria de reconeixements d'imatges.

7 Prova de concepte

7.1 Descripció del problema i les dades

Només l'any 2013, *The United States Postal Services* va processar més de 150 milions d'enviaments. Un volum massa gros per ser classificats manualment. No cal mirar molts anys enrere, quan al 1997 només el 10% del correu postal va arribar correctament a la seva destinació de forma automàtica. Així doncs, el problema està servit. La necessitat d'un sistema capaç de reconèixer dígit i caràcters escrits a mà resulta evident.

A través del *feedback*, el Servei Postal va ser capaç d'entrenar equips automàtics per la lectura amb precisió de l'escriptura humana. Avui en dia, més del 98% de tot el correu és processat correctament amb l'ajuda de l'aprenentatge automàtic i les màquines.

A partir d'aquesta aplicació busquem crear un sistema que sigui capaç de reconèixer dígit escrits a mà. Per fer-ho, implementarem l'algorisme de propagació cap enrere presentat a la memòria d'aquest treball.

Les dades les obtenim de la llibreria *Scikit Learn* ja que conté el conjunt de dades *digits*. Aquestes dades corresponen a una mostra simplificada de la base de dades MNIST. La base de dades original, MNIST, conté desenes de milers d'imatges (de mida 28x28 píxels) de dígit escrits a mà escanejades, juntament amb les seves classificacions correctes. El nom, MNIST, es deu a que és un subconjunt modificat de dos conjunts de dades recollides per l'Institut Nacional d'Estàndards i Tecnologia dels Estats Units, *NIST*. MNIST conté les dades separades en dues parts. La primera conté 60000 imatges per usar a la fase d'entrenament, i la segona conté 10000 imatges per usar a la fase d'avaluació.

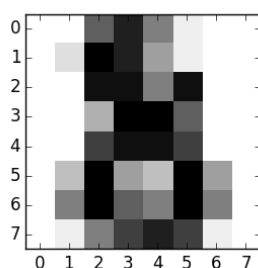


Figura 26: Imatge d'un dígit importat.

El conjunt importat conté 1797 imatges, cadascuna d'elles correspon a un dígit escrit a mà i té mida 8x8 píxels. Per tal de poder treballar amb les figures 8x8, necessitem transformar les dades en vectors de mida 64, que posteriorment usarem per entrenar i avaluar el sistema. Per això, i com a experiment d'aprenentatge automàtic, necessitem dos conjunts de dades de dígit escrits a mà a mode

d'exemple. El primer conjunt, *training set*, l'usarem per la fase d'entrenament i serà més nombrós que el segon *test set*, que serà usat per la fase d'avaluació. En aquest cas hem fet la divisió de les dades segons 75% *training* - 25% *test*.

Com ja sabem, el nostre objectiu és implementar un algorisme que ens permetri trobar valor pels pesos per tal que la sortida de la xarxa sigui $objectiu[i]$ per a totes les dades d'entrenament i , on el vector *objectiu* denota el vector dels valors reals dels dígit a reconèixer.

7.2 Arquitectura

Estem implementant el sistema de reconeixement de dígit manuscrits usant xarxes neuronals, i començarem per l'arquitectura més simple, formada per capa d'entrada, una sola capa oculta i la capa de sortida. Les dades d'entrada són imatges de mida 8x8 píxels, que hem vectoritzat. Per tant, les dades que entraran al sistema són vectors de mida 64, així doncs, la nostra capa d'entrada ha de tenir també 64 unitats.

La capa de sortida pretén predir de quin dígit es tracta, i ho farem amb una sola unitat que, té com a objectiu prendre valors enters entre 0 i 9, ambdós inclosos. Així doncs, a través del resultat de la unitat d'aquesta última capa, sabrem quin és el dígit reconegut pel sistema.

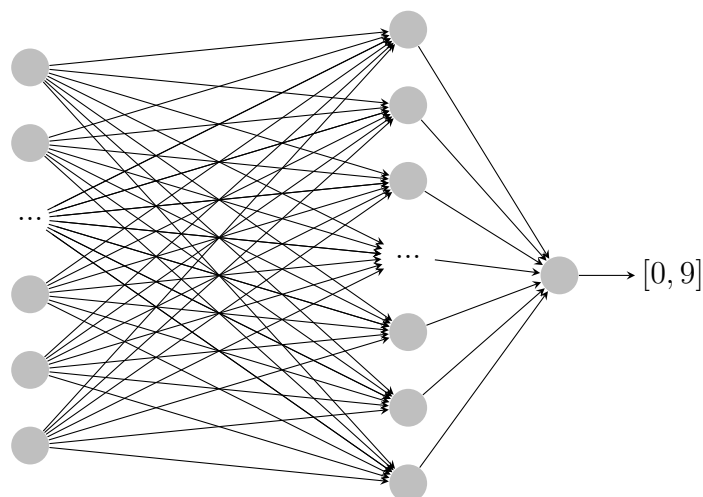


Figura 27: Xarxa neuronal de dues capes usada per entrenar el sistema de reconeixement de dígit. Té 64 unitats a la capa d'entrada i una única unitat de sortida.

La nostra xarxa serà doncs de la forma representada a la *Figura 1*, de la que variarem el nombre de nodes de la capa oculta segons els resultats obtinguts i les necessitats del problema.

Posteriorment, estudiarem els resultats obtinguts afegint una capa oculta a la xarxa. Així doncs, tindrem arquitectura $[64, m, n, 1]$, amb m el nombre d'unitats de la primera capa oculta, i n les unitats de la segona capa oculta.

7.3 Paràmetres experimentals i resultats

Hi ha paràmetres que queden fixats pels requeriments del problema i les dades. Els paràmetres que podem ajustar són la quantitat de capes ocultes, el nombre d'unitats de cada capa oculta, la taxa d'aprenentatge (*learning rate*) i el nombre d'iteracions que repetim el procediment d'aprenentatge.

Les eines que usarem per veure el rendiment del sistema són matrius de confusió per la fase d'entrenament i la fase de test, així com la representació gràfica de la funció de cost, la *LOSS*. Recordem que la matriu de confusió C per a problemes de classificació multiclasse és tal que l'element $C(i, j)$ és el nombre d'observacions que sabem que pertanyen a la classe i i s'han predit com a classe j . Busquem, doncs, aconseguir matrius de confusió diagonals, fet que representaria obtenir una classificació perfecta.

Pel que fa a la representació de la *LOSS*, cal tenir en compte que és una representació dels valors instantanis, obtinguts per una única mostra de les dades i , per tant, és molt sorollosa. Un bon comportament d'aquesta representació de la funció de cost correspon a la disminució del valor instantani.

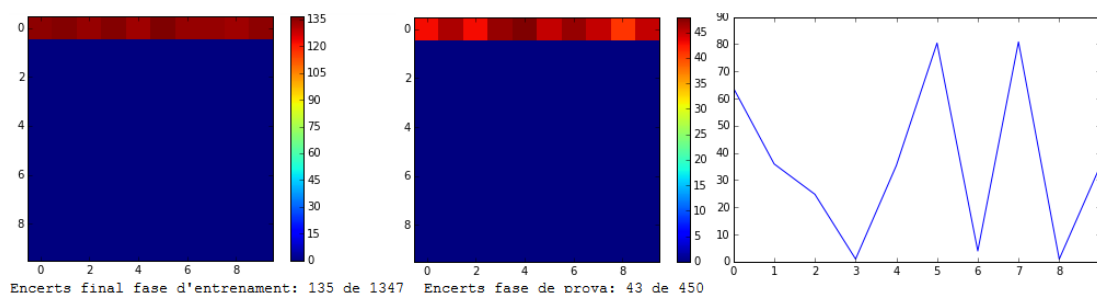


Figura 28: Esquerra: matriu de confusió per la fase d'entrenament. Mig: matriu de confusió per la fase d'avaluació. Dreta: gràfica de l'evolució de la funció de cost (*LOSS*). Tot amb nombre d'iteracions = 10.

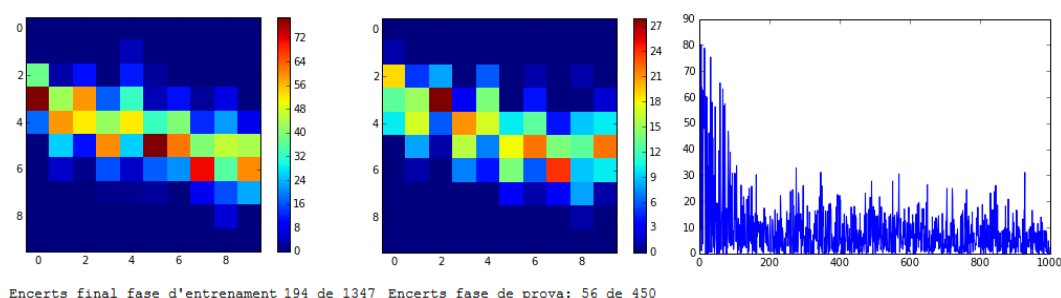


Figura 29: Esquerra: matriu de confusió per la fase d'entrenament. Mig: matriu de confusió per la fase d'avaluació. Dreta: gràfica de l'evolució de la funció de cost (*LOSS*). Tot amb nombre d'iteracions = 1000.

D'entrada el paràmetre que sembla menys evident que influeixi en l'aprenentatge pot ser el nombre d'iteracions. És per això que comencem observant els resultats obtinguts per una mateixa arquitectura i taxa d'aprenentatge però amb nombre d'iteracions diferents.

A *Figura 28* i la *Figura 29* observem les matrius de confusió de la fase test i de la fase entrenament, així com la gràfica de la LOSS. En tots dos casos estem implementant una arquitectura $[64,50,1]$ amb taxa d'aprenentatge de $1e - 4$. La diferència entre les dues imatges és el nombre d'iteracions, ja que la primera ha n'ha executat i la segona 1000. Així doncs, observem que el nombre d'iteracions sí resulta un paràmetre important per aconseguir un bon aprenentatge. A partir d'ara el tindrem en compte i valorem positivament que sigui de gran magnitud, sempre i quan executi l'algorisme amb un temps raonable.

Fem ara una comparativa semblant respecte la taxa d'aprenentatge.

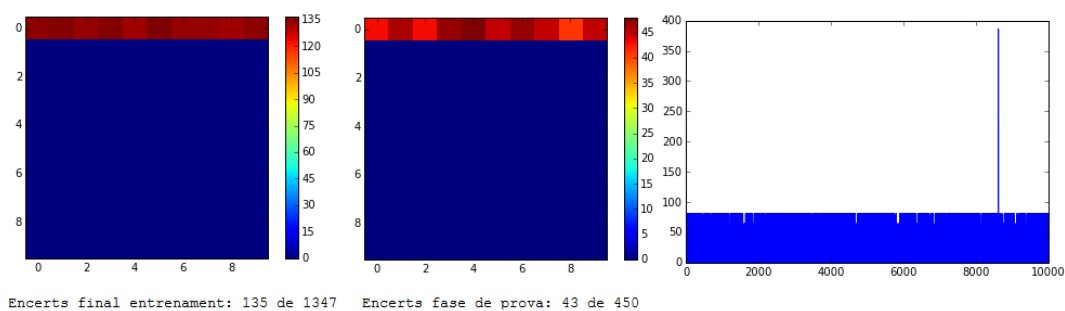


Figura 30: Matrius de confusió amb nombre d'iteracions = 10000, taxa d'aprenentatge 10^{-2} i arquitectura $[64,50,1]$.

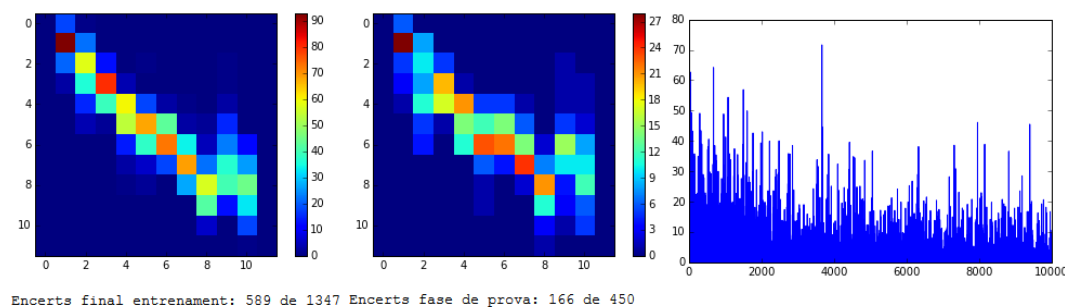


Figura 31: Matrius de confusió amb nombre d'iteracions = 10000, taxa d'aprenentatge 10^{-3} i arquitectura $[64,50,1]$.

Pel que fa a la taxa d'aprenentatge observem que un valor massa gran pot fer que el sistema no aprengui amb l'experiència, i un valor massa petit tampoc ens proporciona bons resultats. Així doncs, i basant-nos en l'experiència obtinguda, un bon valor per aquest paràmetre seria de l'ordre de $1e - 4$. Observem que el nombre d'iteracions ens interessa tenir-lo el més gran possible, però ens trobem

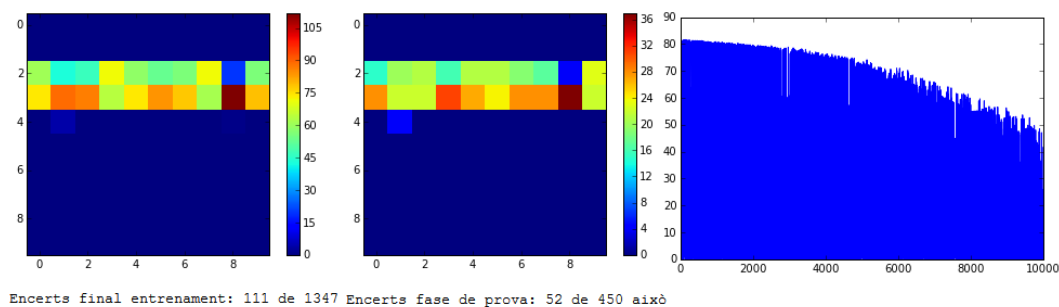


Figura 32: Matrius de confusió amb nombre d'iteracions = 10000, taxa d'aprenentatge 10^{-6} i arquitectura $[64,50,1]$.

amb la limitació del temps d'execució de l'experiment, ja que un valor massa elevat ens farà tardar molt en resoldre el problema. En canvi, però, la taxa d'aprenentatge sembla que queda acotada.

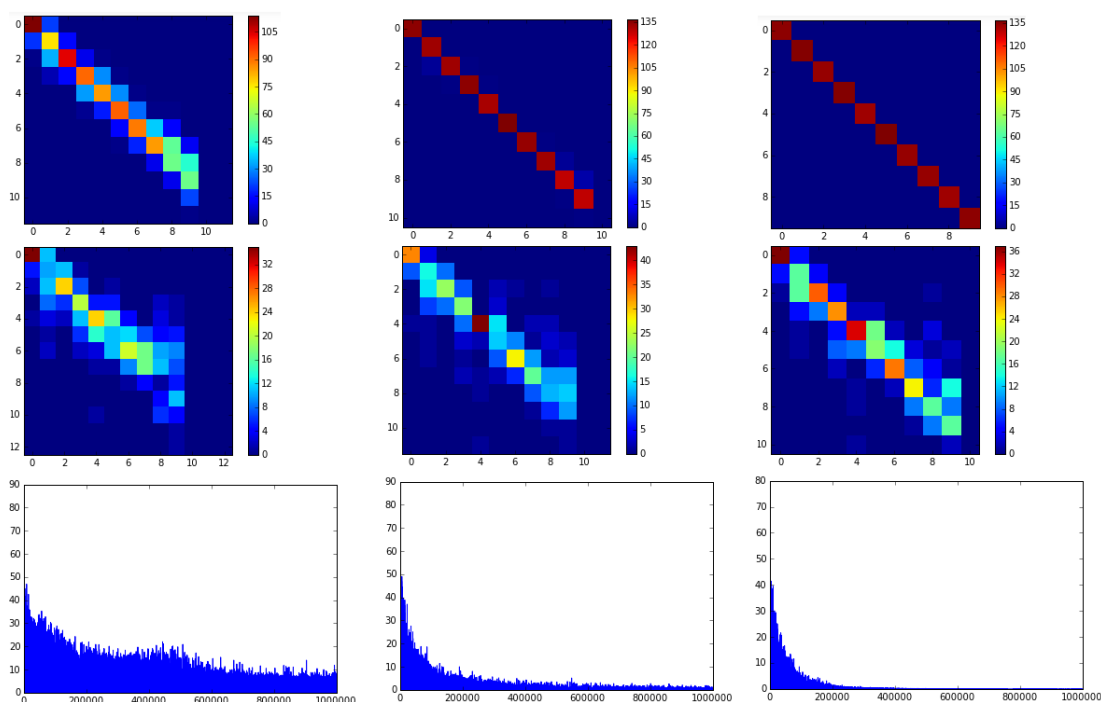


Figura 33: Primera columna: arquitectura $[64,10,1]$, segona columna $[64,50,1]$, tercera correspon a $[64,500,1]$. Taxa aprenentatge $= 1e - 4$ i nombre iteracions=1000000.

A la comparativa entre diferents mides de capa oculta de la *Figura 33* observem que com més neurones tingui una capa oculta, millors resultats obtenim. Val a dir que una quantitat més gran d'unitats significa moltes més connexions, és a dir, més pesos. Això implica que tindrem més derivades a calcular i pesos

a actualitzar, fet que farà anar el procés més lent. Per tant, com en el cas del nombre d'iteracions, ens interessa un valor ben gran que ens permeti executar l'algorisme amb un temps raonable.

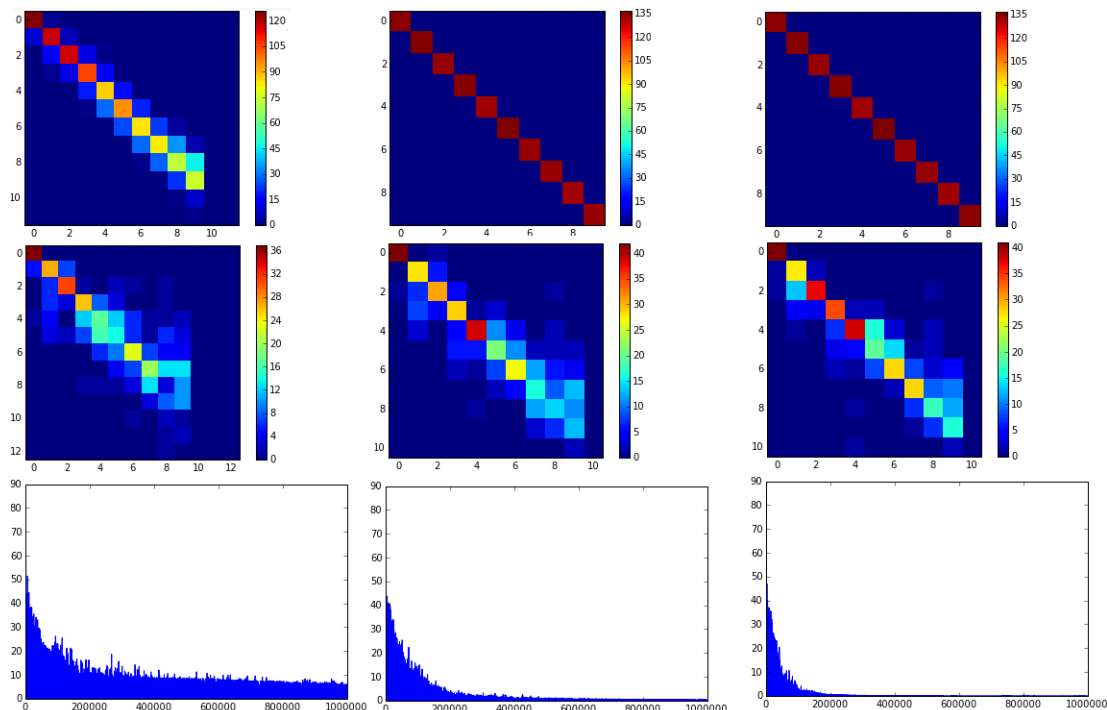


Figura 34: Primera columna: arquitectura $[64,10,10,1]$, segona columna $[64,50,50,1]$, tercera correspon a $[64,500,500,1]$. Taxa aprenentatge $=1e-4$ i nombre iteracions $=1000000$.

A la *Figura 34* trobem el mateix tipus de comparativa que a la *Figura 33*, però en aquest cas amb dues capes ocultes. Podem observar que els resultats són lleugerament millors amb més profunditat de capes però, també podem veure que altres paràmetres com la taxa d'aprenentatge i el nombre d'iteracions tenen més influència en els resultats.

	10 neurones	50 neurones	500 neurones
1 capa oculta	38.67%	49.78%	56%
2 capes ocultes	46.22%	57.78%	63.33%

Figura 35: Comparativa del percentatge d'encert en la fase d'avaluació per a diferents arquitectures. En tots els casos tenim taxa d'aprenentatge $1e-4$ i nombre d'iteracions 1000000 .

Una pregunta natural a fer-nos en aquest punt és si val la pena seguir estudiant xarxes neuronals artificials amb més profunditat de capes ocultes. Així doncs, hem volgut experimentar-ho.

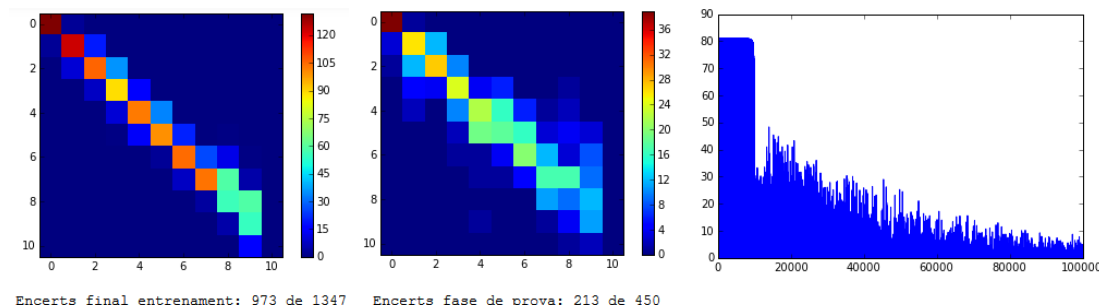


Figura 36: Arquitectura [64,100,50,50,1], taxa d'aprenentatge $1e - 4$, nombre d'iteracions = 1000000.

Observem doncs a la *Figura 36* que els resultats no milloren respecte les xarxes amb dues capes ocultes o fins i tot de les xarxes amb una sola capa oculta.

7.4 Conclusions

Un classificador aleatori pel cas del reconeixement de dígit tindrà una taxa d'encert d'un 10%. En aquest sentit, hem arribat a construir un sistema que és capaç de reconèixer correctament el 63% de les dades noves. Així doncs, hem assolit el nostre objectiu d'aprenentatge amb escreix.

Per tant, ens quedem amb les xarxes neuronals artificials que tinguin o bé una sola capa oculta amb moltes unitats (+500), o bé dues capes ocultes amb un nombre més limitat d'unitats ($\sim (100, 200)$) ja que la profunditat, en aquest problema en concret, no aporta la solució.

A la part pràctica, com a la teòrica, vam començar fent la construcció d'un perceptró multicapa per així passar a aplicar-hi l'algorisme d'aprenentatge del qual hem observat els resultats. Per a sistemes de reconeixement d'objectes més complexes, hauríem de recórrer a altres tipus de xarxes neuronals, com per exemple les convolucionals.

Malgrat ha costat, hem aconseguit implementar l'algorisme de propagació cap enrere pel problema del reconeixement de dígit escrits a mà. Això ha permès una comprensió de la influència dels diferents paràmetres així com del funcionament de l'algorisme en sí.

En aquest punt podem concloure que hem assolit els continguts teòrics necessaris per a la comprensió de les xarxes neuronals artificials i el seu aprenentatge, en concret mitjançant l'algorisme de propagació cap enrere, així com per la correcta implementació d'un sistema de reconeixement de dígit escrits a mà.

Referències

- [1] <http://www.enciclopedia.cat>
- [2] Batut, C.; Belabas, K.; Bernardi, D.; Cohen, H.; Olivier, M.: User's guide to *PARI-GP*,
pari.math.u-bordeaux.fr/pub/pari/manuals/2.3.3/users.pdf, 2000.
- [3] McCulloch and Pitts.
[//www.mind.ilstu.edu/curriculum/modOverview.php?modGUI=212](http://www.mind.ilstu.edu/curriculum/modOverview.php?modGUI=212)
- [4] Michael Nielsen: A visual proof that neural nets can compute any function.
<http://neuralnetworksanddeeplearning.com/chap4.html>
- [5] Lei Jimmy Ba; Rich Caruana: Do Deep Nets Really Need to be Deep?
<http://arxiv.org/abs/1312.6184>
- [6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep Learning, Book in preparation for MIT Press.
<http://goodfeli.github.io/dlbook/>, 2016.
- [7] comp.ai.neural-nets FAQ, Part 3 of 7: Generalization Section - How many hidden layers should I use?
<http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-9.html>
- [8] <http://www.cs.toronto.edu/~kriz/>
- [9] Centre de terminologia de la llengua catalana.
<http://www.termcat.cat>
- [10] Francisco Escolano, Miguel Ángel Cazorla et. al.: Inteligencia artificial. Modelos, técnicas y áreas de aplicación. Thomson Editores Spain Paraninfo S.A, 2003. ISBN: 84-9732-183-9. UB: 0701173487
- [11] Geoffrey Hinton: Neural Networks for Machine Learning.
<https://class.coursera.org/neuralnets-2012-001>
- [12] Andrew Ng: Machine Learning.
<https://www.coursera.org/learn/machine-learning/>
- [13] Antonio López Peña, Ernest Valveny, Maria Vanrell: Detección de objetos. Universitat Autònoma de Barcelona.
<https://www.coursera.org/learn/deteccion-objetos/>
- [14] Lluís Garrido. Pàgina personal: <http://www.ecm.ub.es/~garrido/xarxescasca/gomez/sld0>
- [15] Numpy quickstart tutorial.
<https://docs.scipy.org/doc/numpy-dev/user/quickstart.html>

- [16] Andrej Karpathy: Hacker's guide to Neural Networks.
<http://karpathy.github.io/neuralnets/>
- [17] <http://deeplearning.net/tutorial/mlp.html>