Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Investigació sobre l'estat de l'art en xarxes neuronals artificials, xarxes neuronals convolucionals i aprenentatge profund

Document: Memòria

Alumne: Eduard Feliu Baena

Tutor: Beatriz Lopez Ibañez
Departament: Enginyeria Elèctrica, Electrònica i Automàtica
Àrea: Enginyeria de Sistemes i Automàtica

Convocatòria (mes/any) 09/2016

Index

1 Introducció **5**

1.0.1 Motivació 5

1.0.2 Objectius 6

1.1 Que és l'aprenentatge automàtic? 6

1.1.1 Tipus d'aprenentatge 6

1.2 Conjunts de dades utilitzats 8

1.2.1 Rendiment dels estudiants 9

1.2.2 Microxips 10

1.3 Notació matemàtica 11

2 Regressió lineal **13**

2.1 Representació de la hipòtesi 13

2.2 Funció objectiu 14

2.3 Gradient descent 15

2.3.1 Preprocés de l'algoritme gradient descent 16

2.4 Exemple 17

3 Regressió logística **19**

3.1 Motivacions 19

3.2 Representació de la hipòtesi per classificacions binàries 22

3.3 Representació de la hipòtesi per classificacions multiclasse 23

3.4 Funció objectiu 24

3.5 Intuïció sobre la funció objectiu 24

3.6 Minimitzar la funció de cost 26

3.7 Sobreajustament 27

3.7.1 Regularització 27
3.8 Exemple 28
3.8.1 Interpretació dels resultats 30

4 Xarxes neuronals multicapa supervisades 33
4.1 Motivacions 33
4.2 Representació de la hipòtesi 34
4.3 *Backpropagation* 36
4.3.1 Algoritme 38

5 Deep learning 39
5.1 Introducció 39
5.1.1 El problema del *vanishing gradient* 40
5.1.2 Recerca en *deep learning* 41
5.1.3 Factors externs claus en l'èxit del Deep learning 41
5.2 Funcions d'activació 42
5.3 Regularització *dropout* 45
5.4 Optimització de l'aprenentatge 47
5.5 Xarxa neuronal convolucional 49
5.5.1 Motivacions 49
5.5.2 Capes 50
5.5.3 Cas d'estudi VGGnet 56
5.6 Xarxa neuronal recurrent 58
5.6.1 Motivació 58
5.6.2 Exemple d'una arquitectura completa 59
5.6.3 *Backpropagation* 61
5.6.4 Modelant seqüències condicionades a un context 61
5.6.5 Xarxa recurrent bidireccional 62
5.6.6 Arquitectura *Encoder-Decoder* o *Sequence-to-Sequence* . . . 62
5.6.7 Xarxa recurrent multicapa 64
5.6.8 LSTM 65

6 Conclusions **75**

1 Introducció

El present treball de fi de grau explorarà l'àrea de recerca coneguda com a *deep learning*. Es començarà introduint conceptes que seran necessaris per poder entendre el *deep learning*. S'introduiran conceptes senzills com l'aprenentatge supervisat i altres conceptes bàsics d'aprenentatge automàtic. Finalment s'entrarà en la recerca que s'està duent a terme actualment en *deep learning*.

1.0.1 Motivació

La meva motivació per fer aquest treball més teòric, és que des del meu punt de vista fer un treball de final de grau sobre una aplicació o una web no m'emplenava, ja que és el que molt probablement s'espera de molts de nosaltres que fem en el transcurs de la nostra vida laboral. El que vull dir, és que la vida laboral ens portarà a fer aquesta classe de feines i tard o d'hora s'aprendran. En canvi, és en la universitat on s'ha d'aprofitar per desenvolupar treballs més teòrics que potser algun dia són els que faran que tinguem alguna idea interessant per desenvolupar un projecte personal o professional. Concretament vaig triar el *deep learning* perque havia llegit sobre un concepte que em va semblar molt interessant: un sistema d'aprenentatge *end-to-end*. La majoria dels sistemes d'aprenentatge que coneixia depenen en gran mesura de la representació de les dades d'entrada per obtenir bons resultats. En canvi, amb *deep learning* podem aconseguir representacions de dades noves que facilitin la tasca a desenvolupar. Un bon exemple d'aquest fet són les xarxes neuronals convolucionals.

1.0.2 Objectius

L'objectiu del treball és aprendre. Apendre sobre *machine learning* i *deep learning*. Personalment crec que l'augment de les dades disponibles fa que el *deep learning* sigui una àrea molt interessant per descobrir.

1.1 Que és l'aprenentatge automàtic?

L'aprenentatge automàtic [1] [13] és una branca de la intel·ligència artificial que es basa en la capacitat de crear programes que aprenen sense que algú hagi programat aquest coneixement explícitament. L'aprenentatge automàtic té sentit quan es presenta un problema de difícil solució analítica, ja que el problema és ambigu i contradictori fins a cert punt.

1.1.1 Tipus d'aprenentatge

L'aprenentatge consisteix a descobrir una sèrie de patrons en les dades que no s'havien observat. És una definició molt oberta que dóna lloc a diferents estratègies d'aprenentatge per tal de resoldre problemes diferents.

Aprenentatge supervisat

L'aprenentatge supervisat es dóna quan es disposa d'un exemple del qual ja se sap el resultat. La notació utilitzada per referenciar l'exemple i-essim del conjunt de dades és la següent: $(x^{(i)}, y^{(i)})$, on $x^{(i)}$ representa el conjunt de variables descriptores i $y^{(i)}$ representa la sortida esperada o predicció. Dos exemples d'aprenentatge supervisat són la classificació i la regressió.

Aprenentatge no supervisat

En l'aprenentatge no supervisat, a diferència de l'aprenentatge supervisat, els exemples estan formats únicament amb variables explicatives $x^{(i)}$. El fet de no tenir etiquetes $y^{(i)}$ fa que l'aprenentatge explori la cerca de correlacions; és a dir,

és capaç de localitzar certa estructura en les dades. Dos exemples d'aprenentatge no supervisat són la reducció de la dimensionalitat i el *clustering*.

1.2 Conjunts de dades utilitzats

En aquest secció s'introduiran els conjunts de dades utilitzats durant el desenvolupament del treball. Els conjunts de dades utilitzats són els següents: rendiment dels estudiants i microxips.

1.2.1 Rendiment dels estudiants

S'utilitzarà el conjunt de dades proporcionat per Paulo Cortez [4] per motivar el primer tema del treball, la regressió lineal. S'implementarà la predicció del resultat d'un examen final d'un estudiant en funció del resultat dels exàmens parcials d'un conjunt d'estudiants. És un exemple senzill, però constitueix un bon punt de partida per introduir diversos conceptes i preparar el terreny per a temes més complexos com la regressió logística o les xarxes neuronals multi-capa supervisades. Enllaç a les dades: `https://archive.ics.uci.edu/ml/datasets/Student+Performance`

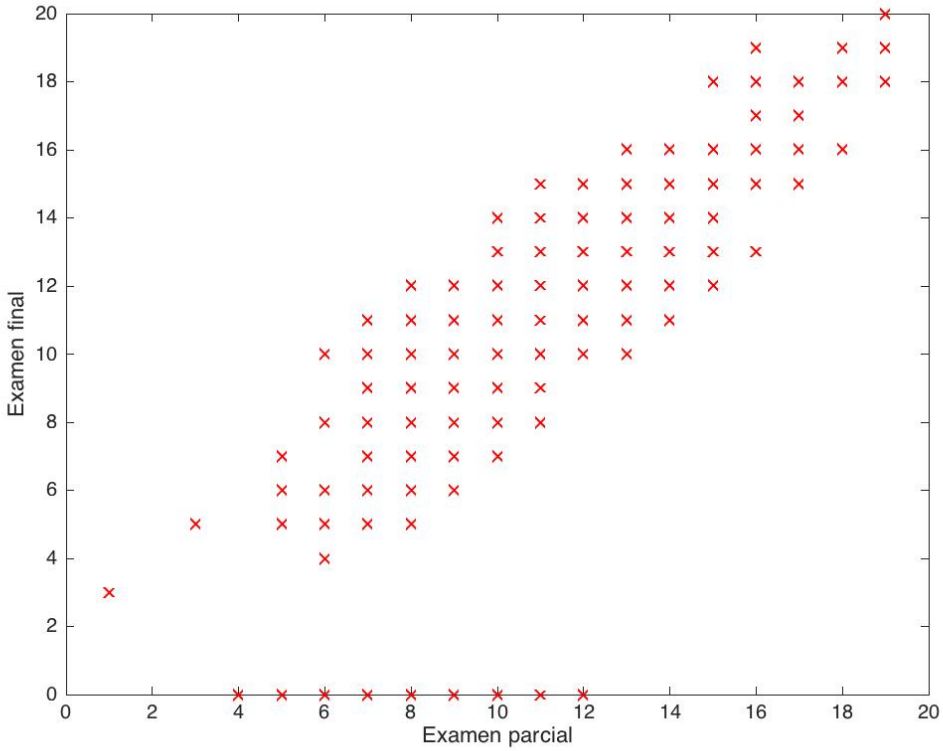

Figura 1.1: Notes de matemàtiques d'un examen final i un parcial d'alumnes de secundària.

1.2.2 Microxips

S'utilitzarà un conjunt de dades del curs de *Machine Learning* [13] de la Universitat de Stanford. El conjunt de dades consisteix en dos tests de qualitat per a cada microxip i mostra si el microxip va ser acceptat o no. És molt similar a l'exemple de l'admissió dels estudiants, ja que els dos problemes són classificacions binaries. La diferència principal és que el conjunt de dades actual no és linealment separable. Això fa que sigui un bon exemple per parlar de transformacions no lineals sobre les variables descriptores. També s'utilitzarà aquest conjunt de dades per introduir de forma pràctica el sobreajustament i la regularització L2 en el context de la regressió logística.

enllaç a les dades: `http://openclassroom.stanford.edu/MainFolder/courses/MachineLearning/exercises/ex5materials/ex5Data.zip`

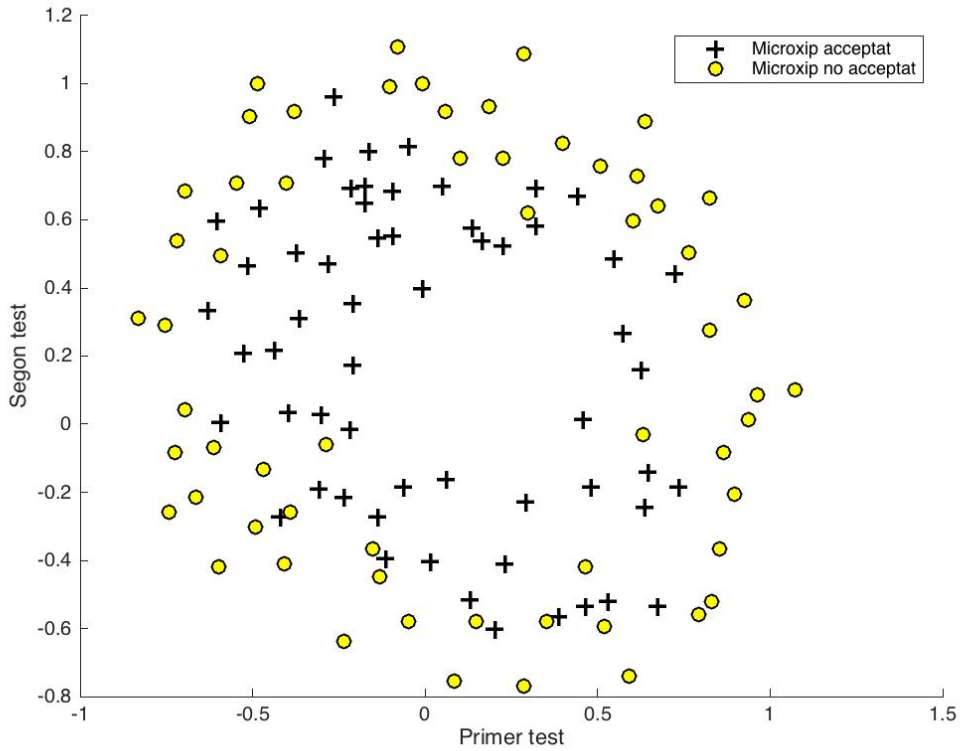

Figura 1.2: Acceptació dels microxips en funció de dos tests de qualitat

1.3 Notació matemàtica

En aquest apartat s'introduirà la notació genèrica utilitzada en el treball. La idea és mantenir una notació consistent amb els criteris aquí exposats per facilitar la llegibilitat de la formulació matemàtica. La notació més específica s'introduirà en cada un dels capítols.

m = nombre d'exemples utilitzats per realitzar l'aprenentatge

x = variables d'entrada

y = variable de sortida

(x, y) = exemple d'aprenentatge supervisat

$(x^{(i)}, y^{(i)})$ = i-essim exemple d'aprenentatge supervisat

$x_j^{(i)}$ = variable d'entrada j de l'i-essim exemple

θ = Conjunt de paràmetres d'un model d'aprenentatge automàtic

σ = funció sigmoide

$J(\theta)$ = funció de cost d'un model d'aprenentatge automàtic utilitzant un conjunt de paràmetres θ

2 Regressió lineal

La regressió lineal [1] [13] és un bon punt de partida per introduir l'aprenentatge automàtic. La regressió lineal modela la relació entre una variable contínua i dependent y i una o més variables explicatives $x_1, x_2, .., x_n$. Per exemplificar la regressió lineal s'utilitzarà el primer conjunt de dades, presentat en la subsecció 1.2.1. En aquest primer exemple es vol predir el resultat d'un examen final d'un estudiant a partir d'un conjunt d'exàmens parcials anteriors. La notació emprada en aquest exemple serà la següent: $(x^{(1)}_{parcial}, y^{(1)}_{final}), (x^{(2)}_{parcial}, y^{(2)}_{final}), ..., (x^{(m)}_{parcial}, y^{(m)}_{final})$.

2.1 Representació de la hipòtesi

Una hipòtesi és una proposició acceptable que ha estat formulada a través de la recol·lecció de dades. En el cas de la regressió lineal, la hipòtesi és una combinació lineal de totes les variables descriptores x. En el nostre cas, només tenim una variable descriptora $x_{parcial}$. S'ha escollit un exemple amb una sola variable descriptora per facilitar la visualització gràfica de la funció de cost, que s'explicarà en la següent secció.

$$h_\theta(x) = \sum_{i=1}^{n} \theta_i x_i = \theta^t x \tag{2.1}$$

On $x \in \mathbb{R}^{n+1}$ i $\theta \in \mathbb{R}^{n+1}$. Com s'observa en l'equació 2.1, per facilitar la notació compacta $\theta^t x$ s'ha d'afegir la variable $x_0 = 1$. Aquesta notació ens aporta dos beneficis: una notació més compacta i un rendiment més elevat, ja que una bona llibreria d'àlgebra lineal pot paral·lelitzar les multiplicacions de vectors o matrius.

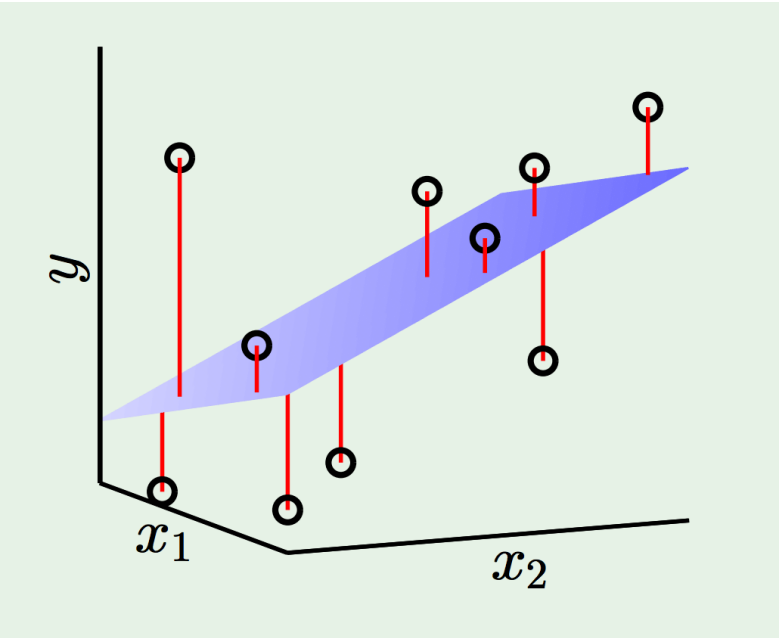

Figura 2.1: Del llibre *Learning from data* [1]. Les línies vermelles representen el cost entre la nostra hipòtesi, en aquest cas el pla de color blau, i els nostres exemples.

2.2 Funció objectiu

En la majoria dels algoritmes d'aprenentatge s'utilitza una funció anomenada funció de cost. L'objectiu d'aquesta funció és avaluar el rendiment del nostre model. Una funció de cost determina la diferència entre la hipòtesi $h_\theta(x^{(i)})$ i exemples que ja tenim etiquetats $(x^{(i)}, y^{(i)})$. És una forma de determinar la qualitat d'un model. En el cas de la regressió lineal, la funció de cost queda definida en la següent equació:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{2.2}$$

La regressió lineal utilitza una de les funcions de cost més senzilles, l'error quadràtic mitjà. L'error quadràtic mitjà, presentat en l'equació 2.2, defineix el cost com la mitjana del sumatori de la diferència entre la hipòtesi i el valor real al quadrat. L'objectiu de la fase d'aprenentatge és minimitzar la funció de cost, ja que d'aquesta forma la diferència entre la hipòtesi i els valors reals serà petita. Quant més petita sigui aquesta diferència més bona serà la predicció.

2.3 Gradient descent

Un cop definida la funció de cost és necessari definir un algoritme d'optimització per minimitzar la funció de cost i obtenir millors prediccions. L'algoritme *gradient descent* és una tècnica utilitzada per minimitzar funcions convexes. L'objectiu de l'algoritme *gradient descent* és trobar una combinació de paràmetres θ que minimitzin la funció de cost. En el primer pas de l'algoritme 1 es calcula el

Algorithm 1: Gradient descent

Donat un conjunt de valors inicials al vector θ

repeat

1. $\Delta\theta := -\nabla J(\theta)$.
2. Selecciona el tamany de pas α
3. Actualitza $\theta := \theta + \alpha\Delta\theta$

until criteri de parada està satisfet

gradient de la funció de cost. El gradient és un vector que té com a components les derivades parcials de la funció de cost respecte cada un dels paràmetres que s'han de minimitzar. El gradient indica quina és la direcció on és produeix un canvi més elevat en la funció de cost. Un exemple senzill seria una superfície convexa de dues dimensions on la dimensió y representa la funció de cost i la dimensió x representa un paràmetre que es vol optimitzar. Si el paràmetre respecte a la funció de cost es troba situat a la part dreta de la superfície convexa, la derivada serà positiva. El fet que el primer pas negui el gradient fa que s'acabi sumant un valor negatiu al paràmetre que volem optimitzar desplaçant el paràmetre, poc o molt això ja depèn de la variable α que pondera el desplaçament, cap al centre. Donant per entès que en el centre de la superfície convexa la funció de cost és mínima. En el cas contrari, si la inicialització del paràmetre ens situa en la part esquerra de la superfície la derivada serà negativa. Aquest fet en combinació amb la negació del primer pas ens desplaçarà cap al centre, de forma simètrica que en el cas anterior. Alhora d'escollir la mida de pas, si la mida que es selecciona és massa elevada, l'algoritme podria no convergir. En cas contrari, si es selecciona un pas molt petit,

l'algoritme pot tardar molt a arribar al mínim global. L'estratègia més adequada és escollir una mida de pas relativament gran en el moment d'iniciar l'algoritme i anar reduint el pas a mesura que s'aproxima al mínim global de la funció.

2.3.1 Preprocés de l'algoritme gradient descent

Alhora de cercar el mínim d'una funció, un problema habitual és el fet de tenir variables descriptores amb escales de valors molt diferents. L'algoritme gradient descent convergeix de forma més ràpida si totes les variables d'entrada x tenen una escala similar. Per mantenir les variables descriptores amb una escala similar s'utilitzarà la següent equació:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{2.3}$$

2.4 Exemple

Per tal d'il·lustrar tots els conceptes que s'han aplicat en aquest capítol s'ha implementat la regressió lineal amb Matlab utilitzant el primer dels conjunts de dades presentat en la subsecció 1.2.1. El conjunt de dades del rendiment dels estudiants. L'objectiu és predir el resultat d'un examen final en funció d'un conjunt d'exàmens parcials. La implementació del codi es troba en la carpeta matlab/regressio_lineal. S'ha implementat l'entrada de dades, la funció de cost per avaluar el rendiment de la hipòtesi i l'algoritme *gradient descent*. En les figures 2.2 i 2.3 es pot observar el resultat d'aplicar iterativament l'algoritme d'optimització *gradient descent* per intentar ajustar la hipòtesi al conjunt de dades d'exemple. Els exàmens estan avaluats amb valors entre 0 i 20. La hipòtesi que s'utilitzarà en aquest exemple és la següent:

$$h_\theta(x) = \theta_0 + \theta_1 x_{parcial}$$

En aquest exemple s'ha establert el nombre d'iteracions de l'algoritme *gradient descent* i quina α seleccionar de forma manual. L'objectiu d'aquest exemple és il·lustrar l'algoritme *gradient descent* en el context de la regressió lineal.

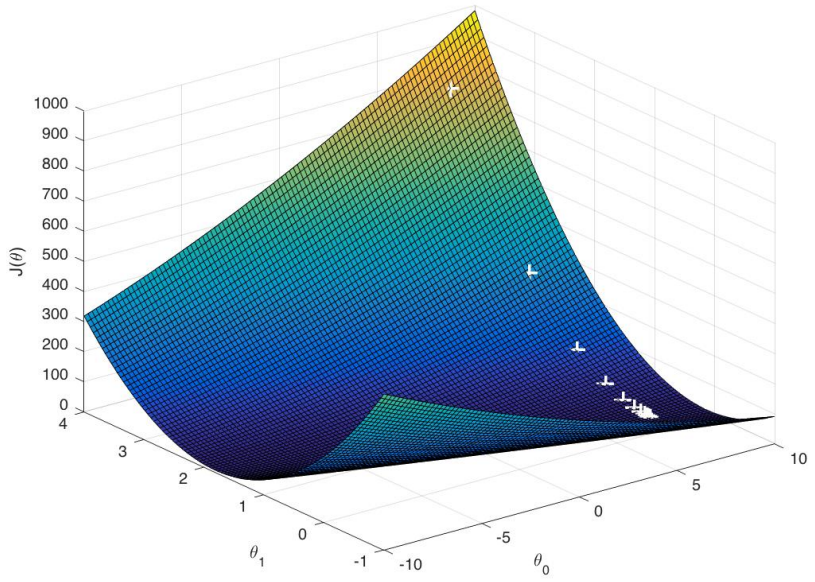

Figura 2.2: Cada punt de la superfície representa una iteració de l'algoritme *gradient descent*. Com es pot veure, al cap de poques iteracions la funció de cost obté valors pròxims a 0.

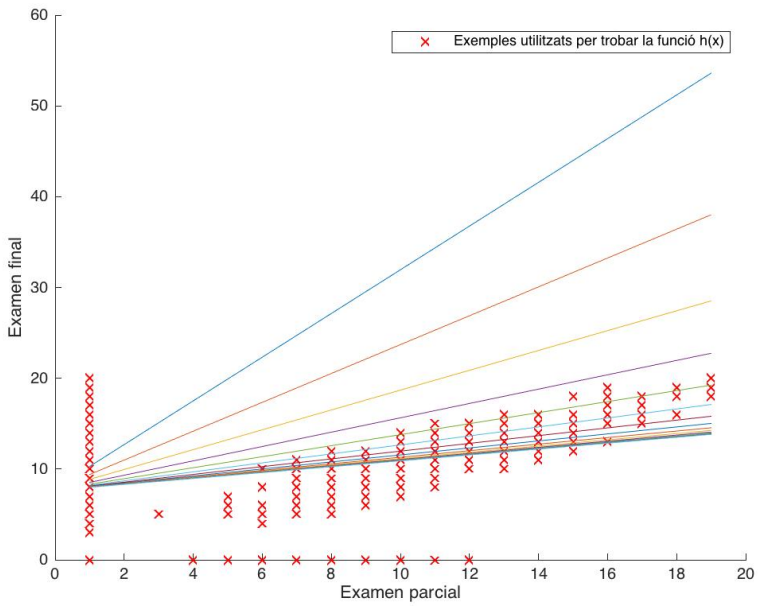

Figura 2.3: Cada línia representa una iteració de l'algoritme *gradient descent*. La primera iteració, la línia blava, no s'ajusta a les dades d'exemple. Al cap de poques iteracions s'obté un resultat acceptable.

3 Regressió logística

La regressió logística [1] [13] s'utilitza en problemes de classificació. Per exemple, a l'hora de classificar si un tumor és maligne o benigne o per determinar si un correu electrònic es pot considerar correu brossa o no. La primera part del capítol se centrarà en classificacions binàries.

$$y \in \{0, 1\}$$

3.1 Motivacions

Seguint amb l'exemple dels tumors, s'intentarà il·lustrar amb un exemple perquè s'utilitza la regressió logística en classificacions binàries. Seria possible utilitzar la regressió lineal per fer classificacions binàries? Si s'estableix un llindar com en l'equació 3.1, la regressió lineal també sembla ser adequada per fer classificacions binàries.

$$y = \begin{cases} 1, & \text{if } h_\theta > 0.5 \\ 0, & \text{altrament} \end{cases} \tag{3.1}$$

En l'exemple que es desenvoluparà, la x representa la mida del tumor i s'utilitzarà per predir la y. La y és una variable discreta amb dos valors que indiquen si el pacient té un tumor benigne $y = 0$ o maligne $y = 1$.

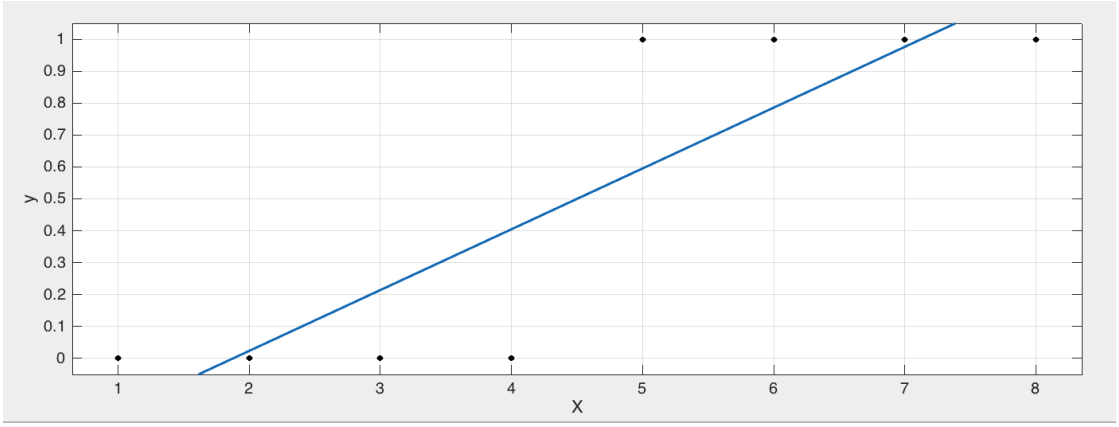

Figura 3.1: Regressió lineal correctament ajustada als exemples

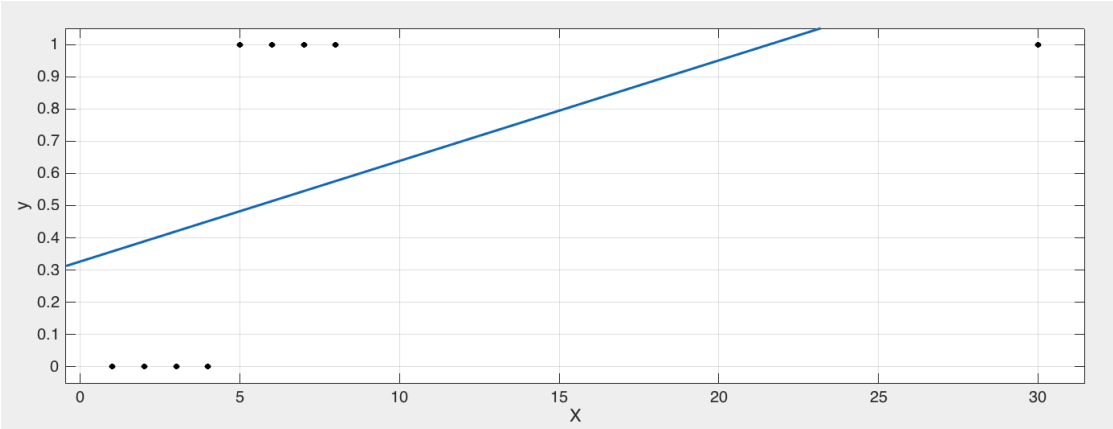

Figura 3.2: Regressió lineal amb un nou exemple

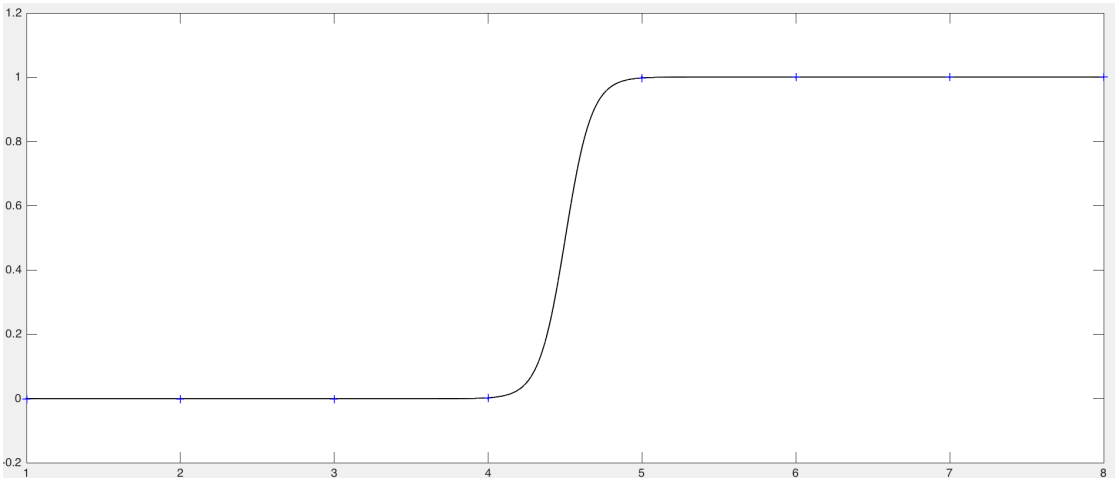

Figura 3.3: Regressió logistica amb els exemples de la figura 3.1

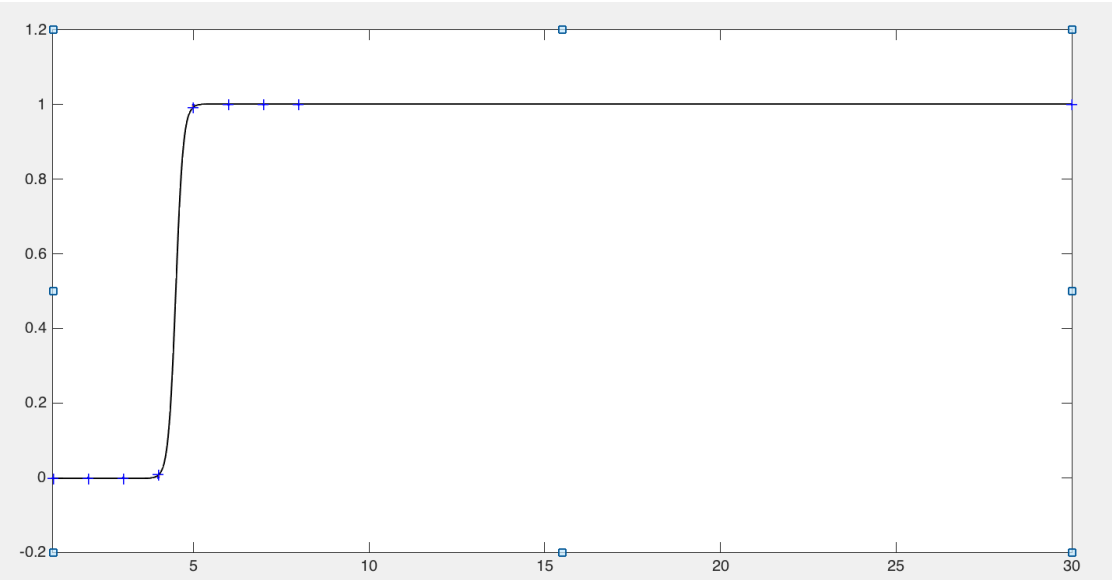

Figura 3.4: Regressió logistica amb un nou exemple, com en la figura 3.2

En la figura 3.1, la regressió lineal està correctament ajustada als exemples utilitzats. Com es pot observar, els quatre primers exemples donen prediccions inferiors al llindar marcat i, efectivament, el seu valor real és $y = 0$. Els quatre exemples restants, amb un valor real $y = 1$, donen valors superiors al llindar establert, que és 0.5. Si s'observa la figura 3.1, es pot veure que la regressió lineal sembla adequada per fer classificacions binàries. En la figura 3.1 s'observa una tendència: els tumors són malignes quan el valor x és superior a 4. En la figura 3.2, s'ha afegit un tumor maligne amb una mida de 30. Aquest nou tumor no canvia la tendència explicada, però la regressió lineal en aquest cas no és capaç de predir correctament tots els exemples. Concretament en el cinquè exemple es realitza una predicció errònia. En canvi, la regressió logística s'ajusta correctament a tots els exemples, com s'observa a les figures 3.3 i 3.4.

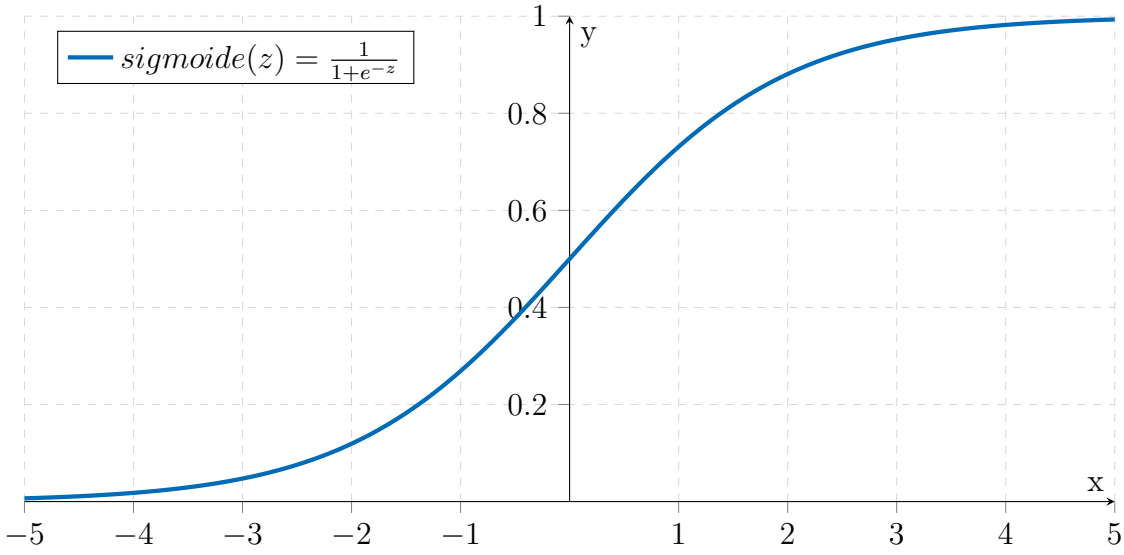

Figura 3.5: Funció sigmoide

3.2 Representació de la hipòtesi per classificacions binàries

La representació de la hipòtesi o del model equival a la regressió lineal però afegint l'aplicació d'una funció sigmoide o logística a la hipòtesi. El fet d'afegir l'aplicació de la funció sigmoide fa que la hipòtesi sigui la probabilitat estimada que y=1, condicionada a les variables predictores x i als paràmetres θ. Formalment, la definirem de la següent manera:

$$h_\theta(x) = sigmoide(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} = P(y = 1|x; \theta)$$

La funció sigmoide (Figura 3.5) té una asímptota que tendeix a zero per la part negativa i una asímptota que tendeix a 1 per la part positiva. Aquest fet limita els valors de la hipòtesi en l'interval comprès entre 0 i 1.

3.3 Representació de la hipòtesi per classificacions multiclasse

La regressió logística multinomial generalitza la regressió logística per problemes multiclasse on la variable depenent és una variable discreta amb més de dos valors. A cada valor se li assigna una probabilitat. La suma d'aquestes probabilitats sempre suma 1, ja que s'aplica una normalització en el denominador.

$$h_\theta^{(i)}(x) = P(y = i|x;\theta) = softmax(\theta^t x) = \frac{\exp(\theta^t x)_i}{\sum_{n=1}^{N}(\exp(\theta^t x)_n)}$$

Un exemple concret sobre un classificador amb tres classes podria ser el següent:

$$softmax(\theta^t x) = softmax([2, 1, 0.1]) = [0.7, 0.2, 0.1]$$

3.4 Funció objectiu

Una funció de cost o objectiu determina la diferència entre la hipòtesi i els exemples etiquetats. És una forma de determinar la qualitat d'un model. En el cas de la regressió logística, s'avalua el cost del model amb la funció de cost $J(\theta)$:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] \tag{3.2}$$

La funció de cost 3.2, anomenada *cross-entropy*, és més complexa que la funció de cost analitzada en el capítol de la regressió lineal. Per tal de proporcionar la intuïció necessària per entendre la necessitat d'una funció de cost més complexa, primer s'analitzarà la derivada parcial $\partial J(\theta)/\partial\theta$ en el cas de l'error quadràtic mitjà 3.3 i la mateixa derivada en el cas d'emprar la funció de cost *cross-entropy* 3.4.

$$\frac{\partial J_{error_quadratic}(\theta)}{\partial\theta} = (h_\theta - y)\sigma'(\theta^T x) \tag{3.3}$$

$$\frac{\partial J_{cross_entropy}(\theta)}{\partial\theta} = \frac{1}{n}\sum_{x} x_j(h_\theta - y). \tag{3.4}$$

L'equació 3.4 és molt intuïtiva. Bàsicament proporciona un gradient més elevat en cas d'obtenir un error més gran $h_\theta(x) - y$. Intuïtivament és el que s'esperava. En canvi, la dinàmica de l'equació 3.3 és molt diferent. El terme $\sigma'(\theta^T x)$ indica la derivada de la funció sigmoïdal. Aquesta derivada, en qualsevol de les dues asímptotes de la funció sigmoïdal, obtindrà un valor proper a zero. Això fa que la velocitat de l'aprenentatge no sigui l'adequat. A més a més, és poc intuïtiu.

3.5 Intuïció sobre la funció objectiu

En aquest apartat és descompondrà la funció de cost *cross-entropy* per entendre exactament quin és el seu comportament. La funció de cost es pot descompondre en dos casos. Quan $y = 0$, veure figura 3.7, o quan $y = 1$ veure figura 3.6.

$$J(\theta) = \begin{cases} -\log(h_\theta(x^{(i)})), & \text{if } y = 1 \\ \log(1 - h_\theta(x^{(i)})), & \text{if } y = 0 \end{cases} \tag{3.5}$$

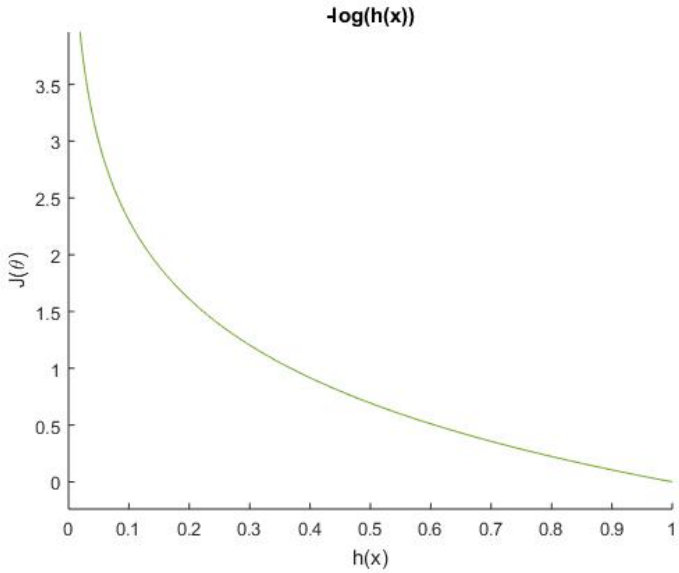

Figura 3.6: Quan $y = 1$, la figura 3.6 captura la intuïció que, en el cas que la hipòtesi sigui igual a 1, el cost és zero. La gràfica també captura la intuïció que el cost s'incrementa a mesura que anem reduint el valor de la nostra hipòtesi, fins al cas extrem de tenir una hipòtesi amb valor igual a zero i, per tant, un cost infinit.

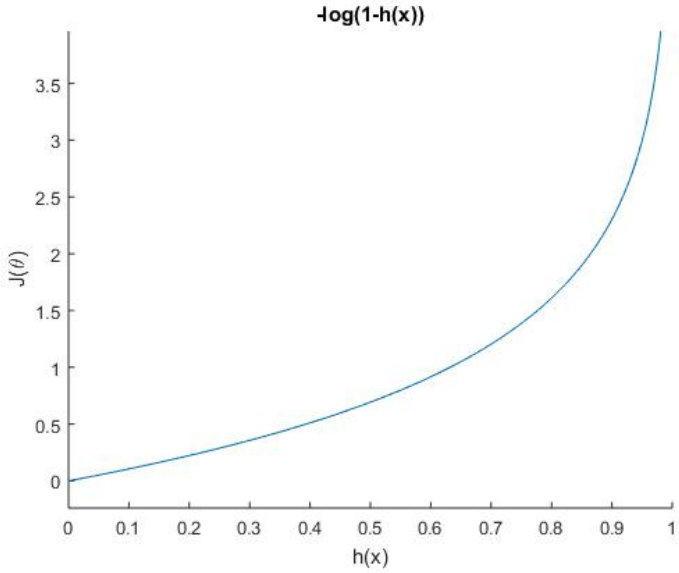

Figura 3.7: En el cas que $y = 0$ es pot observar que, quan la hipòtesi sigui igual a 0 i la y de l'exemple que estem avaluant sigui 0, el cost serà zero. També s'observa que el cost es va incrementant en augmentar el valor de la nostra hipòtesi, fins al punt de tenir una hipòtesi de valor igual a 1 i, per tant, un cost infinit.

3.6 Minimitzar la funció de cost

Per minimitzar la funció de cost, s'utilitzarà el mateix algoritme mostrat en l'apartat de la regressió logística, l'algoritme *gradient descent* 1. Tots els paràmetres θ_n s'han d'actualitzar simultàniament. Com a detall d'implementació, en aquest apartat podem veure l'algoritme a més baix nivell. El gradient és un vector format per les derivades parcials respecte a cada un dels paràmetres.

Algorithm 2: Algoritme *gradient descent* aplicat a regressió logística

repeat

$\theta_0 = \theta_0 - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$

$\theta_1 = \theta_1 - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$

....

$\theta_n = \theta_n - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_n^{(i)}$

until *convergence*;

3.7 Sobreajustament

El sobreajustament [1] és una mesura comparativa que consisteix en l'ajustament excessiu del model a les nostres dades. Un model té sobreajustament si l'avaluació de la funció de cost en el conjunt de dades d'entrenament dóna valors baixos, mentre que en el conjunt de dades de test s'obtenen valors elevats. En observar aquesta dinàmica es podria argumentar que el model s'ajusta excessivament a les dades d'entrenament. Els conjunts amb poques dades o amb dades sorolloses són sensibles al sobreajustament.

3.7.1 Regularització

La regularització és una família de tècniques amb l'objectiu de reduir el sobreajustament de les dades en models d'aprenentatge automàtic. Moltes tècniques de regularització es basen a restringir la funció que aprèn el model a partir de les dades, anomenada hipòtesi. Una forma de restringir la hipòtesi és eliminar els atributs que es considerin menys rellevants utilitzant *feature selection*. Aquesta forma de regularització s'anomena *hard constraint*, ja que s'elimina part de les dades disponibles. Una segona tècnica molt utilitzada en el context de les xarxes neuronals és la regularització *weight decay* o L2. Aquesta tècnica restringeix la contribució de cada atribut. La regularització L2 serà tractada amb detall en la secció 3.8.

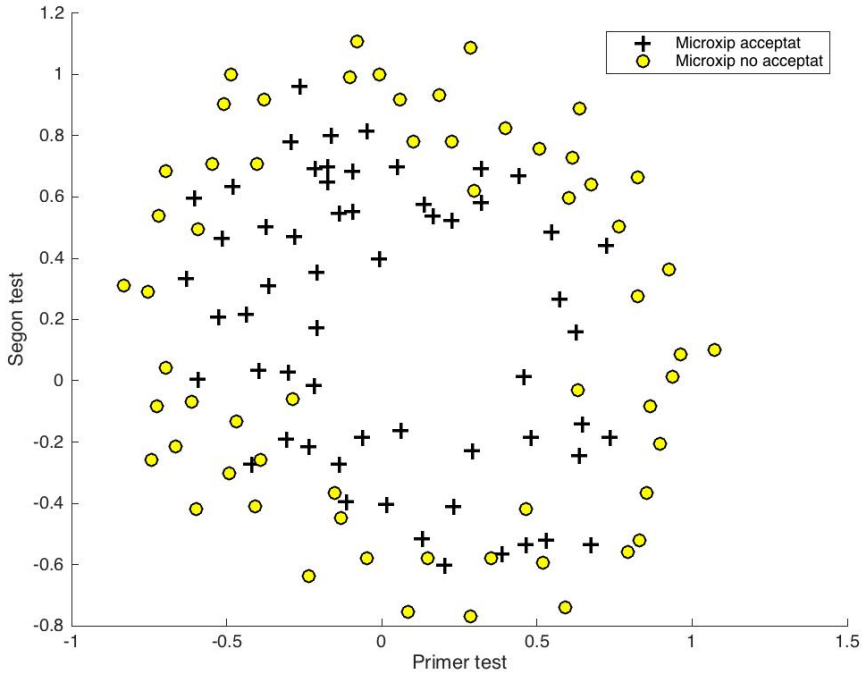

Figura 3.8: Acceptació dels microxips en funció de dos tests de qualitat

3.8 Exemple

En la subsecció 3.7 s'han introduït els conceptes de sobreajustament i regularització. En aquest apartat es tractarà la regularització en el context de la regressió logística i s'exposarà un problema on l'ús de la regularització és adequat. Per motivar l'ús de la regularització s'utilitzarà el segon conjunt de dades, que mostra l'acceptació d'un microxip en funció de dos tests de qualitat. Com s'observa en la figura 3.8, els microxips de les dues classes no són linealment separables. Com que la regressió logística és un model lineal, es necessitarà una representació de dades més potent per poder separar les dues classes. En aquest cas concret s'utilitzaran els termes polinòmics de *primer_test* i *segon_test* fins a la sisena potència.

$$x = \begin{bmatrix} 1 \\ primer_test \\ segon_test \\ primer_test^2 \\ primer_test * segon_test \\ segon_test^2 \\ primer_test^3 \\ \vdots \\ primer_test * segon_test^5 \\ segon_test^6 \end{bmatrix}$$

Per entendre com és possible aconseguir hipòtesis no lineals en un model lineal recomano llegir l'apartat 3.4 (*Nonlinear transformations*) del llibre *Learning from data* [1]. En aplicar transformacions no lineals a les variables descriptores $x_1, x_2, .., x_n$, es generen noves variables $z_1 = x_1^2, z_2 = x_2^2, .., z_n = x_n^2$. En funció de l'operació aplicada a les noves variables és possible que el nou espai de característiques Z sigui linealment separable. D'aquesta forma es podrà utilitzar un model lineal com la regressió logística. Un cop definides les variables descriptores, s'utilitzarà la representació de la hipòtesi de la regressió logística presentada en la secció 3.2, l'algoritme *gradient descent* per l'optimització dels paràmetres 2 i s'utilitzarà una versió modificada de la funció de cost *cross-entropy* 3.4, que contempla un factor de regularització L2.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \quad (3.6)$$

On el paràmetre λ controla l'equilibri entre les dues parts de la funció de cost. Al minimitzar el valor de la funcio $J(\theta)$, en el cas de tenir un valor λ molt alt, la funció de cost seria $\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$ i la minimització d'aquesta funció és equivalent a assignar un valor igual a zero a tots els paràmetres θ. En canvi, si el paràmetre λ és igual a zero, obtenim la funció de cost tradicional de la regressió logística (secció 3.4), i de tots els possibles valors λ que podem seleccionar. Tota la implementació es troba en la carpeta matlab/regressio_logistica_regularitzada.

3.8.1 Interpretació dels resultats

Després d'entrenar quatre models amb quatre valors λ diferents, en la figura 3.13 s'observa que, amb un valor $\lambda = 0$, la frontera de decisió que separa els exemples positius dels negatius s'ajusta massa a les dades i, per tant, podem pensar de forma intuïtiva que no generalitzarà correctament. En aquest cas, podem assegurar que el nostre model té una alta variància. En l'altre cas extrem, $\lambda = 10$, podem observar que s'ajusta massa poc a les dades i, per tant, el model pateix un biaix gran. Els altres dos casos, $\lambda = 0.01$ i $\lambda = 1$, i molts dels valors compresos en aquest rang generalitzen millor. La idea que ha de quedar clara d'aquest apartat és que amb una dosi molt petita de regularització es pot disminuir en gran mesura la variància del model. En aquest exemple no s'ha mencionat el cost final assolit per cada un dels models. Evidentment, el model que aconsegueix un cost més baix en les dades d'entrenament és el model $\lambda = 0$, ja que s'ajusta excessivament a les nostres dades. No obstant això, el més important és el cost que assolit en el conjunt de dades de test.

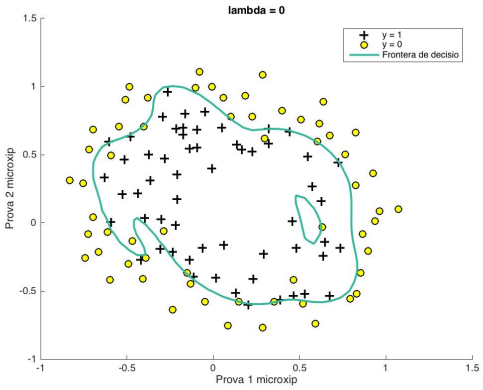

Figura 3.9: $\lambda = 0$

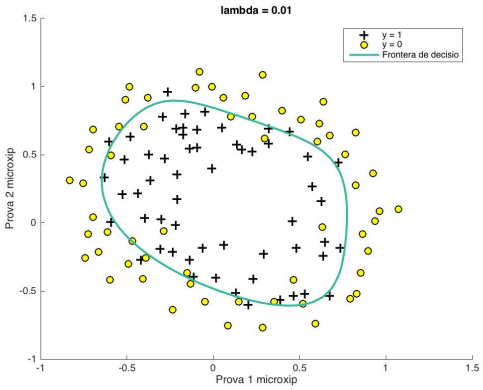

Figura 3.10: $\lambda = 0.01$

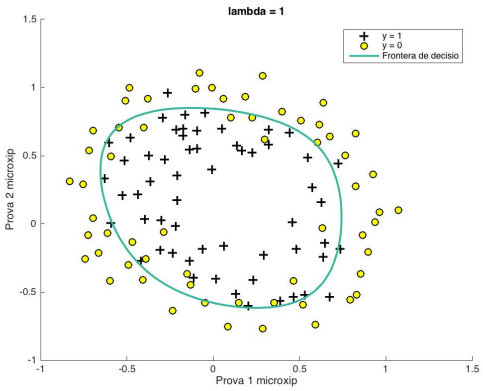

Figura 3.11: $\lambda = 1$

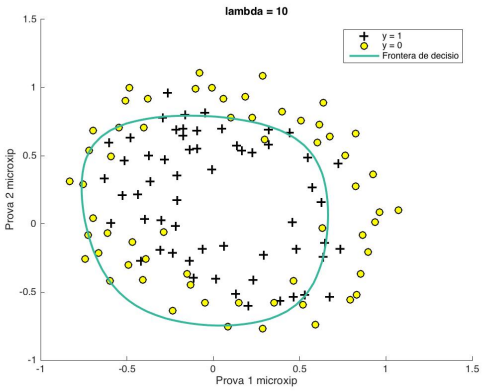

Figura 3.12: $\lambda = 10$

Figura 3.13: Resultats amb diferents valors λ de la funció de cost $J(\theta)$

4 Xarxes neuronals multicapa supervisades

4.1 Motivacions

La motivació principal de l'ús de les xarxes neuronals multicapa [2] [1] és aprendre una millor representació de les dades d'entrada. D'aquesta forma, les dades d'entrada són projectades a una nova representació que facilita la tasca de classificació que la xarxa neuronal està portant a terme. En la figura 4.1 s'observa que la regressió logística no pot separar correctament l'exemple que es visualitza. En afegir una capa oculta, com en la figura 4.2, s'observa que la xarxa multicapa ha après una representació de les dades que és linealment separable. Aquest exemple és fàcil de visualitzar, ja que només tenim dues neurones en la capa oculta i això facilita la visualització de la representació de les dades que està aprenent aquesta capa oculta. En augmentar el nombre de neurones, s'han d'aplicar tècniques de reducció de la dimensionalitat per visualitzar les representacions que aprenen les capes ocultes. [20].

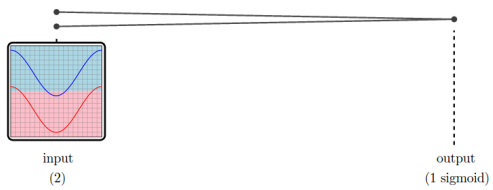

Figura 4.1: Regressió logística

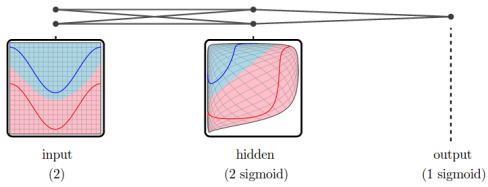

Figura 4.2: Xarxa neuronal multicapa

Figura 4.3: Del blog d'en Christopher Olah [14]

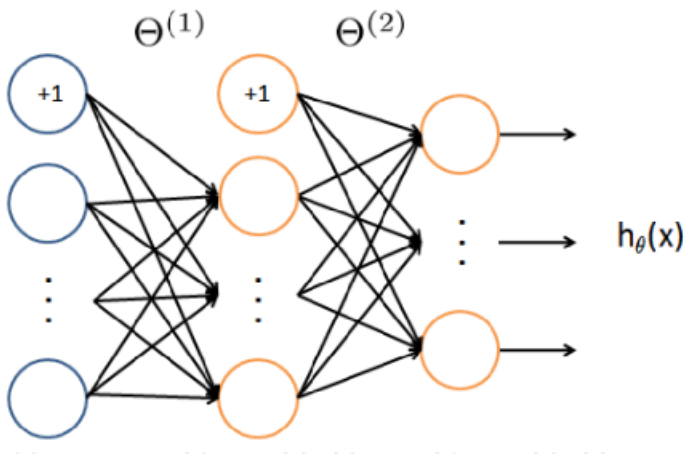

Figura 4.4: Del curs de *machine learning* [13]. Arquitectura d'una xarxa neuronal multicapa

4.2 Representació de la hipòtesi

L'equació 4.1 defineix l'activació de la neurona j en la capa l.

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \tag{4.1}$$

On w_{jk}^l és el paràmetre entre la neurona k de la capa $(l-1)$ i la neurona j de la capa l. a_k^{l-1} és l'activació de la neurona k de la capa $l-1$. b_j^l és el biaix de la neurona j de la capa l. Al vectoritzar l'equació 4.1 es definirà la matriu de pesos w^l per cada capa l. El valor en la fila j i la columna k serà w_{jk}^l. També es definirà per cada capa l un vector biaix b^l i finalment es definirà per cada capa l un vector d'activacions a^l.

$$a^l = \sigma(w^l a^{l-1} + b^l) \tag{4.2}$$

L'equació vectoritzada 4.2 és més intuïtiva i ens aporta una visió més global de com les activacions d'una capa es relacionen amb les activacions de la capa anterior. A l'hora de computar a^l, es definirà la variable $z^l = w^l a^{l-1} + b^l$ que s'utilitzarà en l'algoritme *backtracking* presentat en la següent secció. La representació de la hipòtesi de la xarxa multicapa queda definida en l'equació 4.3.

$$h_\theta(x) = a^L(x) \tag{4.3}$$

On L és el nombre de capes de la xarxa i $a^L(x)$ és el vector d'activacions de la sortida de la xarxa quan l'entrada és x.

4.3 *Backpropagation*

L'algoritme *backpropagation* va ser introduït en els anys 70, però no es va començar a utilitzar efectivament en xarxes neuronals fins a l'any 1986 [15]. Avui en dia l'algoritme *backpropagation* és l'algoritme principal a l'hora de realitzar la fase d'aprenentatge d'una xarxa neuronal multicapa. L'algoritme *backpropagation* calcula les derivades parcials de la funció de cost respecte qualsevol paràmetre de la xarxa. Aquestes derivades parcials o gradients indiquen com de ràpid varia la funció de cost quan s'introdueixen petits canvis en els paràmetres. Per computar les derivades parcials s'introduirà una variable nova, δ_j^l, que és l'error de la neurona j en la capa l.

$$\delta_j^l \equiv \frac{\partial J(\Theta)}{\partial z_j^l} \tag{4.4}$$

On Θ representa tots els paràmetres de la xarxa. L'algoritme de *backpropagation* es basa fonamentalment en quatre equacions que s'utilitzaran per computar els errors δ_j^l i el gradient de la funció de cost per cada paràmetre. La primera equació defineix l'error en la capa de sortida L.

$$\delta_j^L = \frac{\partial J(\Theta)}{\partial a_j^L}\sigma'(z_j^L). \tag{4.5}$$

L'equació 4.5 és molt intuïtiva. La subexpresió $\partial J(\Theta)/\partial a_j^L$ mesura la rapidesa amb la qual la funció cost canvia en funció de la j-essima neurona de sortida. La subexpressió $\sigma'(z_j^L)$ mesura la rapidesa amb la qual la funció σ canvia en introduir petits canvis en el valor z_j^L. Podem expressar la mateixa equació de forma vectoritzada, millorant així el seu rendiment amb l'equació 4.6.

$$\delta^L = \nabla_a J(\Theta) \odot \sigma'(z^L). \tag{4.6}$$

On $\nabla_a J(\Theta)$ és un vector compost per les derivades parcials $\partial J(\Theta)/\partial a_j^L$. Equació per computar l'error δ^l en funció de l'error de la capa superior δ^{l+1}.

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l), \tag{4.7}$$

L'equació 4.7, suposant que disposem de l'error δ^{l+1} en la capa $l+1$, descriu com l'error es propaga per les connexions de la capa $l+1$ fins a la capa l utilitzant la

subexpressió ($(w^{l+1})^T\delta^{l+1}$). Finalment es propaga a través de la funció d'activació $\sigma(z^l)$, computant δ^l. En combinar l'equació 4.5 amb l'equació 4.7, es pot computar l'error per qualsevol capa de la xarxa. Primer es començarà aplicant l'equació 4.5 a l'última capa i després s'aplicarà l'equació 4.7 per computar la resta d'errors: δ^{L-1},δ^{L-2},...,δ^1. Tot seguit es defineix l'equació que expressa la velocitat de canvi del cost respecte qualsevol biaix de la xarxa.

$$\frac{\partial J(\Theta)}{\partial b_j^l} = \delta_j^l \tag{4.8}$$

La següent equació expressa la velocitat de canvi del cost respecte dels paràmetres de la xarxa.

$$\frac{\partial J(\Theta)}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l. \tag{4.9}$$

L'equació 4.9 descriu com computar $\partial J(\Theta)\partial w_{jk}^l$ en funció de l'error de la capa superior l i de l'activació de la capa $l-1$. Cal tenir en compte que si el valor de a_k^{l-1} és gaire petit, l'aprenentatge serà més lent. És a dir, les activacions properes a zero disminueixen la velocitat d'aprenentatge. En el cas particular de la funció d'activació sigmoide, quan les activacions $\sigma(z_j^L)$ són aproximadament 0 o 1. El terme $\sigma'(z_j^L)$ en l'equació 4.5 dóna valors propers a zero. Per descriure aquesta dinàmica se sol dir que la neurona està saturada. Com a resultat d'aquesta saturació, la neurona està aprenent molt lentament.

4.3.1 Algoritme

L'algoritme *backpropagation* ens proporciona una forma de computar els gradients de la funció de cost respecte als paràmetres de la xarxa. En aquesta subsecció s'utilitzaran les equacions 4.5, 4.7, 4.8 i 4.9 en format d'algoritme.

1. Entrada x: assignem x a l'activació a^1
2. Pas endavant: per cada $l = 2, 3, \ldots, L$ computem $z^l = w^l a^{l-1} + b^l$ i $a^l = \sigma(z^l)$
3. Computem l'error δ^L: $\delta^L = \nabla_a J(\Theta) \odot \sigma'(z^L)$
4. Propaguem l'error: per cada $l = L-1, L-2, \ldots, 2$ computem $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
5. Sortida: computem els gradients de la funció de cost respecte als paràmetres de la xarxa. $\frac{\partial J(\Theta)}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ i $\frac{\partial J(\Theta)}{\partial b^l_j} = \delta^l_j$

Finalment s'utilitzarà l'algoritme *gradient descent* per actualitzar els paràmetres.

5 Deep learning

5.1 Introducció

Que és el *deep learning*? Fent una cerca es poden trobar moltes definicions d'aquest concepte. La definició més general entén el *deep learning* com la recerca feta en el camp de les xarxes neuronals des de l'any 2006. En l'any 2006 es va presentar un article [7] en el qual es presentava una nova tècnica per entrenar xarxes neuronals compostes de múltiples capes ocultes utilitzant aprenentatge no supervisat. Aquesta nova tècnica va despertar de nou l'interès en les xarxes neuronals. En l'any 1970, amb la creació de l'algoritme *backpropagation*, es va provar d'implementar xarxes neuronals amb múltiples capes ocultes utilitzant aprenentatge supervisat sense obtenir bons resultats. En l'any 1991, Sepp Hochreiter va descobrir el principal motiu dels fracassos anteriors, un problema anomenat *vanishing gradient*. El *vanishing gradient* va ser una de les causes principals que va frenar l'èxit de les xarxes neuronals multicapa en l'època anterior al *deep learning*.

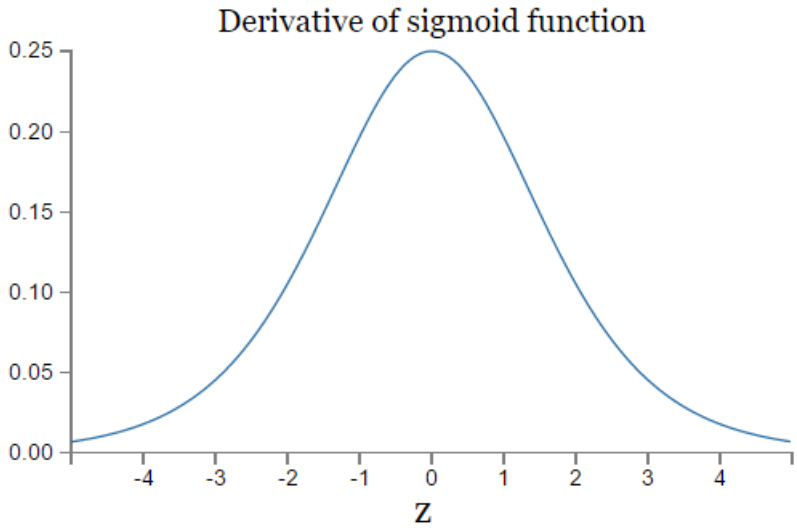

Figura 5.1: L'eix y representa els valors de la derivada de la funció sigmoide respecte Z

5.1.1 El problema del *vanishing gradient*

Per explicar el *vanishing gradient* s'utilitzarà una xarxa neuronal amb un valor d'entrada, tres capes ocultes amb una única neurona en cada capa i una sortida. Per entendre el problema s'estudiarà el gradient $\partial J(\Theta)/\partial b_1$ utilitzat per actualitzar el biaix de la neurona en la primera capa oculta.

$$\frac{\partial J(\Theta)}{\partial b_1} = \sigma'(z_1)\, w_2\sigma'(z_2)\, w_3\sigma'(z_3)\, w_4\sigma'(z_4)\, \frac{\partial J(\Theta)}{\partial a_4} \tag{5.1}$$

Exceptuant l'últim terme $\frac{\partial J(\Theta)}{\partial a_4}$, l'equació 5.1 està composta de productes d'expressions amb la següent forma: $w_j\sigma'(z_j)$. La figura 5.1 mostra que el valor màxim que pot obtenir l'expressió $\sigma'(z_j)$ és $1/4$. Això fa que en molts casos el gradient obtingui un valor igual a zero durant el pas de propagació de l'error, ja que com més capes tingui la xarxa més vegades s'haurà de multiplicar el valor $\sigma'(z_j)$ i això pot derivar en un gradient igual a zero, aturant així l'aprenentatge de la xarxa neuronal. Aquest problema apareix tant en les xarxes neuronals multicapa com en les xarxes recurrents.

5.1.2 Recerca en *deep learning*

La recerca de l'última dècada en xarxes neuronals s'ha centrat en els següents apartats:

- Cerca de funcions de cost més efectives, com per exemple la funció de cost *cross-entropy* introduïda anteriorment.
- Noves funcions d'activació no lineals.
- Noves arquitectures de xarxes neuronals com la xarxa neuronal convolucional o la xarxa neuronal recurrent LSTM.
- Exploració de tècniques de regularització més potents com el *dropout*.
- Cerca d'algoritmes d'optimització més potents que el *gradient descent*.
- Utilització d'aprenentatge no supervisat per a la inicialització dels paràmetres de xarxes neuronals multicapa.

5.1.3 Factors externs claus en l'èxit del Deep learning

El primer factor clau que ha propiciat l'èxit del *deep learning* és la quantitat de dades disponibles. Conjunts de dades com l'*ImageNet* [5], amb més de catorze milions d'imatges etiquetades, proporcionen quantitats de dades massives. Utilitzar conjunts de dades grans és important per entrenar correctament xarxes neuronals amb múltiples capes ocultes. El segon factor clau ha estat la utilització de GPUs a l'hora d'entrenar les xarxes neuronals. L'arquitectura paral·lela de la GPU està pensada per computar operacions amb vectors i matrius, ja que aquestes operacions són necessàries en computació gràfica i en els videojocs. Les xarxes neuronals utilitzen de forma molt intensiva la multiplicació de vectors i matrius.

5.2 Funcions d'activació

Funció d'activació sigmoide

La funció d'activació utilitzada en l'època anterior a l'explosió del *deep learning* és la funció sigmoide. En la presentació del problema *vanishing gradient* s'argumenta el problema principal derivat de la utilització d'aquesta funció d'activació. El problema és que les neurones amb valors propers a les asímptotes de la funció sigmoide no propaguen correctament els gradients. El segon problema que presenta la funció sigmoide és que l'expressió involucra el càlcul d'una funció exponencial costosa de calcular.

Funció d'activació tangent hiperbòlica o tanh

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.2}$$

La funció d'activació tanh és una versió escalada de la funció d'activació sigmoide. La funció tanh està centrada a l'origen i els valors de sortida de la funció estan compresos entre $[-1, 1]$. Aquesta funció hereta els problemes ja presentats en la funció sigmoide.

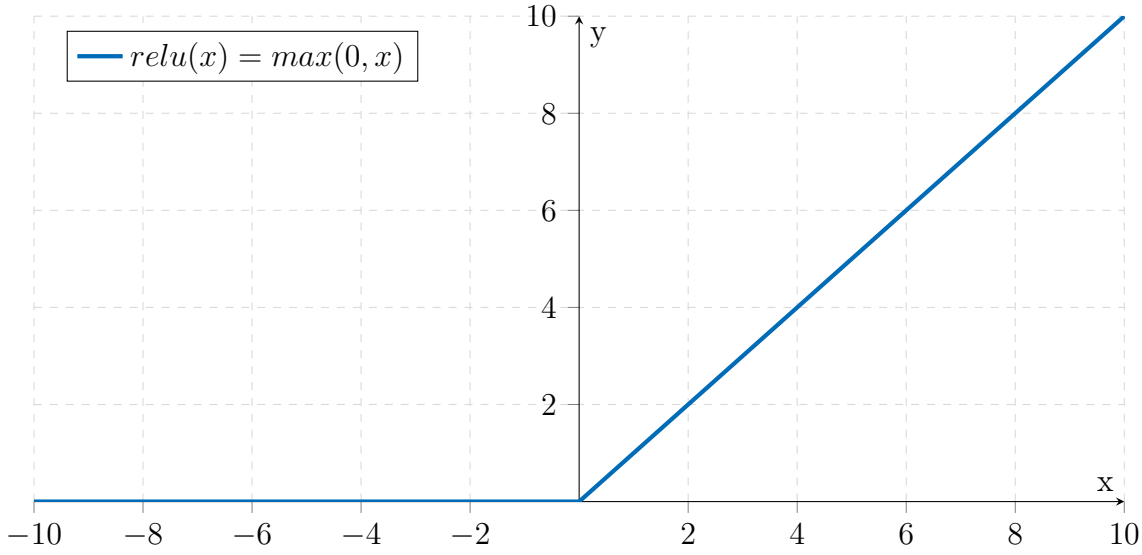

Figura 5.2: Funció ReLU

Funció d'activació Rectified Linear Unit o ReLU

$$relu(x) = max(0, x) \tag{5.3}$$

En l'any 2012 es va publicar l'article [10] que presenta la funció d'activació anomenada ReLU. En els experiments duts a terme en l'article s'observa que s'incrementa la velocitat d'aprenentatge dels paràmetres per sis. Per entendre l'efectivitat de la funció d'activació ReLU s'ha d'observar que, en la figura 5.2, la derivada sempre és igual a 1 en la regió positiva. Aquest fet resol el problema del *vanishing gradient*. La funció ReLU és computacionalment molt efectiva i és la funció d'activació més utilitzada en *deep learning*. Un problema associat a la funció d'activació ReLU és que, en la regió negativa de la funció, la derivada és igual a zero. En el cas que la combinació d'inicialització de paràmetres faci que la sortida de la neurona sempre sigui negativa, aquella neurona no propagarà mai el gradient. Aquest fet s'anomena *dead* ReLU.

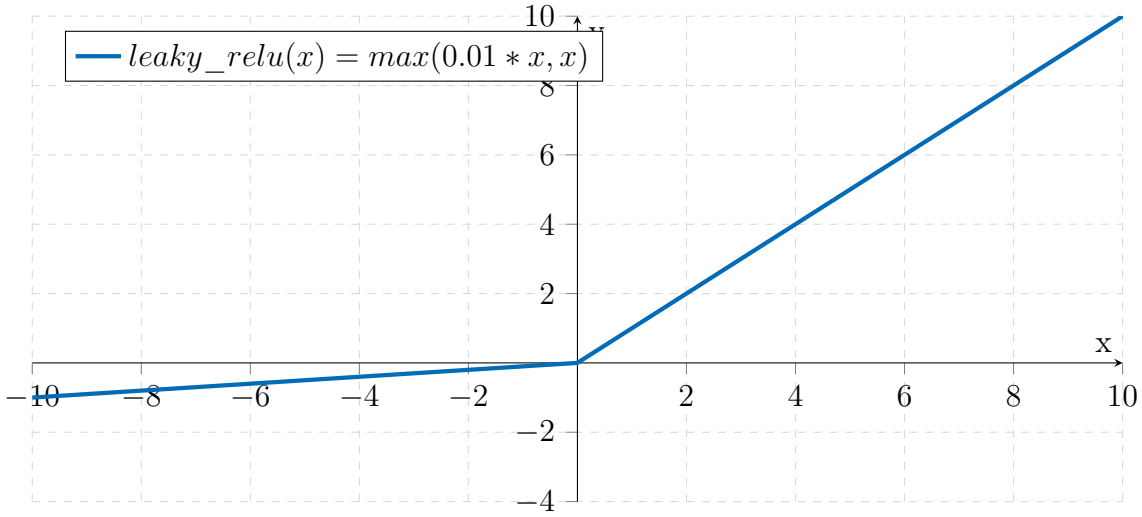

Figura 5.3: Funció d'activació *leaky* ReLU

Funció d'activació leaky ReLU

$$leaky_relu(x) = max(0.01 * x, x) \tag{5.4}$$

La funció d'activació *leaky* ReLU obté tots els beneficis de la funció d'activació ReLU i evita el problema *dead* ReLU, ja que la derivada en la zona negativa de la funció d'activació serà diferent de zero. En la figura 5.3 s'observa el comportament de la funció *leaky* ReLU.

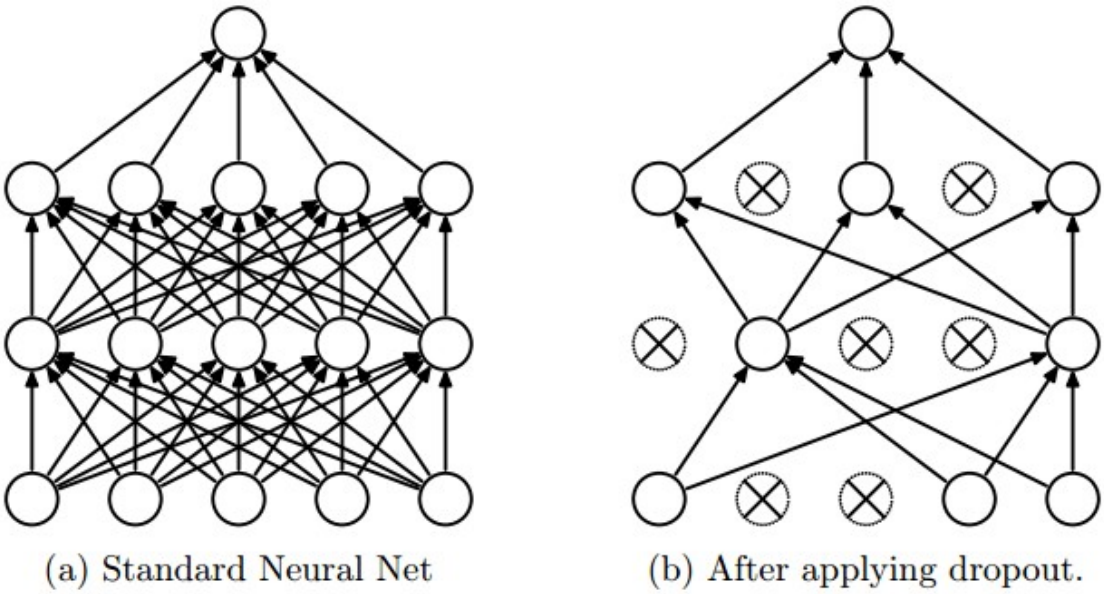

Figura 5.4: De l'article [19]. Aplicació de la tècnica de regularització *dropout*

5.3 Regularització *dropout*

La tècnica *dropout* [19] és una tècnica de regularització diferent. A diferència de la regularització L2, el *dropout* no modifica la funció de cost. El que fa és modificar l'arquitectura de la xarxa neuronal. En aquesta secció es descriuran els mecanismes bàsics del *dropout*. Per explicar aquest concepte, se suposarà que s'està entrenant una xarxa neuronal. En cada pas endavant, descrit en la subsecció 4.3.1, s'eliminarà de forma aleatòria i temporal la meitat de les neurones de les capes ocultes. L'entrada i la sortida es mantindran intactes. En cada iteració de l'algoritme *backpropagation* només es propagarà l'error en les neurones que no s'han eliminat. Un cop s'hagi finalitzat la propagació endarrere i s'hagin actualitzat els paràmetres de la xarxa, el procés es repetirà. Primer es restauraran les neurones de la capa oculta eliminades i després es tornarà a eliminar la meitat de les neurones de forma aleatòria. Aquest procés es repeteix durant tot el procés d'aprenentatge.

Per què el *dropout* redueix el sobreajustament de la xarxa?

Si s'entrenen múltiples xarxes neuronals utilitzant el mateix conjunt de dades, el resultat d'aquestes xarxes pot ser diferent. La diferència es pot donar per la inicialització dels paràmetres o per una arquitectura diferent. En obtenir diferents models, podem utilitzar algun mecanisme de votació per decidir quina és la sortida més adequada. Per exemple, es suposarà que s'han entrenat cinc xarxes neuronals diferents amb l'objectiu de classificar dígits. En cas que tres de les cinc xarxes neuronals estiguin classificant un dígit com un tres, probablement l'entrada de les xarxes neuronals serà un tres. Aquesta tècnica d'agregar diferents models és una forma potent de reduir el sobreajustament però és molt costosa computacionalment, ja que s'han d'entrenar cinc xarxes neuronals per poder obtenir un benefici. Aquesta tècnica redueix el sobreajustament, ja que cada xarxa pot sobreajustar-se de formes diferents i probablement una votació amb diferents models pot reduir el sobreajustament global. Quan s'aplica *dropout* en cada iteració de l'algoritme *backpropagation* s'utilitza un subconjunt de neurones ocultes diferent i es pot intuïr que és com si s'apliques a una xarxa neuronal diferent. Un cop s'ha acabat la fase d'aprenentage s'utilitzen totes les neurones de les capes ocultes. L'efecte final que proporciona el *dropout* és com el d'una votació o mitjana entre diferents models i aquest motiu fa que redueixi el sobreajustament. Els resultats que proporciona el *dropout* són molt bons i és una tècnica de regularització a tenir en compte en *deep learning*.

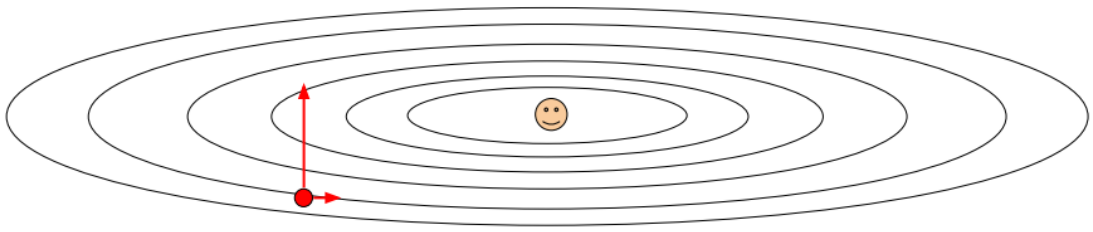

Figura 5.5: Del curs [8]. Superfície de la funció de cost amb un gradient elevat representat amb una fletxa vermella més llarga, que és la direcció vertical, i un gradient més petit representat amb la fletxa vermella petita que és la direcció horitzontal.

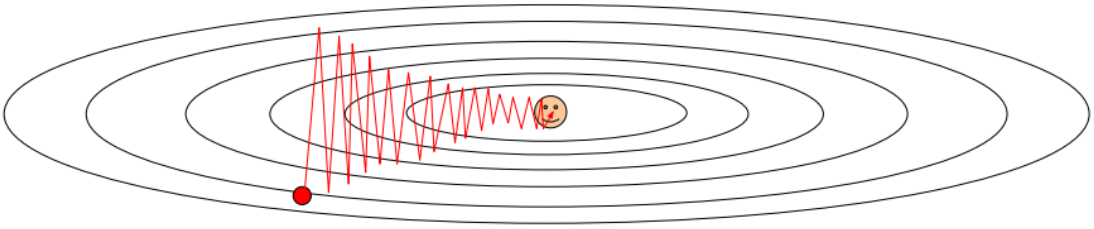

Figura 5.6: Del curs [8]. Aplicació de l'algoritme *gradient descent* en la funció de cost de la figura 5.5

5.4 Optimització de l'aprenentatge

L'algoritme per optimitzar l'aprenentatge ja presentat en la secció 2.3 és el *gradient descent*. L'ús d'aquest algoritme genera un problema.

Problema al utilitzar l'algoritme *gradient descent*

En la figura 5.5, s'observa la superfície d'una funció de cost que es vol optimitzar. Se suposarà que la superfície és molt empinada verticalment i amb una amplada petita. En la figura 5.5, el gradient es representa amb una fletxa vermella. El gradient en la direcció vertical és més gran, ja que la direcció vertical ens proporciona una variació més elevada de la funció de cost. En canvi, el gradient en la direcció horitzontal és més petit. La figura 5.6 il·lustra el problema derivat d'optimitzar la funció de cost en una superfície d'aquestes característiques, l'avenç en la direcció horitzontal és molt lent.

Solució: *momentum*

La tècnica *momentum* modifica l'algoritme *gradient descent*. En primer lloc, s'introdueix la noció de velocitat v. En utilitzar *momentum*, el gradient modifica la velocitat i no la posició directament. En segon lloc s'introduirà la noció de fricció μ que pretén reduir de forma gradual la velocitat.

Algoritme *gradient descent*

$$\theta := \theta - \alpha \nabla J(\theta)$$

Algoritme *gradient descent* amb *momentum*

$$v = \mu v - \alpha \nabla J(\theta) \tag{5.5}$$

$$\theta = \theta + v \tag{5.6}$$

Per entendre el significat de l'equació 5.5. Primer es considerarà el cas on $\mu = 1$. Aquest valor significa que la fricció es inexistent. En aquest cas el gradient $\nabla J(\theta)$ modifica la velocitat v, i la velocitat controla el rati de canvi dels paràmetres θ. Intuïtivament es genera velocitat en agregar el gradient de forma repetitiva. Quan la direcció, en cada pas de l'algoritme *gradient descent*, és constant es adequat incrementar la velocitat. Però quan aquesta direcció canvia, s'ha de poder disminuir la velocitat. En cas contrari es podria fer un pas equivocat. La disminució de la velocitat està controlada pel paràmetre μ. Concretament l'expressió $1 - \mu$ defineix la quantitat de fricció. Quan $\mu = 0$ la fricció és tan elevada que no es genera velocitat i s'obté l'algoritme *gradient descent* sense *momentum*. En la pràctica [8] es solen escollir valors μ entre 0.5 i 0.99.

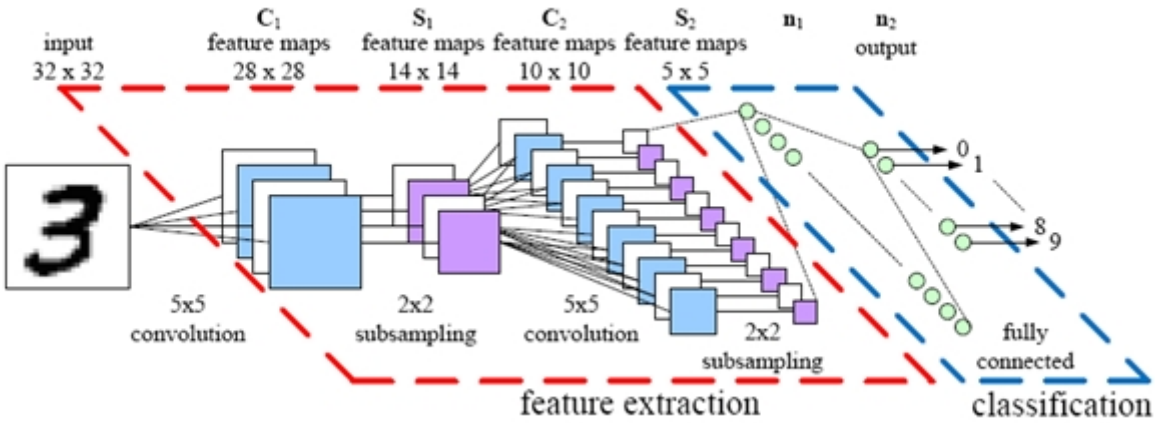

Figura 5.7: Arquitectura d'una xarxa neuronal convolucional

5.5 Xarxa neuronal convolucional

5.5.1 Motivacions

Actualment, les xarxes neuronals convolucionals [8] són l'estat de l'art en tasques com la classificació o segmentació d'imatges en l'àrea de visió per computador. Gran part de la recerca anterior a l'explosió del *deep learning* es basava en l'extracció de característiques de les imatges, que posteriorment s'utilitzaven en tasques de classificació. Aquesta metodologia constava d'una part manual, decidir quines característiques extreure, i una part automàtica, ja que el classificador és entrenat utilitzant aprenentatge supervisat. La motivació principal del model presentat és que, tant en l'extracció de les característiques com en la classificació, s'utilitza aprenentatge supervisat simplificant la metodologia. Un altre avantatge significatiu és que aquesta nova arquitectura permet la compartició de paràmetres. La compartició de paràmetres permet localitzar la mateixa característica, per exemple la roda d'un cotxe, en qualsevol posició de la imatge. Això fa que la xarxa sigui més robusta, ja que és invariant a la posició dels objectes en la imatge.

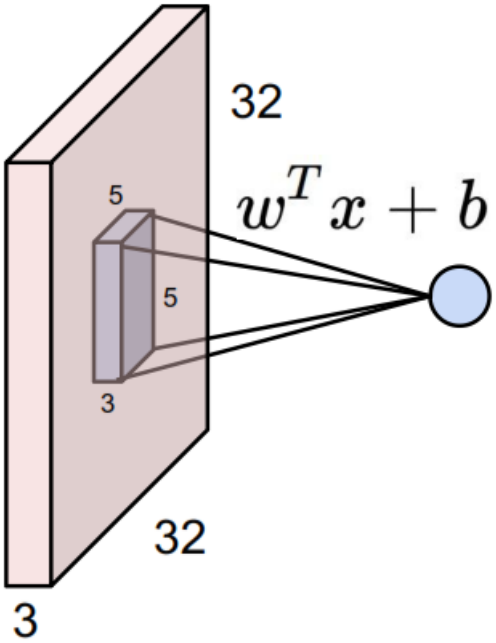

Figura 5.8: Del curs [8]. Aplicació d'un filtre en la part central d'una imatge en una capa convolucional. On $x \in \mathbb{R}^{5x5x3}$ és la part de la imatge on apliquem el filtre i $W \in \mathbb{R}^{5x5x3}$ són els paràmetres del filtre.

5.5.2 Capes

Com s'observa en la figura 5.7, una xarxa convolucional la podem separar en dues parts: l'extractor de característiques i el classificador. En aquesta subsecció s'introduiran les diferents capes que conformen una xarxa neuronal convolucional. En la capa totalment connectada, presentada anteriorment, les entrades s'estructuren com una línia vertical de neurones. En les xarxes convolucionals és útil pensar en cada capa de l'extractor de característiques com un volum de neurones. En el cas d'una imatge, aquest volum seria alçada x amplada x RGB. Com ja és habitual, es connectarà la capa d'entrada a una capa oculta de neurones.

Capa convolucional

Hi ha dos conceptes principals que definiran la capa convolucional: els filtres i els mapes de característiques. De forma intuïtiva, el propòsit principal d'una capa convolucional és detectar característiques visuals de les imatges: arestes, línies, gotes de color, etc. Aquesta tasca és portada a terme per una primera

capa de neurones ocultes que estan connectades a la capa d'entrada. En la capa convolucional no es connecten totes les neurones d'entrada a la capa oculta, només es fa per petites zones localitzades en l'espai de les neurones d'entrada. Visualment, aquest fet queda representat en la figura 5.8. En la figura, cada neurona de la capa oculta està connectada a una petita regió de 5x5x3 neurones de la capa d'entrada. Intuïtivament, es pot pensar en una finestra de grandària 5x5x3 que va recorrent tota l'entrada de dimensió 32x32x3 que representa la imatge. Aquesta finestra es va desplaçant per tota la capa d'entrada i en cada desplaçament de la finestra hi ha una neurona en la capa oculta que processa aquesta informació. Inicialment, la finestra se situa en la cantonada superior a l'esquerra de la imatge i computa l'activació de la primera neurona de la capa oculta. Equació que calcula l'activació de neurona j, k de la capa oculta:

$$ReLU\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4}\sum_{d=0}^{2} w_{l,m,d} a_{j+l,k+m,d}\right). \tag{5.7}$$

On $b \in \mathbb{R}$ és el biaix compartit. $w_{l,m,d} \in \mathbb{R}^{5,5,3}$ és el volum de paràmetres compartits del filtre. $a_{x,y,z}$ s'utilitza per denotar l'entrada en la posició x per l'alçada, y per l'amplada i z per la profunditat. Finalment s'aplica la funció d'activació ReLU per obtenir la neurona j, k de la capa oculta. Un cop computada la primera neurona, la finestra es desplaçarà una posició cap a la dreta per connectar les 5x5x3 neurones de la capa d'entrada amb la segona neurona de la capa oculta. I així, successivament, anem recorrent tot l'espai de la capa d'entrada, d'esquerra a dreta i de dalt a baix. Analitzant una mica el cas concret que s'ha proposat s'observa que, si tenim una entrada de 32x32x3 píxels i una finestra de dimensió 5x5x3, aquesta combinació defineix un espai de 28x28x1 neurones en la primera capa oculta. Un detall important de la capa convolucional és que s'usa el mateix volum de paràmetres w i el mateix biaix b per a totes les neurones de la capa oculta. En el nostre cas, per les 32x32x3 neurones. Aquesta compartició de paràmetres redueix de manera dràstica el nombre de paràmetres que tindria una capa totalment connectada. Aquest volum de paràmetres compartit juntament amb el biaix s'anomena *kernel* o *filter* en el context de les xarxes convolucionals.

Un *kernel* definit per la matriu w i el biaix b només permet detectar una

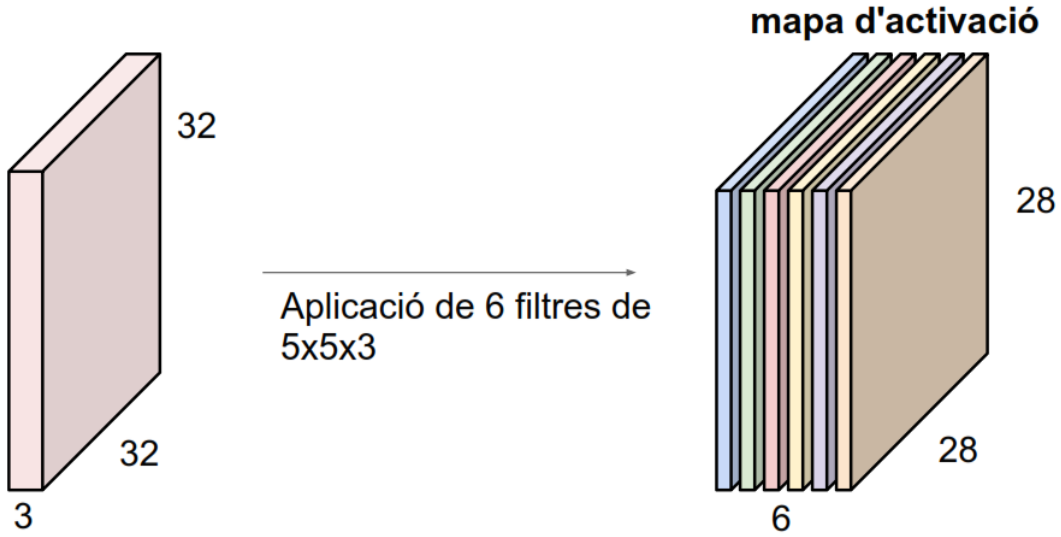

Figura 5.9: Del curs [8]. Aplicació de sis filtres sobre l'exemple de la figura 5.8.

característica concreta. Per tant, per poder realitzar reconeixement d'imatges s'utilitzaran múltiples *kernels*, un per a cada característica que es vulgui detectar. Per aquest motiu, una capa convolucional completa comprèn diversos *kernels*. En la figura 5.9 s'observa com l'aplicació de sis filtres genera un volum d'activació de 28x28x6 neurones.

Paràmetres que modifiquen el comportament de la capa convolucional

- El paràmetre *stride* determina quantes posicions es desplaça el filtre abans de computar una activació. En la figura 5.9 s'ha utilitzat un *stride*=1. Utilitzar valors *stride* més elevats evita certa superposició entre un filtre i el següent i redueix la mida del mapa d'activació resultant.

- El paràmetre *padding* soluciona el següent problema: en aplicar els filtres sobre el volum d'entrada, l'alçada i l'amplada del volum es redueix com s'observa en la figura 5.9. Aquest fet en no és gaire desitjable en alguns casos , ja que ens limita la capacitat d'agregar més capes convolucionals a la xarxa. Per resoldre aquest problema s'afegeix un marge anomenat *padding*, per tal que els mapes d'activació preservin l'alçada i l'amplada de l'entrada. Aquest marge es pot visualitzar com un embolcall de zeros que rodeja la imatge. Per calcular l'alçada o l'amplada d'un mapa d'activació s'utilitzarà

l'equació 5.8:

$$mida_mapa_activacions = \frac{(mida_volum - mida_filtre + 2 * padding)}{stride} + 1 \tag{5.8}$$

On $mida_volum$ representa l'alçada o l'amplada del volum d'entrada i $mida_filtre$ representa l'alçada o l'amplada del filtre. La variable $mida_mapa_activacions$ ha de donar obligatòriament un valor enter. En cas contrari, les mides del volum d'entrada i del filtre en combinació amb el valor de desplaçament *stride* i el valor *padding* fan que el filtre no es pugui aplicar correctament a tota la imatge.

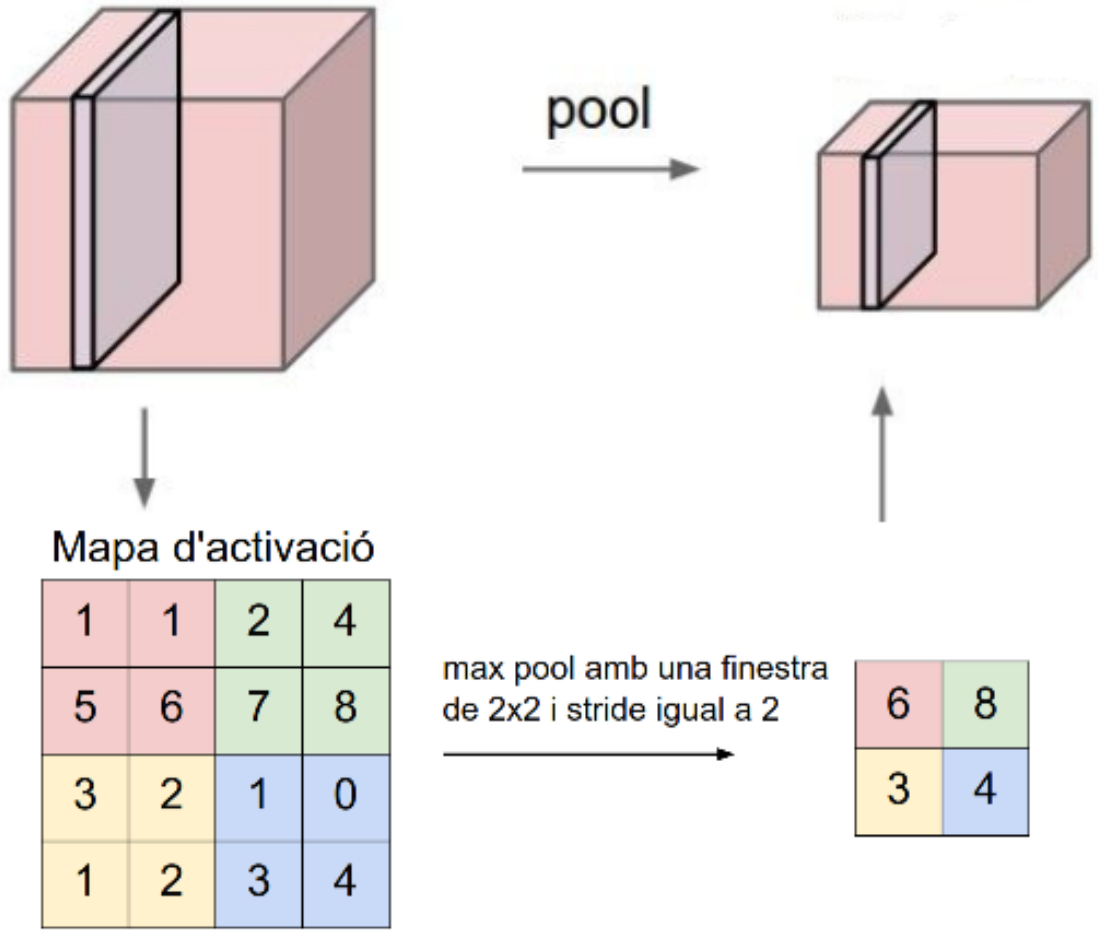

Figura 5.10: Del curs [8]. Aplicació de la capa de pooling

Capa de *pooling*

Normalment, la capa de *pooling* està situada entre successives capes convolucionals. La seva funció principal és reduir la dimensió resultant de la capa anterior i d'aquesta forma reduir el nombre de paràmetres de la xarxa. En reduir el nombre de paràmetres, s'observa un benefici pel que fa al sobreajustament de la xarxa. La capa de *pooling* opera per separat en cada mapa d'activació i aplica funcions d'agregació com MAX o AVG i desplaça una finestra com en el cas dels filtres, aplicant així l'operació d'agregació. En la figura 5.10 s'observa com l'operació de *pooling* redueix a la meitat l'alçada i l'amplada de l'entrada. La figura també mostra un exemple concret aplicant la funció d'agregació MAX.

Capa FC

L'arquitectura de xarxes convolucionals agrega una o dues capes en les quals les neurones estan totalment connectades amb la capa anterior. Aquesta part de l'arquitectura és el classificador. Aquest classificador podria ser reemplaçat per un altre model com ara el *support vector machine*.

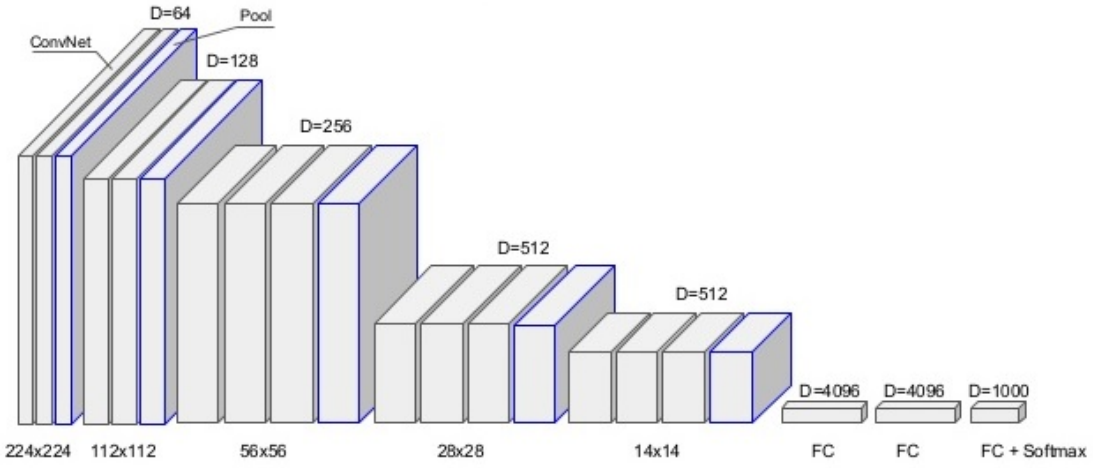

Figura 5.11: Arquitectura de la xarxa VGGNET

5.5.3 Cas d'estudi VGGnet

En aquesta subsecció s'estudiarà l'arquitectura d'una xarxa neuronal convolucional [17] que va guanyar la competició ILSVRC [16] l'any 2014. La seva contribució principal va ser demostrar que la profunditat de la xarxa és clau per millorar el rendiment. L'arquitectura de la xarxa consta de 16 capes i un dels components més atractius de l'arquitectura és la seva simplicitat. Tota la xarxa està composta de capes convolucionals amb filtres d'alçada i amplada 3x3 amb *stride* i *padding* d'una unitat. La decisió d'afegir un *padding* igual a 1 és molt encertada, ja que permet agregar diferents capes convolucionals preservant l'alçada i l'amplada del volum d'activacions on apliquen la capa convolucional. Com es pot observar en la taula 5.5.3, cada dues o tres capes convolucionals s'aplica una capa de *pooling* amb finestra de 2x2 que utilitza l'operació d'agregació *max* amb un *stride* igual a dues unitats. La capa de *pooling* mai utilitza *padding*, ja que la funció de la capa de *pooling* és reduir paràmetres. La capa de *pooling* utilitzada en aquesta arquitectura redueix a la meitat l'alçada i l'amplada del volum on s'aplica. Com s'observa en la taula 5.5.3, després d'aplicar cada capa de *pooling* es redueix molt el consum de memòria. Al reduir la memòria podem afegir més capes. En la part final de la xarxa hi ha tres capes totalment connectades en les quals s'hi troba gran part dels paràmetres de la xarxa. L'última de les capes classifica entre les 1000 classes de la competició ILSVRC.

Capa	Mida	Memòria	Paràmetres
Imatge	[224x224x3]	224*224*3=150K	0
CONV3-64	[224x224x64]	224*224*64=3.2M	(3*3*3)*64 = 1,728
CONV3-64	[224x224x64]	224*224*64=3.2M	(3*3*64)*64 = 36,864
POOL2	[112x112x64]	112*112*64=800K	0
CONV3-128	[112x112x128]	112*112*128=1.6M	(3*3*64)*128 = 73,728
CONV3-128	[112x112x128]	112*112*128=1.6M	(3*3*128)*128 = 147,456
POOL2	[56x56x128]	56*56*128=400K	0
CONV3-256	[56x56x256]	56*56*256=800K	(3*3*128)*256 = 294,912
CONV3-256	[56x56x256]	56*56*256=800K	(3*3*256)*256 = 589,824
CONV3-256	[56x56x256]	56*56*256=800K	(3*3*256)*256 = 589,824
POOL2	[28x28x256]	28*28*256=200K	0
CONV3-512	[28x28x512]	28*28*512=400K	(3*3*256)*512 = 1,179,648
CONV3-512	[28x28x512]	28*28*512=400K	(3*3*512)*512 = 2,359,296
CONV3-512	[28x28x512]	28*28*512=400K	(3*3*512)*512 = 2,359,296
POOL2	[14x14x512]	14*14*512=100K	0
CONV3-512	[14x14x512]	14*14*512=100K	(3*3*512)*512 = 2,359,296
CONV3-512	[14x14x512]	14*14*512=100K	(3*3*512)*512 = 2,359,296
CONV3-512	[14x14x512]	14*14*512=100K	(3*3*512)*512 = 2,359,296
POOL2	[7x7x512]	7*7*512=25K	0
FC	[1x1x4096]	4096	7*7*512*4096 = 102,760,448
FC	[1x1x4096]	4096	4096*4096 = 16,777,216
FC	[1x1x1000]	1000	4096*1000 = 4,096,000

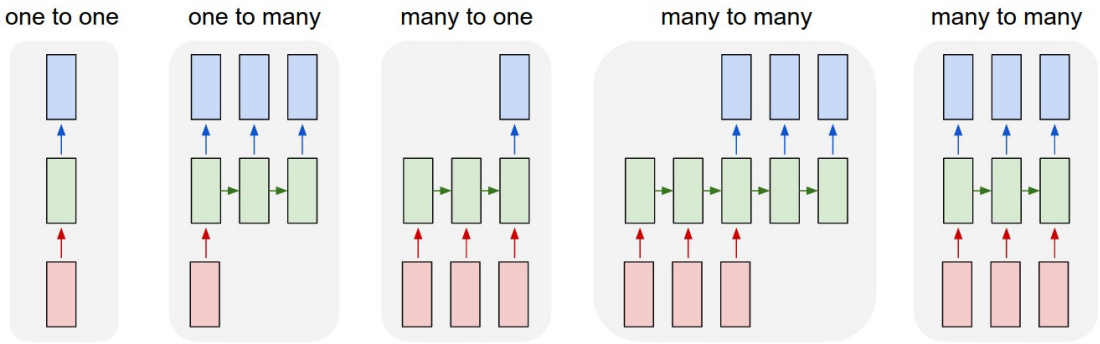

Figura 5.12: Del blog [9]. Diferents exemples de xarxes recurrents desplegades en funció del temps. Cada rectangle representa un vector, les fletxes representen funcions (per exemple, una transformació afí). Els vectors d'entrada estan representats de color vermell; els vectors de sortida, de color blau, i els vectors verds mantenen el context de la xarxa recurrent

5.6 Xarxa neuronal recurrent

5.6.1 Motivació

Les xarxes neuronals recurrents són una família de xarxes neuronals especialitzada en el processament de seqüències $(x^{(1)},x^{(2)},...,x^{(t)})$, on t representa un instant de temps. Una motivació important d'aquesta arquitectura és el concepte de compartició de paràmetres, com en el cas de les xarxes convolucionals. La compartició de paràmetres de les dues arquitectures és diferent. Per exemplificar la diferència, suposarem que es disposa d'una capa convolucional d'una dimensió que processa una seqüència temporal de dades d'entrada. L'aplicació de la capa convolucional permet la compartició de paràmetres en diferents instants de temps. La sortida de la capa convolucional serà una seqüència en la qual cada membre de la sortida serà funció d'una petita part del veïnatge de l'entrada. En canvi, en el cas de les xarxes recurrents, cada sortida és funció de les entrades anteriors, és a dir, del context de la seqüència i de l'entrada actual. El fet de poder fer prediccions en funció del context i de l'entrada actual fa que sigui una arquitectura realment interessant. El context s'expressa en forma de vector $h^{(t)} \in \mathbb{R}^{n_h}$ i s'anomena estat ocult. La n_h limita la capacitat de la xarxa recurrent d'expressar un context més potent.

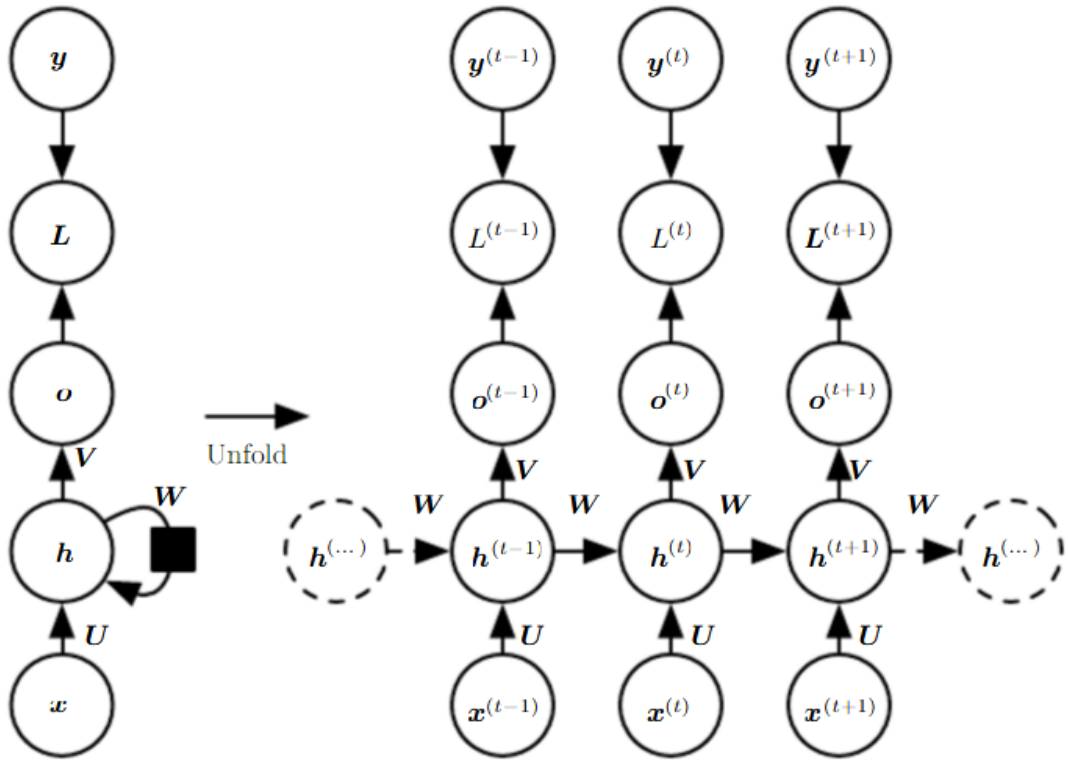

Figura 5.13: Del llibre [2]. Arquitectura de xarxa recurrent. A la dreta s'observa la versió desplegada en diferents instants de temps de la xarxa recurrent

L'equació 5.9 actualitza l'estat ocult $h^{(t)}$ a partir de l'estat ocult anterior $h^{(t-1)}$ i l'entrada actual $x^{(t)}$ aplicant una funció f que dependrà de l'arquitectura de la xarxa recurrent.

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{5.9}$$

5.6.2 Exemple d'una arquitectura completa

En aquesta subsecció es presentarà una arquitectura de xarxa recurrent completa. En la figura 5.13 s'observa l'arquitectura de la xarxa recurrent. Per cada entrada $x^{(t)} \in \mathbb{R}^{n_x}$, la xarxa produirà una sortida $y^{(t)} \in \mathbb{R}^{n_y}$. S'assumirà que el valor de sortida és un valor discret. Una forma natural de representar la sortida és assignant una probabilitat per cada possible valor de la variable discreta. Per computar la sortida $y^{(t)}$ primer s'ha d'actualitzar l'estat ocult $h^{(t)}$ aplicant la funció f que queda

representada en l'equació 5.10.

$$h^{(t)} = tanh(b + Wh^{(t-1)} + Ux^{(t)}) \tag{5.10}$$

L'equació 5.10 comença computant dues transformacions afins. La primera transformació afí s'aplica sobre l'estat ocult anterior $h^{(t-1)}$ utilitzant la matriu de paràmetres $W \in \mathbb{R}^{n_h,n_h}$. La segona transformació afí s'aplica sobre l'entrada $x^{(t)}$ utilitzant la matriu de paràmetres $U \in \mathbb{R}^{n_h,n_x}$. Al resultat de sumar les dues transformacions afins i el biaix $b \in \mathbb{R}^{n_h}$ s'aplica la funció d'activació *tanh* i s'obté com a resultat l'estat ocult actual $h^{(t)}$.

$$o^{(t)} = c + Vh^{(t)} \tag{5.11}$$

Un cop calculat l'estat ocult actual, l'equació 5.11 transformarà l'estat ocult amb un vector de sortida $o^{(t)} \in \mathbb{R}^{n_y}$, que contindrà l'evidència per cada possible valor de sortida de la xarxa recurrent. On n_y és la quantitat de valors que pot representar la variable discreta de sortida.

$$\hat{y}^{(t)} = softmax(o(t)) \tag{5.12}$$

Finalment transformem les evidències $o^{(t)}$ en un vector de probabilitats normalitzades $\hat{y}^{(t)}$ utilitzant la funció d'activació *softmax*. Tal com s'il·lustra en l'arquitectura de la figura 5.13, en cada instant de temps es processa un valor $x^{(t)}$ i es produeix un valor $y^{(t)}$. La funció de cost a partir d'una parella de seqüències $x^{(1)}$,...,$x^{(t)}$ i $y^{(1)}$,...,$y^{(t)}$ serà la suma dels valors de la funció de cost en cada instant de temps. Per exemple, si la a funció de cost que s'utilitza és la funció *cross-entropy*, s'obté la següent equació:

$$J(\Theta) = -\sum_t \log p_{model}(y^{(t)}|x^{(1)}, ..., x^{(t)}) \tag{5.13}$$

On $p_{model}(y^{(t)})|x^{(1)}, ..., x^{(t)})$ es calcula al llegir la component del vector de sortida $\hat{y}^{(t)}$ en la posició de l'etiqueta correcta $y^{(t)}$. Θ representa tots els paràmetres de l'arquitectura.

5.6.3 *Backpropagation*

Per computar els gradients en una xarxa recurrent s'aplica l'algoritme general de *backpropagation* amb una variació. L'algoritme de *backpropagation* no pot gestionar les connexions recurrents i, per tant, s'aplicarà l'algoritme en la versió desplegada de la xarxa recurrent de la figura 5.13. En la figura s'observa la xarxa recurrent com un graf de computació on els nodes estan indexats en funció del temps: $x^{(t)}$, $h^{(t)}$, $o^{(t)}$ i $L^{(t)}$ (En el treball es defineix la funció de cost amb una J, en la figura està representada amb una L). Les arestes transporten els paràmetres necessaris per computar les operacions dels nodes. Per cada node N s'ha de computar el gradient $\nabla_N J$ recursivament aplicant la regla de la cadena de càlcul. Aquesta versió del *backpropagation* s'anomena *backpropagation through time*. Hi ha una variant d'aquesta tècnica que consisteix només a propagar els gradients cada t instants de temps. Aquesta versió s'anomena *truncated backpropagation through time* i és l'opció recomanada per a seqüències molt llargues.

5.6.4 Modelant seqüències condicionades a un context

En aquesta subsecció es presenta una variant de l'arquitectura presentada. En aquesta variant s'introdueix un context en forma de vector, que es pot agregar de diferents maneres:

- Com una entrada afegida en cada instant de temps.
- Com l'estat inicial $h^{(0)}$.
- Com els dos casos anteriors.

5.6.5 Xarxa recurrent bidireccional

L'arquitectura de xarxa recurrent presentada fins al moment només és capaç de capturar informació del passat $x^{(1)}$,...,$x^{(t-1)}$ i del valor actual $x^{(t)}$. En algunes aplicacions pot ser interessant que la predicció del valor $y(t)$ depengui de tota la seqüència. Com el propi nom indica, una xarxa bidireccional combina una xarxa recurrent que es desplaça endavant a través del temps, que és la xarxa presentada fins al moment. La primera xarxa es combina amb una segona xarxa recurrent que es mou a la inversa i que comença al final de la seqüència en l'instant de temps t i es mou cap a l'inici de la seqüència.

5.6.6 Arquitectura *Encoder-Decoder* o *Sequence-to-Sequence*

Una arquitectura interessant per les seves aplicacions en la traducció automàtica i en el reconeixement de veu és l'arquitectura *Encoder-Decoder* observada en la figura 5.14. Aquesta arquitectura està formada per dues xarxes recurrents. La primera s'anomena *encoder* i processa l'entrada, per exemple una frase en anglès. L'*encoder* emet un context C que habitualment és l'últim estat oculta $h^{(t)}$ de la xarxa *encoder* que sumaritza el contingut de l'entrada $x^{(1)}, ..., x^{(n_x)}$. En aquest cas concret, cada $x^{(i)}$ serà una paraula en angles. La segona xarxa recurrent s'anomena *decoder* i està condicionada pel context C, tal com s'ha presentat en la subsecció 5.6.4, i per la traducció parcial que genera el *decoder*. Finalment, el *decoder* genera la sortida $y^{(1)}, ..., y^{(n_y)}$, que en el nostre cap pot ser una traducció en català on $y^{(1)}$ és una paraula en català. L'avantatge d'aquesta arquitectura és que l'entrada de l'*encoder* i la sortida del *decoder* poden ser seqüències de mides diferents. En el nostre cas, n_x equival a la llargada de la seqüència d'entrada i n_y a la llargada de la seqüència de sortida. En el context de la traducció automàtica té molt de sentit que l'entrada i la sortida siguin seqüències amb mides diferents, ja que potser la traducció d'una paraula en un context determinat requereix més d'una paraula per tal de poder expressar el mateix concepte en un altre idioma. Finalment, les dues xarxes s'entrenen amb aprenentatge supervisat per maximitzar la probabilitat mitjana $P(y^{(1)}, ..., y^{(n_y)}|x^{(1)}, ..., x^{(n_x)})$ per cada parella de seqüències

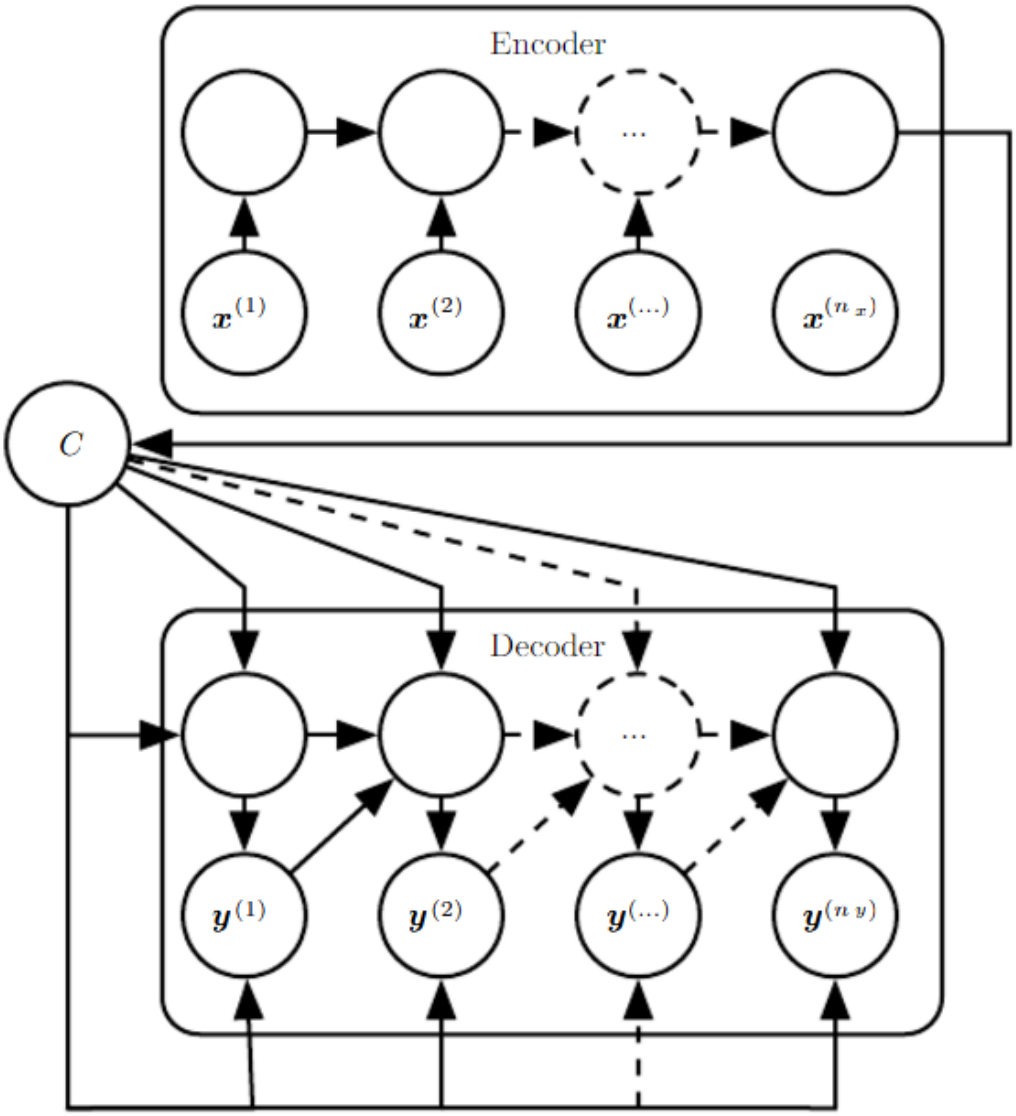

Figura 5.14: Del llibre [2]. Arquitectura *Encoder-Decoder*

(x, y) en el conjunt de dades d'entrenament. Una limitació d'aquesta arquitectura ocorre quan el context C de la sortida de l'*encoder* té una capacitat massa petita per resumir correctament una seqüència molt llarga. Aquest problema es pot adreçar utilitzant un concepte conegut com a *attention mechanism*.

5.6.7 Xarxa recurrent multicapa

Les xarxes recurrents multicapa presenten els mateixos beneficis que les xarxes neuronals multicapa, ja que poden expressar funcions més complexes. Això és gràcies a la capacitat de transformar les dades d'entrada en representacions que facilitin la tasca que porta a terme la xarxa recurrent.

5.6.8 LSTM

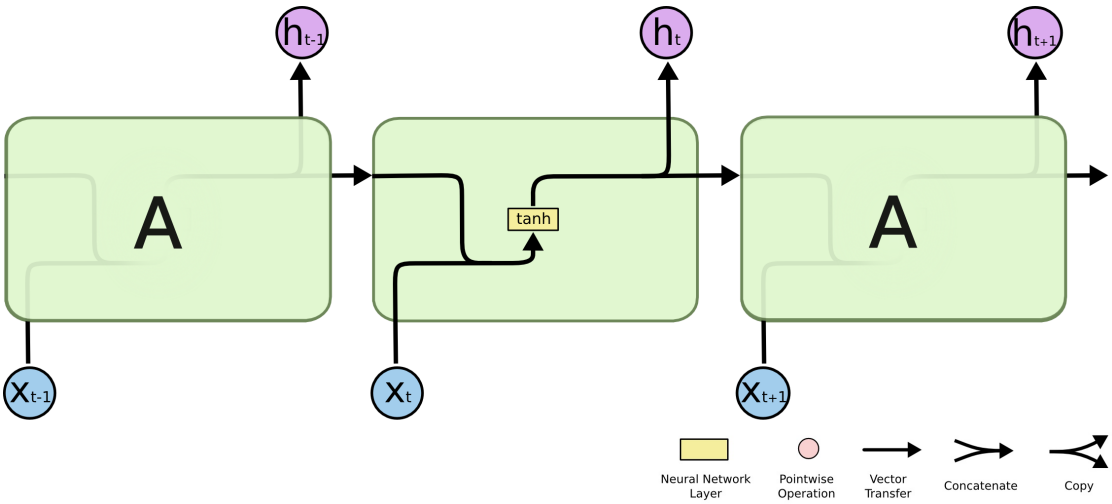

Figura 5.15: Arquitectura desplegada en diferents instants de temps d'una xarxa recurrent

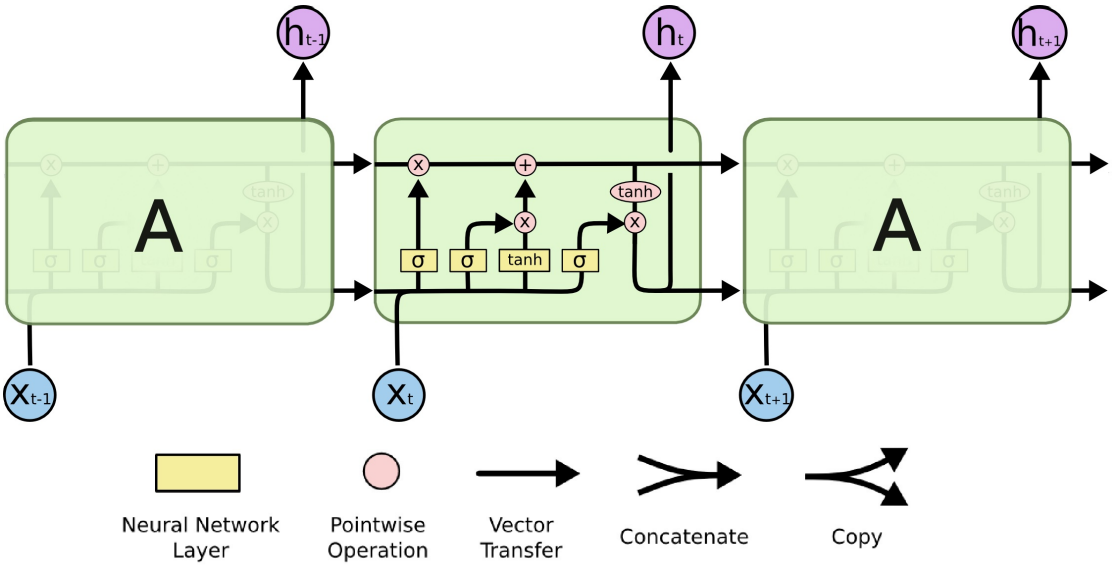

Figura 5.16: Arquitectura desplegada en diferents instants de temps d'una xarxa LSTM

Motivació

En aplicar l'algoritme *backpropagation through time* en la fase d'aprenentatge d'una xarxa recurrent, els gradients no es propaguen correctament. Això es deu al problema del *vanishing gradient*, introduït anteriorment. Per visualitzar aquest problema en el cas de les xarxes recurrents, s'ha implementat una xarxa recurrent en Python exemplificant la magnitud d'aquest problema. El codi Python es troba a la carpeta python/vanishing_gradient_xarxa_recurrent. En l'exemple s'ha generat una xarxa recurrent amb 25 neurones en la capa oculta utilitzant la funció d'activació ReLU. En aquest cas concret s'han obviat les entrades, ja que només es pretén observar la dinàmica dels gradients en la fase de *backpropagation through time*. En la fase de *backpropagation through time* es mostra per pantalla la norma del gradient de la capa oculta en cada instant de temps.

- Instant de temps 199: 8.04725655494e-09
- Instant de temps 198: 2.88195188106e-17
- Instant de temps 197: 1.42815302481e-25
- Instant de temps 196: 6.44451778954e-34
- Instant de temps 195: 3.00004430194e-42
- Instant de temps 194: 1.38257418413e-50
- Instant de temps 193: 6.39243496235e-59
- Instant de temps 192: 2.95246984779e-67
- Instant de temps 191: 1.36412174356e-75
- Instant de temps 190: 6.30191748587e-84
- Instant de temps 189: 2.91144009611e-92
- Instant de temps 188: 1.3450485663e-100
- Instant de temps 187: 6.21397790118e-109

- Instant de temps 186: 2.87078662914e-117
- Instant de temps 185: 1.32627106918e-125
- Instant de temps 184: 6.12722208919e-134
- Instant de temps 183: 2.83070730801e-142
- Instant de temps 182: 1.30775474431e-150
- Instant de temps 181: 6.0416794898e-159
- Instant de temps 180: 0.0
- ...
- Instant de temps 0: 0.0

Després de 19 instants de temps s'ha produït el problema del *vanishing gradient*. Ja que el vector gradient és igual a zero i ja no propaga cap informació. Per pal·liar aquest problema s'utilitza l'arquitectura LSTM. En les figures 5.15 i 5.16 s'observa una visió global de les diferències entre les dues arquitectures.

La cel·la C

El primer element nou de l'arquitectura LSTM és la cel·la C. La cel·la $C \in \mathbb{R}^{n_h}$ és una memòria on la xarxa recurrent pot esborrar i gravar informació en cada instant de temps de forma controlada per unes estructures anomenades *gates*. Les *gates* decideixen quina informació s'esborra de la cel·la C, quina informació s'afegeix a la cel·la C i quina informació s'utilitza de la cel·la C per construir l'estat ocult de la xarxa recurrent h. Cada *gate* està formada per una capa oculta de neurones i una funció d'activació sigmoide.

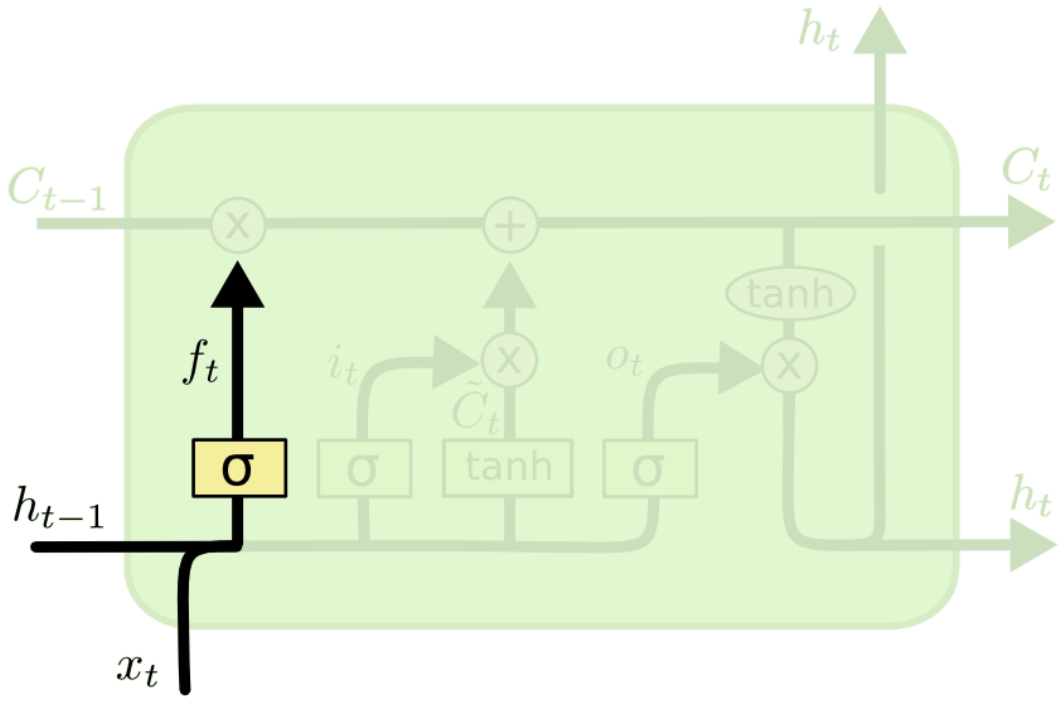

Figura 5.17: *Gate* que controla quina informació esborrar de la cel·la $C^{(t)}$

Forget gate

El primer pas de l'arquitectura LSTM és decidir quina informació volem eliminar de la cel·la $C^{(t-1)}$. La primera porta que interactua amb el sistema és la *forget gate*. En la figura 5.17 s'observen de forma ressaltada els elements que interaccionen amb aquesta porta. La *forget gate*, calculada en l'equació 5.14, examina els valors $h^{(t-1)}$ i $x^{(t)}$ i respon amb un valor d'entre 0 i 1 per cada valor contingut en la cel·la $C^{(t-1)}$. En multiplicar el valor $f^{(t)}$ per la cel·la $C^{(t-1)}$ s'obté l'eliminació de la cel·la en funció de l'estat anterior $h^{(t-1)}$ i de l'entrada actual $x^{(t)}$.

$$f^{(t)} = \sigma(W_f \begin{bmatrix} h^{(t-1)} \\ x^{(t)} \end{bmatrix} + b_f) \tag{5.14}$$

On s'aplica una transformació afí sobre l'entrada $x^{(t)} \in \mathbb{R}^{n_h}$ i l'estat ocult $h^{(t-1)} \in \mathbb{R}^{n_h}$ utilitzant la matriu de paràmetres $W_f \in \mathbb{R}^{n_h, n_h*2}$. Finalment se suma el biaix b_f al resultat de la transformació afí i s'aplica la funció sigmoide, donant com a resultat $f^{(t)} \in [0,1]^{(n_h)}$.

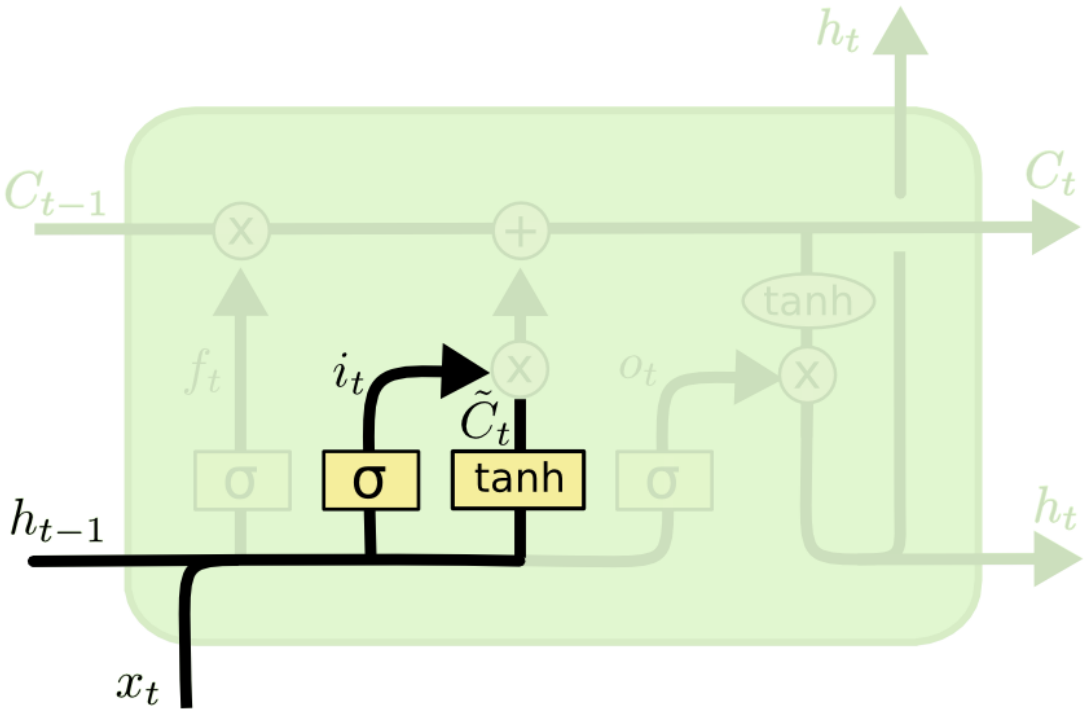

Figura 5.18: Porta que controla la informació agregada a la cel·la $C(t)$

Input gate

El següent pas (vegeu la figura 5.18) és decidir quina informació s'actualitzarà a la cel·la $C^(t)$. Aquest pas es pot dividir en dues parts. En primer lloc, es disposa d'una porta anomenada *input gate*, definida en l'equació 5.15, que decideix quina informació es gravarà en la cel·la $C^(t)$.

$$i^{(t)} = \sigma(W_i \begin{bmatrix} h^{(t-1)} \\ x^{(t)} \end{bmatrix} + b_i) \tag{5.15}$$

On s'aplica una transformació afí sobre l'entrada $x^{(t)} \in \mathbb{R}^{n_h}$ i l'estat ocult $h^{(t-1)} \in \mathbb{R}^{n_h}$, utilitzant la matriu de paràmetres $W_i \in \mathbb{R}^{n_h, n_h*2}$. Finalment se suma el biaix b_i al resultat de la transformació afí i s'aplica la funció sigmoide, que dóna com a resultat $i^{(t)} \in [0,1]^{(n_h)}$. Un cop ja s'ha decidit quanta informació s'actualitzarà amb la *input gate*, cal decidir quina informació s'actualitzarà. L'equació 5.16 defineix quina informació s'actualitzarà.

$$\widetilde{C}^{(t)} = tanh(W_c \begin{bmatrix} h^{(t-1)} \\ x^{(t)} \end{bmatrix} + b_c) \tag{5.16}$$

On s'aplica una transformació afí sobre l'entrada $x^{(t)} \in \mathbb{R}^{n_h}$ i l'estat ocult $h^{(t-1)} \in \mathbb{R}^{n_h}$ utilitzant la matriu de paràmetres $W_c \in \mathbb{R}^{n_h, n_h*2}$. Finalment se suma el biaix b_c al resultat de la transformació afí i s'aplica la funció d'activació tanh o ReLU, obtenint com a resultat la informació que cal actualitzar, $\widetilde{C} \in \mathbb{R}^{n_h}$.

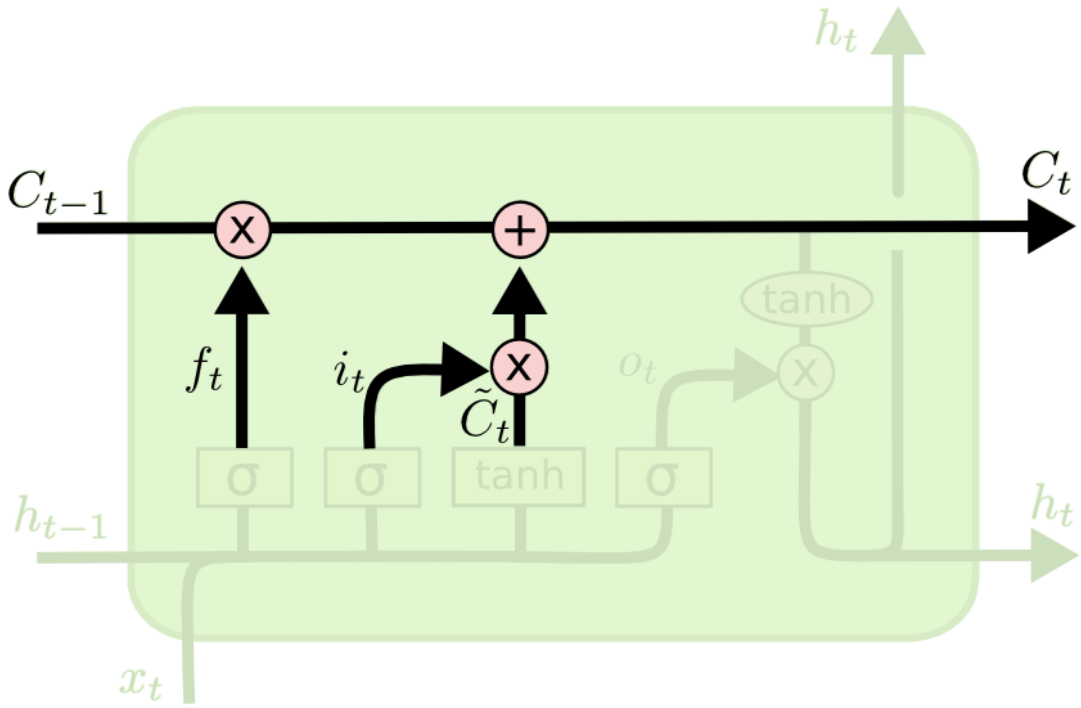

Figura 5.19: Cel·la C introduïda en l'arquitectura LSTM

Actualitzar la cel·la C

El següent pas és actualitzar l'estat $C^{(t-1)}$ per tal d'obtenir l'estat $C^{(t)}$. Primer s'esborra la informació de la cel·la $C^{(t-1)}$ amb l'expressió $f^{(t)} * C^{(t-1)}$ i després s'actualitza la cel·la $C^{(t)}$ amb l'expressió $i^{(t)} * \widetilde{C}^{(t)}$.

$$C^{(t)} = f^{(t)} * C^{(t-1)} + i^{(t)} * \widetilde{C}^{(t)} \tag{5.17}$$

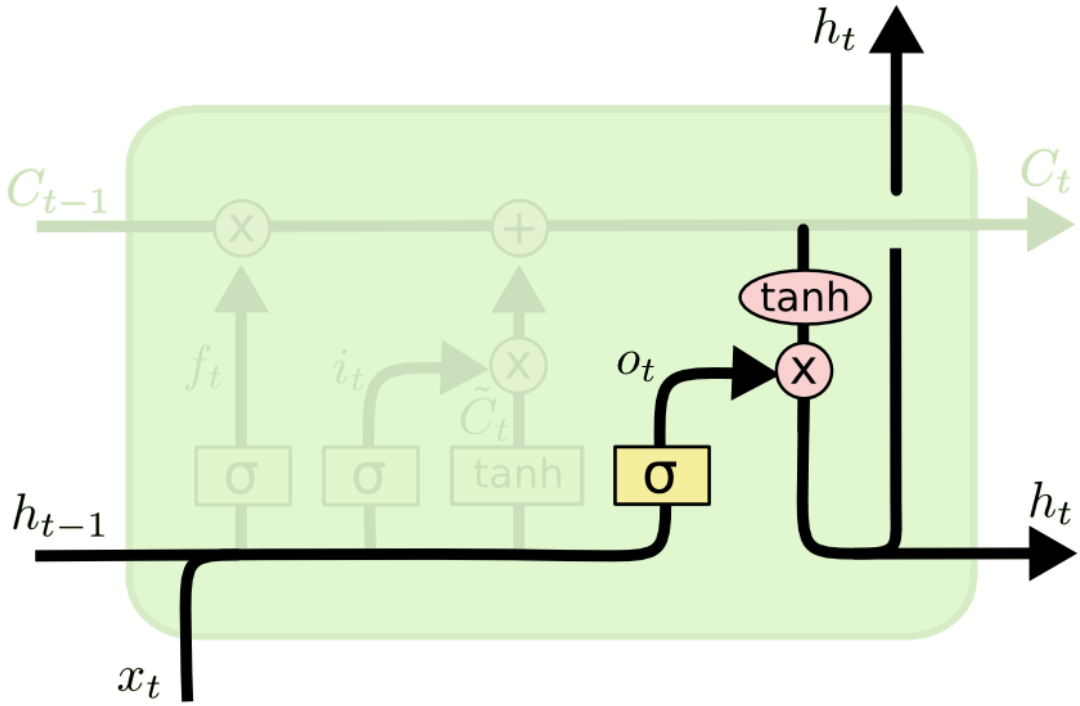

Figura 5.20: Porta que actualitza l'estat ocult $h^{(t)}$

Actualitzar l'estat ocult $h^{(t)}$

L'últim pas és actualitzar l'estat ocult $h^{(t)}$, il·lustrat en la figura 5.20. L'estat $h^{(t)}$, basat en l'estat de la cel·la $C^{(t)}$, serà una versió filtrada per l'última de les portes, anomenada *output gate*. L'*output gate* regularà quina informació de la cel·la $C(t)$ s'utilitza per actualitzar la capa oculta $h(t)$. L'equació 5.18 defineix l'*output gate*, que decidirà quanta informació de la $C^{(t)}$ s'assignarà al nou estat ocult $h^{(t)}$.

$$o^{(t)} = \sigma(W_o \begin{bmatrix} h^{(t-1)} \\ x^{(t)} \end{bmatrix} + b_o) \tag{5.18}$$

On s'aplica una transformació afí sobre l'entrada $x^{(t)} \in \mathbb{R}^{n_h}$ i l'estat ocult $h^{(t-1)} \in \mathbb{R}^{n_h}$ utilitzant la matriu de paràmetres $W_o \in \mathbb{R}^{n_h, n_h*2}$. Finalment se suma el biaix b_o al resultat de la transformació afí i s'aplica la funció sigmoide donant com a resultat $o^{(t)} \in [0,1]^{(n_h)}$. Finalment, l'equació 5.19 defineix el nou estat ocult $h^{(t)}$.

$$h_{(t)} = o_{(t)} * tanh(C^t) \tag{5.19}$$

Actualment, les xarxes LSTM són l'estat de l'art en xarxes neuronals recurrents. Tot el que s'ha vist en aquesta secció és aplicable utilitzant l'arquitectura LSTM.

Per exemple, podem crear xarxes LSTM multicapa o xarxes bidireccionals LSTM. Avui dia s'està investigant per aconseguir un model que mantingui certa estabilitat pel que fa als problemes esmentats del *vanishing gradient*, però alhora intentar reduir la complexitat introduïda en la xarxa LSTM.

6 Conclusions

El *deep learning* és una eina eficaç per tasques on s'utilitzen dades d'una complexitat elevada, com podria ser el cas de veu, text o imatge. Aquestes dades fan que sigui necessària la utilització de xarxes neuronals multicapa o arquitectures especialitzades com les xarxes convolucionals. Crec que ha estat un treball complicat de realitzar. Ja que el *deep learning* és una àrea de recerca molt nova i és complicat trobar informació. Molta de la informació és de difícil interpretació a causa de la complexitat del tema. En la majoria dels temes he hagut de contrastar moltes fonts abans d'entendre cada un dels conceptes i després ser capaç de transmetre el concepte. Crec que la redacció del treball m'ha ajudat a entendre la dificultat de transmetre conceptes de complexitat elevada. Aquest treball no només m'ha servit per descobrir el *deep learning* sinó per profunditzar en l'àrea de l'aprenentatge automàtic. Sóc molt conscient que aquest treball probablement és diferent de la majoria de treballs presentats, possiblement té un enfocament més teòric. Crec que la vida laboral tendeix a ser massa pràctica i que en l'etapa universitària hem d'intentar adquirir coneixements que segurament seran més difícils de trobar en el món laboral.

Bibliografia

[1] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012.

[2] Ian Goodfellow Yoshua Bengio and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[3] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[4] Paulo Cortez and Alice Silva. Using data mining to predict secondary school student performance. *Proceedings of 5th FUture BUsiness TEChnology Conference (FUBUTEC 2008) pp. 5-12*, 2008.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] Kaibo Duan, S. Sathiya Keerthi, Wei Chu, Shirish Krishnaj Shevade, and Aun Neow Poo. *Multiple Classifier Systems: 4th International Workshop, MCS 2003 Guildford, UK, June 11–13, 2003 Proceedings*, chapter Multi-category Classification by Soft-Max Combination of Binary Classifiers, pages 125–134. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[7] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[8] Andrej Karpathy. Stanford University CS231n: Convolutional Neural Networks for Visual Recognition.

[9] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`, 2015.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[11] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[12] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[13] Andrew Ng. Stanford cs229 - machine learning. https://see.stanford.edu/Course/CS229, 2008.

[14] Christopher Olah. Neural networks, manifolds, and topology. `http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/`, 2016.

[15] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[16] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[18] Richard Socher, James Hong, Sameep Bagadia, David Dindi, B. Ramsundar, N. Arivazhagan, and Qiaojing Yan. Cs224d: Deep learning for natural language processing.

[19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[20] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualization large-scale and high-dimensional data. *CoRR*, abs/1602.00370, 2016.