Treball final de grau

GRAU DE MATEMÀTIQUES

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

XARXES NEURONALS RECURRENTS PER A SÈRIES TEMPORALS

Autor: Núria Casals Lladó

Director: Dr. Josep Fortiana Gregori
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 27 de juny de 2018

Abstract

Recurrent neural networks (RNNs) have been widely used for processing sequential data and are capable of learning long-term dependencies. This project proceeds from its inception, studying the behaviour of the simplest Deep Learning structures, to learning issues associated with time series data analysis to finally achieve more complex architectures: Long Short-Term Memory and Gated Recurrent Units. A model with a Gated Recurrent Unit has been implemented to forecast time series data associated with electricity consumption.

Resum

Les xarxes neuronals recurrents (RNNs) han estat àmpliament utilitzades per processar dades seqüencials i són capaces d'aprendre llargues dependències de temps. Aquest projecte procedeix des del principi amb l'estudi del comportament de les estructures més senzilles de Deep Learning. S'analitzen les problemàtiques associades a l'hora de tractar sèries temporals, per arribar finalment a estructures més complexes: les unitats Long Short-Term Memory i Gated Recurrent Unit. S'ha implementat un model amb una Gated Recurrent Unit per fer predicció de dades temporals associades al consum d'electricitat.

2010 Mathematics Subject Classification. 65Y10, 68T01, 90C35

Agraïments

Vull agrair al meu tutor Josep Fortiana per embarcar-se en aquest camp i per aconsellar-me quan ho he necessitat. A la meva família, pel seu suport incondicional. A les amistats que he fet a la universitat, per haver compartit aquesta etapa tan emocionant. A Holaluz, per donar-me la oportunitat de créixer dins d'una empresa que vol canviar el món. Als meus companys de feina, per fer de treballar una aventura.

Índex

1 Introducció

El context

Les xarxes neuronals van començar com un model del funcionament del cervell representat com a circuits connectats per simular un comportament intel·ligent. Va ser al 1943, amb un simple circuit elèctric del neurofisiòleg Warren McCulloch i el matemàtic Walter Pitts. A la dècada de 1950, els investigadors van començar a traduir aquestes xarxes a sistemes computacionals, i la primera xarxa es va implementar amb èxit al MIT el 1954.

El 1958, Frank Rosenblatt, en un intent d'entendre i quantificar un sistema de decisions en ulls de mosques, va proposar la idea d'un Perceptró. La bellesa d'aquest Perceptró radica en el fet que els seus pesos eren "apresos"de manera que es minimitzava la diferència entre la predicció desitjada i la real.

El 1969, Minsky i Papert van publicar el llibre "Perceptrons". El llibre va argumentar de manera concloent que l'aproximació de la percepció única de Rosenblatt a les xarxes neuronals no es pot traduir eficaçment en xarxes neuronals de múltiples capes. L'efecte d'aquest text va ser el principi d'una etapa de 10-12 anys sense publicacions al respecte, anomenada l'hivern de la intel·ligència artificial.

No obstant això, durant la dècada dels 80, un gran descobriment d'un concepte ja existent des dels anys 60 que va ajudar a sortir les xarxes neuronals d'aquesta etapa: la propagació cap enrere. Aquesta, juntament amb el mètode del descens del gradient, formen la columna vertebral de les xarxes neuronals. Mentres el descens seguint el gradient actualitza constantment i mou els pesos cap al mínim de la funció de cost, la propagació cap enrere avalua el gradient del cost respecte els pesos, la llargada i la direcció dels quals s'utilitza pel descens del gradient per corregir (aprendre) els pesos.

Avui en dia, les xarxes neuronals estan a l'ordre del dia de la intel·ligència artificial. La seva promesa sembla molt brillant ja que la pròpia naturalesa és la prova que funcionen. Tot i així, el seu futur va estretament lligat amb els avenços tecnològics, sobretot pel que fa a les potències de les màquines.

El projecte

Vaig tenir la idea de fer un treball sobre xarxes neuronals després de cursar les assignatures optatives de la menció d'estadística a la Universitat Politècnica de Catalunya. En aquestes vaig conèixer camps emergents relacionats amb el tractament i la modelització de grans mostres de dades. Més concretament, moltes tècniques de modelització mitjançant algorismes d'aprenentatge que em van resultar molt atractives. Les xarxes neuronals però, em van semblar un camp molt ampli i desconegut, que a la vegada veia que se'n podien extreure bons resultats. Aquest fet doncs, em va motivar a seguir aprenent en aquesta direcció des d'un punt de vista formal, que és el que m'ha ensenyat cursar el grau de matemàtiques.

La intenció d'aquesta memòria és entendre des d'un punt de vista matemàtic el funcionament de les xarxes neuronals recurrents. Això implica comprendre la seva estructura, els problemes més freqüents i les seves limitacions.

En línia amb aquest objectiu, he aprofitat l'experiència a una empresa, *Holaluz*, que m'ha donat la llibertat, les eines i les dades necessàries per poder experimentar amb els models sobre els quals prèviament havia teoritzat. En concret, les dades disponibles són de caire temporal i per això, tot i que tenen nombroses aplicacions, s'ha acotat l'estudi de les xarxes neuronals a casos de regressió per a sèries temporals.

La part experimental ha sigut una part important per completar l'aprenentatge i ser conscient de molts dels aspectes estudiats, que sota el meu punt de vista no podia faltar en un treball sobre xarxes neuronals. Tot i així, l'enfoc principal que li he volgut donar en aquest treball és l'estudi dins el marc teòric d'aquest tipus d'algorismes.

Estructura de la Memòria

La memòria comença amb un capítol d'introducció als algorismes d'aprenentatge, que és el marc on s'engloben les xarxes neuronals. D'aquesta manera es comparteixen conceptes transversals en aquesta disciplina que marquen el punt de partida del treball.

La següent secció introdueix l'arquitectura del tipus més bàsic de xarxes neuronals, que són les xarxes neuronals orientades cap endavant. Aquestes xarxes són les més conegudes i més senzilles a nivell de concepte i ens serveixen per posar les bases per estructures més complexes que s'explicaran a posteriori.

Seguint amb aquest tipus de xarxes neuronals, en la secció de Propagació cap endavant s'explica la primera fase de l'algorisme. També es presenta un teorema fonamental per a aquest punt.

A continuació, a la secció d'Aprenentatge s'explica la segona fase de l'algorisme, que és la propagació cap enrere. Aquest és el punt on es produeix l'aprenentatge dels paràmetres del model, i per tant cal introduir conceptes com la funció d'error (o funció de cost) i el mètode d'optimització d'aquesta, el descens seguint el gradient.

L'últim bloc teòric de la memòria va dedicat a les xarxes neuronals recurrents. Així doncs, es presenta la seva arquitectura com una variant de la presentada anteriorment i també el pas de la propagació cap enrere en aquest tipus de xarxes neuronals. El bloc segueix amb un apartat dedicat a problemes relacionats amb l'estructura d'aquestes xarxes i presenta unes versions 'tancades' d'aquestes, amb estructures més complicades que neixen amb la finalitat d'esquivar els problemes mencionats.

Finalment, es presenta el cas d'estudi pràctic. Es presenten les dades estudiades, el tractament d'aquestes, el model utilitzat en la implementació i finalment els resultats obtinguts. Aquests resultats es comparen amb un model base per poder tenir una referència de cara a la seva interpretació.

2 Algorismes d'aprenentatge

Les xarxes neuronals són un cas dels coneguts Algorismes d'aprenentatge (*Machine Learning*). Un algorisme d'aprenentatge és un algorisme capaç d'aprendre de les dades. Però el concepte d'aprenentatge és molt ampli. Una definició molt encertada és la que donen Ian Goodfellow, Yoshua Bengio i Aaron Courville [5]:

"Es diu que un programa aprèn de l'experiència E respecte una classe de tasques T i una mesura de la bondat de l'ajust P si la mesura de les tasques T mesurades per P, milloren amb l'experiència E."

Per a aquesta secció, m'he basat en els capítols introductoris [16] i [5], capítols 5.1 i 5.2.

2.1 Tasca, bondat d'ajust i experiència

Tasca (T)

Les tasques de *machine learning* són algorismes que processen exemples (també anomenats observacions). Un exemple és una col·lecció de variables mesurades quantitativament sobre algun objecte o esdeveniment que volem que l'algorisme processi. Normalment es representen els exemples com a vectors $x \in \mathbb{R}^n$ on cada component x_i és el valor d'una variable en l'exemple.

La classificació i la regressió són els dos tipus de tasques més utilitzades en els algorismes d'aprenentatge. En el cas d'aquest treball, la tasca es tractarà d'una regressió. La regressió és un tipus de tasca on l'algorisme espera predir un valor numèric donat un input. Per resoldre aquesta tasca, l'algorisme té com a sortida una funció $f : \mathbb{R}^m \longrightarrow \mathbb{R}$.

Mesura de la bondat (P)

De cara a avaluar les habilitats d'un algorisme d'aprenentatge es necessita dissenyar una mesura quantitativa del seu comportament.

Habitualment, és interessant veure com de bé o malament l'algorisme d'aprenentatge es comporta amb dades que no ha vist abans. Això determina com de bé funcionarà en futurs casos. És per això que mitjançant una funció de cost s'avalua la bondat de l'ajust en un conjunt d'exemples diferents dels utilitzats per entrenar l'algorisme. En la secció **5.1.1 Cost dels errors** s'aprofundeix en aquestes mesures.

Experiència (E)

L'experiència és un concepte general no formal molt important en els algorismes d'aprenentatge.

Com s'ha comentat anteriorment, els algorismes d'aprenentatge processen una col·lecció d'exemples. Aquests exemples contenen variables que es componen en variables explicatives i una variable resposta.

Un exemple molt utilitzat per la investigació és la col·lecció de dades *Iris dataset* (Fisher, 1936). És una col·lecció de mesures de diferents parts d'un total de 150 plantes. Cada planta observada és un exemple de la col·lecció. Les **variables explicatives** d'aquestes dades són la llargada del sèpal, la profunditat del sèpal, la llargada del pètal i la profunditat del pètal. La **variable resposta** és l'espècie a la qual pertany cada planta.

Podríem explicar l'experiència de l'algorisme com l'aprenentatge que porta a terme quan, observant les variables explicatives, les dades "ensenyen"a l'algorisme a quina variable resposta estan associades. És a dir, la variable resposta guia a l'algorisme de què fer.

En el cas concret de les Xarxes Neuronals, a la secció **5 Aprenentatge** s'explica com l'algorisme aprèn de les dades a través de l'experiència.

2.2 Capacitat, overfitting i underfitting

L'objectiu principal en l'entrenament d'algorismes d'aprenentatge és poder predir dades que no s'han vist (o processat) anteriorment. La capacitat de donar bons resultats amb exemples no observats es diu **generalització**.

Habitualment, quan s'entrena un model d'autoaprenentatge, es té accés a un conjunt de dades i es calcula l'error del model respecte aquestes dades, anomenat **error d'entrenament**, de manera que es vol reduir aquest error.

De moment, sembla que fins aquí estem tractant un problema d'optimització. El que separa els algorismes d'aprenentatge de l'optimització és que l'objectiu final és minimitzar **l'error de generalització**. Aquest es defineix com el valor esperat de l'error del model sobre dades que no ha processat durant l'entrenament. Típicament s'estima l'error de generalització del model mesurant el seu error en un conjunt d'exemples independents dels utilitzats per l'entrenament, les **dades de validació**.

Quan s'entrena un model d'aprenentatge, no es fixen els paràmetres. Segons les **dades d'entrenament**, s'escullen els paràmetres que redueixen l'error d'entrenament, després s'utilitzen aquests paràmetres amb les dades de validació. Lògicament, dins d'aquest procés, s'espera que l'error de generalització (sobre les dades de validació) sigui més gran o igual que l'error d'entrenament.

Hi ha dos factors que determinen com de "bo"és l'algorisme d'aprenentatge:

1. Fa l'error d'entrenament petit.
2. Fa que la diferència entre l'error d'entrenament i l'error de generalització sigui petita.

Aquests dos factors corresponen a dos reptes centrals del machine learning: l'*underfitting* i l'*overfitting*.

Per una banda, l'underfitting passa quan el model no és capaç de obtenir un error d'entrenament suficientment baix. Per altra banda, l'overfitting passa quan la diferència de l'error d'entrenament i el de generalització és molt gran, cosa que

significa que el model ha ajustat tant bé les dades d'entrenament que al veure un conjunt d'exemples diferents no s'ajusta bé.

Es pot controlar la probabilitat d'overfit i d'underfit alterant la **capacitat** del model. Informalment, la capacitat d'un model és l'habilitat d'ajustar-se a una gran varietat de funcions. Els models amb capacitat baixa no tenen un bon error d'entrenament i s'associen a models amb underfitting. Els models amb capacitat alta poden tenir un error d'entrenament tant bo que els paràmetres obtinguts no serveixen per observacions fora de les dades d'entrenament, cosa que porta a l'overfitting.

3 Arquitectura

En aquesta secció es presenta l'estructura de les xarxes neuronals orientades cap endavant (també anomenades xarxes neuronals *feedforward* o *Multi Layer Perceptron*). Aquestes serveixen com a base per, posteriorment, poder presentar les Xarxes Neuronals Recurrents.

3.1 Estructura

Sigui X una col·lecció d'exemples amb variables explicatives i una variable resposta, una xarxa neuronal orientada cap endavant defineix una funció $y = f(x;\theta)$ que aprèn el valor del paràmetre θ, que resulta de la millor aproximació de la funció que defineixen les dades a modelar.

Durant l'entrenament de la xarxa neuronal, l'algorisme vol aproximar una funció $f(x)$ a la funció que defineixen les dades $f^*(x)$. En les xarxes neuronals feedforward no hi ha bucles, la informació sempre flueix cap endavant i mai cap enrere. En contraposició, hi ha altres estructures com les xarxes neuronals recurrents que permeten bucles en la seva estructura, d'aquestes en parlarem més endavant.

El capítol 6 de [5] suggereix que el model es pot representar com un graf acíclic que descriu les interaccions que pateixen els exemples d'entrada fins arribar a la variable resposta. La imatge següent presenta un diagrama d'una xarxa neuronal orientada cap endavant.

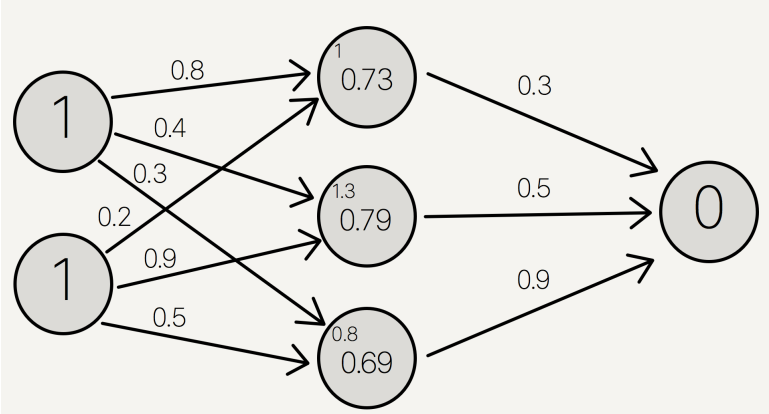


Figura 1: Detall dels valors que pren una xarxa neuronal orientada cap endavant amb una capa oculta.

Anomenem X al conjunt de dades organitzades en variables explicatives i una variable resposta. A l'esquerra de la figura 1 es té l'anomenada **capa d'entrada**. Està formada per diferents unitats, cada unitat representa a una variable explicativa d'X.

A la dreta del tot es té la **capa de sortida**, que representa la variable resposta d'X. La capa de sortida té un nombre específic d'unitats que s'explicarà a continuació.

En aquesta capa, els valors resultants són els valors estimats de la variable resposta que l'algorisme assigna a cada exemple processat.

Les capes que no són d'entrada o de sortida s'anomenen **capes ocultes**. S'anomenen així ja que tot i que l'entrada i la sortida corresponen a esdeveniments visibles a la realitat emmagatzemats en dades, els valors de les capes ocultes no són esdeveniments que es poden observar directament.

Cada unitat que forma la capa oculta representa una agregació d'informació donada per les dades d'entrada (o de la capa oculta anterior si és que n'hi ha més d'una). Aquesta agregació és el resultat d'aplicar una **funció d'activació** no lineal a la combinació lineal dels valors resultants de la capa prèvia amb un conjunt de pesos associats. En el transcurs de la memòria s'explicaran amb més profunditat els detalls. Cada unitat permet al model capturar interaccions entre variables. És a dir, com més unitats hi hagi, més interaccions es podran capturar.

El nombre de capes de la xarxa neuronal defineix la **profunditat** del model. És d'aquest concepte que neix el terme *Deep Learning*.

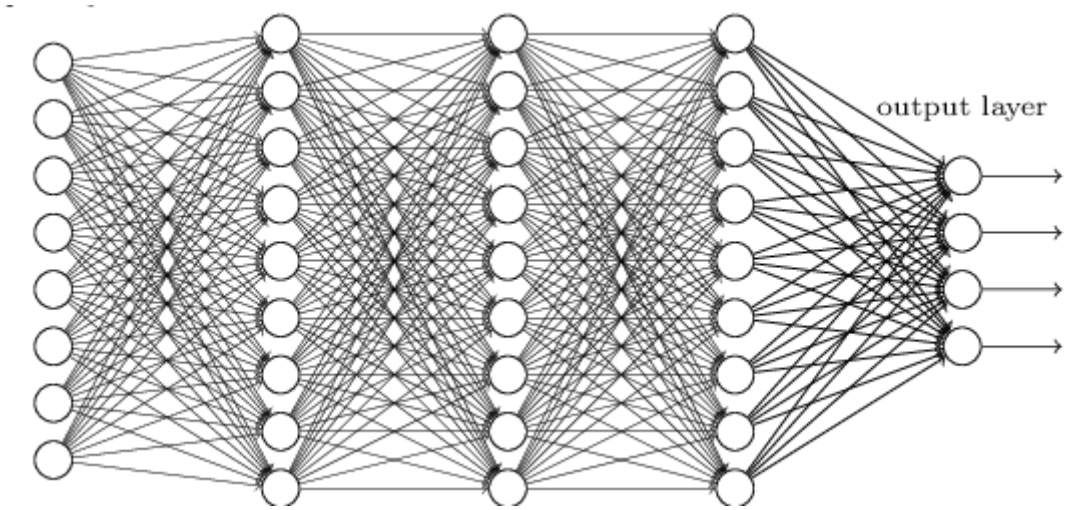


Figura 2: Xarxa neuronal orientada cap endavant amb tres capes ocultes. Imatge extreta de [14].

La figura 2 intenta il·lustrar que una capa actua com un conjunt d'unitats que actuen en paral·lel, cadascuna, representant una funció $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. Cada unitat s'anomena **neurona**, ja que el comportament de rebre un input de diferents unitats i computar un valor d'activació pot recordar a l'activitat neuronal del cervell.

Per acabar, la dimensió de les capes ocultes, és a dir, el nombre de neurones que té cada capa oculta defineix l'**amplada** del model.

Capa de sortida

Cada xarxa neuronal té exactament una capa de sortida. La mida d'aquesta capa, és a dir, el nombre de neurones que la formen, està completament determinada per la configuració de les dades que es volen modelar.

Si la xarxa neuronal modela unes dades que formen part d'un problema de regressió (variable resposta contínua), aleshores la capa de sortida té una única neurona que, per a cada exemple, l'activació d'aquesta és el valor associat de la variable resposta estimada per l'algorisme.

Si la xarxa neuronal entrena dades que formen part d'un problema de classificació amb k classes en la variable resposta, aleshores la capa de sortida està formada per k neurones, una per a cada categoria de la variable resposta. En aquest cas, els valors resultants per a cada unitat són les probabilitats de que l'exemple processat pertanyi a cada classe de la variable resposta. Això és degut a la utilització de la funció *softmax*, molt utilitzada en problemes de classificació.

3.2 Funcions d'activació

Una funció d'activació és una funció utilitzada per transformar la combinació lineal de pesos i inputs d'una neurona en un senyal de sortida. Les funcions d'activació s'utilitzen per introduir no-linearitat al model, és a dir, per assegurar que l'espai de sortida de la xarxa neuronal és diferent del d'entrada.

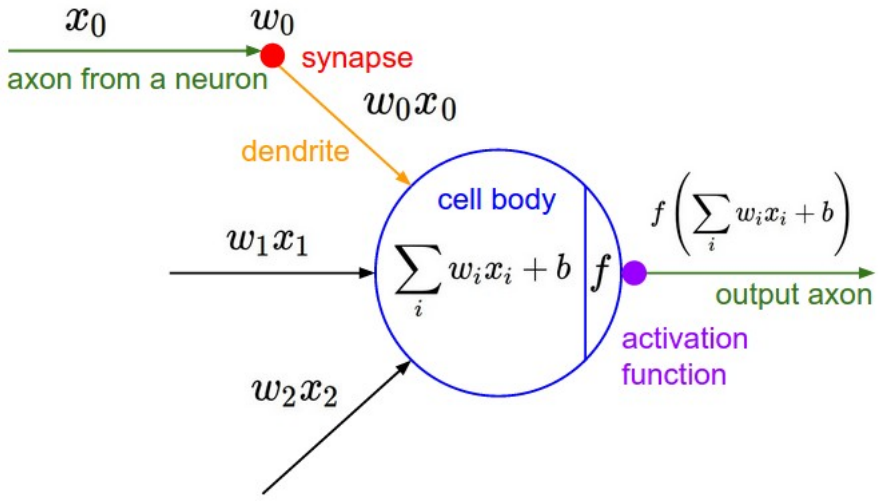


Figura 3: Esquema de les funcions d'activació dins les xarxes neuronals orientades cap endavant.

Les imatges i els continguts d'aquesta secció estan basats en [7].

Hi ha molts tipus de funcions d'activació proposades al llarg dels últims anys (més de 640). A continuació es presenten dues funcions d'activació molt utilitzades en les capes ocultes de xarxes neuronals per a problemes de regressió.

Funció logística

La funció logística ve definida de la següent manera:

$$\sigma : \mathbb{R} \longrightarrow (0, 1),$$
$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

La funció logística pren valors reals i els concentra entre en un rang d'entre 0 i 1, $\sigma(x) \in (0, 1)$. En particular, nombres molt negatius esdevenen quasi 0 i nombres molt positius esdevenen quasi 1. També s'anomena *squashing function* per aquest efecte.

Cal destacar que la funció logística *satura els gradients*, veure secció 6.3. És a dir, les neurones amb funcions d'activació logístiques tenen la propietat que quan es són 0 o 1, el gradient de la funció logística ($\dot{\sigma}(x) = \sigma(x)(1 - \sigma(x))$, veure figura 4) és quasi zero en aquestes regions.

Durant el procés de propagació cap enrere que porten a terme les xarxes neuronals i del qual se'n parlarà més endavant, es calculen els gradients de l'activació de les neurones i s'utilitzen per actualitzar els pesos per tal d'obtenir una solució més acurada. Si el gradient en qüestió és petit, el procés d'aprenentatge pot ser molt lent arribant a "matar"el gradient i no provocar cap millora en els pesos. Per això és important inicialitzar els pesos de les capes amb funcions d'activació logístiques per prevenir aquesta saturació.

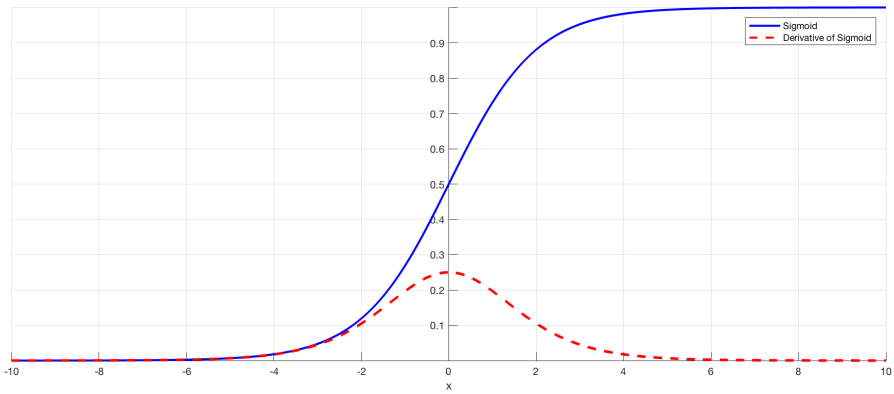


Figura 4: Funció logística i la seva derivada.

Tangent hiperbòlica

La tangent hiperbòlica és una alternativa a la funció logística. Ve definida per la fórmula:

$$\tanh : \mathbb{R} \longrightarrow (-1, 1),$$
$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}.$$

També condensa el conjunt real en un rang d'entre -1 i 1, $\tanh(x) \in (-1, 1)$.

La derivada de la tangent hiperbòlica és $\dot{\tanh}(x) = 1 - \tanh^2(x)$ (veure figura 5). S'interpreta que, com la funció logística, també pot saturar els gradients ja que té una derivada de forma semblant.

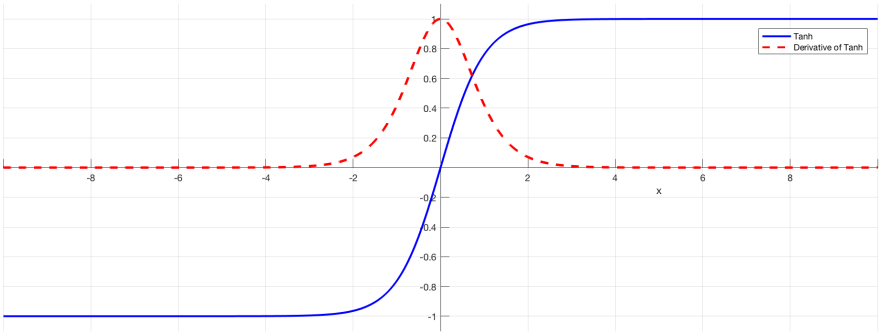


Figura 5: Funció tangent hiperbòlica i la seva derivada.

Es pot veure que $\sigma(x)$ i tanh són dues funcions diferenciables en tot el seu domini, així que el càlcul dels gradients que s'explicarà més endavant no serà problemàtic.

3.3 Notació

Aquesta secció va dedicada a definir una notació per referir-nos als pesos de la xarxa neuronal de manera no ambigua. La notació presentada s'ha extret del capítol 2 de [19].

Definició 3.1. *Sigui una xarxa neuronal amb $L \in \mathbb{N}$ capes i n_l el nombre de neurones a la capa l amb $1 \leq l \leq L$. Aleshores,*

- $w^l_{jk} \in \mathbb{R}$ *és el pes de la connexió de la neurona k de la capa l-1 a la neurona j de la capa l.*
- $b^l_j \in \mathbb{R}$ *és el biaix de la neurona j de la capa l.*
- $a^l_j \in \mathbb{R}$ *és el resultat de l'activació de la neurona j de la capa l.*

On $1 \leq k \leq n_k$ i $1 \leq j \leq n_j$.

Per reescriure aquesta expressió en forma matricial, presentem la següent definició equivalent:

Definició 3.2. *Sigui L el nombre de capes d'una xarxa neuronal i $1 \leq l \leq L$. Aleshores,*

- w^l *és la matriu de pesos que entren en la capa l de mida $n_l \times n_{l-1}$.*
- b^l *és el vector de mida n_l de biaixos de la capa l.*
- a^l *és el vector de mida n_l de resultats de l'activació de la capa l.*

4 Propagació cap endavant

Si σ_l és la funció d'activació de la capa l, utilitzant la notació anterior tenim que el valor d'activació a la capa l és:

$$a_j^l = \sigma_l(\sum_k w_{jk}^l a_k^{l-1} + b_j^l).$$

Per la definició prèvia 3.2, podem reescriure l'expressió anterior com:

$$a^l = \sigma_l(w^l a^{l-1} + b^l) = \sigma_l(z^l).$$

A l'element z^l l'anomenarem *input escalat* pels pesos.

4.1 Aproximació universal

Existeix un teorema d'aproximació universal per a les xarxes neuronals orientades cap endavant. Va ser demostrat per Cybenko el 1989 [4] utilitzant la funció d'activació logística i per Hornik [9] amb qualsevol funció d'activació.

Abans però, cal una definició prèvia.

Definició 4.1. *(Funció squashing) Una funció $\Psi : \mathbb{R} \longrightarrow [a, b]$ amb $a, b \in \mathbb{R}$ es diu que és squashing si:*

- *És no decreixent*
- $\Psi \xrightarrow{+\infty} b$ *i* $\Psi \xrightarrow{-\infty} a$

Les funcions d'activació esmentades en la secció 3.2 , la funció logística i la tangent hiperbòlica, són funcions *squashing*.

Teorema. *(Aproximació Universal) Sigui $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ amb $n, m < \infty$ una funció contínua en un compacte K. Existeix una xarxa neuronal feedforward amb una capa de sortida lineal i amb una capa oculta amb una funció d'activació "squashing" que pot aproximar f amb un error arbitràriament petit $\epsilon > 0$, suposant que la xarxa neuronal té suficients neurones en la capa oculta.*

Suposem donada una funció $f(x)$ que volem aproximar amb error $\epsilon > 0$ fixat. El teorema ens garanteix que, utilitzant suficients neurones en la capa oculta, sempre existeix una xarxa neuronal la qual simuli una funció $f^*(x)$ que satisfà $|f^*(x) - f(x)| < \epsilon$, per totes les entrades x.

El Teorema d'Aproximació Universal significa que sigui quina sigui la funció que s'intenta aprendre, es sap que existeix una xarxa neuronal capaç de representar la funció. Tot i així, no garanteix que l'algorisme d'aprenentatge sigui capaç d'aprendre aquesta funció. No existeix un procediment universal per examinar les dades d'entrenament i computar una funció que es pugui generalitzar en tots els punts.

5 Aprenentatge

5.1 Cost dels errors

5.1.1 Funció de cost

La funció de cost és una part important tant per a les xarxes neuronals com per qualsevol algorisme d'aprenentatge. Aquesta és utilitzada per mesurar la dissimilitud entre els valors predits per l'algorisme $\hat{y}$ i els valors reals y. És una funció no negativa i la robustesa i la qualitat del model millora a mesura que decreix el valor d'aquesta funció.

Per a problemes de regressió, la funció de cost més utilitzada és l'error quadràtic mig o *Mean Squared Error* (MSE).

Definició 5.1. *Sigui $n \in \mathbb{N}$ el nombre total de casos d'entrenament. La funció error quadràtic mig (MSE) és defineix com:*

$$C(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2.$$

Si s'utilitza la notació presentada anteriorment, podem expressar la funció anterior com:

$$C(w, b) = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - (a^L)^{(i)})^2.$$

On L és el nombre total de capes de la xarxa neuronals i $(a^L)^{(i)}$ és el valor de sortida de l'algorisme donat per l'activació de l'última capa. w denota la col·lecció dels pesos de la xarxa, b els biaixos i n és el nombre total de casos d'entrenament.

Definició 5.2. *Sigui C la funció de cost de la xarxa neuronal, es defineix el vector de canvi en l'error associat a la capa l com:*

$$\delta^l = (\delta_1^l, ..., \delta_{n_j}^l) = (\frac{\partial C}{\partial z_1^l}, \ldots, \frac{\partial C}{\partial z_{n_j}^l}).$$

5.1.2 Hipòtesis sobre la funció de cost

Per a la secció 5.3 **Propagació cap enrere** és necessari fer dues assumpcions sobre la funció de cost C. Ens les explica el capítol 2 de [19].

La primera hipòtesi és que la funció de cost es pugui escriure com una mitjana $C = \frac{1}{n} \sum_i C_i$ on i=1,. . . ,n.

En el cas particular de l'error quadràtics mitjà, $C_i = (\hat{y} - y)^2$ és la funció per un cas d'entrenament qualsevol.

La raó per la qual necessitem aquesta hipòtesi és perquè a la propagació cap enrere es computen les derivades parcials $\frac{\partial C_x}{\partial w}$ i $\frac{\partial C_x}{\partial b}$ de cada cas d'entrenament en particular i, finalment, es computa $\frac{\partial C}{\partial w}$ i $\frac{\partial C}{\partial b}$ fent la mitjana de tots els casos.

La segona hipòtesi sobre la funció de cost és que ha de poder ser escrita com una funció de les sortides (*outputs*) de la xarxa neuronal.

La funció d'error quadràtic mig satisfà aquest requisit, notant $C_i = (y - a^L)^2$ com la funció de cost d'un cas d'entrenament particular.

Clarament, aquesta funció també depèn del valor de la variable resposta y. Tot i així, cal remarcar que un cop fixat el cas d'entrenament i, la variable resposta per aquell cas particular també és fixada. Així doncs, no és un paràmetre que es pugui modificar canviant els pesos, o els biaixos de cap manera, és a dir, no és un paràmetre que la xarxa neuronal pugui aprendre.

5.2 Descens seguint el gradient

La optimització consisteix en minimitzar la funció de cost $C(x)$ alterant x.

Sigui $C : \mathbb{R}^n \longrightarrow \mathbb{R}$ una funció de cost dues vegades diferenciable. Anomenem $C^* = min_x C(x)$ i $x^* = argmin_x C(x)$.

Per minimitzar $C(x)$ podríem resoldre $\bigtriangledown C(x^*) = 0$, però analíticament pot arribar a ser complicat, més quan parlem de funcions definides en un espai de dimensió n considerablement gran.

Com a alternativa, el descens seguint el gradient o *gradient descent* és un mètode iteratiu que computa una seqüència de punts $x^{(0)}, x^{(1)}, \ldots$ tals que
$C(x^{(k)}) \longrightarrow C(x^*)$ quan $k \longrightarrow \infty$.

La seqüència de valors es genera repetint:

$$x^{(k)} = x^{(k-1)} - \gamma \bigtriangledown C(x^{(k-1)}), \tag{5.1}$$

on $k = 1, 2, \ldots$, $\gamma \in \mathbb{R}$ és la constant d'aprenentatge i $x^{(0)}$ s'inicialitza de manera aleatòria.

L'algorisme s'atura quan no es produeixen canvis en els punts de la successió respecte una tolerància predeterminada ϵ (i.e. $| \bigtriangledown C(x^{(k+1)}) - \bigtriangledown C(x^{(k)})| < \epsilon$) o bé quan completa un nombre predeterminat d'iteracions k.

Es pot veure que $C(x^{(k)}) \leq C(x^{(k-1)})$ aplicant l'aproximació de primer ordre de Taylor en el punt $C(x^{(k-1)}))$ de la manera següent:

$$\begin{aligned} C(x^{(k)}) =& C(x^{(k-1)} - \gamma \bigtriangledown C(x^{(k-1)})) \approx \\ \approx C(x^{(k-1)}) &- \gamma \bigtriangledown C(x^{(k-1)})^\intercal \bigtriangledown C(x^{(k-1)}) \leq C(x^{(k-1)}). \end{aligned}$$

En la pràctica, si les dades que tenim són massa grans, s'utilitza una variant del descens seguint el gradient anomenada *Stochastic Gradient Descend* (SGD). Aquest

algorisme estima el gradient de la funció de cost avaluant-la només sobre una mostra aleatòria de les dades.

Naturalesa dels mínims

Seguint els punts generats pel *Gradient Descent*, ens garanteix arribar a prop dels valors òptims per minimitzar les funcions. Tot i així, arribar al mínim global o local depèn de si la funció que es vol aproximar és o no convexa.

Definició 5.3. *Sigui* $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ *i* $dom(f)$ *el seu domini, es diu que* f *és una funció convexa si:*

1) $dom(f)$ *és un conjunt convex.*

2) $\forall\ x_1,\ x_2 \in dom(f)$ *i* $\theta \in (0,1)$ *es compleix que* $f(\theta x_1 + (1-\theta)x_2) \leq \theta f(x_1) + (1-\theta)f(x_2)$.

La funció de cost MSE (definició5.1) és clarament convexa respecte $\hat{y}$. Tot i així, no podem controlar $\hat{y}$ directament. Podem controlar els pesos del model, que al seu torn canvien $\hat{y}$.

Sigui $\hat{y} = f(w, b, x)$, on w és el vector de pesos, x és l'exemple d'entrada i f és la funció que defineixen les dades. Aleshores la funció que ens interessa és $g(x, y, w, b) = C(\hat{y}, y) = C(f(w, b, x), y)$.

La funció f és la distribució que segueixen les dades d'entrada, per tant, en general no és convexa i a conseqüència g tampoc.

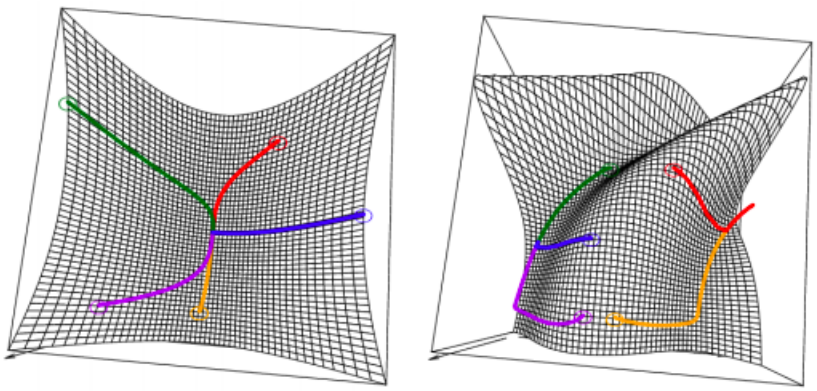


Figura 6: A l'esquerra, exemple d'una funció convexa. A la dreta, exemple d'una funció no convexa. Imatge extreta de [25].

5.3 Propagació cap enrere

5.3.1 Notes introductòries

El concepte de propagació cap enrere va ser originalment introduït als anys 1970, però la seva importància no va ser del tot apreciada fins el 1986 que es va publicar un famós paper de David Rumelhart, Geoffrey Hinton, i Ronald Williams [24].

El paper descriu algunes xarxes neuronals on la propagació cap enrere era computacionalment més ràpida que intents previs. Avui, l'algorisme de propagació cap enrere és el referent de l'aprenentatge de les xarxes neuronals.

La idea de l'algorisme és una expressió de les derivades parcials de la funció de cost respecte qualsevol pes de la xarxa neuronal. L'expressió explica ràpidament com canvia el la funció de cost quan modifiquem els pesos i els biaixos.

L'algorisme de propagació cap enrere que es descriu aquí només és un mètode de diferenciació automàtica inspirat en el capítol 2 de [19]. Altres enfocaments avaluen les subexpressions de la regla de la cadena en diferents ordres. En general, determinar l'ordre d'avaluació dels resultats en el menor cost computacional és un problema difícil.

La propagació cap enrere presentada no és, doncs, l'única manera o la forma òptima de la computació del gradient, però és un mètode molt pràctic que funciona. En el futur, la tecnologia de diferenciació automàtica per a xarxes neuronals profundes millorarà a mesura que els professionals de l'aprenentatge profund s'adonin dels avenços en el camp tant ampli de la diferenciació automàtica.

Tot i que l'expressió sigui en certa manera complicada, també té una bellesa en ella. Permet donar una interpretació natural i detallada de cada paràmetre, i veure com canviant els pesos i els biaixos es modifica el comportament de la xarxa. És per això que en aquest capítol l'estudiarem en detall.

5.3.2 Inicialització dels paràmetres

La corba de Gauss és la funció de densitat de probabilitat de la llei normal.

La funció que la defineix és la següent:

$$f(x) = \frac{1}{\sigma\sqrt[2]{2\pi}} \exp \frac{-(x-\mu)^2}{2\sigma^2}.$$

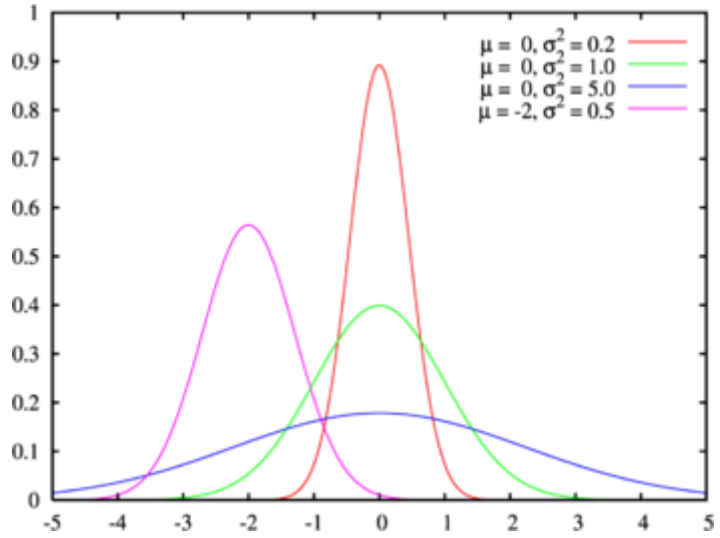


Figura 7: Densitat de probabilitat de la distribució normal amb diferents valors μ i σ.

Segons el capítol 3 de [19], en general, els paràmetres biaix (b) i pesos (W) de la xarxa neuronal s'inicialitzen de manera aleatòria seguint una distribució normal amb mitjana $\mu = 0$ i desviació estàndard $\sigma = 1$. Aquesta inicialització és necessària per tenir un punt de partida de l'algorisme que es presenta a continuació.

5.3.3 Regla de la cadena

Veurem que la propagació cap enrere consisteix en, sistemàticament, aplicar la regla de la cadena del càlcul diferencial. Seguidament es presenta l'enunciat en la forma que l'aplicarem posteriorment.

Siguin $m, n, k \in \mathbb{N}$ i $f : \mathbb{R}^m \longrightarrow \mathbb{R}^k$ i $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ dues aplicacions diferenciables.

Sigui $x \in \mathbb{R}^n$, $u = g(x) = (g_1(x), \ldots, g_m(x)) \in \mathbb{R}^m$ i $y = f(u) = (f_1(u), \ldots, f_k(u)) \in \mathbb{R}^k$.

Aleshores,

$$\frac{\partial y}{\partial x_i} = \sum_{l=1}^{m} \frac{\partial y}{\partial u_l} \frac{\partial u_l}{\partial x_i} = \Delta y \frac{\partial u}{\partial x_i}, \tag{5.2}$$

on $\Delta y = (\frac{\partial y}{\partial u_1}, \ldots, \frac{\partial y}{\partial u_m})$.

5.3.4 Algorisme

A continuació es presenten quatre equacions fonamentals per dur a terme l'aprenentatge de la xarxa neuronal i computar el gradient de la funció de cost.

L'algorisme de propagació cap enrere està basat en operacions comunes (suma de vectors, multiplicació de vectors per matrius, etc). Tot i així, una de les operacions és menys coneguda. Es defineix a continuació.

Definició 5.4. *(Producte de Hadamard $\odot$) Siguin A, B dues matrius de dimensió $m \times n$. Els elements resultants del producte de Hadamard entre A i B són:*

$$(A \odot B)_{i,j} = (A)_{i,j}(B)_{i,j}. \tag{5.3}$$

Proposició 5.5. *Equació de l'error en la capa de sortida.*

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\dot{\sigma}(z_j^L). \tag{5.4}$$

Demostració. Per la definició 5.2 tenim $\delta_j^L = \dfrac{\partial C}{\partial z_j^L}$ sent $j \in 1, ..., n_L$.

Aplicant la regla de la cadena, reexpressem la derivada parcial en termes de derivades parcials.

$$\delta_j^L = \sum_{k=1}^{n_L} \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L}.$$

Clarament, a_k^L depèn de z_j^L quan k=j i $\dfrac{\partial a_k^L}{\partial z_j^L} = 0 \ \forall k \neq j$ per a la definició (3.2).

Així doncs, podem simplificar l'equació anterior per:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}.$$

Com que $a_j^L = \sigma(z_j^L)$ aleshores $\delta_j^L = \dfrac{\partial C}{\partial a_j^L}\dot{\sigma}(z_j^L)$. □

Corol·lari 5.6. *La proposició anterior és equivalent a:*

$$\delta^L = \bigtriangledown_a C \odot \dot{\sigma}(z^L), \tag{5.5}$$

On $\bigtriangledown_a$ és el vector de derivades parcials $\dfrac{\partial C}{\partial a_j^L}$.

Proposició 5.7. *Podem expressar l'error δ^l en termes de l'error de la següent capa δ^{l+1} de la següent manera:*

$$\delta^l = ((w^{l+1})^\intercal \delta^{l+1}) \odot \dot{\sigma}(z^l).$$

Demostració. Volem reescriure $\delta_j^l = \dfrac{\partial C}{\partial z_j^l}$ en termes de $\delta_k^{l+1} = \dfrac{\partial C}{\partial z_k^{l+1}}$. Ho podem fer utilitzant la regla de la cadena:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l}. \tag{5.6}$$

Notem que $z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$. Derivant obtenim:

$$\frac{\partial z_k^{l+1}}{\partial z_j^{l+1}} = w_{kj}^{l+1} \dot{\sigma}(z_j^l).$$

Substituint-ho a l'expressió obtenim:

$$\delta_j^l = \sum_j \delta_k^{l+1} w_k^{l+1} j \dot{\sigma}(z_j^l).$$

□

Combinant les proposicions 5.6 i 5.7 podem calcular l'error δ^l de qualsevol capa l=1,...,L de la xarxa. Es comença utilitzant 5.6 per computar δ^L, on L és l'índex de la capa de sortida, seguidament s'aplica 5.7 per computar δ^{L-1}. Altra vegada, s'utilitza 5.7 per calcular δ^{L-2} i així successivament propagant-se cap enrere de la xarxa.

Proposició 5.8. *L'equació del canvi de la funció de cost respecte qualsevol biaix de la xarxa és:*

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

Demostració. Per la definició 3.3 de l'error associat a la capa l d'una xarxa neuronal tenim,

$$\delta_j^l = \frac{\partial C}{z_j^l} = \frac{\partial C}{\partial b_j^l} \frac{\partial b_j^l}{\partial z_j^l}.$$

La definició 3.2 ens diu, en altres paraules, que $z_j^l = \sum_k w_{kj}^l a_k^{l-1} + b_j^l$ i per tant $b_j^l = z_j^l - \sum_k w_{kj}^l a_k^{l-1}$. On $a_k^{l-1} = \sigma(z_k^{l-1})$ no depèn de z^l.

Així doncs $\frac{\partial b_j^l}{\partial z_j^l} = 1$ i, tornant a l'expressió anterior, tenim que

$$\delta_j^l = \frac{\partial C}{\partial b_j^l} \frac{\partial b_j^l}{\partial z_j^l} = \frac{\partial C}{\partial b_j^l},$$

com volíem veure. □

Proposició 5.9. *L'equació del canvi de la funció de cost respecte qualsevol pes de la xarxa és:*

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l.$$

Es presenta la propagació cap enrere en forma d'algorisme.

Input: Vector de l'exemple d'entrada x. Es calculen els valors d'activació a^1 de la primera capa.

Propagació cap endavant: Per a cada capa l=2,...,L es calcula $z^l = w^l a^{l-1} + b^l$ i $a^l = \sigma(z^l)$.

Càlcul de l'error: Per a la capa de sortida es calcula $\delta^L = \bigtriangledown_a C \odot \dot{\sigma}(z^l)$ amb la proposició 5.4.

Propagació cap enrere: Per l=L-1, L-2,...,2 es calculen els canvis dels errors δ^l de les capes ocultes amb la proposició 5.7.

Output: Amb els errors calculats en l'anterior pas es computen les equacions gradient de la funció de cost respecte el biaix ($\bigtriangledown_b C$) i els pesos ($\bigtriangledown_w C$) amb les proposicions 5.8 i 5.9 respectivament.

Aplicació del descens seguint el gradient: S'actualitzen els valors dels pesos i del biaix amb l'equació 5.1.

Es repeteix la seqüència fins que s'arribi al nombre d'iteracions desitjades o a una tolerància fixada.

6 Xarxes Neuronals Recurrents

En una xarxa neuronal orientada cap endavant s'assumeix que els exemples d'entrada són independents entre ells però per a moltes tasques això és una mala idea. Si es vol predir la següent paraula d'una frase és millor saber quines paraules venien abans d'aquesta. Així doncs, per poder fer predicció amb dades de caràcter temporal es necessita una estructura una mica més complexa per tractar la tasca.

En aquest apartat del treball s'explica una variació de les xarxes neuronals orientades cap endavant: les xarxes neuronals recurrents (RNN). Les xarxes neuronals recurrents es diuen així perquè tenen en compte una ordenació seqüencial de les dades d'entrada. Podem pensar amb les RNN com una unitat amb memòria que captura la informació que s'ha calculat anteriorment. En teoria, les RNN poden utilitzar informació de sèries arbitràriament grans, però a la pràctica estan limitades a uns quants passos anteriors, com després es veurà.

Altres aplicacions de les xarxes neuronals recurrents són el reconeixement de veu, traducció de textos (*machine translation*) o el reconeixement i caracterització d'imatges.

En aquesta secció coneixerem, primer, les xarxes neuronals recurrents (o *Vanilla Recurrent Neural Networks*) i l'adaptació de l'algorisme de propagació cap enrere en aquestes. També veurem els problemes associats a aquest tipus d'estructures i finalment presentarem unes versions "tancades"que neixen com a solució dels problemes anteriors.

6.1 Arquitectura

Es diu xarxa neuronal recurrent a la xarxa neuronal amb una capa recurrent. S'entèn com a capa recurrent aquella capà que té connexions que retroalimenten l'activació de les seves neurones amb el resultat de l'activació de les mateixes neurones un instant de temps endarrere. Més endavant es veurà una expressió formal.

Una xarxa neuronal recurrent opera amb seqüències que contenen vectors $x^{(t)}$ en el mateix moment de temps t, que va de 1 a τ, amb τ fixat. Així doncs, les dades d'entrada completes per a la xarxa neuronal venen donades per una matriu X de dimensió $m \times \tau$, on τ és la llargada de la sèrie de temps i m és el nombre de variables que conformen les dades d'entrada.

Per exemple, ens podem disposar a tractar un conjunt de dades on la variable resposta y és el nombre d'habitants en un cert país per any. Si tenim 10 anys de dades, aleshores $\tau = 10$. També tenim variables explicatives que ajuden a la predicció de la variable resposta per l'any següent, com poden ser índexs relacionats amb la natalitat, la mortalitat, la immigració o l'emigració. Si tenim un total de 4 variables explicatives, aleshores $m = 4$.

En termes de notació, definim $X = (x^{(1)}, \ldots, x^{(\tau)})$ la matriu de vectors columna on $x^{(i)} = (x_1^{(i)}, \ldots, x_m^{(i)})^\intercal$.

El diagrama de sota ens mostra una RNN sent *desenrotllada* cap a una xarxa

neuronal completa. Pel concepte de desenrotllar s'entén com escriure la xarxa neuronal per la seqüència completa. Així és una xarxa neuronal recurrent típica:

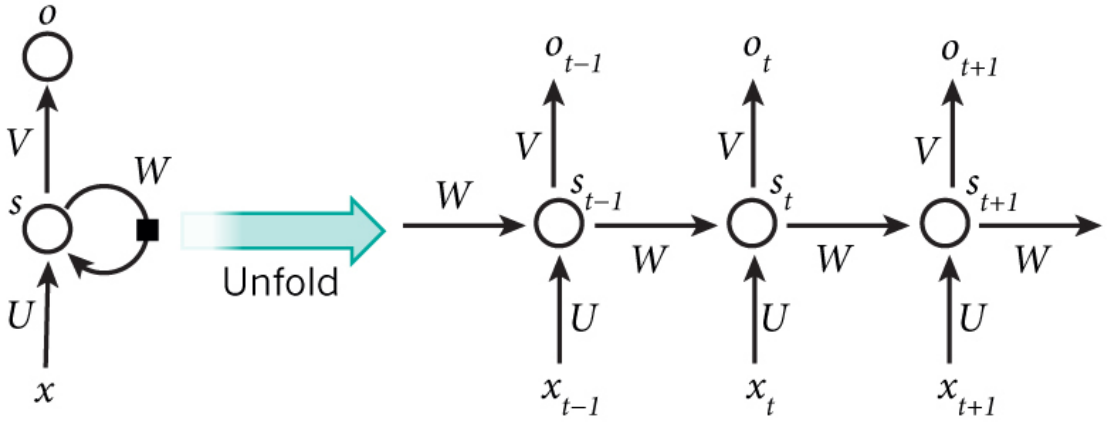


Figura 8: Imatge extreta de [13].

La notació següent està extreta del capítol 10.2 de [5].

Definició 6.1. *Sigui n el nombre de neurones en la capa recurrent, sigui σ la funció d'activació de la Capa recurrent i ρ la funció d'activació de l'última capa. Aleshores es defineix:*

- $a^{(t)} = b + Ws^{(t-1)} + Ux^{(t)}$ *vector real de dimensió n.*
- $s^{(t)} = \sigma(a^{(t)})$ *vector real de dimensió n.*
- $o^{(t)} = c + Vs^{(t)}$ *escalar real.*
- $\hat{y}^{(t)} = \rho(o^{(t)})$ *escalar real.*

La dimensió dels paràmetres depèn de la dimensió de la variable resposta $y^{(t)}$. En el cas d'aquest treball, la regressió, es té $y^{(t)} \in \mathbb{R}$ i per tant les dimensions resultants són les següents:

- b $\in \mathbb{R}^n$ (biaix de la capa oculta).
- c $\in \mathbb{R}$ biaix de la capa de sortida.
- $U \in \mathbb{R}^n \times \mathbb{R}^m$ matrius de pesos.
- $W \in \mathbb{R}^n \times \mathbb{R}^n$ matriu de pesos.
- $V \in \mathbb{R}^n$ vector de pesos.

Cal especificar també, que l'operació $s^{(t)} = \sigma(a^{(t)})$ es porta a terme element a element (*elementwise*).

L'error total d'una seqüència de valors $x^{(t)} \in \mathbb{R}^m$ juntament amb les seves variables respostes $\hat{y}^{(t)}$ és simplement la mitjana de tots els errors durant tots els intervals de temps $C(y, \hat{y}) = \frac{1}{n} \sum_{t=1}^{\tau} C(y^{(t)}, \hat{y}^{(t)})$.

$s^{(t)} \in \mathbb{R}^n$ i se l'anomena l'estat ocult a l'instant de temps t, es calcula basant-se en els estats ocults anteriors del instant de temps actual. La funció d'activació σ utilitzada acostuma a ser una funció no lineal com la tangent hiperbòlica o la funció logística. $s^{(-1)}$ necessita ser calculat pel primer estat ocult i típicament s'inicialitza tot a zeros.

Es pot pensar es l'estat ocult $s^{(t)}$ com la memòria de la xarxa neuronal. $s^{(t)}$ captura la informació del que ha passat en tots els instants de temps anteriors. El resultat $o^{(t)}$ de la capa recurrent es calcula únicament basat en la memòria en el temps t.

Paràmetres compartits

Per passar de xarxes neuronals *feedforward* a xarxes neuronals recurrents, es necessita aprofitar-se d'una una de les primeres idees trobades en l'aprenentatge automàtic i els models estadístics dels anys 1980: compartir els paràmetres al llarg de diferents parts d'un model.

Hem vist que les xarxes neuronals tradicionals (orientades cap endavant o *feedforward*) utilitzen diferents matrius de paràmetres en cada capa. Les xarxes neuronals recurrents comparteixen els mateixos paràmetres (les matrius U, V i W de la figura 8) en tots els instants de temps. Això reflecteix el fet que es computi la mateixa tasca en cada moment, només amb diferents exemples d'entrada. Aquest fet redueix en gran mesura el nombre total de paràmetres requerits per l'aprenentatge.

Si tinguéssim paràmetres separats per a cada valor de l'índex de temps, no podríem generalitzar les longituds de seqüència que no es veuen durant l'entrenament, ni compartir la força estadística a través de diferents longituds de seqüència i en diferents posicions a temps. El fet de compartir és especialment important quan es pot produir una informació específica en múltiples posicions dins de la seqüència.

6.2 Propagació cap enrere a través del temps

L'entrenament d'una xarxa neuronal recurrent és similar a l'entrenament d'una xarxa neuronal orientada cap endavant. També s'utilitza l'algorisme de propagació cap enrere però amb algunes modificacions ja que, com s'ha vist, els paràmetres es comparteixen per tots els instants de temps i el gradient de l'error en cada pas no només depèn dels càlculs i activacions en aquell instant sinó també dels instants anteriors. Per exemple, si es vol calcular el gradient a $t = 4$ es necessita propagar cap enrere 3 passos per sumar els gradients. Aquesta variació de l'algorisme és coneguda com Propagació cap enrere a través del temps o *Backpropagation through time (BPTT)*.

La diferència clau amb la propagació cap enrere utilitzada en xarxes neuronals orientades cap endavant és que es sumen tots els gradients a cada instant de temps. En una xarxa neuronal tradicional no es comparteixen paràmetres entre capes i per tant no es necessita sumar-los.

L'objectiu de l'entrenament és calcular el gradient de la funció de cost per a tots els instants de temps respecte les matrius V, W i U.

Per aquesta secció utilitzarem la notació de [1]. Suposarem que la funció de cost C és la funció esmentada anteriorment, la mitjana dels quadrats dels errors (MSE):

$$C(y, \hat{y}) = \frac{1}{n} \sum_{t=1}^{\tau} (y^{(t)} - \hat{y}^{(t)})^2.$$

Notem que aquesta funció difereix una mica de l'equació 5.1 en la secció 5.1 **Cost dels errors** ja que, en aquest cas, C es calcula per a un cas d'entrenament particular al llarg de tots els instants de temps a diferència del càlcul al llarg de tots els casos d'entrenament.

Per altra banda, s'utilitzarà $\sigma = \tanh$ com a funció d'activació per a les capes ocultes, i $\rho(x) = Ax$ com a funció d'activació lineal per a la capa de sortida, on $A \in \mathbb{R}$.

Per comoditat en càlculs posteriors, es defineix $E^{(t)} = (y^{(t)} - \hat{y}^{(t)})^2$ l'error d'entrenament per un t fixat.

L'objectiu d'aquesta secció és veure com es calculen els gradients d'E respecte els paràmetres que conformen la xarxa neuronal recurrent, per tal de poder aplicar el mètode del descens seguint el gradient per minimitzar la funció de cost.

El vector V

El vector V està present en la funció $\hat{y}^{(t)}$ definida a l'inici de la secció. Aplicant la regla de la cadena, s'obté:

$$\frac{\partial E^{(t)}}{\partial V} = \frac{\partial E^{(t)}}{\partial \hat{y}^{(t)}} \cdot \frac{\partial \hat{y}^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial V}. \tag{6.1}$$

Derivant E respecte l'estimació de $y^{(t)}$ s'obté $\dfrac{\partial E^{(t)}}{\partial \hat{y}^{(t)}} = -2(y^{(t)} - \hat{y}^{(t)})$.

Derivant $y^{(t)}$ respecte $o^{(t)}$ s'obté la derivada de la funció d'activació de la capa de sortida ρ, que és $\rho'(x) = A$.

L'últim factor de l'expressió 6.1 és $s^{(t)}$.

Així doncs, l'expressió anterior es redueix a:

$$\frac{\partial E^{(t)}}{\partial V} = -2(y^{(t)} - \hat{y}^{(t)}) A s^{(t)}.$$

La matriu W

La matriu de paràmetres W apareix com a argument de $s^{(t)}$, aleshores, per calcular el gradient desitjat, s'haurà de passar per $s^{(t)}$ i per $\hat{y}^{(t)}$.

De la mateixa manera que abans, obtenim en notació matricial:

$$\frac{\partial E^{(t)}}{\partial W} = \frac{\partial E^{(t)}}{\partial \hat{y}^{(t)}} \cdot \frac{\partial \hat{y}^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial s^{(t)}} \frac{\partial s^{(t)}}{\partial W}.$$

S'expressarà la derivada anterior d'una forma més detallada però necessària per càlculs posteriors. Es presenta la derivada d'E respecte l'element ij-èssim de la matriu de paràmetres W com:

$$\frac{\partial E^{(t)}}{\partial W_{ij}} = \frac{\partial E^{(t)}}{\partial \hat{y}^{(t_k)}} \cdot \frac{\partial \hat{y}^{(t_k)}}{\partial o^{(t_l)}} \cdot \frac{\partial o^{(t_l)}}{\partial s^{(t_m)}} \cdot \frac{\partial s^{(t_m)}}{\partial W_{ij}}, \tag{6.2}$$

on $i, j, t_k, t_l, t_m \in 1, \ldots, \tau$.

Els tres primers termes de l'expressió 6.2 ja s'han calculat abans, amb la nova notació quedarien de la següent manera:

- $\frac{\partial E^{(t)}}{\partial \hat{y}^{(t_k)}} = -2(y^{(t_k)} - \hat{y}^{(t_k)})$
- $\frac{\partial \hat{y}^{(t_k)}}{\partial o^{(t_l)}} = A_{kl}$
- $\frac{\partial s^{(t_m)}}{\partial W_{ij}} = V_{lm}$

Per calcular l'últim terme de l'expressió, cal fer notar la dependència implícita de $s^{(t)}$ a W a través de $s^{(t-1)}$. Aleshores s'obté:

$$\frac{\partial s^{(t_m)}}{\partial W_{ij}} = \frac{\partial s^{(t_m)}}{\partial W_{ij}} + \frac{\partial s^{(t_m)}}{\partial s^{(t-1_n)}} \cdot \frac{\partial s^{(t-1_n)}}{\partial W_{ij}}.$$

Podem tornar mirar un instant de temps enrere i tenim:

$$\frac{\partial s^{(t_m)}}{\partial W_{ij}} = \frac{\partial s^{(t_m)}}{\partial W_{ij}} + \frac{\partial s^{(t_m)}}{\partial s^{(t-1_n)}} \cdot \frac{\partial s^{(t-1_n)}}{\partial W_{ij}} + \frac{\partial s^{(t_m)}}{\partial s^{(t-1_n)}} \cdot \frac{\partial s^{(t-1_n)}}{\partial s^{(t-2_p)}} \cdot \frac{\partial s^{(t-2_p)}}{\partial W_{ij}}.$$

Aquesta dependència s'ha d'aplicar recurrentment fins arribar a $s^{(-1)}$, l'estat inicial presentat anteriorment que s'inicialitza al començar l'entrenament.

L'últim terme es col·lapsa a $\frac{\partial s^{(t_m)}}{\partial s^{(t-2_n)}} \cdot \frac{\partial s^{(t-2_n)}}{\partial W_{ij}}$ i es pot convertir el primer terme en $\frac{\partial s^{(t_m)}}{\partial s^{(t_n)}} \cdot \frac{\partial s^{(t_n)}}{\partial W_{ij}}$. Aleshores, s'aconsegueix una forma compacta:

$$\frac{\partial s^{(t_m)}}{\partial W_{ij}} = \frac{\partial s^{(t_m)}}{\partial s^{(r_n)}} \cdot \frac{\partial s^{(r_n)}}{\partial W_{ij}},$$

on la suma sobre tots els valors de r més petits que t amb l'índex n. Més clarament, es pot escriure com:

$$\frac{\partial s^{(t_m)}}{\partial W_{ij}} = \sum_{r=0}^{t} \frac{\partial s^{(t_m)}}{\partial s^{(r_n)}} \cdot \frac{\partial s^{(r_n)}}{\partial W_{ij}},$$

combinant tots els camps, obtenim la derivada d'E respecte W:

$$\frac{\partial E^{(t)}}{\partial W_{ij}} = -2(y^{(t_k)} - \hat{y}^{(t_k)}) A_{kl} V_{lm} \sum_{r=0}^{t} \frac{\partial s^{(t_m)}}{\partial s^{(r_n)}} \cdot \frac{\partial s^{(r_n)}}{\partial W_{ij}}.$$

La matriu U

Calcular el gradient de la matriu de paràmetres U és similar a fer-ho per W ja que les dues necessiten derivades seqüencials del vector $s^{(t)}$. Es té:

$$\frac{\partial E^{(t)}}{\partial U_{ij}} = \frac{\partial E^{(t)}}{\partial \hat{y}^{(t_k)}} \cdot \frac{\partial \hat{y}^{(t_k)}}{\partial o^{(t_l)}} \cdot \frac{\partial o^{(t_l)}}{\partial s^{(t_m)}} \cdot \frac{\partial s^{(t_m)}}{\partial U_{ij}}.$$

Es calcula l'últim terme seguint el mateix procediment que a W, es troba que:

$$\frac{\partial s^{(t_m)}}{\partial U_{ij}} = \sum_{r=0}^{t} \frac{\partial s^{(t_m)}}{\partial s^{(r_n)}} \cdot \frac{\partial s^{(r_n)}}{\partial U_{ij}}.$$

Aleshores es té:

$$\frac{\partial E^{(t)}}{\partial U_{ij}} = \left(\hat{y}^{(t_l)} - y^{(t_l)}\right) V_{lm} \sum_{r=0}^{t} \frac{\partial s^{(t_m)}}{\partial s^{(r_n)}} \cdot \frac{\partial s^{(r_n)}}{\partial U_{ij}}.$$

La diferència entre U i W apareix durant la implementació, ja que els valors de $\frac{\partial s^{(r_n)}}{\partial U_{ij}}$ i $\frac{\partial s^{(r_n)}}{\partial W_{ij}}$ són diferents.

6.3 Explosió i anul·lació del gradient

Les xarxes neuronals recurrents propaguen matrius de paràmetres d'un instant de temps al següent. Un dels objectius d'aquesta tipologia de xarxes és ser capaços de propagar informació a través d'instants de temps llargs.

Com s'ha vist a la secció anterior de BPTT, aquesta propagació de resultats en llargs instants de temps es tradueix en llargues sèries de multiplicacions de matrius. Aleshores, els paràmetres poden explotar o anul·lar-se.

S'anomena **explosió del gradient** quan aquest creix cap a infinit abans de que s'acabi el procés de propagació cap enrere a través del temps. A l'hora d'implementar

les xarxes neuronals, és fàcil detectar el problema de l'explosió de gradients ja que aquests es convertiran en NaN (not a number) i el programa pararà de funcionar.

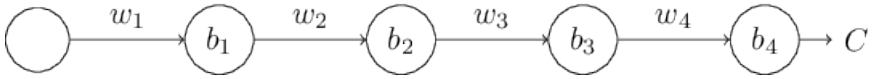
S'anomena **anul·lació del gradient** quan els valors del gradient disminueixen exponencialment ràpid fins a arribar a zero després d'uns instants de temps. Aquest fenomen és difícil de detectar ja que, a diferència de l'explosió del gradient, les seves conseqüències són reduir la qualitat de l'aprenentatge de manera que també s'incrementa el nombre d'iteracions per aconseguir un error desitjat.

Quan es parla de reduir la qualitat de l'aprenentatge es vol dir que quan el gradient de la funció de cost respecte un pes és zero en una iteració, aquest no contribueix en l'aprenentatge ja que les matrius de paràmetres no queden modificades mitjançant l'algorisme del descens del gradient.

L'anul·lació i l'explosió del gradient no són exclusius per a xarxes neuronals recurrents, també tendeixen a passar en xarxes neuronals orientades cap endavant molt profundes. El que passa es que les RNN tendeixen a ser molt profundes i aleshores el problema és més comú.

6.3.1 Causes de l'anul·lació del gradient

Per veure amb més detall en quin punt es produeix l'anul·lació del gradient es posa d'exemple extret del capítol 5 de [19] d'una xarxa neuronal orientada cap endavant amb 3 capes ocultes:



On w_i són els pesos de la xarxa en la capa i, i b_i són els biaixos de la xarxa en la capa i, i considerem σ la funció d'activació logística comentada anteriorment.

Estudiarem el gradient associat a la primera capa. Obtindrem una expressió per $\frac{\partial C}{\partial b_1}$ i mitjançant aquesta expressió podrem entendre les causes del problema de l'anul·lació del gradient.

Imaginem que fem un petit canvi Δb_1 al biaix b_1. Això provoca una sèrie de canvis en cadena a la resta de la xarxa neuronal. Intentem trobar una expressió del gradient $\frac{\partial C}{\partial b_1}$ seguint aquests canvis.

Com que $a_1 = \sigma(z_1) = \sigma(w_1 a_0 + b_1)$ aleshores

$$\Delta a_1 \approx \frac{\partial \sigma(w_1 a_0 + b_1)}{\partial b_1} \Delta b_1 = \dot{\sigma}(z_1) \Delta b_1,$$

on $\dot{\sigma}$ és la derivada de primer ordre de la funció d'activació. Aquesta modificació Δa_1 introdueix un canvi en $z_2 = w_2 a_1 + b_2$ i per tant, $\Delta z_2 \approx \frac{\partial z_2}{\partial a_1} \Delta a_1 = w_2 \Delta a_1$. Combinant les expressions per les dues capes, obtenim $\Delta z_2 \approx \sigma(z_1) w_2 \Delta b_1$.

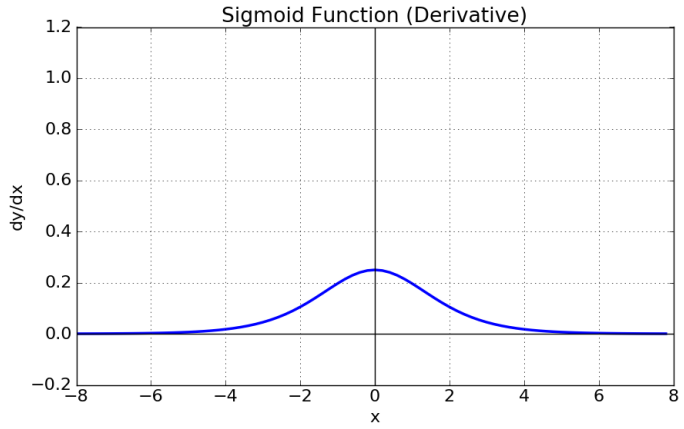
Si seguim fent els canvis que es propaguen al llarg de les capes per Δb_1 fins a tenir ∂C obtenim:

$$\partial C \approx \dot{\sigma}(z_1) w_2 \dot{\sigma}(z_2) w_3 \dot{\sigma}(z_3) w_4 \dot{\sigma}(z_4) \frac{\partial C}{\partial a_4} \Delta b_1.$$

Dividint per Δb_1 obtenim l'expressió desitjada del gradient

$$\frac{\partial C}{\partial b_1} = \dot{\sigma}(z_1) w_2 \dot{\sigma}(z_2) w_3 \dot{\sigma}(z_3) w_4 \dot{\sigma}(z_4) \frac{\partial C}{\partial a_4}. \tag{6.3}$$

Excepte l'últim terme, tota l'expressió és un producte de termes $w_j \dot{\sigma}(z_j)$. Per veure el comportament d'aquests termes, mirem la forma de $\dot{\sigma}$.



Observem que $\max(\dot{\sigma}) = 1/4$ en tot el domini. Si s'utilitza la inicialització dels paràmetres habituals (veure secció 5.3.2) aleshores $|w_j| < 1$ en la primera iteració i $|w_j \dot{\sigma}(z_j)| < 1/4$. Si es fa el producte de diferents termes, aquest decreixerà exponencialment.

Comparem l'expressió 6.3 amb $\dfrac{\partial C}{\partial b_3}$, on:

$$\frac{\partial C}{\partial b_3} = \dot{\sigma}(z_3) w_4 \dot{\sigma}(z_4) \frac{\partial C}{\partial a_4}. \tag{6.4}$$

Veiem que entre 6.4 i 6.3 es comparteixen els tres últims termes però 6.3 té dos termes extres que, si considerem $|w_j \dot{\sigma}(z_j)| < 1/4$, aleshores l'expressió 6.3 és aproximadament 16 cops més petita que 6.4. Aquest és l'origen de l'anul·lació dels gradients.

Exemple per l'explosió de gradients

Aprofitem la xarxa neuronal profunda presentada anteriorment per donar un exemple senzill de l'explosió de gradients. Fixarem els paràmetres de la xarxa neuronal de manera que el gradient exploti. Tot i que l'exemple sigui forçat, ens permet veure que l'explosió de gradients pot passar realment.

Primer de tot, triem grans pesos per la xarxa: $w_1 = w_2 = w_3 = w_4 = 100$. Després, triem els biaixos tals que els termes $\dot{\sigma}(z_j)$ no siguin petits. Això és senzill, es necessita que $z_j = 0$ sigui zero per a cada neurona (i per tant $\dot{\sigma}(z_j) = 1/4$). Per exemple, volem $z_1 = w_1 a_0 + b_1 = 0$. Assignant els valors $b_1 = -100a_0$. Podem aplicar la mateixa idea per seleccionar els altres biaixos de la xarxa. D'aquesta manera, tots els termes $w_j \dot{\sigma}(z_j)$ són iguals a $100 \times 1/4 = 25$. Així doncs, ja que $w_j \dot{\sigma}(z_j) >> 1$ obtenim, en poques iteracions i un nombre considerable de capes, l'explosió dels gradients.

6.3.2 Solucions

Acotació del gradient

L'acotació del gradient, també coneguda com *gradient clipping*, serveix per resoldre el problema de l'explosió dels gradients.

Va ser introduïda per Thomas Mikolov [21] amb una idea senzilla d'acotar els gradients a un nombre petit quan aquests explotessin.

Actualment, el que s'entén per acotar els gradients és acotar la norma $||g||$ del gradient g just abans de que el paràmetre s'actualitzi. Tal com es presenta al capítol 10.11 de [5],

$$Si||g|| > v \Rightarrow g \longleftarrow \frac{gv}{||g||}.$$

On v és un valor llindar i g és el gradient pertinent que s'utilitza per actualitzar els paràmetres.

D'aquesta manera, el gradient de tots els paràmetres es renormalitza amb un simple factor d'escala v, aquest mètode garanteix que l'actualització no perdi la direcció del gradient.

Inicialització de les matrius de pesos

En apartat anterior s'ha vist que una acció fonamental per les xarxes neuronals recurrents és la repetició de multiplicacions entre matrius. Aquests càlculs signifiquen que la matriu resultant és exponencial al llarg de les capes de la xarxa neuronal. Aquest fet pot ser un problema d'estabilitat numèrica, amb resultats d'explotar o desaparèixer ràpid.

Un dels factors més influents en aquest fet és el càlcul repetit de l'estat ocult $s^{(t)}$ utilitzant la mateixa matriu de pesos durant molts instants de temps.

Per resoldre el problema de l'anul·lació del gradient, en lloc d'inicialitzar les matrius de pesos W, U i V de la manera usual vista abans, [15] proposa utilitzar matrius ortogonals.

Definició 6.2. *Una matriu ortogonal és una matriu la qual les seves files són ortogonals entre elles i les seves columnes també:*

$$A^\intercal A = AA^\intercal = I.$$

Les matrius ortogonals tenen moltes propietats interessants però la més important per aquest problema la que es presenta a continuació.

Proposició 6.3. *Tots els valors propis d'una matriu ortogonal tenen valor absolut igual a 1.*

Demostració. Sigui A una matriu quadrada real ortogonal de dimensió n. Sigui $\lambda \in \mathbb{R}^n$ un valor propi, $v \in \mathbb{R}$ el vector propi associat i $\|.\| : \mathbb{R}^n \longrightarrow \mathbb{R}$ una norma.

Es té que $Av = \lambda v$. Aleshores $\|Av\|^2 = \|\lambda v\|^2 = |\lambda|^2\|v\|^2$.

Prenent el terme de l'esquerra de l'expressió es té $\|Av\|^2 = (Av)^\top(Av) = v^\top A^\top Av = v\top v = \|v\|^2$.

Així doncs, $\|v\|^2 = |\lambda|^2\|v\|^2$. Com que v és un vector propi, es té que $\|v\| \neq 0 \implies |\lambda|^2 = 1 \implies |\lambda| = 1$. □

Això significa que, sense importar quants cops es repeteix el càlcul, el resultat de la multiplicació de matrius repetidament no explota ni s'anul·la.

Per posar un exemple il·lustratiu, fixem-nos en un model simplificat. Suposem una Xarxa Neuronal Recurrent amb inputs nuls, sense biaix, amb una funció activació $f(x) = x$ i amb l'estat ocult inicialitzat a la matriu identitat: $s^{(0)} = I$

Proposició 6.4. *Si les següents propietats són certes:*

$$s^{(t)} = f(Ws^{(t-1)} + Vx^{(t)}) = f(Ws^{(t-1)}) = Ws^{(t-1)},$$

$$s^{(0)} = I,$$

$$V = 0.$$

Aleshores, per inducció es té que $s^{(t)} = W^t$.

Demostració. Per $t = 0$, es té que $s^{(0)} = W^0 = I$.

Per $t = 1$ es té que $s^{(1)} = f(Ws^{(0)} + Vx^{(0)}) = f(Ws^{(0)}) = Ws^{(0)} = W$.

Suposant que l'expressió és certa per t, per t+1, es té que $s^{(t+1)} = f(Ws^{(t)} + Vx^{(t+1)}) = f(Ws^{(t)}) = Ws^{(t)} = WW^t = W^{(t+1)}$. □

Amb aquest resultat i utilitzant que $W^t = Q\Lambda^t Q^{-1}$ on Λ és la matriu diagonal de valors propis i Q és la matriu canvi de base composta pels vectors propis de W, es pot veure que el càlcul no tendirà a infinit ni zero, evitant així els problemes esmentats prèviament.

Es pot obtenir una matriu ortogonal aleatòria $n \times n$ (uniformement distribuïda) utilitzant la factorització QR de una matriu gaussiana aleatòria d'elements i.i.d (independents idènticament distribuïts) $n \times n$ de mitjana 0 i variància 1.

6.4 Versions tancades de RNN

Aquesta secció es basa en el capítol 10.10 de [5] i en les imatges de [20].

6.4.1 El problema de les llargues dependències de temps

Un dels atractius de les Xarxes Neuronals Recurrents és la capacitat de connectar informació prèvia amb la tasca que s'està processant en el moment.

De vegades només es necessita informació recent per processar una tasca. Per exemple, considerem un model de llenguatge que intenta predir la següent paraula d'una frase en base a les anteriors. Si s'intenta predir l'última paraula de "Els núvols estan al *cel*", no es necessita un context. És bastant probable que la següent paraula serà *cel*.

En aquest cas, la distància entre la informació rellevant i la tasca actual és petita, però hi ha casos en que es necessita més context.

Per exemple, si s'intenta predir l'última paraula del text "Vaig néixer a França (...) Parlo *francès*". La informació més recent ("parlo") indica que probablement la següent paraula sigui una llengua, però si es vol saber quin idioma és, hem de moure'ns més enrere per arribar al context de França. En aquest cas, pot ser que la distància entre la informació rellevant i el punt actual sigui molt llarga. Desafortunadament, com més creix aquesta distància, més difícil és aprendre a connectar la informació per a les xarxes neuronals recurrents.

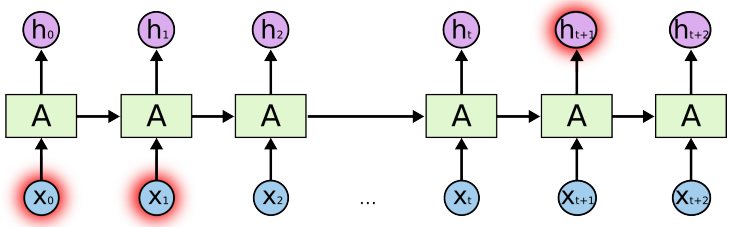


Figura 9: Il·lustració d'una RNN amb una distància gran entre la informació rellevant i el punt que s'intenta predir.

Aquestes unitats permeten acumular informació, com evidències per a una variable en particular, durant un llarg període de temps. Tot i així, un cop s'ha utilitzat la informació, per la xarxa neuronal pot ser útil *oblidar-la*. Necessitem un mecanisme per oblidar els estats anteriors en lloc de decidir manualment quan fer-ho. Es vol que la xarxa neuronal recurrent aprengui a decidir quan fer-ho.

Com s'ha intentat il·lustrar amb l'exemple anterior, les xarxes neuronals recurrents funcionen bé amb dependències de temps curtes. Però, en general, fallen al entendre dependències a llarg plaç. Les versions tancades de les xarxes neuronals recurrents, que tenen unitats dissenyades per oblidar i actualitzar informació rellevant, actualment són la solució al problema.

Les variants més efectives utilitzades en aplicacions pràctiques són les versions tancades de les RNN o *Gated Recurrent Neural Networks*. Aquestes neixen de la idea de crear camins en el temps que tenen derivades que no exploten ni s'anul·len.

Els dos tipus més importants de d'aquestes versions tancades són:

- Long Short Term-Memories (LSTM). Van ser introduïdes per S.Hochreiter i J.Schmidhuber el 1997 [8] i són extensament utilitzades. Degut a la seva alta complexitat, funcionen molt bé en llargues dependències de temps.
- Gated Recurrent Units (GRU). Van ser introduïdes recentment (2014) per K.Cho. [3]. Són més simples i ràpides que LSTM i conseqüentment s'optimitzen més ràpid.

6.4.2 Long Short-Term Memory

Com s'ha vist abans, totes les xarxes neuronals recurrents tenen forma de cadena de mòduls de xarxa neuronal. En les xarxes neuronals recurrents estàndard, aquests mòduls tenen una estructura molt simple formada només per una funció d'activació.

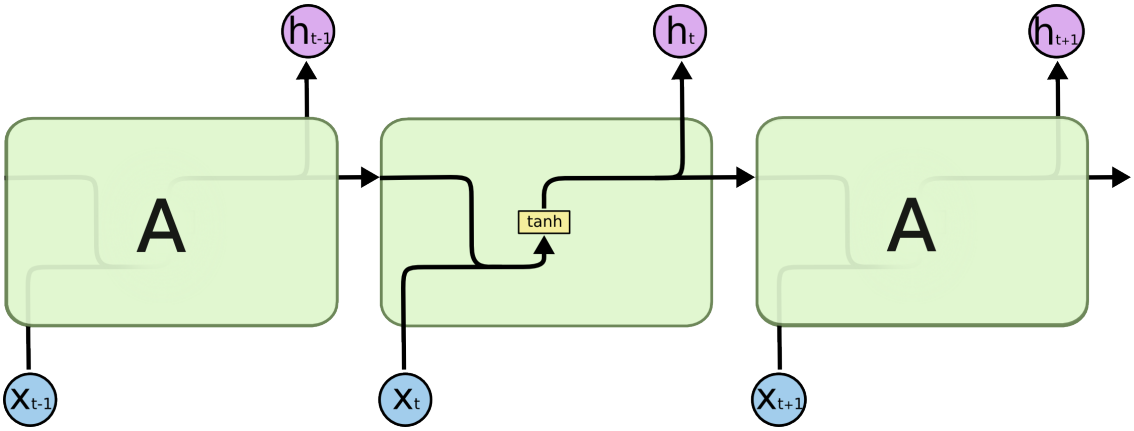


Figura 10: RNN estàndard amb la funció d'activació tanh

Les xarxes neuronals recurrents *Long Short-Term Memory* (LSTM) també tenen aquesta estructura en cadena, però el mòdul que es repeteix és diferent. En lloc de tenir una capa simple n'hi ha quatre, que interactuen d'una manera molt particular.

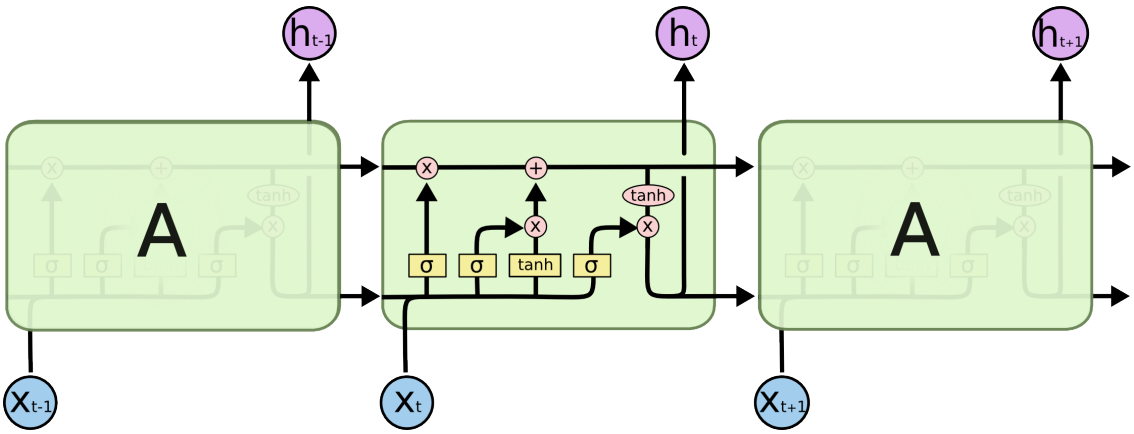


Figura 11: Esquema d'una xarxa LSTM. Cada línia és un vector d'entrada $x^{(t)}$, des de la sortida d'un mòdul a l'entrada de l'altre. Els cercles roses representen operacions de vectors. Les caixes grogues són capes de la xarxa neuronal de dins el mòdul. Les bifurcacions de línies denoten que el contingut es copia a diferents llocs.

Un dels conceptes claus de les xarxes LSTM és l'**estat de la cèl·lula** $C^{(t)}$ (*cell state*), representat com la línia horitzontal de la següent figura.

L'estat de la cèl·lula és una espècie de cinta transportadora. Com veurem a continuació, corre per tota la cadena amb només algunes interaccions lineals. Amb aquest mecanisme, és fàcil que la informació corri per la cadena sense canvis.

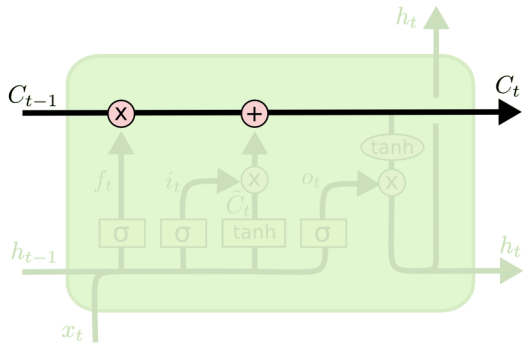


Figura 12: Estat de la cèl·lula

Com s'ha comentat abans, les LSTM tenen la capacitat d'esborrar o afegir informació a l'estat de la cèl·lula. Aquest procés es regula mitjançant estructures anomenades **portes**.

Les portes són una via per deixar passar la informació de manera opcional. Estan compostes per una xarxa neuronal amb una funció d'activació *squashing* i una operació vectorial. D'aquesta capa se'n obté un nombre entre zero i u, que descriu quina quantitat de capa component s'ha de deixar passar. El valor zero significa que la informació processada no és important, i per tant l'oblidarem. El valor u significa que deixarem passar tota la informació en aquest mòdul. Les xarxes LSTM estan compostes per tres d'aquestes portes, que controlen i regulen l'estat de la cèl·lula. A continuació es descriu el procés per obtenir $C^{(t)}$ un cop calculat $C^{(t-1)}$.

LSTM pas a pas

(1) El primer pas és decidir quina informació s'esborrarà de l'estat de la cèl·lula. Aquesta decisió la fa una capa amb una funció d'activació logística que s'anomena **forget gate layer**. Aquesta utilitza $h^{(t-1)}$ i $x^{(t)}$.

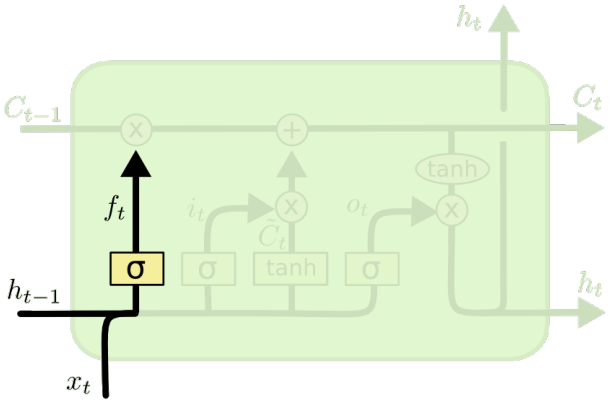


Figura 13: Pas 1

Es pot traduir aquest primer pas amb l'expressió algebraica següent:

$$f_j^{(t)} = \sigma(b_j^f + \sum_k U_{j,k}^f x_k^{(t)} + \sum_k W_{j,k}^f h_k^{(t-1)}).$$

On $x^{(t)}$ és l'actual vector de les dades d'entrada i $h^{(t-1)}$ és l'anterior estat ocult. b^f, U^f i W^f són respectivament els biaixos i les matrius de pesos de la capa actual.

(2) El següent pas és decidir quina de la nova informació es guardarà en l'estat de la cèl·lula. Això té dues parts. Primer, una capa amb funció d'activació logística anomenada **input gate layer** decideix quins dels valors anteriors s'actualitzen. Després, una capa amb funció d'activació tanh crea un vector dels valors nous candidats a afegir-se, $\tilde{C}^{(t)}$.

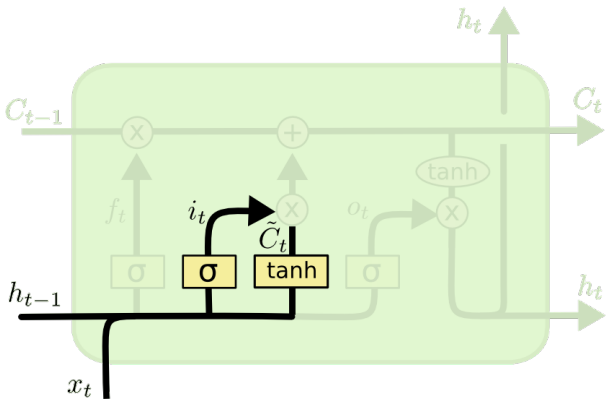


Figura 14: Pas 2

$$i_j^{(t)} = \sigma(b_j^i + \sum_k U_{j,k}^i x_k^{(t)} + \sum_k W_{j,k}^i h_k^{(t-1)}).$$

$$\tilde{C}_j^{(t)} = \tanh(b_j^c + \sum_k U_{j,k}^c x_k^{(t)} + \sum_k W_{j,k}^c h_k^{(t-1)}).$$

(3) En el tercer pas es combinaran els dos resultats anteriors per actualitzar l'estat $C^{(t-1)}$ al nou $C^{(t)}$. El procés és multiplicar l'estat antic per $f^{(t)}$, oblidant els termes que anteriorment s'han decidit. Després, s'afegeix el nou vector de candidats $\tilde{C}^{(t)}$, escalat per la quantitat decidida en el pas anterior.

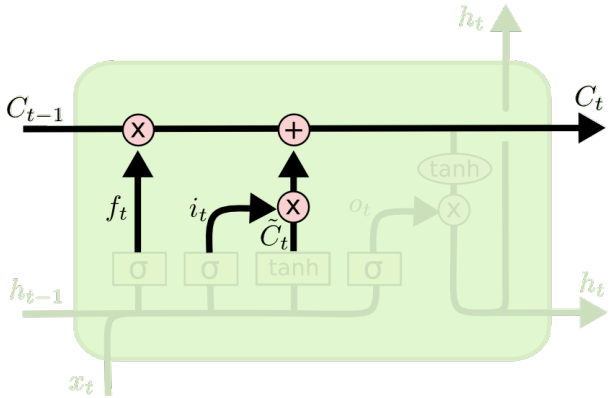


Figura 15: Pas 3

$$C_j^{(t)} = f_j^{(t)} C_j^{(t-1)} + i_j^{(t)} \tilde{C}_j^{(t)}.$$

(4) Finalment, es necessita decidir quina serà la sortida del mòdul. Aquesta sortida $o^{(t)}$ està basada en l'estat de la cèl·lula $C^{(t)}$, però és una versió filtrada.

Primer de tot, es processa una capa logística que decideix quines parts de l'estat de la cèl·lula es conserven. Després, es passa $C^{(t)}$ per la funció tanh (per forçar que els valors estiguin acotats entre -1 i 1) i es multiplica per la sortida de la capa logística anterior, per extreure finalment les parts que s'han decidit.

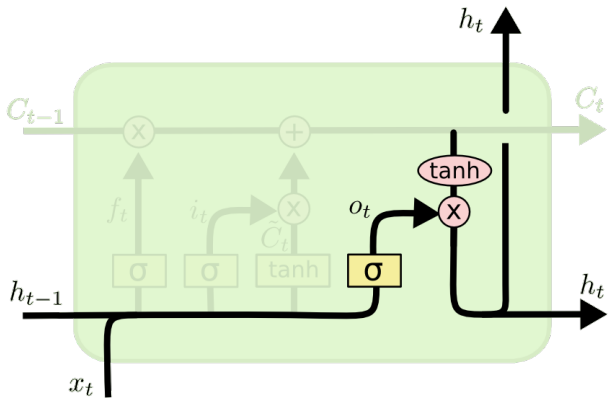


Figura 16: Pas 4

$$o_j^{(t)} = \sigma(b_j^o + \sum_k U_{j,k}^o x_k^{(t)} + \sum_k W_{j,k}^o h_k^{(t-1)}),$$

$$h_j^{(t)} = \tanh(C_j^{(t)}) o_j^{(t)}.$$

En resum, amb equacions simplificades tenim:

(1) $f^{(t)} = \sigma(b^f + U^f x^{(t)} + W^f h^{(t-1)})$,

(2) $i^{(t)} = \sigma(b^i + U^i x^{(t)} + W^i h^{(t-1)})$,
$\tilde{C}^{(t)} = \tanh(b^c + U^c x^{(t)} + W^c h^{(t-1)})$,

(3) $C^{(t)} = f^{(t)} \odot C^{(t-1)} + t^{(t)} \odot \tilde{C}^{(t)}$,

(4) $o^{(t)} = \sigma(b^o + U^o x^{(t)} + W^o h^{(t-1)})$,
$h^{(t)} = \tanh(C^{(t)}) \odot o^{(t)}$.

On $o^{(t)}$, $f^{(t)}$, b^o, b^f, i, b^i, b^c, $h^{(t)}$, $C^{(t)}$ i $\tilde{C}^{(t)}$ són vectors de dimensió n. U^c, U^o, U^f i U^i són matrius de dimensió $n \times m$. W^o, W^f, W^i i W^c són matrius de dimensió $n \times n$. Sent n el nombre de neurones dins de la unitat LSTM i m la dimensió de $x^{(t)}$ (nombre de variables).

6.4.3 Gated Recurrent Units

En aquest punt ens podem preguntar: Quines peces de l'arquitectura LSTM són necessàries? Quines altres arquitectures es poden dissenyar per tal que la xarxa neuronal controli els instants de temps per poder oblidar el comportament de les seves unitats?

Recentment s'han donat algunes respostes a aquestes preguntes amb unes unitats anomenades **Gated Recurrent Units** (GRU). La principal diferència amb les LSTM és que una única unitat controla el factor d'oblit (*reset*) i la decisió d'actualitzar l'estat h. Les equacions que conformen aquestes unitats són les següents:

$$\hat{h}_i^{(t)} = \tanh(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}),$$

$$h_i^{(t)} = z_i^{(t)} \hat{h}_i^{(t)} + (1 - z_i^{(t-1)}) h_i^{(t-1)},$$

on z és la capa d'actualització i r la capa d'oblit. Els seus valors venen definits com l'usual:

$$z_i^{(t)} = \sigma(b_i^z + \sum_j U_{i,j}^z x_j^{(t)} + \sum_j W_{i,j}^z h_j^{(t-1)}),$$

$$r_i^{(t)} = \sigma(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t-1)}),$$

on r, z, b, $h^{(t)}$, $\hat{h}^{(t)}$, b^z i b^r són vectors de dimensió n. W, W^z i W^r són matrius de dimensió $n \times n$. U, U^z i U^r són matrius de dimensió $n \times m$. Sent n el nombre de neurones dins de la unitat GRU.

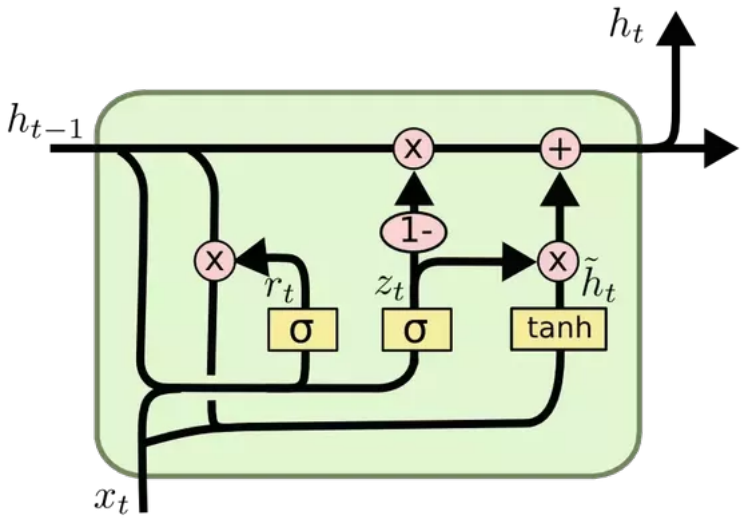


Figura 17: Cèl·lula GRU

Les capes z i r poden ignorar individualment parts del vector s'estat h. Similarment a les LSTM, aquestes capes escullen si copiar el contingut de l'estat o ignorar-lo mitjançant la funció logística.

La capa de *reset* r introdueix una no linealitat de més, tenint en compte la relació entre l'estat passat i el futur.

Es poden dissenyar moltes variants més al voltant d'aquest tema. Tot i així, diferents investigacions (com [6] i [11]) sobre aquestes variacions no han trobat una millora significativa respecte GRU i LSTM.

7 Cas d'estudi: Predicció de sèries temporals

En aquesta secció aplicarem la part teòrica del treball per a una col·lecció de dades concretes. Aquesta part s'ha fet en col·laboració amb una comercialitzadora energètica d'energia renovable: *Holaluz*. Si haguéssim de definir el problema a resoldre amb una frase, seria la següent:

Predir a un termini de temps X el consum d'electricitat agregat per tarifa i zona geogràfica dels clients de la cartera d'Holaluz.

Per anonimitzar les dades, no es farà públic la tarifa ni la zona geogràfica que s'ha escollit per portar a terme aquest cas d'estudi. A més a més, s'ha fet una transformació lineal de les dades originals per motius de protecció de dades.

La tarifa és un concepte del mercat elèctric que divideix els consumidors en segments que s'estableixen segons la tensió, la discriminació horària (quan el preu de l'electricitat varia segons l'hora consumida) i la potència contractada.

En les seccions següents es presentaran les dades utilitzades, el tractament d'aquestes, el model utilitzat i els resultats obtinguts. En aquesta última secció, s'ha comparat els resultats obtinguts implementant una xarxa neuronal recurrent amb un model de sèries temporals *Autoregressive integrated moving average* (ARIMA) per tal d'utilitzar-lo de base a l'hora d'interpretar els resultats. Els models ARIMA són un model clàssic per a sèries temporals.

La implementació del model s'ha portat a terme amb el paquet *Keras* [2] dins del llenguatge *Python* [22], a l'annex s'hi pot trobar el codi. Keras és una API de xarxes neurals d'alt nivell. Està escrita en Python i s'ha utilitzat la versió que funciona sobre de *TensorFlow*.

7.1 Naturalesa de les dades

Per qüestions d'entorn, les dades que disposem es poden diferenciar en dos tipus. Per això, s'ha separat les dades en dues col·leccions i s'ha dividit el problema principal en dos.

El motiu d'aquesta separació és que, gran part de les dades que es tenen venen informades dels comptadors dels clients. Hi ha dos tipus de comptadors:

- Els comptadors telemesurats, que envien una dada de consum en kWh de manera horària.
- Els comptadors no telemesurats, que no envien dades al sistema i es rep informació mitjançant lectures (en kWh) que genera la distribuïdora elèctrica en un període d'un mes.

Així doncs, tenim una col·lecció de dades amb informació horària i una altra amb informació mensual.

month	period	total active cups
2017-03	P1	10533
total cups with consumptions	weighted labor days	demand mean
10461	69	90235.93
demand sd	weighted temperature min	total consumption
7422.27	8.1	13143065

Taula 1: Exemple de la col·lecció de dades no telemesurades. Temperatura mesurada en graus Celsius. El període correspon a un conjunt d'hores del dia, en total hi ha tres períodes.

datetime	total cups with consumptions	weighted labor days
2018-03-29 16:00:00	2409	1.99
temperature	relative humidity	temperature max
285.66	176.53	287.92
temperature min	shortwave radiation	wind speed
280.67	552.3	4.37
windchill	heat index	demand
39.53	88.32	76899
total consumption		
34428.903		

Taula 2: Exemple de la col·lecció de dades telemesurades. Temperatura mesurada en graus Kelvin, velocitat del vent en m/s i radiació curta d'ona en W/m^2. *Windchill* i *heat index* són índexs que reflecteixen les sensacions tèrmiques de fred i calor respectivament.

Tant en la taula 1 com en la taula 2 es pot veure un exemple de cada col·lecció de dades. Els *cups* (Código Unificado del Punto de Suministro) són els punts de subministrament que la companyia té donats d'alta en la tarifa i la zona geogràfica corresponent.

Es pot veure un conjunt de variables meteorològiques que s'han inclòs al model per la seva alta correlació amb el consum d'electricitat, extretes de *Global Forecast System*[18]. Per últim, la variable *demand* és una dada que prové de Red Eléctrica Española[26], sent una estimació, a grans trets, de la demanda del sistema elèctric en general.

La variable resposta s'anomena *Total consumption* i és la suma del consum en kWh dels cups per la tarifa i la zona geogràfica en qüestió.

Com es pot comprovar, en el cas de les dades telemesurades, es té una agregació total del consum cada hora. En les dades no telemesurades es té una agregació total

del consum mensual i per període. El període correspon a un conjunt d'hores del dia, en total hi ha tres períodes.

Aquest fet implica directament una gran diferència en la mida de les dues col·leccions de dades. En el cas de les dades no telemesurades, tenim 84 exemples per a cada període i la primera dada és del juliol del 2011. Les dades telemesurades consten de 7766 exemples que comencen a l'1 de juny de 2016.

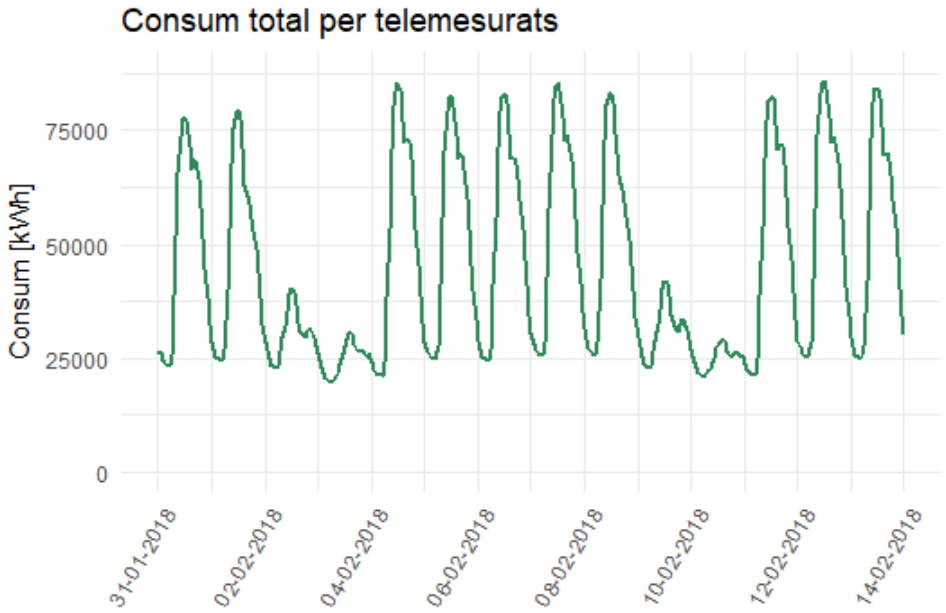


Figura 18: Consum agregat per comptadors telemesurats.

La figura 18 mostra el consum total d'electricitat a nivell horari durant 15 dies. Es pot apreciar un comportament molt diferenciat entre dia i nit i entre caps de setmana i dies laborables.

La figura 19 mostra el consum d'electricitat del segment en qüestió a nivell mensual en un període d'un any. Es pot veure un comportament creixent ja que la cartera de clients ha anat augmentant. També s'aprecia que el període P2 conté les hores puntes de consum d'electricitat.

7.2 Preprocessat

Per poder utilitzar la informació que s'ha presentat, és necessari fer un preprocessat de les dades.

Primer de tot, degut a la gran varietat de variables que tenim i la diversitat de magnituds, una bona pràctica per qualsevol algorisme de *machine learning* és normalitzar les dades.

Normalitzar les dades és un concepte molt ampli que es pot fer de moltes maneres. En aquest cas s'ha transformat les dades de manera que tinguin mitjana igual a 0

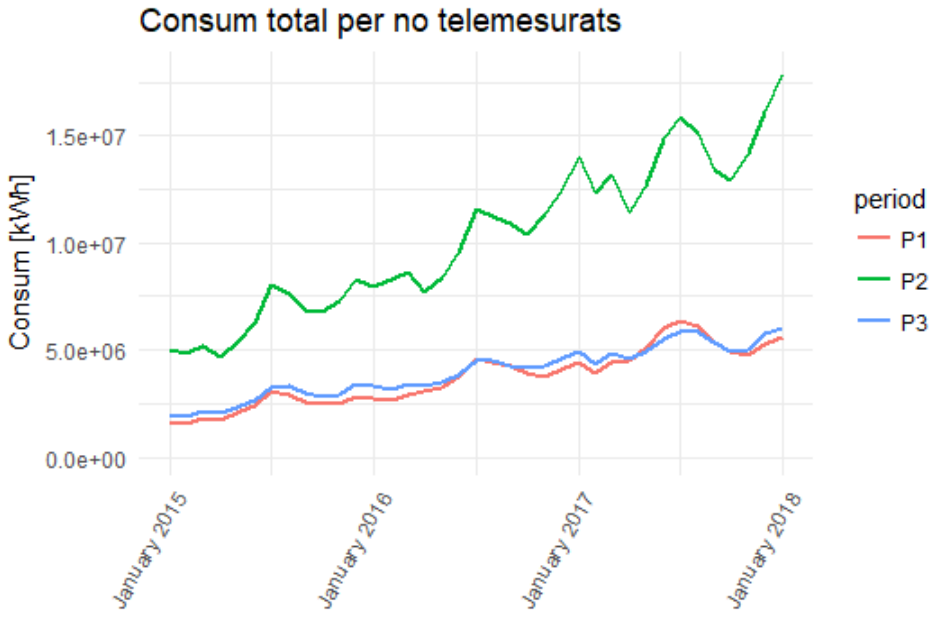


Figura 19: Consum agregat per període per comptadors no telemesurats.

i desviació típica igual a 1.

Així doncs, si X és la col·lecció de dades, la nova col·lecció de dades normalitzada $\tilde{X}$ serà:

$$\tilde{X} = \frac{X - \mu}{\sigma},$$

on μ i σ corresponen a la mitjana i la variància mostrals de la col·lecció de dades X. D'aquesta manera, la mitjana de $\tilde{X}$ és zero i la desviació típica és 1.

Una manera de validar el model és separar cada col·lecció en dues parts, la part d'**entrenament** i la part de **validació**. La primera s'utilitza perquè el model fixi els seus paràmetres, en el nostre cas, per fixar els valors de les matrius i els vectors de pesos. La segona part s'utilitza per veure el comportament del model fixat amb unes dades desconegudes que no ha processat durant l'entrenament, d'aquesta manera es calcula l'error generalitzat (veure secció 2.2).

En el nostre cas, la separació de les dues col·leccions de dades és la següent:

- Dades telemesurades : La col·lecció de validació consta de 144 hores d'exemples i la part d'entrenament conté els 6 mesos anteriors de dades. Això equival a dir que es fa una predicció a 6 dies vista amb un històric de 6 mesos.
- Dades no telemesurades: La col·lecció de validació conté 3 mesos d'exemples i la part d'entrenament conté totes les dades anteriors a aquests mesos en qüestió. Això equival a fer una predicció a 3 mesos vista amb totes les dades històriques.

En els problemes com el nostre, on es tracten dades de caràcter temporal, la separació de les dades entre validació i entrenament es fa de manera ordenada. És a dir, s'escullen els punts amb que es farà la validació, i a partir d'aquests s'agafa un volum més gran de dades anterior als punts de validació. Per exemple, si volem validar el model amb els mesos de agost i setembre de 2017, s'entrenarà el model amb les dades anteriors a aquests mesos.

De cara a assegurar que el model no comet *overfitting*, en el cas de les dades telemesurades s'han triat tres col·leccions de validació que consten dels dies **29 de juliol al 3 d'agost de 2017, 3 al 8 d'octubre de 2017 i 1 al 6 d'abril del 2018**. Aquesta selecció s'ha fet intentant cobrir diferents estacions de l'any i agafant dies laborables i caps de setmana. D'aquesta manera, s'intenta evitar que el model funcioni molt bé amb un tipus de dades i malament amb un altre tipus.

En el cas de les dades no telemesurades, s'ha agafat com a validació els mesos de **setembre, octubre i novembre de l'any 2017**. A diferència del cas anterior, no s'ha fet la validació múltiple a causa del poc nombre de dades que es té.

7.3 Model

En aquesta secció es detallaran alguns dels paràmetres que defineixen el model de predicció que hem programat.

En ambdós models (telemesurat i no telemesurat) s'ha utilitzat una arquitectura amb dues capes ocultes: la primera capa és capa recurrent GRU amb 80 unitats i la següent és una capa densa no recurrent amb 50 neurones.

S'ha utilitzat la funció d'activació tanh per a l'activació de les dues capes ocultes. La capa de sortida no té funció d'activació.

També s'ha utilitzat un mètode d'optimització anomenat **Adam** (en podem trobar més detalls a [12]) . És una versió computacionalment més eficient i amb menys requeriments de memòria del mètode d'optimització del descens seguint el gradient explicat a la secció 5.2. Aquest optimitzador ha utilitzat una constant d'aprenentatge γ de 0.001.

Un altre paràmetre que s'ha hagut de definir és la **mida del batch** (*batch size*), que consisteix en el nombre de mostres que es propaguen per la xarxa neuronal independentment, en paral·lel. En el cas d'entrenament, els paràmetres del model s'actualitzen cada cop que un batch és processat (no cada cop que un exemple es processa).

Generalment, un batch aproxima millor la distribució de les dades d'entrada que fer-ho d'un en un. Com més gran sigui el batch, millor serà l'aproximació. Tanmateix, també és veritat que el batch tardarà més temps en processar-se i només generarà una única actualització dels pesos. Per fer prediccions, es recomana triar una mida de batch que sigui tan gran com es pugui permetre sense sortir de la memòria de la màquina. En els nostres models s'ha fixat la mida del batch a 6.

També s'ha fixat **nombre d'èpoques**. Aquest paràmetre és el nombre de vegades en que la xarxa neuronal processa tota la col·lecció d'entrenament. S'ha fixat a 100.

Per últim, cal comentar com s'ha inicialitzat els pesos de la xarxa. Per la capa GRU, s'ha utilitzat matrius ortogonals generades aleatòriament com s'explica a la secció 6.3.2. Els pesos de la capa densa s'han inicialitzat seguint una distribució normal amb desviació típica de $\sqrt[2]{\frac{2}{in + out}}$ on $in = 80$ i $out = 50$ són les dimensions d'entrada i de sortida de la capa.

7.3.1 Nombre de paràmetres

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, 80)	22080
dense_1 (Dense)	(None, 50)	4050
dense_2 (Dense)	(None, 1)	51

Total params: 26,181
Trainable params: 26,181
Non-trainable params: 0

Figura 20: Resum del model telemesurat proporcionat per *Keras*.

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, 80)	20880
dense_1 (Dense)	(None, 50)	4050
dense_2 (Dense)	(None, 1)	51

Total params: 24,981
Trainable params: 24,981
Non-trainable params: 0

Figura 21: Resum del model no telemesurat proporcionat per *Keras*.

A les figures 20 i 21 podem veure el total de paràmetres desglossat per capes que el model ha d'aprendre. Aquestes xifres quadren amb les dimensions mostrades a la secció 6.4.3. En ambdós casos tenim que el nombre d'unitats de la capa recurrent és $n = 80$, en el cas telemesurat tenim que el nombre de variables explicatives és $m = 11$ i en el cas no telemesurat $m = 6$. Així doncs, el nombre de paràmetres de la capa recurrent es calculen de la següent forma:

$$\begin{aligned} \#Parametres_{GRU} = dim(b) + dim(b^z) + dim(b^r) + dim(U) + dim(U^z) + dim(U^r) \\ + dim(W) + dim(W^z) + dim(W^r) = \\ = 3 \times n + 3 \times n \times m + 3 \times n \times n, \end{aligned}$$

Que és 22080 per les dades telemesurades i 20880 per les dades no telemesurades. El nombre de paràmetres associades a la segona capa oculta no recurrent és, per ambdós casos:

$$\begin{aligned}\#Parametres_{Dense} =& \#UnitatsGru \times \#NeuronesCapaDensa + dim(biaix_{Dense}) = \\ =& 80 \times 50 + 50 = 4050.\end{aligned}$$

Finalment, el nombre de paràmetres associats a la capa de sortida per ambdós casos és:

$$\begin{aligned}\#Parametres_{Sortida} =& \#NeuronesCapaDensa \times \#NeuronesCapaSortida + dim(biaix_{Sortida}) = \\ =& 50 \times 1 + 1 = 51.\end{aligned}$$

Per altra banda, el nombre de paràmetres utilitzats en el model ARIMA són els donats per un polinomi de primer ordre i un de segon ordre en el cas del model per a les dades no telemesurades i tres polinomis (un de primer, un de segon, i un de cinquè ordre) en el cas de les dades telemesurades.

7.4 Resultats i discussió

Per qüestions de negoci, l'error que interessava analitzar en aquesta tasca és el *Mean Absolute Percentage Error* (MAPE).

$$MAPE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{y_i} \cdot \frac{100}{n}.$$

Aquest error mesura en valor absolut la desviació del valor predit respecte el valor real i després ho escala sobre 100 per poder comparar diferents magnituds.

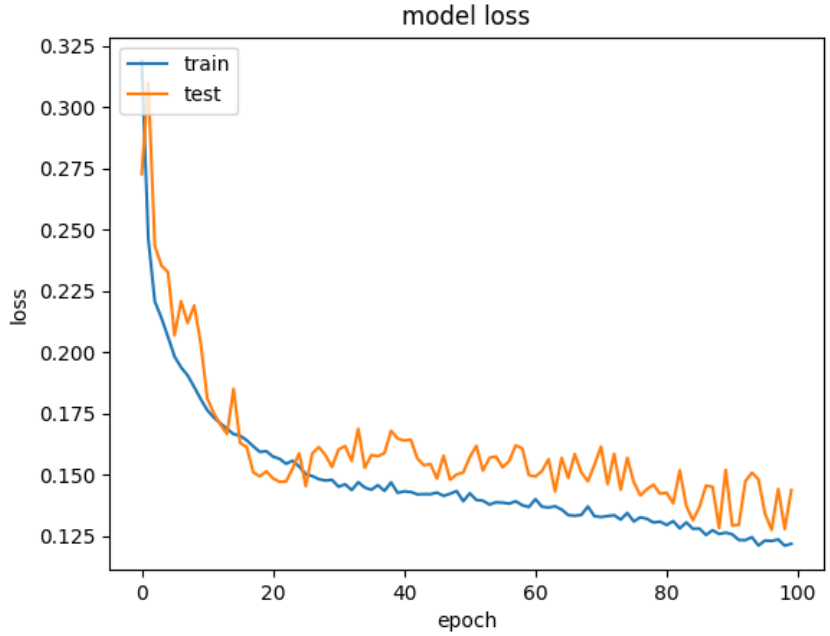


Figura 22: Evolució de l'error del model respecte les èpoques en la col·lecció de validació d'abril.

La figura 22 mostra l'evolució del model telemesurat respecte les èpoques fixades. En l'eix de les ordenades es veu el valor del MAPE en tant per 1. Podem apreciar una reducció significativa de l'error al llarg de les èpoques. També podem observar que la forma de la línia corresponent a les dades d'entrenament (train) és molt més suau que la línia de l'error corresponent a les dades de validació (test). Aquest fet és molt habitual ja que el model no ha utilitzat les dades de validació per entrenar-se i la seva performance no és tant bona.

model	validació	període	MAPE	desviació típica error
RNN	Set-nov 2017	P1	5.34	0.27
	Set-nov 2017	P2	1.79	0.067
	Set-nov 2017	P3	2.02	0.06
ARIMA	Set-nov 2017	P1	5.01	0.03
	Set-nov 2017	P2	0.86	0.002
	Set-nov 2017	P3	1.67	0.014

Taula 3: Error pel model de les dades no telemesurades.

model	validació	MAPE	desviació típica MAPE	MRE
RNN	Agost 2017	4.97	4.51	-1.03
	Octubre 2017	8.36	7.27	-5.02
	Abril 2018	6.34	1.65	-0.58
ARIMA	Agost 2017	15.36	0.12	-8.06
	Octubre 2017	19.08	0.13	-5.2
	Abril 2018	20.92	0.13	-12.62

Taula 4: Error pel model de les dades telemesurades.

A les taules 3 i 4 s'hi pot veure els resultats de la implementació del model definit a la secció anterior (RNN) en comparació amb el model base (ARIMA).
A la taula 4 s'ha inclòs l'error relatiu mitjà (MRE) que es calcula:

$$MRE = \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)}{y_i} \cdot \frac{100}{n}.$$

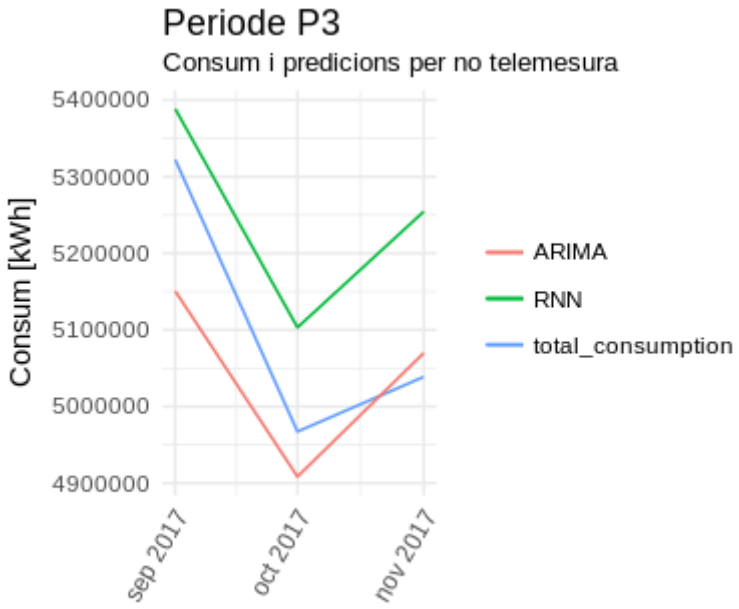


Figura 23: Resultats en les dades de validació per la col·lecció no telemesurada.

Als gràfics 23 i 24 es pot veure la diferència de prediccions entre els dos models amb el consum real.

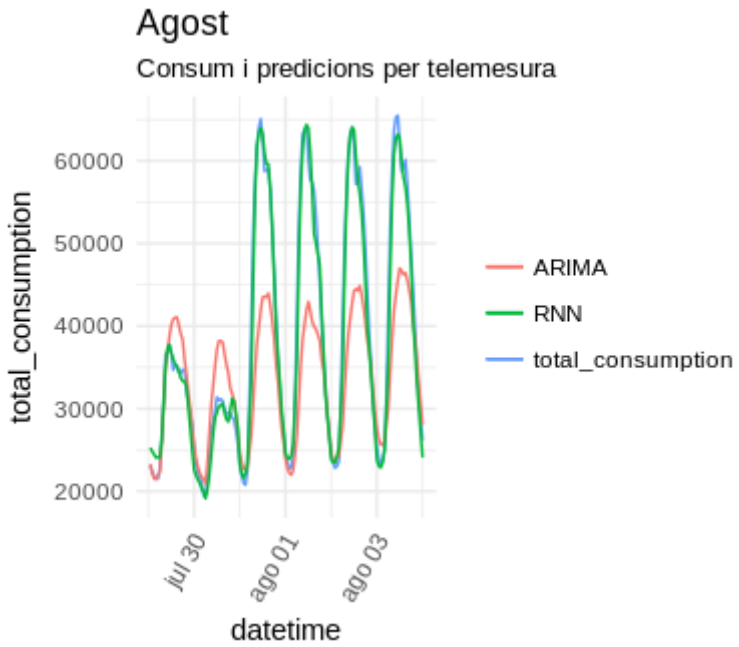


Figura 24: Resultats en les dades de validació per la col·lecció telemesurada.

A l'hora de comentar els resultats presentats prèviament, ens podem fer algunes preguntes:

1. Per que la bondat del model ARIMA és superior en les dades no telemesurades i la bondat del model RNN en les dades telemesurades?

   Pel que fa al bon ajust de l'ARIMA respecte el model RNN en les dades no telemesurades, atribueixo aquesta diferència al nombre de d'exemples que té el model (84). Hem vist a la secció 7.3.1 que el nombre de paràmetres que té el model RNN és 20880. És clar, doncs, que aquest model està encarat a processar volums molt més grans de dades i que és difícil poder aprendre 20880 paràmetres amb 84 exemples i donnt un error petit.

   Aquest fet contrasta amb la bondat del model RNN respecte el model ARIMA en les dades telemesurades, que s'entrenen amb 6 mesos d'històric (és a dir, amb uns 4320 exemples). Això fa que el model RNN, al tenir més paràmetres és un model més flexible i té un major ajust que el model ARIMA, que consta de molts menys paràmetres.

2. Per què, en ambdós models, l'error amb les dades telemesurades és més alt?

   Aquest fet l'atribueixo a la naturalesa de les dades. Si ens mirem les figures 18 i 19, podem veure que les dades a nivell horari són molt més complexes que les dades a nivell mensual ja que es poden apreciar petits canvis intra horaris. Des del meu punt de vista, aquesta complexitat fa que l'error irreductible sigui més alt. En particular, a la taula 4 veiem el model ARIMA obté error molt alts (en mitjana del 18%), que corrobora que amb models menys flexibles com aquest és més difícil captar aquests petits canvis.

3. Per què la desviació típica de l'error és més alta, en general, en el cas del model RNN?

   Sembla ser que aquest fet també és una qüestió de la diferència amb el nombre de paràmetres entre els models. Això ho explica el capítol 2 del llibre [10] on parla de la compensació entre la variància i el biaix dels models.

   Informalment, la **variància** mesura com de lluny un conjunt de nombres estan dispersats de la seva mitjana. Si el model te variància alta aleshores petits canvis en les dades d'entrenament poden afectar molt a la predicció final.

   Per altra banda, es refereix al **biaix** com l'error que s'introdueix modelant dades reals amb models simples, com per exemple l'ARIMA.

   Com a regla general, com més flexible sigui el model (més paràmetres tingui), la variància serà més gran i el biaix serà més baix.

4. Per què el període P1 en les dades no telemesurades té un error significativament més alt que els altres períodes?

   Per aquest fet no s'ha trobat una explicació concreta. A la figura 25 podem veure un zoom de les dades telemesurades per tal de mirar-les en les dates de validació (setembre-novembre 2017). Observem que els períodes P1 i P3

tenen comportaments molt semblants. S'ha fet un anàlisi de les variables explicatives durant aquell període i no s'ha trobat tendències estranyes.

L'únic factor que li podem atribuir és que el període P1, a diferència del P3 que sempre es manté igual, canvia d'hivern (18-22h) a estiu (11-15h). Per tant pot ser que durant l'entrenament del model aquest fet sigui un factor de confusió.

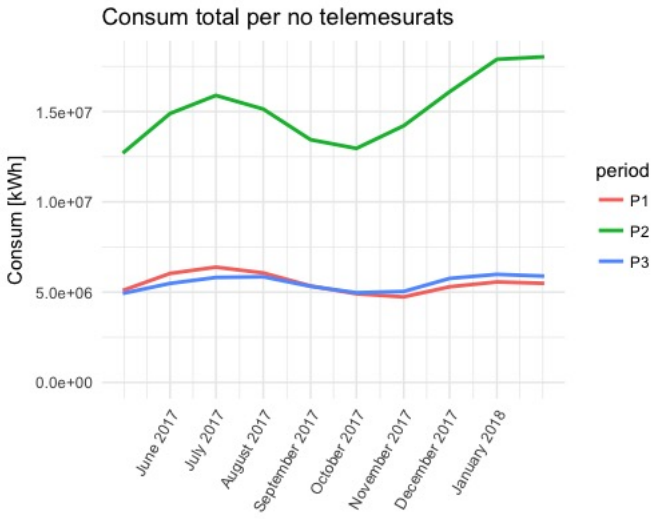


Figura 25: Zoom de la figura 19.

5. Per què l'error dels models amb dades telemesurades és més gran a l'octubre?

   Es pot observar que l'error dels models ARIMA i RNN en la col·lecció de dades horàries és més alt.

   Entre els experts amb models de consum d'electricitat a *Holaluz*, és comú dir que durant les èpoques on hi ha canvis d'estacions la bondat del model disminueix. Explorem aquest fet.

   Si mirem la figura 26 podem veure com les variables meteorològiques que s'han dibuixat tenen una tendència a la baixa (temperatura màxima i mínima, radiació solar i sensació tèrmica). En canvi, es pot apreciar que la línia que dibuixa el consum total no experimenta aquesta tendència, o l'experimenta molt suaument.

   Aquest factor sembla molt determinant a l'hora de fer una predicció pel mes d'octubre, ja que sembla que les variables explicatives experimenten uns canvis que el model també atribueix a la variable resposta i la prediu, en mitjana, menor que el real (veure MRE de la taula 4). Sembla que aquest és el fet que porta a augmentar l'error de manera significativa.

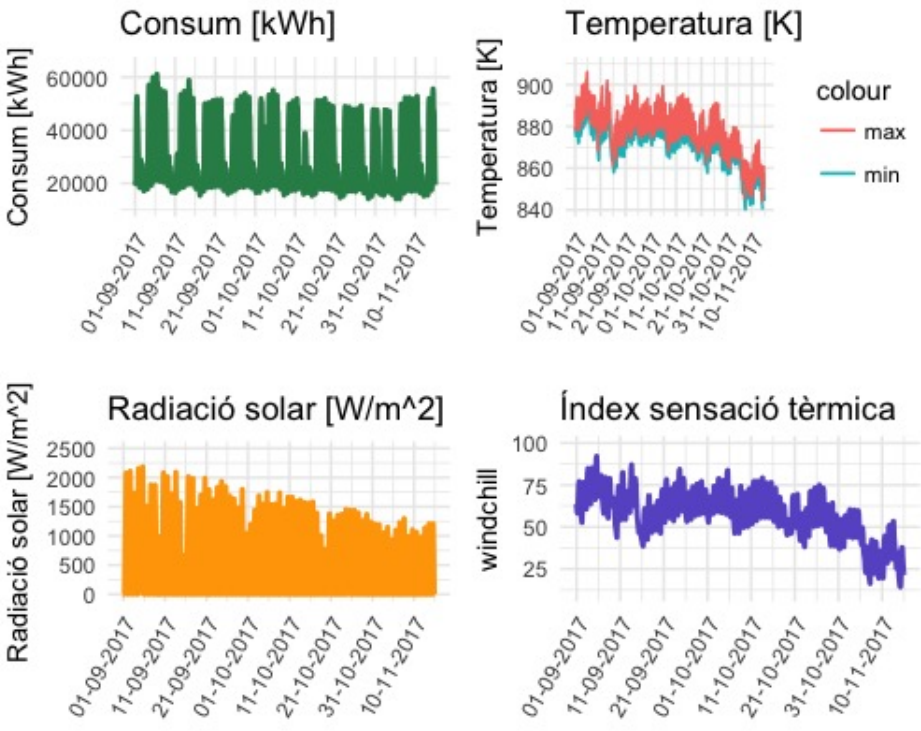


Figura 26: Tendència de la variable resposta (Consum) i algunes variables explicatives meteorològiques durant l'octubre. Dades telemesurades.

8 Conclusions

En aquesta memòria s'ha presentat el funcionament de les xarxes neuronals recurrents a nivell matemàtic. Per fer-ho, primer s'ha explorat l'estructura més senzilla, les xarxes neuronals orientades cap endavant, que han permès posar en context i entendre posteriorment les xarxes neuronals recurrents. Al seu torn, l'estudi d'aquestes i els seus problemes associats han donat sentit al posterior estudi de les versions tancades de les xarxes neuronals recurrents, que han conclòs l'estudi teòric i s'han utilitzat per la implementació pràctica.

El principal fet que vull destacar és la gran extensió del *Deep Learning* com a camp d'estudi. Durant la confecció del treball he anat trobant moltes fonts, totes explicades amb diferents enfocaments, que han suposat un esforç de lectura però a la vegada han enriquit el treball. Principalment, he notat aquesta diversitat d'enfocaments en el camp de la propagació cap enrere i la propagació cap enrere a través del temps.

Pel que fa a la part pràctica, la nombrosa quantitat de fonts m'ha sigut molt útil a l'hora de desenvolupar un codi amb un llenguatge que era desconegut per mi i que ara veig que està ple de possibilitats. A més a més, aquesta part ha estat clau a l'hora d'entendre tota la teoria estudiada prèviament i a la vegada, ser conscient de tot el que em falta per aprendre en aquest camp.

Crec que, tot i no ser l'objectiu del meu treball, el cas d'estudi pràctic té molt futur per explorar i probablement seguiré investigant, dins d'*Holaluz*, possibles canvis.

Per exemple, un possible estudi és implementar LSTM en lloc de GRU per poder valorar quin tipus d'unitat s'ajusta millor al problema, o entendre a nivell teòric els mètodes d'optimització derivats del descens seguint el gradient, com els presentats a *"An overview of gradient descent optimization algorithms"(2016)*[23], per entendre el seu comportament, virtuts, defectes i poder escollir i implementar el que sigui més adient.

També, un aspecte que crec que pot ser molt interessant per a futurs estudis és la inicialització de l'estat $s^{(-1)}$ de les versions tancades de les xarxes neuronals recurrents. L'article *"State initialization for recurrent neural network modeling of time-series data"(2017)*[17], estudia com fer una bona inicialització d'aquest estat ocult i proposa utilitzar una xarxa neuronal orientada cap endavant per inferir-los.

A Annex

A.1 Cas d'estudi I: Dades no telemesurades

```
############## PACKAGES
from numpy import concatenate
from numpy import where
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import GRU
import pandas as pd

############## VARIABLES
month = 10
year = 2017
variables = ['month','total_active_cups','
   total_cups_with_consumptions','weighted_labor_days','
   demand_mean', 'demand_sd','weighted_temperature_min','
   total_consumption']
activation='tanh'
n_iter = 15

############## RETRIEVE DATA
path_to_script = '/home/nuria/Dropbox/TFG/rnn_noTM_input.
   csv'
dataset = pd.read_csv(path_to_script)

############## DEAL WITH DATETIMES
dataset['month'] = pd.to_datetime(dataset['month'])
dataset = dataset.sort_values('month')

############## DEAL WITH NaN's
loc_NaN = where(dataset.isna()) # locating the NaN's
dataset = dataset.drop(dataset.index[loc_NaN[0]]) # drop
   rows which have at least one NaN

############## SELECT LAST UPDATED INFORMATION
dataset = dataset[dataset.date_of_update == max(dataset.
   date_of_update)]

############## PERIODS
datasetP1 = dataset[dataset['period'] == 'P1']
datasetP2 = dataset[dataset['period'] == 'P2']
datasetP3 = dataset[dataset['period'] == 'P3']
```

```
############## GET INDEX OF VALIDATION SPLIT
def get_index_validation_prod(dataset, month, year):
    currentMonth = month
    currentYear = year
    index_today = where((dataset['month'].dt.month ==
  currentMonth) & (dataset['month'].dt.year == currentYear
  ))
    index_today = int(index_today[0])
    return index_today

############## MODEL DEFINITION
def RNN(train_X, train_y, test_X, test_y, scaler):
    # Design network
    model = Sequential()
    model.add(GRU(80, input_shape=(train_X.shape[1],
  train_X.shape[2]), activation=activation))
    model.add(Dense(50,  activation=activation,
  kernel_initializer='glorot_normal'))
    model.add(Dense(1, kernel_initializer='glorot_normal')
  )
    model.compile(loss='mean_absolute_error', optimizer='
  adam', metrics=['accuracy'])

    # Fit model
    nEpochs = 100
    batchSize = 6
    model.fit(train_X, train_y, epochs=nEpochs, batch_size
  =batchSize, validation_data=(test_X, test_y), verbose=1,
    shuffle=False)

    # Make a prediction
    yhat = model.predict(test_X)
    test_X = test_X.reshape((test_X.shape[0], test_X.shape
  [2]))

    # Invert scaling for forecast
    inv_yhat = concatenate((test_X, yhat), axis=1)
    inv_yhat = scaler.inverse_transform(inv_yhat)
    inv_yhat = inv_yhat[:, -1]

    return inv_yhat
```

```
############ MAIN FUNCTION
def launch_model_per_period(dataset, n_iter, month, year):
    # Select used variables
    dataset = dataset[variables]
    index_today = get_index_validation_prod(dataset, month
   , year)

    # Delete rows which month > currentMonth +1
    dataset = dataset.drop(dataset.index[(index_today + 2)
   :])

    # Select m-1, m, m+1
    index_validation = index_today - 1  # where the
   validation split starts

    # Deal with 'date' type
    dataset.index.name = 'month'
    del dataset['month']

    # Normalizing values
    values = dataset.values
    values = values.astype('float32')
    scaler = StandardScaler()
    scaled = scaler.fit_transform(values)

    # Data splitting
    train_X = scaled[:index_validation, :-1]
    test_X = scaled[index_validation:, :-1]
    train_y = scaled[:index_validation, -1]
    test_y = scaled[index_validation:, -1]

    # Reshape input to be 3D: samples, timesteps, features
    train_X = train_X.reshape((train_X.shape[0], 1,
   train_X.shape[1]))
    test_X = test_X.reshape((test_X.shape[0], 1, test_X.
   shape[1]))

    # Prediction average
    prediction_dataframe = pd.DataFrame()
    for x in range(n_iter):
        prediction_vector = RNN(train_X, train_y, test_X,
   test_y, scaler)
        prediction_dataframe = prediction_dataframe.append
   ([prediction_vector])
```

```
    return(prediction_dataframe)

############## LET'S MAKE A PREDICTION!
predictionP1_df = launch_model_per_period(datasetP1,
   n_iter, month, year)
predictionP2_df = launch_model_per_period(datasetP2,
   n_iter, month, year)
predictionP3_df = launch_model_per_period(datasetP3,
   n_iter, month, year)

############# RETRIEVE ERROR
predictionP1 = predictionP1_df.mean(axis=0).tolist()
predictionP2 = predictionP2_df.mean(axis=0).tolist()
predictionP3 = predictionP3_df.mean(axis=0).tolist()

consumptionP1 = datasetP1[datasetP1['month'].isin(['
   2017-09-01', '2017-10-01', '2017-11-01'])].
   total_consumption.tolist()
consumptionP2 = datasetP2[datasetP2['month'].isin(['
   2017-09-01', '2017-10-01', '2017-11-01'])].
   total_consumption.tolist()
consumptionP3 = datasetP3[datasetP3['month'].isin(['
   2017-09-01', '2017-10-01', '2017-11-01'])].
   total_consumption.tolist()

errorP1 = mean_absolute_error(consumptionP1, predictionP1)
   /consumptionP1*100
errorP2 = mean_absolute_error(consumptionP2, predictionP2)
   /consumptionP2*100
errorP3 = mean_absolute_error(consumptionP3, predictionP3)
   /consumptionP3*100

print('P1')
print('MAE:', errorP1.mean())
print("sd(MAE):",errorP1.std())

print('P2')
print('MAE:', errorP2.mean())
print("sd(MAE):", errorP2.std())

print('P3')
print('MAE:', errorP3.mean())
print("sd(MAE):", errorP3.std())

print('Finished!')
```

A.2 Cas d'estudi II: Dades telemesurades

```
############## PACKAGES

from numpy import concatenate
from numpy import where
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import GRU
import datetime as dt
from datetime import timedelta
import pandas as pd

############## VARIABLES
subsystem = 'PENINSULA'
atr = '3.0A'
telemeasured = True
variables = ['datetime', 'total_cups_with_consumptions', '
   weighted_labor_days', 'temperature', 'relative_humidity'
   , 'temperature_max', 'temperature_min', '
   short_wave_radiation', 'wind_speed', 'windchill', '
   heat_index', 'demand', 'total_consumption']
n_days = 6
activation = 'tanh'

############## VALIDATION PERIODS
end_date_august = dt.date(2017, 8, 4)
start_date_august = end_date_august + timedelta(days=-180)
end_date_october = dt.date(2017, 10, 9)
start_date_october = end_date_october + timedelta(days
   =-180)
end_date_april = dt.date(2018, 4, 7)
start_date_april = end_date_april + timedelta(days=-180)

############## FORMATING DATES
start_date_august = start_date_august.strftime('%Y-%m-%d')
end_date_august = end_date_august.strftime('%Y-%m-%d')
start_date_october = start_date_october.strftime('%Y-%m-%d
   ')
end_date_october = end_date_october.strftime('%Y-%m-%d')
start_date_april = start_date_april.strftime('%Y-%m-%d')
end_date_april = end_date_april.strftime('%Y-%m-%d')
```

```
############ RETRIEVE DATA
path_to_script = '/home/nuria/Dropbox/TFG/rnn_TM_input.csv
    '
dataset = pd.read_csv(path_to_script)

############ DEALING WITH DATETIMES
dataset['datetime'] = pd.to_datetime(dataset['datetime'])
dataset = dataset.sort_values('datetime')
# Select used variables
dataset = dataset[variables]

############ MODEL DEFINITION
def RNN(train_X, train_y, test_X, test_y, scaler):
    # Design network
    model = Sequential()
    model.add(GRU(80, input_shape=(train_X.shape[1],
   train_X.shape[2]), activation=activation))
    model.add(Dense(50, activation=activation,
   kernel_initializer='glorot_normal'))
    model.add(Dense(1, kernel_initializer='glorot_normal')
   )
    model.compile(loss='mean_absolute_error', optimizer='
   adam', metrics=['accuracy'])

    # Fit model
    nEpochs = 100
    batchSize = 6
    model.fit(train_X, train_y, epochs=nEpochs, batch_size
   =batchSize, validation_data=(test_X, test_y),
                        verbose=2, shuffle=False)

    # Make a prediction
    yhat = model.predict(test_X)
    test_X = test_X.reshape((test_X.shape[0], test_X.shape
   [2]))

    # Invert scaling for forecast
    inv_yhat = concatenate((test_X, yhat), axis=1)
    inv_yhat = scaler.inverse_transform(inv_yhat)
    inv_yhat = inv_yhat[:, -1]

    return inv_yhat
```

```
############ MAIN FUNCTION
def launch_model(dataset, start_date, end_date):
    # Dealing with NaN's
    loc_NaN = where(dataset.isna()) # locating the NaN's
    dataset = dataset.drop(dataset.index[loc_NaN[0]]) #
   drop the rows which have at least one missing value

    # Index of data between the dates
    index_set = where((dataset['datetime'] >= start_date)
   & (dataset['datetime'] <= end_date))

    # Remove the information above and below the
   validation set
    dataset = dataset.drop(dataset.index[(index_set[0][-1]
    + 1):])
    dataset = dataset.drop(dataset.index[:(index_set
   [0][0])])

    # Index of validation split
    n_days = 6
    n_hours = 24 * n_days
    index_train = dataset.shape[0] - n_hours

    # Dealing with 'date' type
    dataset = dataset.set_index(['datetime'])

    # Normalizing values
    values = dataset.values
    values = values.astype('float32')
    scaler = StandardScaler()
    scaled = scaler.fit_transform(values)

    # Data splitting
    train_X = scaled[:index_train, :-1]
    test_X = scaled[index_train:, :-1]
    train_y = scaled[:index_train, -1]
    test_y = scaled[index_train:, -1]

    # Reshape input to be 3D [samples, timesteps, features
   ]
    train_X = train_X.reshape((train_X.shape[0], 1,
   train_X.shape[1]))
    test_X = test_X.reshape((test_X.shape[0], 1, test_X.
   shape[1]))
```

```
    prediction_mean = RNN(train_X, train_y, test_X, test_y
   , scaler)

    df = pd.DataFrame({'prediction' : prediction_mean, '
   total_consumption' : dataset[index_train:].
   total_consumption})

    return df

############ LET'S MAKE A PREDICTION!
print('starting august')
august = launch_model(dataset, start_date_august,
   end_date_august)
august['mae']=abs(august.total_consumption-august.
   prediction)/august.total_consumption*100
august['rel_error']=(august.total_consumption-august.
   prediction)/august.total_consumption*100

print("MAE:", august.mae.mean())
print("sd(MAE):", august.mae.std())
print("MRE:", august.rel_error.mean())

print('starting october')
october = launch_model(dataset, start_date_october,
   end_date_october)
october['mae']=abs(october.total_consumption-october.
   prediction)/october.total_consumption*100
october['rel_error']=(october.total_consumption-october.
   prediction)/october.total_consumption*100

print("MAE:", october.mae.mean())
print("sd(MAE):", october.mae.std())
print("MRE:", october.rel_error.mean())

print('starting april')
april = launch_model(dataset, start_date_april,
   end_date_april)
april['mae']=abs(april.total_consumption-april.prediction)
   /april.total_consumption*100
april['rel_error']=(april.total_consumption-april.
   prediction)/april.total_consumption*100

print("MAE:", april.mae.mean())
print("sd(MAE):", april.mae.std())
print("MRE:", april.rel_error.mean())
```

Referències

[1] Carter N. Brown. Go2carter GitHub. *nn-learn.* Gradients for a RNN. [consulta: 22 de juny del 2018] Disponible a: `https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf`.

[2] Chollet, F. *Keras* [Recurs electrònic], 2015. [consulta: 22 de juny del 2018]. Disponible a : `https://keras.io`

[3] Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. `arXiv:1412.3555`, 2014.

[4] Cybenko, G. Approximation by superpositions of a sigmoidal function. A:*Math. Control Signal Systems*, 1989, 2: 303. ISSN: 1435-568X.

[5] Goodfellow, I.; Bengio, Y.; Courville, A.. *Deep Learning.* MIT Press, 2016. ISBN: 9780262035613

[6] Greff, K.; Srivastava, R.K.; Koutník, J.; Steunebrink, B.R.; Schmidhuber, J. LSTM: A Search Space Odyssey. A: *Trans. Neural Netw. Learning Syst.*, 2017. Num 10 p. 222-2232.

[7] Hao, Z. *Isaac Changau* [Recurs electrònic] Activation Functions in Neural Networks . 2017-2018, [consulta: 22 de juny del 2018]. Disponible a: `https://isaacchanghau.github.io/post/activation_functions/`.

[8] Hochreiter, S.; Schmidhuber, J. Long short-term memory. A: *Neural computation*, 1997, p. 1735-1780.

[9] Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. A:*Neural Networks.* 2, p. 359-366, 1989.

[10] James, G.; Witten, D.; Hastie, T.; Tibshirani, R; Witten, D. Learning algorithms. A: *An introduction to statistical learning.* New York, 2013. Vol 112.

[11] Jozefowicz, R.; Zaremba, W.; Sutskever, I. An Empirical Exploration of Recurrent Network Architectures. A: *International Conference on Machine Learning*, 2015, p. 2343-2350.

[12] Kingma, Diederik P.; Jimmy Ba. Adam: A method for stochastic optimization. `arXiv:1412.6980`, 2014.

[13] LeCun, Y. ; Bengio, Y. ; Hinton, G. Deep Learning. *Nature*, 2015, 521, p. 436-44. 10.1038/nature14539.

[14] Lekshmi K.R.; Sherly, E. Automatic Speech Recognition using different Neural Network Architectures – A Survey. A: *International Journal of Computer Science and Information Technologies*, 2016, 7(6), p. 2422-2427.

[15] Merity, S. *Smerity* [Recurs electrònic]. Explaining and illustrating orthogonal initialization for recurrent neural networks. [consulta: 22 de juny del 2018]. Disponible a: `https://smerity.com/articles/2016/orthogonal_init.html`.

[16] Tom M. Mitchell. Concept learning and the General-to-Specific Ordering. A: *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997, p. 20-52. ISBN: 0070428077.

[17] Mohajerin, N.; Waslander, S.L. State initialization for recurrent neural network modeling of time-series data. A:*2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, p. 2330-2337.

[18] National Weather Services. *Environmental Modeling Center*. [Recurs electrònic]. Global Forecast System. [consulta: 22 de juny del 2018]. Disponible a: `http://www.emc.ncep.noaa.gov/index.php?branch=GFS`.

[19] Nielsen, M. *Neural Networks and Deep Learning* [Recurs electrònic]. Determination Press, 2015. [consulta: 22 de juny del 2018]. Disponible a: `http://neuralnetworksanddeeplearning.com/`.

[20] Olah, C. *Colah's Blog*, 2015 [Recurs electrònic]. Understanding LSTM Networks. [consulta: 22 de juny del 2018]. Disponible a: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[21] Pascanu, R; Mikolov, T; Bengio, Y. Understanding the exploding gradient problem. A:*CoRR*. `arxiv.org/abs/1211.5063`, 2012.

[22] Python Software Foundation. Python Language Reference, versió 3.6.5. Disponible a: `http://www.python.org`.

[23] Ruder, S. An overview of gradient descent optimization algorithms. A: *CoRR*, 2016. arXiv:1609.04747.

[24] Rumelhart, David E.; Hinton Geoffrey E. ; Williams, Ronald J. Learning representations by back-propagating errors. A: *Nature*, 1986, 323, p. 533–536. ISBN: 0-262-01097-6.

[25] Tibshirani, R. *Gradient Descent* [Recurs electrònic], 2015. [consulta: 22 de juny del 2018]. Disponible a: `http://www.stat.cmu.edu/ ryantibs/convexopt-S15/scribes/05-grad-descent-scribed.pdf`.

[26] Xarxa elèctrica espanyola. *Red Eléctrica de España* [Recurs electrònic]. [consulta: 22 de juny del 2018]. Disponible a: `http://www.ree.es/es/`.