



Projecte final

MÀSTER EN ENGINYERIA INFORMÀTICA

Oscar Galera i Alfaro

Mineria de dades

22 d'abril de 2018

Índex

1 Xarxes neuronals	3
1.1 Què és una xarxa neuronal?	3
1.2 Què és una neurona?	5
1.3 La funció d'activació	9
1.3.1 <i>Threshold function</i>	9
1.3.2 <i>Sigmoid</i>	9
1.3.3 <i>Rectifier function</i>	10
1.3.4 <i>Hyperbolic tangent</i>	10
1.4 Funcions d'activació sobre una xarxa neuronal	11
1.5 Com funcionen?	12
1.6 Com aprenen?	14
1.7 Descens del gradient	17
1.8 Descens del gradient estocàstic	21
1.9 <i>Backpropagation</i>	23
2 Classificació utilitzant una xarxa neuronal	25
2.1 Dades	25
2.2 Problema	27
2.3 Eines a utilitzar	28
2.4 Classificació	30
2.4.1 Processament de les dades	30
2.4.2 Creació de la xarxa neuronal	31
2.4.3 Classificació	33
2.4.4 Avaluar la precisió del model	34
A Codi	36
A.1 Python	36
A.2 R	39

Índex de figures

1	Xarxa neuronal	3
2	Anatomia d'una neurona real.	5
3	Parts d'una neurona.	6
4	Xarxa neuronal simple (Perceptró).	6
5	Pes (<i>weights</i>) de les connexions entre neurones.	7
6	Agregació de valors d'entrada (funció d'activació).	7
7	Integració d'una neurona en una xarxa neuronal.	8
8	<i>Threshold function.</i>	9
9	<i>Sigmoid function.</i>	10
10	<i>Rectifier function.</i>	10
11	<i>Hyperbolic tangent.</i>	11
12	Aplicació de funcions d'activació sobre una xarxa neuronal. .	11
13	Configuració d'una neurona.	13
14	Configuració completa d'una xarxa neuronal senzilla.	13
15	Funció de cost.	14
16	<i>Backpropagation.</i>	15
17	Etapa de càlcul.	15
18	Etapa d'aprenentatge.	16
19	Possibles connexions.	17
20	Sunway taihulight.	18
21	Xarxa neuronal complexa.	18
22	Descens del gradient.	19
23	Passos del descens del gradient.	20
24	Descens del gradient en diferents dimensions.	20
25	Òptim local.	21
26	Descens del gradient estocàstic.	22
27	<i>Forwardpropagation.</i>	23
28	<i>Backpropagation.</i>	23
29	Dades.	26
30	Entrenament de la xarxa.	33
31	Matriu de confusió.	34

1 Xarxes neuronals

En aquesta secció es descriuràn els components més importants que compo-nen una xarxa neuronal.

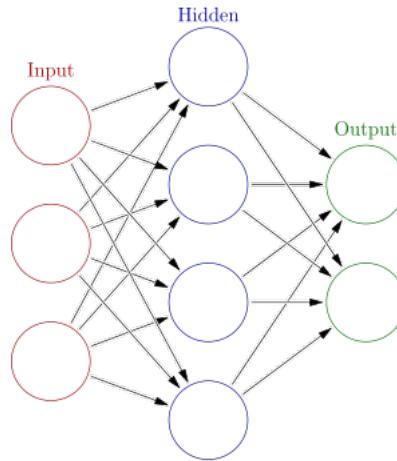


Figura 1: Xarxa neuronal

1.1 Què és una xarxa neuronal?

Una xarxa neuronal, és una estructura de dades dissenyada per a simular (en la mesura del possible) el comportament del cervell humà. El disseny d'aques-ta estructura es fonamenta en diferents capes de neurones interconnectades entre elles i que a través d'un procés iteratiu, són capaces d'ajustar-se i d'a-questa manera adquirir coneixement.

Una xarxa neuronal està composta per capes de neurones, on sempre hi ha una única capa de neurones d'entrada¹, n capes de neurones internes (el nombre de capes depèn de la complexitat del problema a resoldre) i una capa de neurones de sortida.

- **Neurones de la capa d'entrada:** emulen sensors que perceben la informació que es vol processar (variables explicatives).

¹Aquests neurones d'entrada representen les variables independents sobre les quals es vol fer l'anàlisi (variables explicatives).

- **Neurones de les capes internes:** són les neurones que adquireixen el coneixement.
- **Neurones de la capa de sortida:** són les neurones que proporcionen els resultats obtinguts (variables a explicar).

1.2 Què és una neurona?

Les neurones són els blocs bàsics en què es recolzen les xarxes neuronals. La seva funcionalitat treballant de forma individual no serveix de gaire, però si que serveix quan treballen de forma estructurada grans quantitats d'aquestes.

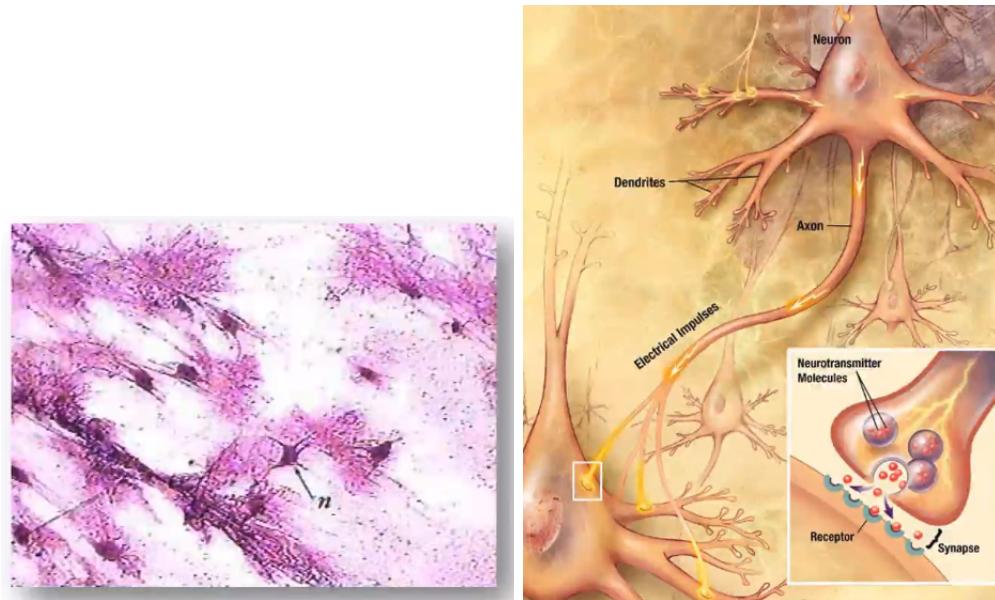


Figura 2: Anatomia d'una neurona real.

Una neurona es compon principalment de:

- Núcli.
- Dendrites (emissor).
- Axó (receptor).

La interconnexió de les neurones, es fa a través de l'enviament i recepció d'impulsos elèctrics (*synapses*).

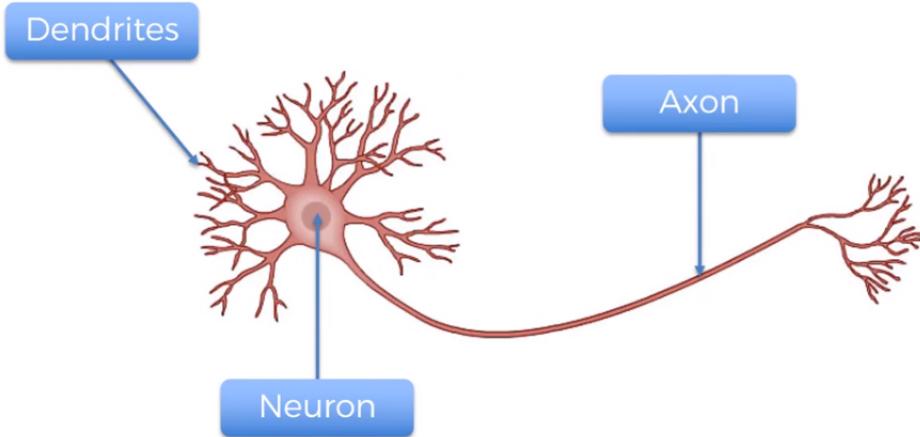


Figura 3: Parts d'una neurona.

Una xarxa neuronal es pot representar a través d'una caixa negra on hi ha un conjunt d'entrades (neurones d'entrada) i una o vàries sortides (neurones de sortida). Dins d'aquesta caixa, hi han diverses capes de neurones interconnectades que amaguen la complexitat de l'estructura.

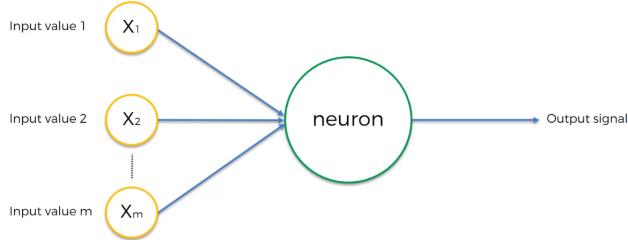


Figura 4: Xarxa neuronal simple (Perceptró).

Les neurones tenen la capacitat d'aprendre, això s'aconsegueix gràcies als pesos (*weights*) assignats a les connexions entre neurones, que determinen el grau d'importància que té cada una de les neurones de la capa anterior. Aquests valors s'actualitzen a través de la tècnica del ***backpropagation*** (secció 1.9) que utilitza el mètode matemàtic del ***descens del gradient*** (secció 1.7 i 1.8).

La configuració de la xarxa neuronal, determinarà el grau de qualitat de

la pròpia xarxa.

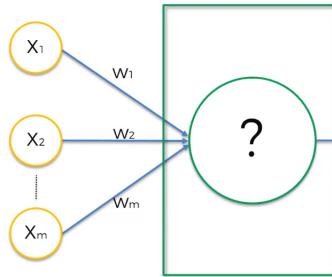


Figura 5: Pes (*weights*) de les connexions entre neurones.

A partir dels valors que rep una neurona de la resta, cal generar un valor que serà la sortida, per això s'ha de fer una agregació de valors a través d'una **funció d'activació** (secció 1.3) que determinarà el grau d'excitació d'aquesta.

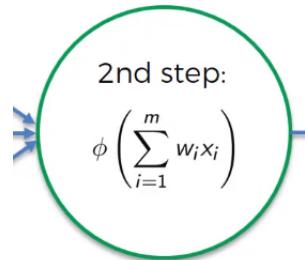


Figura 6: Agregació de valors d'entrada (funció d'activació).

Descrits els diferents components d'una neurona, la integració d'aquesta dins d'una xarxa es pot representar a través de la següent imatge.

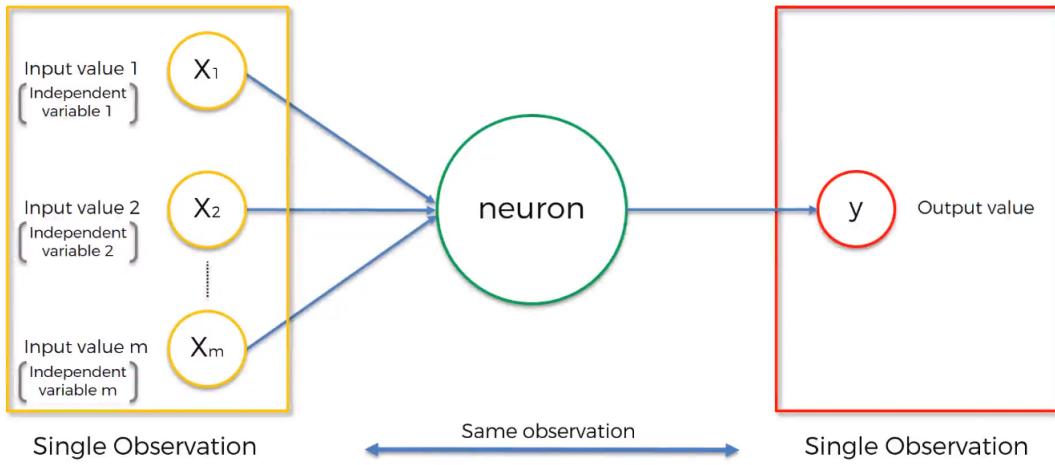


Figura 7: Integració d'una neurona en una xarxa neuronal.

S'ha de tenir en compte que el procés d'aprenentatge d'una xarxa neuronal és iteratiu, que tots els valors de les neurones d'entrada corresponen a una mateixa observació, i el valor de sortida al resultat d'aquesta observació.

1.3 La funció d'activació

Les funcions d'activació serveixen per calcular el valor que emetrà una neurona com a sortida. Les funcions d'activació més típics són:

- *Threshold function.*
- *Sigmoid.*
- *Rectifier function.*
- *Hyperbolic tangent (tanh).*

1.3.1 Threshold function

El rang de possibles valors per aquesta funció és 0 o 1, això fa que sigui una funció molt rígida i que s'adapti perfectament a casos on es vol una sortida binaria.

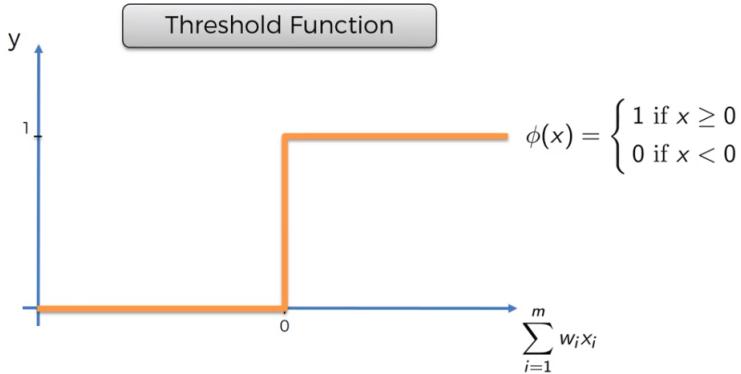


Figura 8: *Threshold function.*

1.3.2 Sigmoid

El rang de valors per aquesta funció va de $(0, 1)$ i es sol utilitzar molt com a funció d'activació en l'última capa de neurones, per calcular probabilitats.

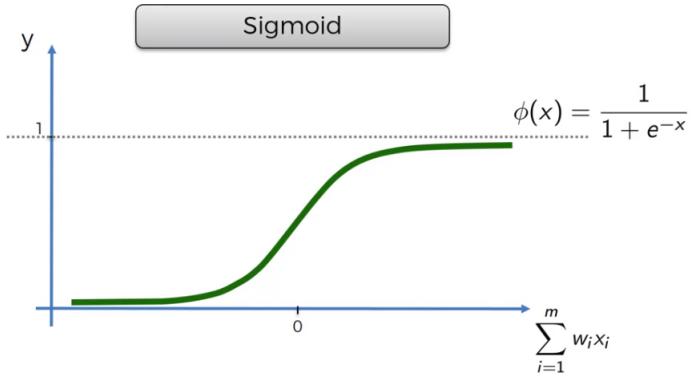


Figura 9: *Sigmoid function.*

És una funció que s'utilitza molt en la regressió logística.

1.3.3 Rectifier function

El rang d'aquesta funció va de $[0, \infty]$, on x correspon a la suma ponderada dels valors d'entrada de la capa de neurones anterior.

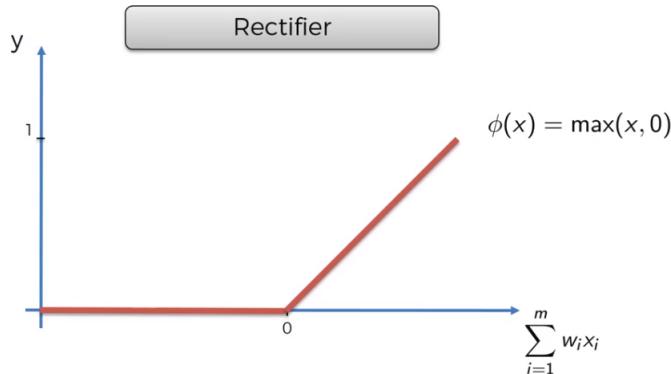


Figura 10: *Rectifier function.*

1.3.4 Hyperbolic tangent

El rang d'aquesta funció va de $(-1, 1)$ i és molt similar a la funció sigmoid.

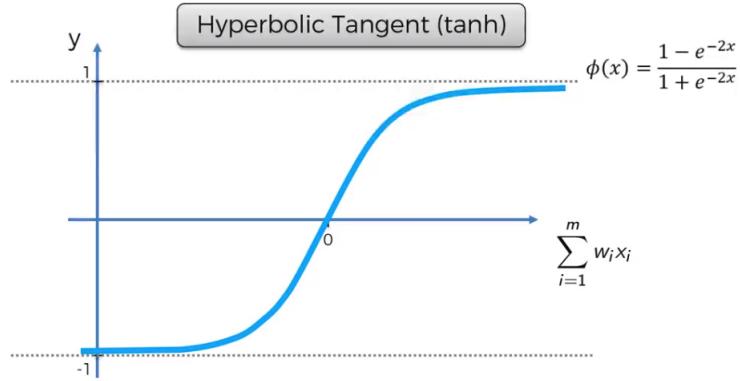


Figura 11: *Hyperbolic tangent*.

1.4 Funcions d'activació sobre una xarxa neuronal

Per a cada neurona interna, s'ha de definir una funció d'activació, per això es poden seguir diferents tècniques.

- Assignar una **mateixa funció d'activació** a tota la xarxa neuronal (rígid però senzill de desenvolupar i mantenir).
- Assignar una **mateixa funció d'activació** a nivell de capa de neurones.
- Assignar una **funció d'activació** **diferent** per a cada neurona (complex però potent).

La tècnica a utilitzar dependrà sempre del tipus de problema a resoldre, de la seva complexitat i de la potència disponible.

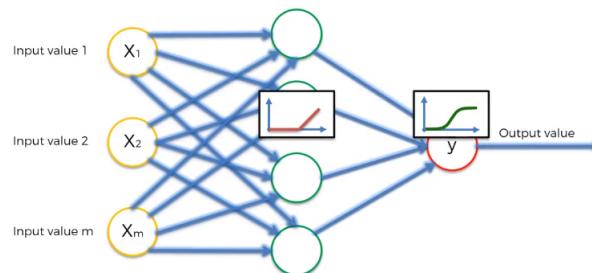


Figura 12: Aplicació de funcions d'activació sobre una xarxa neuronal.

1.5 Com funcionen?

El funcionament de les xarxes neuronals es basa en la configuració que s'ha de fer sobre les capes de neurones internes.

Per veure millor com funcionen, s'utilitzarà un exemple d'una possible aplicació real, en la que es disposa d'un conjunt dades en base a diferents habitatges que tenen els següents paràmetres:

- Dimensions.
- Nombre de lavabos.
- Distància del centre de la ciutat.
- Antiguitat.

Sobre aquestes dades, es vol **especular el preu** que pot tenir un habitatge en base a les seves propietats.

A nivell de neurona, s'ha d'elegir quins paràmetres actuen sobre el seu càcul i crear una connexió que enllaçi les neurones.

En aquest exemple, la primera neurona s'utilitza per influir en l'especulació del preu d'acord amb les mides de l'habitatge i la distància que està del centre de la ciutat.

De forma intuitiva es pot pensar que els habitatges més grans i que estan properes al centre de la ciutat són més cars, i per això, la primera neurona interna influirà en el preu en base aquests dos paràmetres.

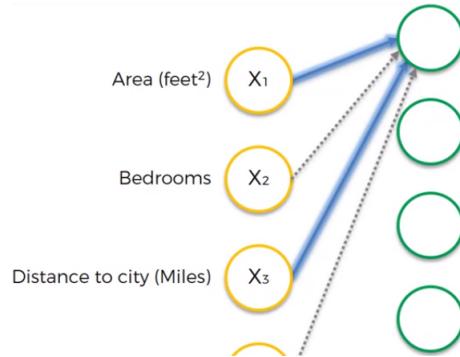


Figura 13: Configuració d'una neurona.

D'aquesta manera, una possible configuració completa d'una xarxa molt senzilla per calcular el preu pot ser com en la següent imatge.

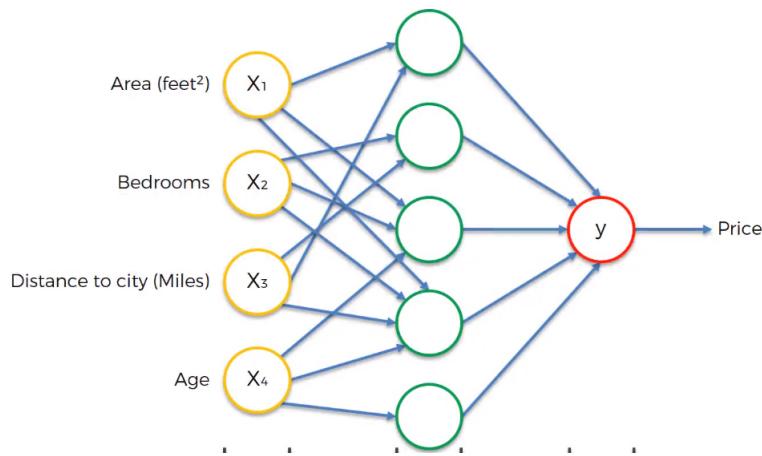


Figura 14: Configuració completa d'una xarxa neuronal senzilla.

Les neurones que no tenen impacte en el càlcul no es representen utilitzant connexions, això es pot interpretar com que tenen un impacte nul sobre aquell càlcul.

1.6 Com aprenen?

Com ja s'ha dit anteriorment, les xarxes neuronals es basen en un mecanisme d'aprenentatge iteratiu amb el qual van adquirint coneixement a mesura que guanyen experiència. Per aquest motiu, és necessari dividir les etapes de funcionament d'una xarxa neuronal en:

- **Etapa d'entrenament:** en aquesta etapa es proporcionen múltiples exemples a partir dels quals s'adquireix la gran majoria del coneixement. S'ha de tenir en compte, que durant aquesta etapa cal saber el resultat correcte per obtenir una xarxa ben entrenada.
- **Etapa de test:** en aquesta etapa es posa a prova la xarxa, proporcionant noves dades i analitzant els resultats. En cas de comptar amb el resultat correcte per aquestes dades, la xarxa pot continuar aprenent.

Durant l'etapa d'entrenament i un cop s'obté el resultat proporcionat de la xarxa, cal comparar-lo amb el resultat real i extreure un valor que representi la distància que hi ha entre el valor predit i el correcte. Aquesta magnitud s'estreu a partir de la funció de cost.

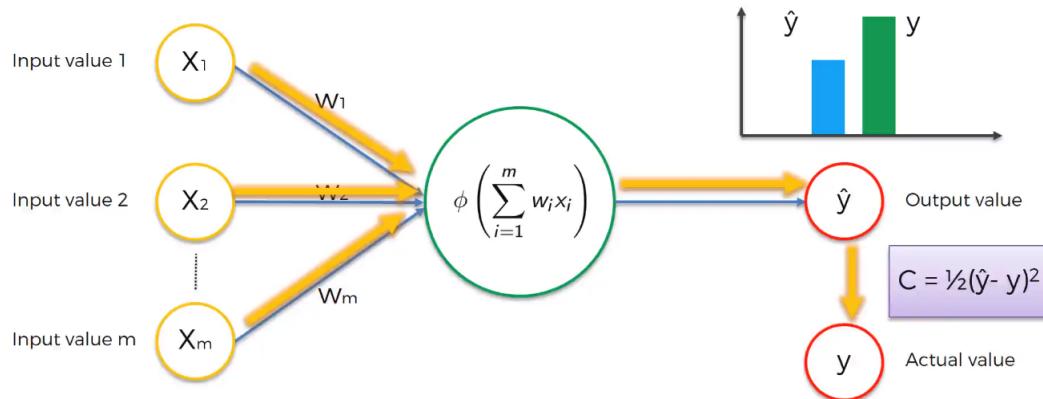


Figura 15: Funció de cost.

El valor proporcionat per la funció de cost, s'utilitza com entrada del mecanisme d'adaptació de la xarxa (*backpropagation*).

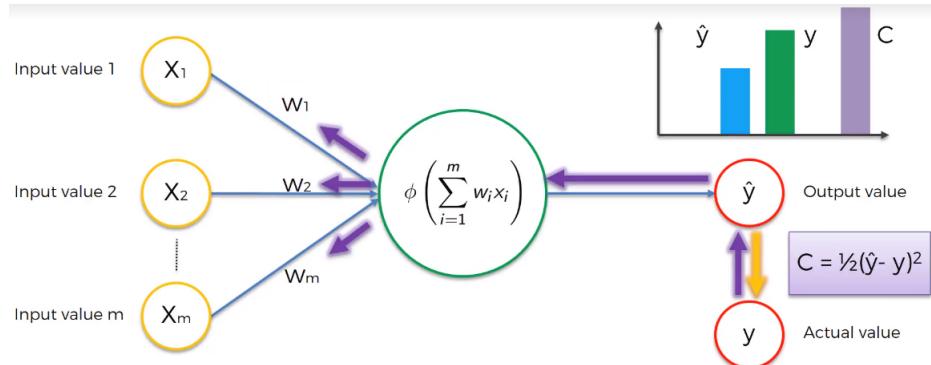


Figura 16: *Backpropagation*.

Un possible exemple, seria la utilització d'una xarxa neuronal per a desenvolupar una regressió que calculés una aproximació de la nota a treure en un examen d'acord amb:

- Nombre d'hores emprades a estudiar.
- Nombre d'hores dormides.
- Última nota obtinguda en el test.

L'etapa d'aprenentatge per aquest problema i donada una xarxa neuronal molt senzilla queda il·lustrada en la següent imatge.

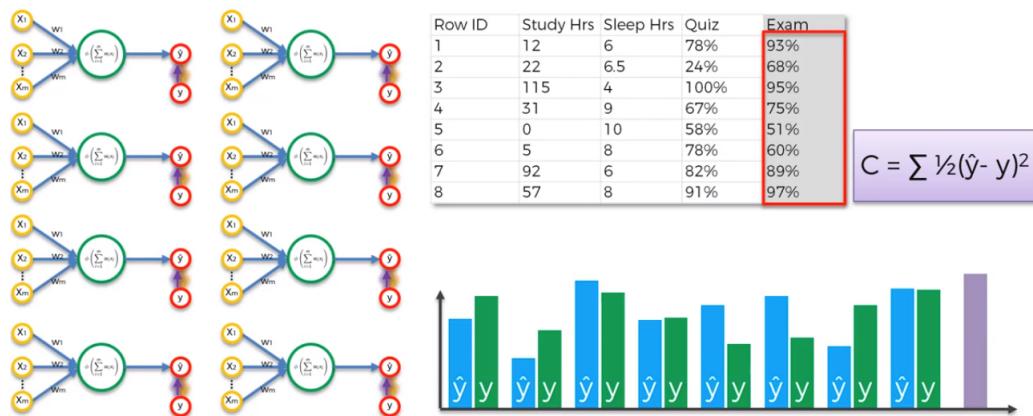


Figura 17: Etapa de càcul.

Un cop obtingut els resultats de la xarxa, s'ha d'aplicar la funció de cost que compara el valor predit amb el valor real i amb aquest resultat aplicar la tècnica de *backpropagation* com es mostra en la següent imatge.

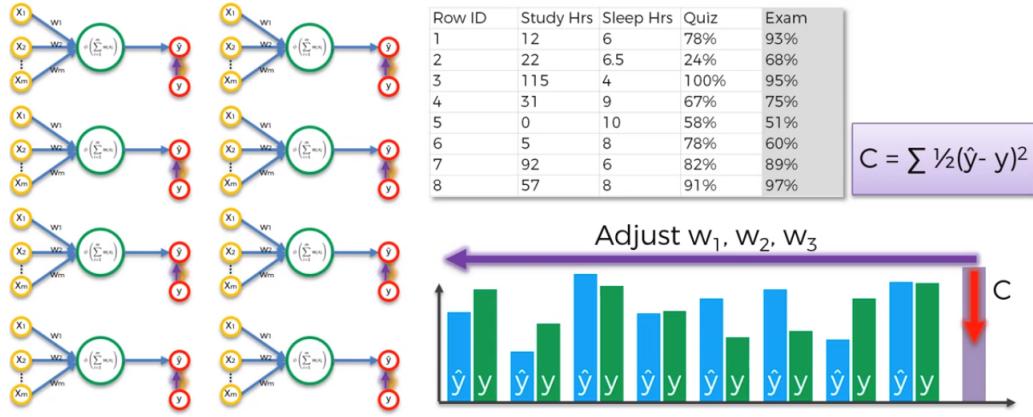


Figura 18: Etapa d'aprenentatge.

És important veure que en les figures 17 i 18 s'està utilitzant la mateixa xarxa neuronal però utilitzant diferents dades, i que la funció de cost per aplicar el *backpropagation* utilitza tots els valors disponibles en el conjunt d'entrenament. ²

²Una iteració sobre totes les dades d'entrenament es coneix com un *epoch*.

1.7 Descens del gradient

Per aconseguir que la xarxa neuronal aprengui, cal modificar els pesos de les *synapses* que connecten les diferents capes de neurones.

Donat una xarxa neuronal simple on hi ha quatre neurones en la capa d'entrada, cinc en la capa interna i una en la capa de sortida, tenim un total de vint-i-cinc possibles connexions.

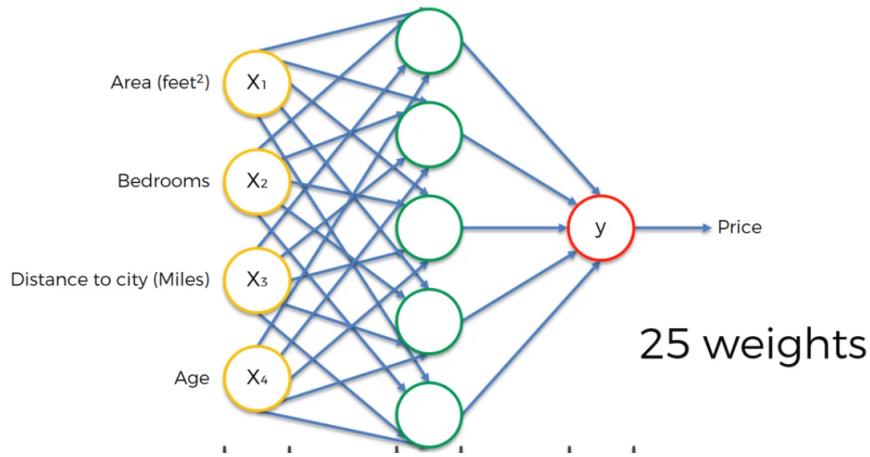


Figura 19: Possibles connexions.

Ara assumim que les variables d'entrada tenen un rang de possibles valors que va de $[1, 1000]$ i volem ajustar el coneixement modificant aleatoriament els vint-i-cinc possibles pesos, això fa que hi hagui 1000^{25} possibles combinacions. Tenint en compte que el **supercomputador més potent a nivell mundial *sunway taihulight*** té una potència de càcul d'aproximadament **93 PFLOPs**³.

³Pot arribar a executar aproximadament $9.3 \cdot 10^{15}$ operacions cada segon.



Figura 20: Sunway taihulight.

Caldrien $1.08 \cdot 10^{58}$ segons, és a dir $3.42 \cdot 10^{50}$ anys per configurar la xarxa.

$$\frac{1000^{25}}{93 \cdot 10^{15}} \approx 1.08 \cdot 10^{58} \text{segons} \approx 3.42 \cdot 10^{50} \text{any}s$$

Tot això, tenint en compte que es pot fer una assignació de pesos per operació, cosa que no pot ser com ja s'ha vist abans.

I si tenim una xarxa més complexa???

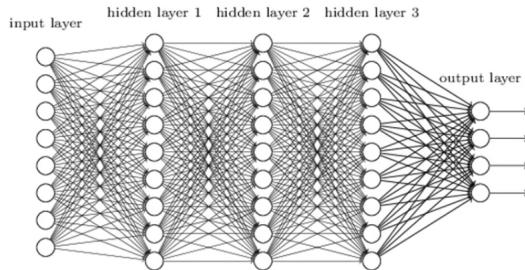


Figura 21: Xarxa neuronal complexa.

Vist que utilitzar força bruta no serveix en aquest problema, cal utilitzar un altre mètode que en aquest cas és el **descens del gradient**.

El descens del gradient és el mètode matemàtic en què es basa la tècnica *backpropagation* per aconseguir absorbir el coneixement adquirit durant la fase d'entrenament.

Aquesta tècnica radica en buscar el camí més curt per reduir la funció de cost, això es fa a través de la derivada de la funció de cost que determina el pendent en un punt donat.

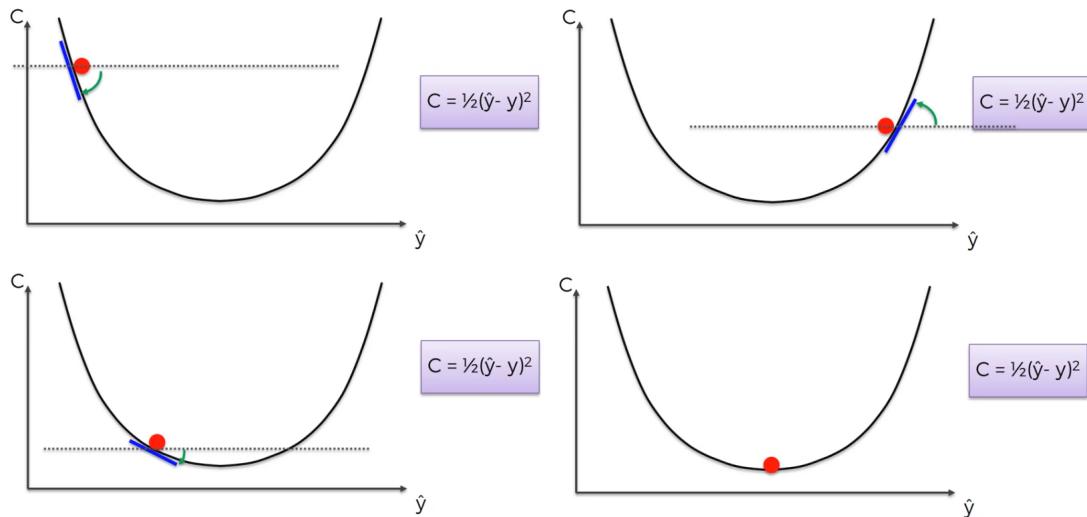


Figura 22: Descens del gradient.

Així doncs, utilitzant la tècnica del descens del gradient i per aquesta funció de cost, podem trobar l'òptim en pocs passos.

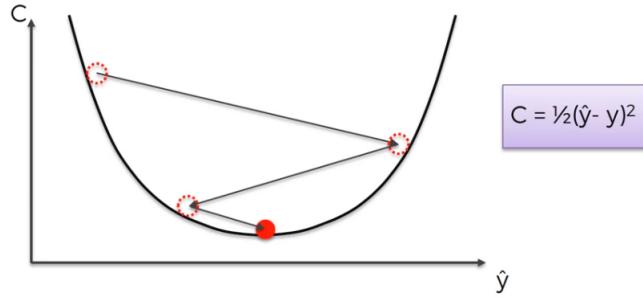


Figura 23: Passos del descens del gradient.

Les següents imatges mostren els passos del descens del gradient tenint en compte diferents dimensions.

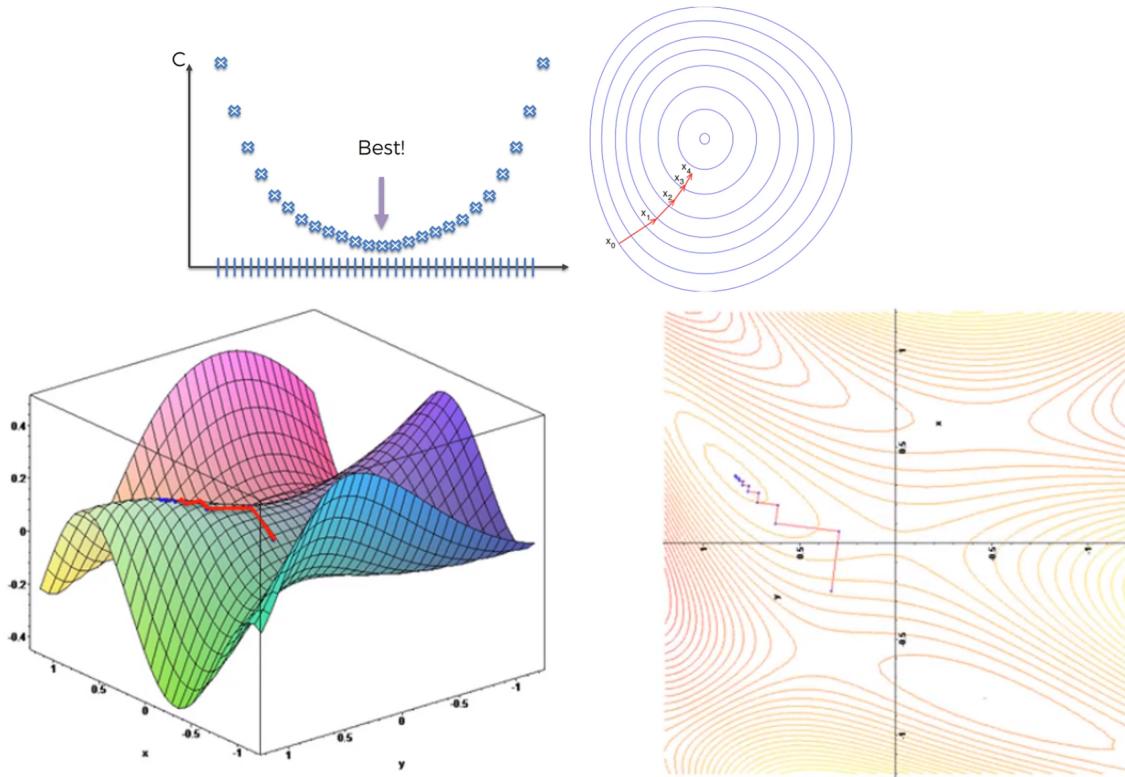


Figura 24: Descens del gradient en diferents dimensions.

1.8 Descens del gradient estocàstic

El típic problema que es pot trobar a l' hora d'utilitzar el mètode del descens del gradient⁴, és el de no arribar a trobar l'òptim global, a causa de que s'ha "caigut" a un òptim local. Això vol dir que no s'arriba a entrenar la xarxa tant com es voldria. Per resoldre aquest problema, es pot utilitzar el mètode del **descens del gradient estocàstic**.

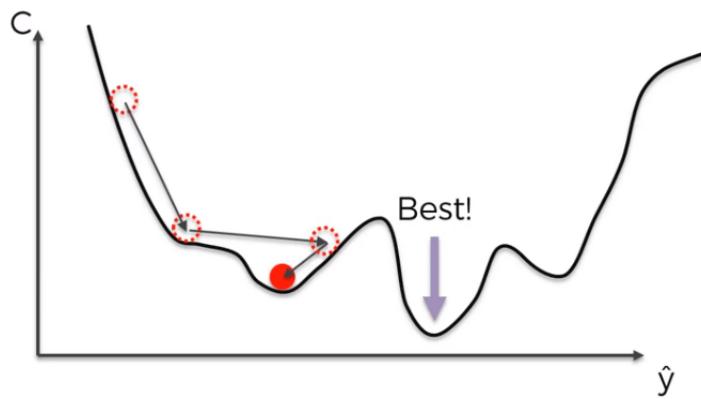


Figura 25: Òptim local.

El descens del gradient estocàstic s'aconsegueix aplicant la funció de *backpropagation* per cada registre processat per la xarxa, d'aquesta manera s'obté una major fluctuació i així hi ha més probabilitat de no "caure" en un òptim local.

⁴Notar que aquest problema no es tindrà si la funció és convexa.

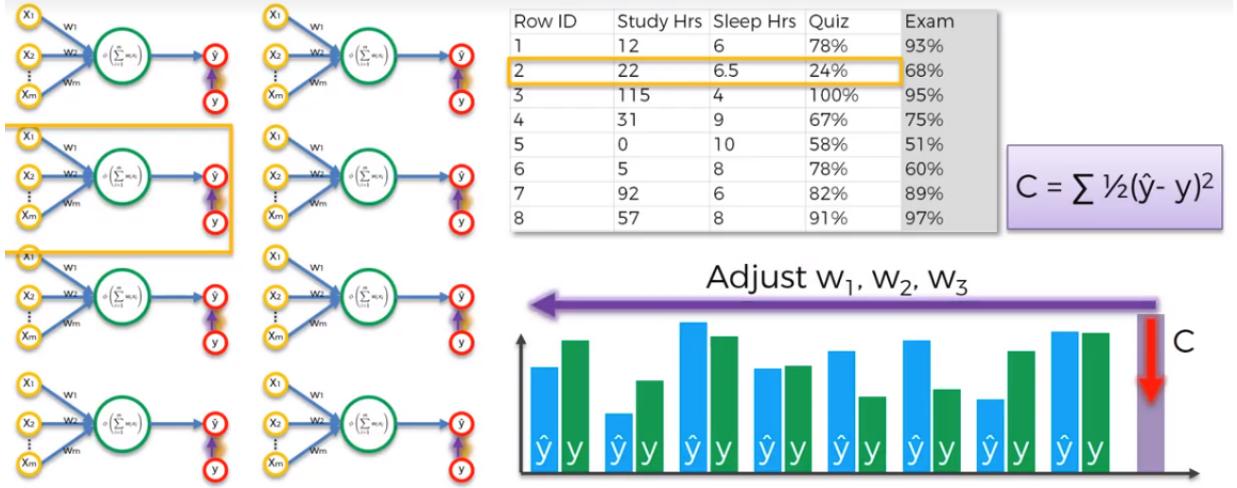


Figura 26: Descens del gradient estocàstic.

És important veure que **aquest mètode és indeterminista**, ja que el resultat dependrà de l'ordre en què es processin els registres.

1.9 Backpropagation

El *Backpropagation* és un algorisme complex a nivell matemàtic que permet ajustar tots els pesos a l'hora, és a dir, permet identificar quins són els pesos que estan causant més error i en quina mesura s'han de modificar.

Durant l'entrenament d'una xarxa neuronal, es poden destacar dues fases.

- La fase de *Forwardpropagation* on entren valors per les neurones d'entrada i es genera un resultat.

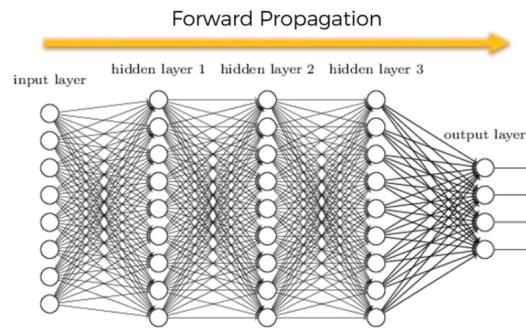


Figura 27: *Forwardpropagation*.

- La fase de *Backpropagation* on es calcula l'error comès a partir del resultat real, i s'intenten ajustar els pesos de les connexions.

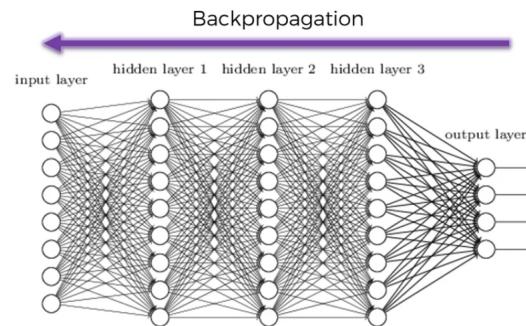


Figura 28: *Backpropagation*.

Els passos que es segueixen durant l'entrenament d'una xarxa són:

1. Iniciar aleatoriament (amb un valor pròxim a 0) els pesos de les connexions.
2. Assignar un valor (de la mateixa observació) a cada una de les neurones de la capa d'entrada.
3. *Forwardpropagation* activant les neurones.
4. Calcular l'error generat a través de la funció de cost.
5. *Backpropagation*, corregint l'error comès.
6. Repetir passos 1-5 fins a arribar a processar totes les observacions.
7. S'ha completat una *epoch*, fer més *epochs* repetint passos 1-6.

2 Classificació utilitzant una xarxa neuronal

2.1 Dades

Les dades amb les quals es treballarà (`Churn_Modelling.csv`), corresponen a 10.000 clients d'un banc (fictici) amb els següents camps:

- **RowNumber:** identificador de registre [1-10.000].
- **CustomerId:** identificador univoc del client.
- **Surname:** Cognom del client.
- **CreditScore:** Nombre de punts.
- **Geography:** País d'origen.
- **Gender:** Sexe.
- **Age:** Edat.
- **Tenure:** Nombre d'anys d'ençà que és client.
- **Balance:** Ingressos fets respecte a les despeses.
- **NumOfProduct:** Nombre de productes del banc que ha adquirit.
- **HasCrCard:** Si té o no targeta de crèdit.
- **IsActiveMember:** Si utilitza algun dels productes amb regularitat.
- **EstimateSalary:** Salari anual estimat.
- **Exited:** Si es preveu que el client abandonarà o no el banc.

En la següent imatge es pot veure un extracte dels 10 primers registres del *dataset*.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	RowNumber	CustomerID	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
2	1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101348.88	1
3	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
4	3	15619304	Ohio	502	France	Female	42	8	159660.8	3	1	0	113931.57	1
5	4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.63	0
6	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	179084.1	0
7	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
8	7	15592531	Bartlett	822	France	Male	50	7	0	2	1	1	10062.8	0
9	8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	0	119346.88	1
10	9	15792365	He	501	France	Male	44	4	142051.07	2	0	1	74940.5	0

Figura 29: Dades.

2.2 Problema

A partir de les dades descrites en la secció 2.1, volem arribar a respondre la següent pregunta mitjançant classificació utilitzant una xarxa neuronal.

Es preveu que en un període de 6 mesos, el client abandoni el banc?

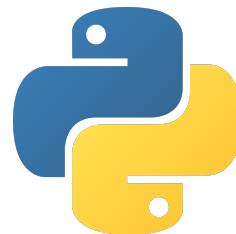
La resposta correcte a aquesta pregunta correspon al valor de l'últim camp *exited*, on:

- 0: El client **no** abandona el banc.
- 1: El client **si** abandona el banc.

2.3 Eines a utilitzar

Per desenvolupar aquest projecte s'utilitzaran les següents eines.

- *Python*: Com a llenguatge de programació (a nivell d'exemple), s'ha decidit utilitzar python perquè és un dels llenguatges més utilitzats en aquest àmbit. El codi complet es pot consultar en l'annex A.1.



- *R*: També s'ha desenvolupat la xarxa neuronal utilitzant el llenguatge de programació R (per completitud del projecte), el codi es pot consultar en l'annex A.2.



- *Theano*: Llibreria de computació numèrica de codi obert, permet efectuar càlculs no només sobre la *CPU*, també sobre la *GPU*.



Instal·lació: `\$pip install --upgrade --no-deps git+git://github.com/Theano/Theano.git`

- *Tensorflow*: Llibreria de computació numèrica de codi obert, inicialment desenvolupada pel *Google brain team* i actualment alliberada “sota” la llicència *Apache license 2.0*.



Instal·lació: https://www.tensorflow.org/versions/r0.12/get_started/os_setup.html

- *Keras*: Llibreria que serveix d'envoltori per les llibreries *Theano* i *Tensorflow*, permetent crear xarxes neuronals amb relativament poques línies de codi.



Instal·lació: `\$pip install --upgrade keras`

S'ha de tenir en compte que les llibreries *Theano* i *Tensorflow* estan orientades al món de la investigació i el desenvolupament professional, i per tant, si s'utilitzen s'ha de desenvolupar la xarxa neuronal des de 0 (la qual cosa suposa un gran esforç i un alt nivell de coneixement sobre la matèria).

2.4 Classificació

Els passos a seguir per aplicar la classificació utilitzant una xarxa neuronal són.

1. Preprocessament de les dades.
2. Creació de la xarxa neuronal.
3. Classificació.
4. Avaluar la precisió del model.

2.4.1 Processament de les dades

En aquest apartat es mostra el codi necessari per efectuar el processament de dades que s'ha de fer abans d'executar la xarxa neuronal.

```
1 # Part 1 – Processament de dades
2
3 # Llibreries
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import pandas as pd
7
8 # Importar dades
9 dataset = pd.read_csv('Churn_Modelling.csv')
10
11 # Seleccio de les variables independents (VI) que influeixen
12 # en la decissió d'abandonar o no el banc.
13 #
14 # Les variables que s'exclouen son:
15 # - RowNumber [0]
16 # - CustomerId [1]
17 # - Surname [2]
18 X = dataset.iloc[:, 3:13].values
19 # Seleccionem la variable dependent
20 y = dataset.iloc[:, 13].values
21
22 # Codificar les variables categoriques
23 from sklearn.preprocessing import LabelEncoder, OneHotEncoder
24 codificador_X_1 = LabelEncoder()
25 X[:, 1] = codificador_X_1.fit_transform(X[:, 1])
26 codificador_X_2 = LabelEncoder()
27 X[:, 2] = codificador_X_2.fit_transform(X[:, 2])
```

```

28
29 # Es crean variables dummy perque no hi ha ordre entre les
   variables
30 # categoriques
31 codificador = OneHotEncoder(categorical_features = [1])
32 X = codificador.fit_transform(X).toarray()
33 # S'exclou la primera variable [0] dummy per eviat la (dummy
   variable trap)
34 X = X[:, 1:]
35
36 # Partir les dades en:
37 # - training set: dades per efectuar l'entrenament de la xarxa
   (8000)
38 # - test set: dades per comprovar l'efectivitat de la xarxa
   (2000)
39 from sklearn.model_selection import train_test_split
40 X_train, X_test, y_train, y_test = train_test_split(X, y,
   test_size = 0.2, random_state = 0)
41
42 # Fer feature scaling per evitar que una (o mes) variable
   independent domini la resta.
43 from sklearn.preprocessing import StandardScaler
44 sc = StandardScaler()
45 X_train = sc.fit_transform(X_train)
46 X_test = sc.transform(X_test)

```

2.4.2 Creació de la xarxa neuronal

En aquest apartat es mostra el codi necessari per efectuar la creació de la xarxa neuronal.

Aquesta xarxa neuronal tindrà tres capes.

- Primera capa “capa d’entrada”: amb **onze neurones** (una per cada variable explicativa).
- Segona capa “capa interna”: amb **sis neurones** (una bona pràctica a seguir, correspon a utilitzar un nombre de neurones igual a la mitja del nombre de neurones de la capa d’entrada i la capa de sortida).
- Tercera capa “capa de sortida”: amb **una única neurona**, perquè volem resoldre un problema de classificació binaria.

```

1 # Part 2 – Construir la xarxa neuronal
2
3 # Importing la llibreria Keras
4 import keras
5 from keras.models import Sequential #iniciar la xarxa
6 from keras.layers import Dense #constuir les capes de la xarxa
7
8 # Iniciar la xarxa
9 xarxa = Sequential()
10
11 # Afegir la capa d'entrada.
12 # - output_dim: nombre de sortides cap a la capa interna de
13 #   neurones (nombre de neurones
14 # que tindra la capa interna , una bona aproximacio es (
15 #   nNeuronesCapaEntrada+nNeuronesCapaSortida)/2.
16 # - init: com iniciar els pesos de les synapses.
17 # - activation: quina funcio d'activacio s'utilitza , en aquest
18 #   cas , s'utilitzara
19 # una funcio de rectificiacio en totes les capes i una funcio
20 #   sigmoide en la capa de sortida.
21 xarxa.add(Dense(output_dim = 6, init = 'uniform', activation =
22 #   'relu', input_dim = 11))
23
24 # Afegir la capa interna.
25 xarxa.add(Dense(output_dim = 6, init = 'uniform', activation =
26 #   'relu'))
27
28 # Afegir la capa de sortida , aquesta utilitza una sigmoide com a
29 #   funcio d'activacio i
30 # com volem obtenir un resultat binari (abandonara o no el banc)
31 #   nomes cal una
32 # neurona que produeixi 0 si no abandonara o 1 si abandonara.
33 xarxa.add(Dense(output_dim = 1, init = 'uniform', activation =
34 #   'sigmoid'))
35
36 # Compilar la xarxa neuronal.
37 # - optimiser: nom de l'algorisme per trobar els pessos optims
38 #   en les synapses , s'utilitza
39 # el descens del gradient estocastic , i una implementacio molt
40 #   efficient d'aquest algorisme
41 # es diu adam.
42 # - loss: Funcio de cost del descens del gradient.
43 # - metrics: Per dir si la xarxa es bona o no es te en compte la
44 #   precissio dels resultats.
45 xarxa.compile(optimizer = 'adam', loss = 'binary_crossentropy',

```

```

metrics = [ 'accuracy' ])

# Entrenament de la xarxa.
# S'APLICA UNA TECNICA DE MICROBATCH (BATCH DE MIDA 10).
# - X_train: variables independents (explicativa).
# - y_train: variables dependents (explicada).
# - batch_size: mida de cada batch a partir del qual s'aplica el
#   backpropagation.
# - nb_epoch: nombre de epochs a processar abans d'acabar d'
#   entrenar la xarxa.

xarxa.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)

```

```

Epoch 32/100
8000/8000 [=====] - 0s - loss: 0.3453 - acc: 0.8592
Epoch 33/100
8000/8000 [=====] - 0s - loss: 0.3446 - acc: 0.8589
Epoch 34/100
8000/8000 [=====] - 0s - loss: 0.3443 - acc: 0.8575
Epoch 35/100
8000/8000 [=====] - 0s - loss: 0.3437 - acc: 0.8599
Epoch 36/100
8000/8000 [=====] - 0s - loss: 0.3433 - acc: 0.8577
Epoch 37/100
8000/8000 [=====] - 0s - loss: 0.3421 - acc: 0.8599
Epoch 38/100
1970/8000 [=====>.....] - ETA: 0s - loss: 0.3333 - acc:
0.8609

```

Figura 30: Entrenament de la xarxa.

És important notar que al final de l'entrenament de la xarxa (en l'últim *epoch*) hi ha hagut una **precisió del 86%**.

2.4.3 Classificació

En aquest apartat es mostra el codi necessari per efectuar la classificació de les dades de test.

```

# Part 3 – Efectura les prediccions.

# Prediccions.
y_pred = xarxa.predict(X_test)
# si la predicció d'abandonament es superior a 0.5 interpretem
# que el client acabara abandonant el banc
y_pred = (y_pred > 0.5)

```

2.4.4 Avaluar la precisió del model

En aquest apartat final, es mostra l'efectivitat de la classificació que ha fet la xarxa, comparant les prediccions amb els valors reals.

```
1 #Part 4 – Comprobar els resultats.  
2 from sklearn.metrics import confusion_matrix  
3 #Per comprovar els resultats utilitzem una matriu de confusió  
4 cm = confusion_matrix(y_test, y_pred)
```

S'ha obtingut una **precisió del 86% aproximadament**. En la següent matriu de confusió es pot veure la relació de resultats.

		0	1
0	1512	83	
1	196	209	

Figura 31: Matriu de confusió.

Referències

- [1] *Curs de machine learning:*
<https://www.udemy.com/machinelearning/learn/v4/overview>
- [2] *Xarxa neuronal:*
https://en.wikipedia.org/wiki/Artificial_neural_network
- [3] *Deep sparse rectifier neural networks:*
<http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf>
- [4] *Funcions de cost més tipiques:*
<https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications>
- [5] *Top 500 supercomputers:*
<https://www.top500.org/system/178764>
- [6] *Descendent del gradient:*
<http://iamtrask.github.io/2015/07/27/python-network-part2/>
- [7] *Descendent del gradient i Backpropagation:*
<http://neuralnetworksanddeeplearning.com/chap2.html>

A Codi

A.1 Python

```
1 # XARXA NEURONAL
2
3 # Part 1 – Processament de dades
4
5 # Llibreries
6 import numpy as np
7 import matplotlib.pyplot as plt
8 import pandas as pd
9
10 # Importar dades
11 dataset = pd.read_csv('Churn_Modelling.csv')
12
13 # Seleccio de les variables independents (VI) que influeixen
14 # en la decissio d'abandonar o no el banc.
15 #
16 # Les variables que s'exclouen son:
17 # - RowNumber [0]
18 # - CustomerId [1]
19 # - Surname [2]
20 X = dataset.iloc[:, 3:13].values
21 # Seleccionem la variable dependent
22 y = dataset.iloc[:, 13].values
23
24 # Codificar les variables categoriques
25 from sklearn.preprocessing import LabelEncoder, OneHotEncoder
26 codificador_X_1 = LabelEncoder()
27 X[:, 1] = codificador_X_1.fit_transform(X[:, 1])
28 codificador_X_2 = LabelEncoder()
29 X[:, 2] = codificador_X_2.fit_transform(X[:, 2])
30
31 # Es crean variables dummy perque no hi ha ordre entre les
32 # variables
33 # categoriques
34 codificador = OneHotEncoder(categorical_features = [1])
35 X = codificador.fit_transform(X).toarray()
36 # S'exclou la primera variable [0] dummy per eviat la (dummy
37 # variable trap)
38 X = X[:, 1:]
39
40 # Partir les dades en:
41 # - training set: dades per efectuar l'entrenament de la xarxa
```

```

(8000)
40 # - test set: dades per comprovar l'efectivitat de la xarxa
    (2000)
41 from sklearn.model_selection import train_test_split
42 X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.2, random_state = 0)
43
44 # Fer feature scaling per evitar que una (o mes) variable
    independent domini la resta.
45 from sklearn.preprocessing import StandardScaler
46 sc = StandardScaler()
47 X_train = sc.fit_transform(X_train)
48 X_test = sc.transform(X_test)
49
50 # Part 2 - Construir la xarxa neuronal
51
52 # Importing la llibreria Keras
53 import keras
54 from keras.models import Sequential #iniciar la xarxa
55 from keras.layers import Dense #constuir les capes de la xarxa
56
57 # Iniciar la xarxa
58 xarxa = Sequential()
59
60 # Afegir la capa d'entrada.
61 # - output_dim: nombre de sortides cap a la capa interna de
    neurones (nombre de neurones
62 # que tindra la capa interna, una bona aproximacio es (
    nNeuronesCapaEntrada+nNeuronesCapaSortida)/2.
63 # - init: com iniciar els pesos de les synapses.
64 # - activation: quina funcio d'activacio s'utilitza , en aquest
    cas, s'utilitzara
65 # una funcio de rectificiacio en totes les capes i una funcio
    sigmoide en la capa de sortida.
66 xarxa.add(Dense(output_dim = 6, init = 'uniform', activation =
    'relu', input_dim = 11))
67
68 # Afegir la capa interna.
69 xarxa.add(Dense(output_dim = 6, init = 'uniform', activation =
    'relu'))
70
71 # Afegir la capa de sortida, aquesta utilitza una sigmoide com a
    funcio d'activacio i
72 # com volem obtenir un resultat binari (abandonara o no el banc)
    nomes cal una

```

```

73 # neurona que produeixi 0 si no abandonara o 1 si abandonara.
74 xarxa.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
75
76 # Compilar la xarxa neuronal.
77 # - optimiser: nom de l'algorisme per trobar els pessos optims
    en les synapses, s'utilitza
78 # el descens del gradient estocastic, i una implementació molt
    eficient d'aquest algorisme
79 # es diu adam.
80 # - loss: Funció de cost del descens del gradient.
81 # - metrics: Per dir si la xarxa es bona o no es té en compte la
    precisió dels resultats.
82 xarxa.compile(optimizer = 'adam', loss = 'binary_crossentropy',
    metrics = ['accuracy'])
83
84 # Entrenament de la xarxa.
85 # S'APLICA UNA TECNICA DE MICROBATCH (BATCH DE MIDA 10).
86 # - X_train: variables independents (explicativa).
87 # - y_train: variables dependents (explicada).
88 # - batch_size: mida de cada batch a partir del qual s'aplica el
    backpropagation.
89 # - nb_epoch: nombre de epochs a processar abans d'acabar d'
    entrenar la xarxa.
90 xarxa.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)
91
92 # Part 3 - Efectura les prediccions.
93
94 # Prediccions.
95 y_pred = xarxa.predict(X_test)
96 # si la predicció d'abandonament és superior a 0.5 interpretarem
    que el client acaba abandonant el banc
97 y_pred = (y_pred > 0.5)
98
99 #Part 4 - Comprobar els resultats.
100 from sklearn.metrics import confusion_matrix
101 #Per comprovar els resultats utilitzem una matriu de confusió
102 cm = confusion_matrix(y_test, y_pred)

```

A.2 R

```
1 # Xarxa neuronal
2
3 # Importar les dades
4 dataset = read.csv('Churn_Modelling.csv')
5 dataset = dataset[4:14]
6
7 # Codificar les variables categoriques
8 dataset$Geography = as.numeric(factor(dataset$Geography,
9                                     levels = c('France', ,
10                                         'Spain', 'Germany'),
11                                         labels = c(1, 2, 3)))
12 dataset$Gender = as.numeric(factor(dataset$Gender,
13                               levels = c('Female', 'Male'),
14                               labels = c(1, 2)))
15
16 # Dividir les dades en entrenament i test.
17 # install.packages('caTools')
18 library(caTools)
19 set.seed(123)
20 split = sample.split(dataset$Exited, SplitRatio = 0.8)
21 training_set = subset(dataset, split == TRUE)
22 test_set = subset(dataset, split == FALSE)
23
24 # Escalar factors per evitar que alguna variable independent
25 # domini a la resta.
26 training_set[-11] = scale(training_set[-11])
27 test_set[-11] = scale(test_set[-11])
28
29 # Creacio de la xarxa
30 # install.packages('h2o')
31 library(h2o)
32 h2o.init(nthreads = -1)
33 model = h2o.deeplearning(y = 'Exited',
34                           training_frame = as.h2o(training_set),
35                           activation = 'Rectifier',
36                           hidden = c(5,5),
37                           epochs = 100,
38                           train_samples_per_iteration = -2)
39
40 # Predir els resultats.
41 y_pred = h2o.predict(model, newdata = as.h2o(test_set[-11]))
42 y_pred = (y_pred > 0.5)
43 y_pred = as.vector(y_pred)
```

```
43
44 # Matriu de confusio
45 cm = table(test_set[, 11], y_pred)
46
47 h2o.shutdown()
```