1 - titolo del progetto: Page Blocks Classification with Neural Networks

2 – elenco degli autori con contributo percentuale: Lo Presti Alessandro 50%, Longo Valerio 50%

3 – classe di problema affrontato: multi-class supervised classification

4 – origine del dataset: https://archive.ics.uci.edu/ml/datasets/Page+Blocks+Classification

5 – descrizione del vettore delle features:

1 - altezza: intero (altezza del blocco)
2 - lunghezza: intero (lunghezza del blocco)
3 - area: intero (lunghezza*altezza)
4 - eccentricità: reale (lunghezza/altezza)
5 - p_black: reale (percentuale di pixel neri all'interno di un blocco)
6 - p_and: reale (percentuale di pixel neri dopo l'applicazione dell'algoritmo RLSA)
7 - media_trans: reale (media dei numeri di transizione da bianco a nero blackpix/wb_trans)
8 - blackpix: intero (numero totale di pixel neri)
9 - blackand: intero (numero totale di pixel dopo l'applicazione dell'algoritmo RLSA)
10 - wb_trans: intero (numero totale di transizioni dal bianco al nero)

6 – desxrizione del vettore degli output:

1 – testo
2 – linea orizzontale
3 – grafica
4 – linea verticale
5 – immagine

7 – descrizione sintetica del problema: applicazione di una rete neurale ad un prolema di classificazione supervisionata multiclasse. Questo progetto prevede l'analisi della rete neurale applicata e il comportamento della stessa al variare dei parametri (numero neuroni, funzione di attivazione, funzione di allenamento) con conseguente spiegazione e analisi dei risultati.

8 – tipo di modello: reti neurali

9 – tipo di algoritmo applicato: backpropagation

10 – tipo di stima: cross-entropy

11 – metodo di validazione: training-set 70%, validation-set 15%, test-set 15%

12 – descrizione sintetica dei risultati: al diminuire del numero di neuroni si verifica un underfitting viceversa aumentandoli in maniera significativa si verifica un overfitting. Le prestazioni della rete sono ottimali utilizzando le impostazioni di default (log-sigmoide, Gradient Descent), ma applicando delle ottimizzazioni nella rete otteniamo dei risultati ancora più soddisfacenti.

13 – linguaggio di programmazione utilizzato: matlab

14 – libreria o package utilizzato: Neural Network Toolbox

Il link attuale del repository GitLab: https://gitlab.com/sapienza/MQI-Project/

APPLICAZIONE DI UNA RETE NEURALE AD UN PROBLEMA DI CLASSIFICAZIONE DI BLOCCHI DI PAGINE

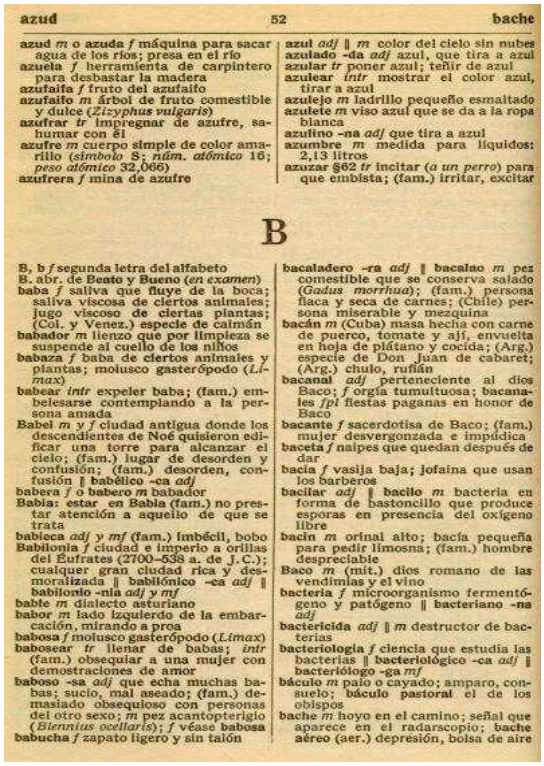
di Alessandro Lo Presti e Valerio Longo

INDICE

1 IL DATASET

1.1 ORIGINE

Dato l'elevato numero di compiti svolti all'interno di un'azienda e data l'accrescente esigenza di rimanere al passo con la tecnologia, è necessario utilizzare delle metodologie atte a semplificare la gestione e la diffusione di informazioni per il raggiungimento degli obiettivi che l'azienda stessa si è prefissata. Digitalizzare le informazioni è quindi uno degli aspetti chiave per quanto concerne l'organizzazione delle attività di una società o un gruppo di persone (gestione dei dati amministrativi, sincronizzazione delle riunioni, ecc.).
Di conseguenza, l'elaborazione di documenti cartacei risulta essere attualmente uno tra i compiti più importanti nell'automazione d'ufficio.
Si tratta di molto di più di una semplice acquisizione di un documento mediante uno scanner.
In generale, un documento cartaceo è una collezione di oggetti stampati (caratteri, colonne, paragrafi, titoli, figure e così via) ciascuno dei quali deve essere rilevato e poi elaborato nel modo più appropriato. Pertanto l'obiettivo principale di un sistema di elaborazione di documenti consiste nel convertire un'immagine di un documento in una appropriata forma simbolica.
Per immagine di documento si intende una rappresentazione visuale di un documento cartaceo, ad esempio: libri, riviste, lettere commerciali, fatture, documenti bancari, ecc. Più recentemente si considerano anche "documenti" come: pagine web, video.


azud 52 bache

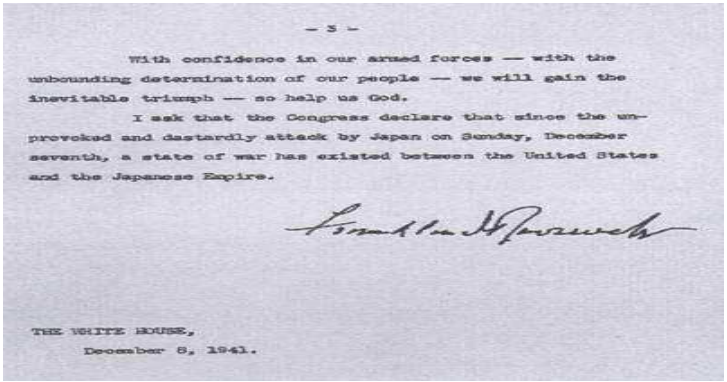
azud *m* o **azuda** *f* máquina para sacar agua de los ríos; presa en el río
azuela *f* herramienta de carpintero para desbastar la madera
azufaifa *f* fruto del azufaifo
azufaifo *m* árbol de fruto comestible y dulce (*Zizyphus vulgaris*)
azufrar *tr* impregnar de azufre, sahumar con él
azufre *m* cuerpo simple de color amarillo (*símbolo* S; *núm. atómico* 16; *peso atómico* 32,066)
azufrera *f* mina de azufre

azul *adj* ‖ *m* color del cielo sin nubes
azulado -da *adj* azul, que tira a azul
azular *tr* poner azul; teñir de azul
azulear *intr* mostrar el color azul, tirar a azul
azulejo *m* ladrillo pequeño esmaltado
azulete *m* viso azul que se da a la ropa blanca
azulino -na *adj* que tira a azul
azumbre *m* medida para líquidos: 2,13 litros
azuzar §62 *tr* incitar (*a un perro*) para que embista; (fam.) irritar, excitar

B

B, b *f* segunda letra del alfabeto
B. abr. de **Beato** y **Bueno** (*en examen*)
baba *f* saliva que fluye de la boca; saliva viscosa de ciertos animales; jugo viscoso de ciertas plantas; (Col. y Venez.) especie de caimán
babador *m* lienzo que por limpieza se suspende al cuello de los niños
babaza *f* baba de ciertos animales y plantas; molusco gasterópodo (*Limax*)
babear *intr* expeler baba; (fam.) embelesarse contemplando a la persona amada
Babel *m* y *f* ciudad antigua donde los descendientes de Noé quisieron edificar una torre para alcanzar el cielo; (fam.) lugar de desorden y confusión; (fam.) desorden, confusión ‖ **babélico -ca** *adj*
babera *f* o **babero** *m* babador
Babia: estar en Babia (fam.) no prestar atención a aquello de que se trata
babieca *adj* y *mf* (fam.) imbécil, bobo
Babilonia *f* ciudad e imperio a orillas del Éufrates (2700–538 a. de J.C.); cualquier gran ciudad rica y desmoralizada ‖ **babilónico -ca** *adj* ‖ **babilonio -nia** *adj* y *mf*
bable *m* dialecto asturiano
babor *m* lado izquierdo de la embarcación, mirando a proa
babosa *f* molusco gasterópodo (*Limax*)
babosear *tr* llenar de babas; *intr* (fam.) obsequiar a una mujer con demostraciones de amor
baboso -sa *adj* que echa muchas babas; sucio, mal aseado; (fam.) demasiado obsequioso con personas del otro sexo; *m* pez acantopterigio (*Blennius ocellaris*); *f* véase **babosa**
babucha *f* zapato ligero y sin talón

bacaladero -ra *adj* ‖ **bacalao** *m* pez comestible que se conserva salado (*Gadus morrhua*); (fam.) persona flaca y seca de carnes; (Chile) persona miserable y mezquina
bacán *m* (Cuba) masa hecha con carne de puerco, tomate y ají, envuelta en hoja de plátano y cocida; (Arg.) especie de Don Juan de cabaret; (Arg.) chulo, rufián
bacanal *adj* perteneciente al dios Baco; *f* orgía tumultuosa; **bacanales** *fpl* fiestas paganas en honor de Baco
bacante *f* sacerdotisa de Baco; (fam.) mujer desvergonzada e impúdica
baceta *f* naipes que quedan después de dar
bacía *f* vasija baja; jofaina que usan los barberos
bacilar *adj* ‖ **bacilo** *m* bacteria en forma de bastoncillo que produce esporas en presencia del oxígeno libre
bacín *m* orinal alto; bacía pequeña para pedir limosna; (fam.) hombre despreciable
Baco *m* (mit.) dios romano de las vendimias y el vino
bacteria *f* microorganismo fermentógeno y patógeno ‖ **bacteriano -na** *adj*
bactericida *adj* ‖ *m* destructor de bacterias
bacteriología *f* ciencia que estudia las bacterias ‖ **bacteriológico -ca** *adj* ‖ **bacteriólogo -ga** *mf*
báculo *m* palo o cayado; amparo, consuelo; **báculo pastoral** el de los obispos
bache *m* hoyo en el camino; señal que aparece en el radarscopio; **bache aéreo** (aer.) depresión, bolsa de aire


- 5 -

With confidence in our armed forces — with the unbounding determination of our people — we will gain the inevitable triumph — so help us God.

I ask that the Congress declare that since the unprovoked and dastardly attack by Japan on Sunday, December seventh, a state of war has existed between the United States and the Japanese Empire.

THE WHITE HOUSE,
December 8, 1941.

Per fare tutto questo, le strutture di un documento devono prima essere definite.
Secondo ODA/ODIF, due formati internazionali standard per documenti, qualsiasi documento è caratterizzato da due strutture differenti rappresentanti entrambe il suo contenuto e la sua organizzazione interna: la struttura geometrica (o layout) e la struttura logica.

La struttura geometrica (o fisica) descrive l'aspetto visivo del documento rappresentando gli oggetti fisici e le loro posizioni reciproche. Generalmente, tale struttura associa i contenuti del documento alla gerarchia degli oggetti costituenti il layout, come linee di testo, linee verticali/orizzontali, elementi grafici, elementi fotografici, colonne, e pagine.
La struttura logica assegna ad ogni oggetto un ruolo. Generalmente, tale struttura associa i contenuti del documento ad una gerarchia di oggetti logici, come il titolo, il sommario, i paragrafi, le sezioni, i capitoli, le tabelle, ecc.
Per esempio, possono essere considerati oggetti facenti parte del layout:

- il tipo di contenuto chiuso (testo, immagini, ecc.);
- la loro posizione assoluta all'interno della pagina secondo un sistema di coordinate ortogonali;
- la loro forma;
- le loro dimensioni, proprietà numeriche delle loro bitmap;

Mentre gli oggetti logici possono essere descritti da:

- il tipo (sommario, virgola, ecc.);
- alcune parole chiave contenute nel testo (data, figura, ecc.);
- proprietà di formattazione (spaziatura, indentazione, ecc.);

In alcuni tipi di documenti le due strutture si sovrappongono, o almeno esiste una corrispondenza tra il tipo di oggetto e la sua posizione e dimensione.
Per un certo stile di stampa la relazione tra una figura e la sua didascalia è sia logica sia fisica dato che la mutua posizione è fissa (es. la didascalia è sotto alla figura).
La relazione tra una figura e uno o più riferimenti ad essa nel testo è solo logica, dato che la posizione fisica è casuale, e la corrispondenza logica tra le due entità può essere determinata solo "leggendo" il testo.
Le relazioni fisico-logico sono le più interessanti. Infatti, consentono di individuare alcune componenti logici di un documento senza necessariamente leggere il contenuto tramite un riconoscitore ottico dei caratteri (OCR), ma utilizzando solo caratteristiche di layout. Per esempio, in una lettera inglese standard, la data è sotto l'indirizzo del mittente, che è a sua volta, nell'angolo in alto a destra. Così, questa semplice informazione di layout può essere proficuamente sfruttata da un sistema di gestione dei documenti per ricostruire la struttura logica di un documento, a partire dalla sua struttura di layout.
Nella letteratura il processo di scomposizione di un'immagine di documento in più componenti (blocchi), senza nessuna conoscenza relativa al formato specifico, è chiamato analisi dei documenti. Esistono diversi approcci al problema di analisi dei documenti, tra i più importanti abbiamo l'approccio top-down e l'approccio bottom-up.
L'approccio top-down prima divide il documento in grandi regioni che a loro volta sono ulteriormente suddivise in sotto-regioni: prima classificare le pagine (per identificare ad esempio le pagine con titolo), in seguito riconoscere i caratteri. Questo approccio si può usare se il layout è noto prima dell'elaborazione delle pagine.

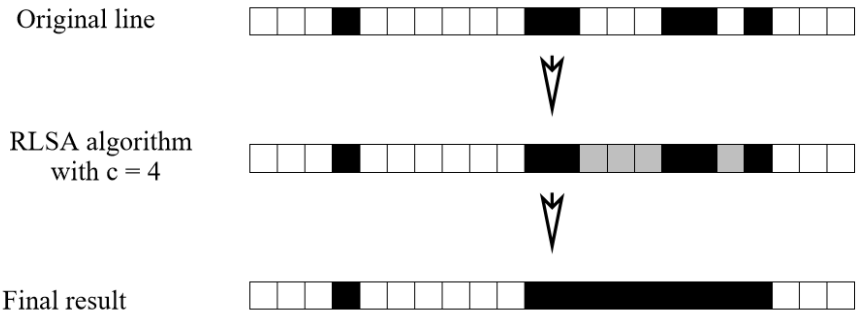
L'approccio bottom-up, invece, prima estrae i singoli componenti e poi li raggruppa insieme. Questo approccio si può usare quando la struttura fisica non è nota, in questo caso tutti gli oggetti nell'immagine (pagina) devono essere localizzati e riconosciuti.

1.1.1 L'ALGORITMO RLSA

L'algoritmo RLSA (Run Length Smoothing Algorithm) è un esempio di approccio top-down. Esso viene utilizzato per fondere insieme aree contigue di pixel neri.
Questo algoritmo è usato spesso per attaccare insieme i caratteri in una parola o le parole in un blocco di testo, di base è applicato ad ogni riga in una immagine B&W.
Ciascuna riga può essere rappresentata come una sequenza di 0 (pixel bianchi) e 1 (pixel neri). L'algoritmo trasforma queste sequenze sulla base delle seguenti regole:

1. Gli 0 sono cambiati a 1 se il numero di pixel 0 contigui è minore di una soglia pre-definita C.
2. Gli 1 non sono cambiati.

La scelta di C è cruciale al fine di ottenere aree omogenee, ossia che includano gli stessi tipi di dati (testo o grafica), inoltre una scelta ottimale del valore consente di raggruppare insieme caratteri in parole e così via.



Dopo il processo di smoothing (Figura 4e) si ottiene il documento segmentato. Ogni blocco è descritto da un insieme di features che permettono la classificazione in testo, linee orizzontale, linee verticali, grafica ed immagini.

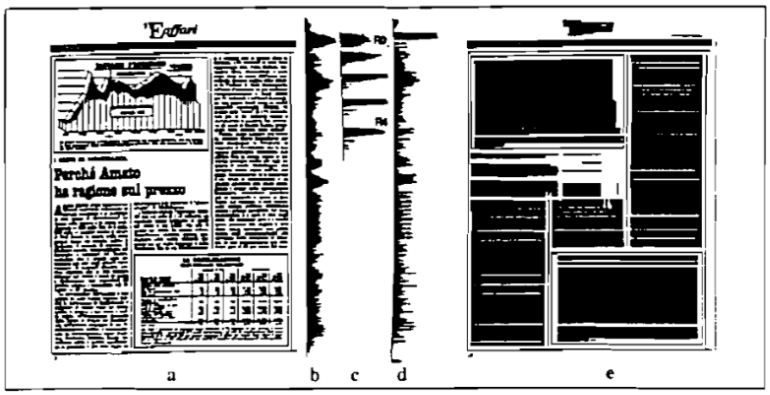


Figure 4. (a) Scanned document, (b) horizontal histogram, (c) sample region R0 and its successive rotations R1, R2, R3, R4, where the last three determine the points of the interpolation, (d) the horizontal histogram after the rotation, (e) result of the smoothing process.

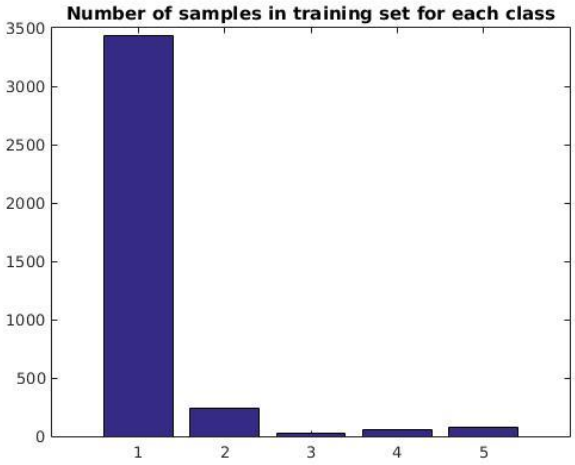
Il dataset che abbiamo utilizzato, il cui link è disponibile nella sezione Riferimenti, è costituito da 5473 campioni, la maggior parte dei quali appartiene alla prima classe.
Questa affermazione è stata possibile tramite la dichiarazione all'interno del codice Matlab delle seguenti variabili:

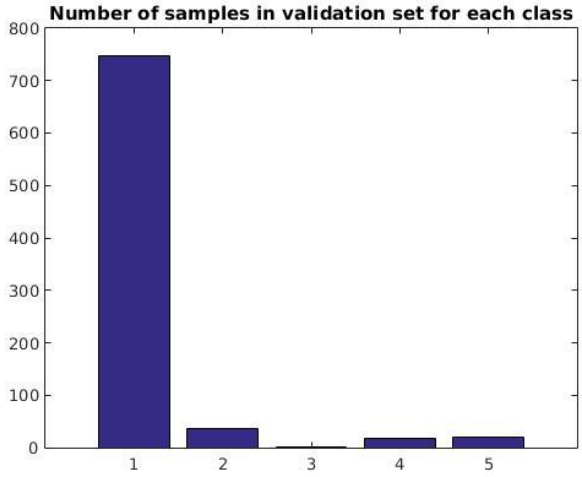
- trFirst, trSecond, trThird, trFourth, trFifth: utilizzate per indicare il numero di campioni, nel training set, appartenenti a ciascuna delle 5 classi;
- vFirst, vSecond, vThird, vFourth, vFifth: utilizzate per indicare il numero di campioni, nel validation set, appartenenti a ciascuna delle 5 classi;
- teFirst, teSecond, teThird, teFourth, teFifth: utilizzate per indicare il numero di campioni, nel test set, appartenenti a ciascuna delle 5 classi.

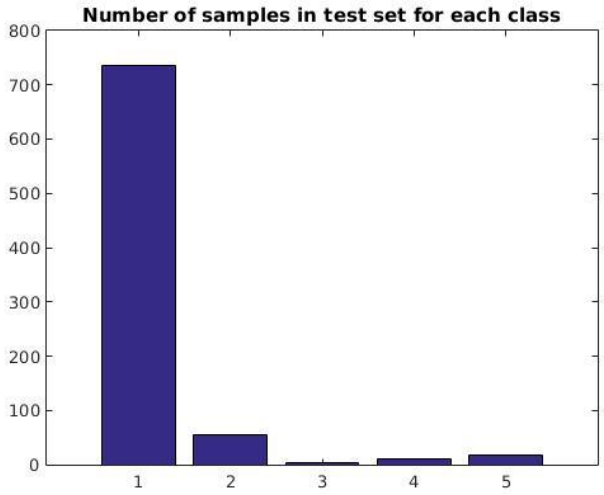
Il Neural Network Toolbox di Matlab, di default, suddivide il dataset in:

- 70% training set;
- 15% validation set;
- 15% test set;

Abbiamo mantenuto questa suddivisione dei dati in quanto la rete neurale ha mostrato ottime prestazioni. Abbiamo inoltre graficato tramite istogrammi i valori assunti dalle variabili precedentemente descritte. I risultati che abbiamo ottenuto, per quanto riguarda la distribuzione delle classi tra il training, il validation e il test set sono riportati nelle tre figure a seguire.







Come è possibile notare la maggior parte dei campioni corrisponde alla prima classe, mentre il minor numero di campioni ricade nella terza classe. Quest'ultima, infatti, nei risultati della confusion matrix, il cui significato verrà spiegato nelle prossime sezioni, in base al numero di neuroni presenti nell'hidden layer, può assumere un valore NaN dovuto al fatto che Matlab non ha distribuito per il test set alcun campione appartenente a questa classe.
Nonostante questa distribuzione dei dati implica che il dataset è sbilanciato, Matlab applica un campionamento random con probabilità uniforme che non altera l'istogramma delle classi e conserva quindi la distribuzione originaria, in sostanza abbiamo all'incirca le stesse percentuali di distribuzione dei campioni per ciascuna delle classi sia nel training set sia nel test set.
Questo fenomeno, infatti, ci ha permesso di non ricorrere a tecniche quali l'underfitting (ossia la rimozione di campioni appartenenti alle classi con una più elevata distribuzione) o l'oversampling (ossia la duplicazione dei campioni appartenenti alle classi con una minore distribuzione).
I risultati ottenuti dalla nostra rete neurale verranno spiegati nelle sezioni successive.

1.2 MANIPOLAZIONE DEI DATI

Adesso che abbiamo motivato il perché della scelta del nostro Dataset, dobbiamo adattare il file e suddividerlo in input (features) e output (classi o target).Le fasi più delicate dell'inizializzazione della rete sono senza dubbio l'analisi e la trasformazione dei dati nelle variabili di input e di output.
Il file page-blocks.txt, direttamente scaricato dal sito:
(https://archive.ics.uci.edu/ml/datasets/Page+Blocks+Classification)
Ha questa struttura:

Features N°1	Features N°2	……	Features N°10	Classe

Di seguito riportiamo le due tabelle descrittive riferite alle features ed alle classi.

N°	FEATURES	TIPO DI DATO	DESCRIZIONE
1	altezza	intero	Altezza del blocco
2	lunghezza	intero	Lunghezza del blocco
3	area	intero	Area del blocco, lunghezza * altezza
4	eccentricità	reale	Eccentricità del blocco, lunghezza / altezza
5	p_black	reale	Percentuale di pixel neri all'interno di un blocco, blackpix / area
6	p_and	reale	Percentuale di pixel neri dopo l'applicazione dell'algoritmo RLSA, blackand / area
7	media_trans	reale	Media dei numeri di transizione da bianco a nero, blackpix/wb_trans
8	blackpix	intero	Numero totale di pixel neri
9	blackand	intero	Numero totale di pixel neri dopo l'applicazione dell'algoritmo RLSA
10	wb_trans	intero	Numero totale di transizioni dal bianco al nero

N°	DESCRIZIONE DELLA CLASSE
1	Testo scritto sul blocco
2	Linea orizzontale del layout
3	Grafica del blocco
4	Linea Verticale del layout
5	Immagine

Nel nostro programma Matlab dovremmo prima importare tale file,

```
filename = 'page-blocks.txt';
dataSet = csvread(filename);
```

e successivamente suddividerlo in features (input) e target (classi).

```
features = dataSet(1:end, 1:end-1);
target = dataSet(1:end, end);
```

Ora però manca un ulteriore passaggio. Infatti la nostra rete neurale non può essere inizializzata direttamente con i dati presi, poiché essi necessitano di una manipolazione dovuta alla struttura intrinseca della rete. Nel Neural Network Toolbox si lavora solamente con dati dai valori di solito nel range [-1,1] oppure [0,1], ciò significa che siamo obbligati a trasformarli. Tale modifica è detta Normalizzazione.

1.2.1 NORMALIZZAZIONE E CODIFICA

La normalizzazione assicura una distribuzione più o meno uniforme tra gli input e gli output. Inoltre i dati standardizzati non solo danno dei risultati più affidabili, ma rendono l'intera rete molto più veloce.

La formula della normalizzazione: $X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$

In Matlab sia le features sia le classi vengono normalizzate tramite il metodo di default "mapminmax" che implementa la formula vista precedentemente. Se usassimo il vettore delle classi che ci ha fornito UCI però, andremmo ad introdurre una metrica tra gli output. La metrica è una funzione legata alla distanza tra una classe ed un' altra. Essa non è assolutamente accettabile nel nostro caso, poiché le classi sono indipendenti le une dalle altre e tra di loro non intercorre alcuna relazione numerica. Risolviamo quindi il problema attraverso una codifica delle classi.
La classe n°1 verrà rappresentata tramite un vettore 1 0 0 0 0, la classe n°2 con un vettore del tipo 0 1 0 0 0 ecc..

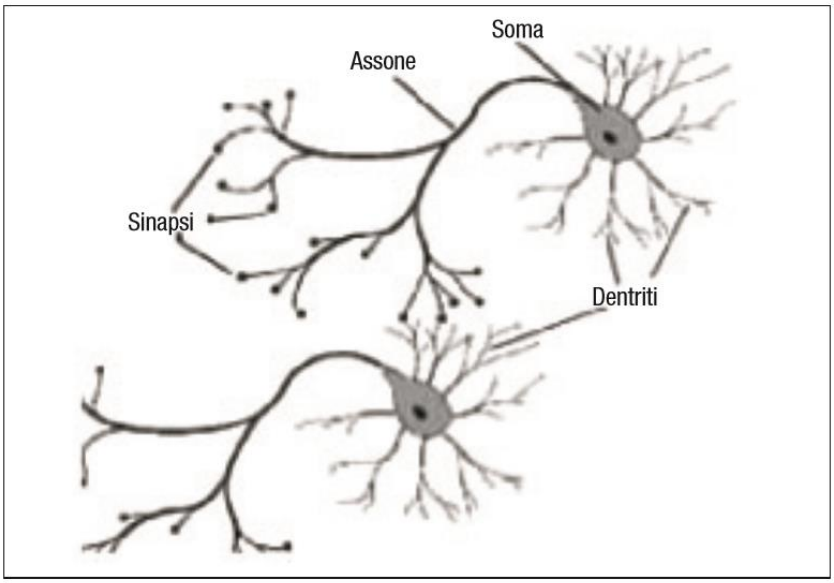
```
featuresDimension = size(features);
nSamples = featuresDimension(2);
nTarget = 5;
targetNorm = zeros(nTarget, nSamples);

for i=1:nSamples
    if target(i) == 1
        targetNorm(1, i) = 1;
    elseif target(i) == 2
        targetNorm(2, i) = 1;
    elseif target(i) == 3
        targetNorm(3, i) = 1;
    elseif target(i) == 4
        targetNorm(4, i) = 1;
    else
        targetNorm(5, i) = 1;
    end
end

target = targetNorm;
```

La nostra matrice "target" adesso avrà tante righe quante sono le classi e tante colonne quante sono le features presenti nel file page-blocks.txt.
Adesso che tutte le classi sono state codificate, tra loro non scorre nessuna relazione e siamo pronti per parlare della nostra Rete Neurale vera e propria.

2 LA NOSTRA RETE NEURALE

2.1 PERCHÉ LA RETE NEURALE?

Prima di parlare del motivo per cui è stata applicata una rete neurale a questo tipo di problema, risulta essere doveroso indicare cosa effettivamente sia e quali sono state le fasi che hanno caratterizzato la nascita delle reti neurali.
Una Artificial Neural Network (o Rete Neurale Artificiale) è una macchina progettata per simulare il funzionamento del cervello umano, implementata fisicamente utilizzando componenti elettronici e che utilizza software dedicati. Prende questo nome in analogia con le reti neurali biologiche dove i neuroni (circa 100 miliardi) sono responsabili dell'attività cerebrale.

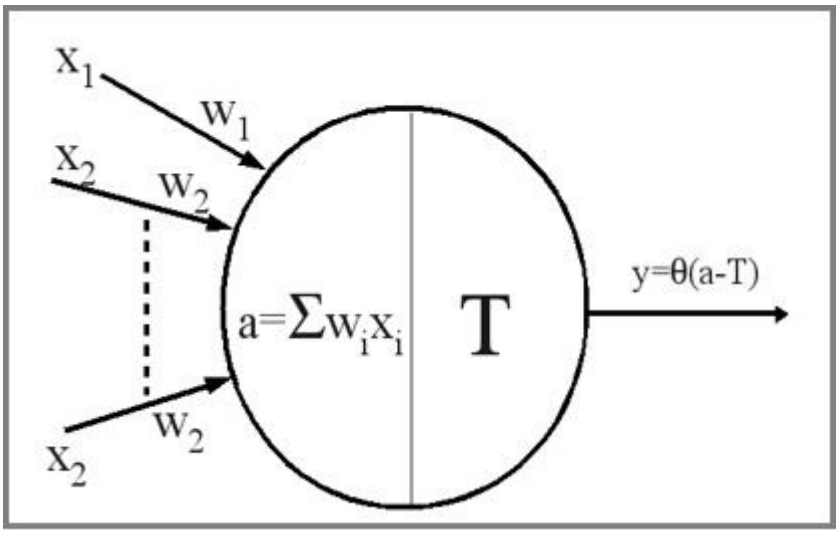


Ciascun neurone è interconnesso a circa altri 10.000 neuroni. Nelle interconnessioni ha luogo la sinapsi, un processo elettrochimico atto a rinforzare o inibire l'interazione cellulare.
Il nucleo somma i segnali di input provenienti dalle sinapsi collegate alle dendriti di altri neuroni. Quando il segnale raggiunge una soglia limite il neurone genera un segnale di output verso altri neuroni. Si dice che il neurone "fa fuoco".
Nel 1890 il filosofo e psicologo William James nel suo "Breve trattato di psicologia" , sembrò anticipare l'idea che l'attività di un neurone dipendesse dalla somma dei suoi stimoli in ingresso, provenienti da altri neuroni e che la forza di tali connessioni fosse influenzata dalla storia passata. Egli formulò per primo alcuni dei principi basilari dell'apprendimento e della memoria.
Nel 1943 McCulloch, uno psichiatra e neuroanatomista, e Pitts, un matematico, descrissero il calcolo logico della rete neurale che unisce la neurofisiologia alla logica matematica.
Secondo il modello McCulloch-Pitts un singolo neurone può essere così schematizzato:

- Il corpo della cellula (o soma) può essere interpretato come una unità di computazione elementare.
- Le connessioni tra bottoni sinaptici e dendriti di un neurone possono essere interpretate come delle linee (canali, connessioni) di input;

- I potenziali d'azione presenti sugli assoni pre-sinaptici possono essere visti come dei segnali di ingresso che possono attivare un canale di input;
- L'assone può essere visto come un canale di output che trasporta un segnale oppure è silente.

Inoltre il numero di neurotrasmettitori rilasciati da un bottone sinaptico quando è presente un potenziale d'azione esprime l'efficacia (o peso) della connessione tra un bottone sinaptico e il dendrite del neurone post-sinaptico. Tale efficacia può essere rappresentata tramite un valore w_i associato a ciascuna linea di input.
Ciascun ingresso può essere rappresentato da una variabile $x_i \in \{0, 1\}$ dove $x_i = 1$ rappresenta la presenza di un potenziale d'azione sul bottone pre-sinaptico, $x_i = 0$ l'assenza.
La presenza o assenza di un potenziale d'azione sull'assone può essere rappresentato da una variabile $y_i \in \{0, 1\}$ associata a ciascun canale d'output.
A partire dalle precedenti considerazioni possiamo descrivere il modello di McCulloch – Pitts nel seguente modo (vedi figura).



Un neurone è visto come una unità di computazione elementare costituita da:

- n linee di ingresso;
- un valore di input $x_i \in \{0, 1\}$, e un peso w_i per ciascuna i-esima linea di ingresso
- una soglia T
- un valore di output $y_i \in \{0, 1\}$.

L'unità di calcolo compie le seguenti operazioni:

- Ad ogni dato momento alcuni canali di input sono attivati, cioè sono alimentati con un input $x_i = 1$;
- L'unità computazionale riceve un input (l'equivalente del potenziale post-sinaptico, PSP) dato dalla somma dei w_i corrispondenti ai canali attivati;
- Tale somma è confrontata con un valore di soglia T. Se la somma supera la soglia T il valore di output dell'unità di calcolo risulta y = 1, altrimenti y = 0.

In definitiva, il valore di output dell'unità di calcolo può essere così formalizzato:

$$y = \theta(h - T) \qquad h = \sum_{i=1,\ldots,n} w_i x_i \qquad \theta(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases}$$

Quindi il neurone artificiale avrà un valore di output pari a 1 quando la somma pesata dei suoi input supera una certa soglia T, altrimenti il suo output sarà pari a 0.
Un altro grande apporto alle reti neurali è avvenuto tramite l'esperimento di Bliss-Limo: usando un coniglio anestetizzato, Bliss e Limo stimolarono con un impulso elettrico un canale neurale; poi misurarono il voltaggio che ne risultava lungo tutto il percorso. Dapprima il voltaggio dell'output fu molto basso, il che stava ad indicare che le connessioni sinaptiche nel circuito erano assai deboli. Ma stimolando ripetutamente il canale con scariche elettriche ad alta frequenza, furono in grado in qualche modo di alzare il volume delle connessioni. Ora, ogni volta che il canale veniva stimolato, i neuroni più a valle rispondevano vigorosamente. In conclusione i neuroni possono essere in qualche modo allenati! Provano ad imparare da esempi, è quindi possibile modificare dinamicamente i pesi sinaptici tramite algoritmi di apprendimento.
Un'altra grande scoperta avvenne nel 1949 quando Donald Hebb formulò un principio per l'addestramento delle connessioni fra neuroni: "Se un neurone A accende un neurone B entro un breve intervallo temporale, la connessione tra A e B è rafforzata, altrimenti è indebolita". Analiticamente ciò significava dare un peso maggiore a certe connessioni.
Nel 1958 Rosenblatt, propose il cosiddetto Percettrone, il primo modello di apprendimento supervisionato. Esso è un classificatore che discrimina gli ingressi in due insiemi linearmente separabili. Il percettrone si rivela utile per il riconoscimento e la classificazione di forme.
Nel 1969 Minsky e Papert, criticarono duramente le potenzialità del percettrone (a quel tempo la rete neurale più nota e oggetto di ricerche) in un celebre libro intitolato "Perceptrons": "Un percettrone non è in grado di fare uno XOR, quindi è troppo limitato per essere interessante". Infatti ciò avrebbe richiesto l'addestramento di neuroni detti "nascosti", cioè neuroni per i quali non esiste un supervisore in grado di condurre l'addestramento.
Negli anni 80 le ANN tornarono alla ribalta con l'introduzione di uno o più livelli intermedi (detti Hidden Layer). Tali reti, in grado di correggere i propri errori, superarono i limiti del Percettrone di Rosenblatt rivitalizzando la ricerca in tale settore.
Dopo un'attenta analisi delle fasi che hanno caratterizzato la nascita e lo sviluppo delle reti neurali è necessario motivare la scelta di questo approccio al problema di classificazione proposto per questo progetto. I modelli prodotti dalle reti neurali, sono molto efficienti, inoltre lavorano in parallelo permettendo, quindi, di raggiungere dei risultati ottimali all'aumentare della complessità del problema e della dimensione dei dati.
Tra le caratteristiche che le distinguono (purché le reti neurali considerate abbiano almeno uno strato nascosto come in questo caso), è il fatto di comportarsi da "approssimatori universali", ovvero possono rappresentare qualsiasi funzione continua con un errore che può essere reso piccolo a piacere in funzione delle variazioni del numero di neuroni nascosti (hidden neurons) come specificato dal teorema di approssimazione universale.
L'unico svantaggio, attualmente molto discusso e al centro di numerosi dibattiti, è il fatto che questi modelli non sono spiegabili in linguaggio simbolico umano: i risultati vanno accettati "così come

sono", da cui anche la definizione inglese delle reti neurali come "black box": in altre parole, a differenza di un sistema algoritmico, dove si può esaminare passo-passo il percorso che dall'input genera l'output, una rete neurale è in grado di generare un risultato valido, o comunque con una alta probabilità di essere accettabile, ma non è possibile spiegare COME e PERCHÉ tale risultato sia stato generato.

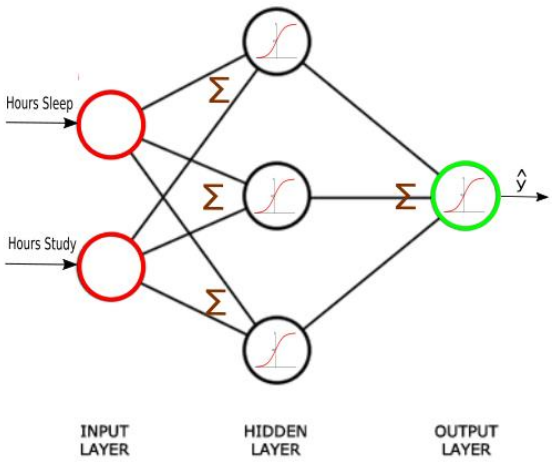
2.2 MORFOLOGIA DELLA NN

2.2.1 FORWARD PROPAGATION

Ora che conosciamo le origini che hanno contraddistinto le reti neurali e abbiamo compreso quali sono le potenzialità di una rete neurale, è necessario spiegare quali sono le componenti che la costituiscono, ma soprattutto come opera una rete neurale nella risoluzione di un problema. Una rete neurale artificiale (ANN) è un insieme di nodi interconnessi tra loro, più precisamente un insieme di strati (ciascuno costituito da uno o più nodi) che eseguono alcune operazioni. Gli strati che costituiscono una rete neurale sono essenzialmente:

- Lo strato degli ingressi (input layer): è il primo strato all'interno di una rete neurale. Esso ha il compito di inviare i dati normalizzati, attraverso le sinapsi all'hidden layer, se presente oppure all'output layer;
- Lo strato nascosto (hidden layer): è uno strato intermedio tra l'input layer e l'output layer. Come già discusso le prime reti neurali non possedevano strati nascosti, utilizzati invece a partire dagli anni 80, è quindi uno strato opzionale. La maggior parte dei problemi sono risolvibili e garantiscono ottime prestazioni con un solo hidden layer, invece, problemi di natura più complessa (ad esempio la classificazione di immagini), richiedono un maggior numero di strati nascosti. Una rete neurale con un numero molto vasto di hidden layer prende il nome di Deep Neural Network, da cui si fa riferimento al Deep Learning.
- Lo strato di uscita (output layer): rappresenta l'ultimo strato di una rete neurale. Esso prende i valori calcolati dall'input layer o dall'hidden layer (l'ultimo se in presenza di più strati nascosti) ed applica una funzione di attivazione, da cui si ottiene il risultato, la predizione secondo la rete neurale.

In questa descrizione dei diversi strati che caratterizzano la rete neurale, abbiamo introdotto alcuni termini che ora spieghiamo in maniera più dettagliata tramite l'illustrazione di una figura.



Come mostrato nell'immagine, è possibile notare che lo strato degli ingressi è costituito da due nodi (il primo nodo rappresenta le ore di sonno, il secondo le ore di studio), lo strato nascosto è semplicemente uno ed è composto da tre nodi, infine, lo strato di uscita è costituito da un solo nodo. Da come appena descritto è facile intuire che i nodi sono rappresentati da un cerchio. La maggior parte dei nodi presenti nella figura sono detti neuroni. I neuroni di una rete neurale artificiale hanno il compito di sommare insieme tutti gli output delle sinapsi entranti nel neurone stesso ed applicare una funzione di attivazione. Abbiamo indicato il termine "la maggior parte" in quanto i nodi che costituiscono lo strato degli ingressi non sono dei neuroni, essi infatti contengono solamente i valori iniziali del dataset normalizzato che vengono poi elaborati dalle sinapsi. Le sinapsi, invece, sono rappresentate in figura dagli archi che connettono i nodi, ciascun nodo è connesso da un arco ad ogni nodo dello strato successivo. Il compito delle sinapsi è quello di prelevare il valore di uscita del nodo corrente, moltiplicarlo per uno specifico peso, e restituirlo in output al nodo dello strato successivo. I pesi sono quei parametri che vengono immagazzinati dalle sinapsi e vengono utilizzati per manipolare i dati, sono loro, insieme al tipo di funzione di attivazione applicata, ad essere responsabili del comportamento del singolo neurone e della rete in generale. Solitamente il valore iniziale dei pesi è casuale o assume un valore uguale per tutti i nodi, in modo da non dare maggior importanza a determinati nodi, successivamente, nella fase di addestramento, assumono dei valori stabiliti in base all'algoritmo utilizzato, come spiegato nelle sezioni successive. Definito per ogni singola unità i un vettore di ingresso:

$$x_i = (x_1, x_2, \cdots, x_n)$$

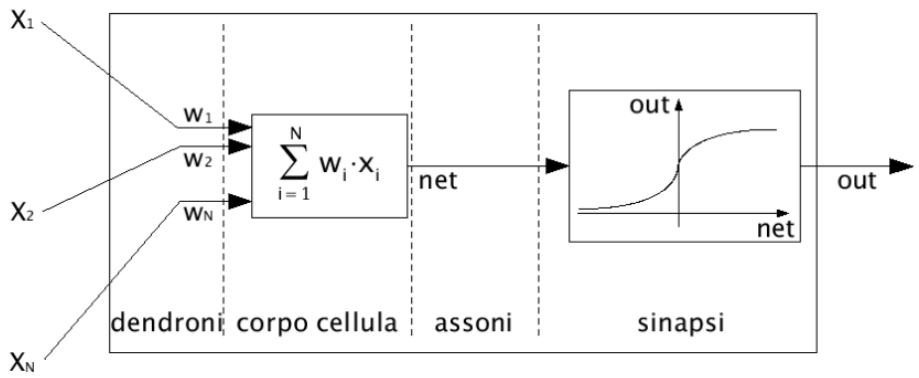
e un vettore di pesi:

$$w_i = (w_{i1}, w_{i2}, \cdots, w_{in})$$

il corpo del neurone compie una somma pesata degli ingressi:

$$net_i = (\sum_{j=1}^{n} w_{ij} x_j) + b_i$$

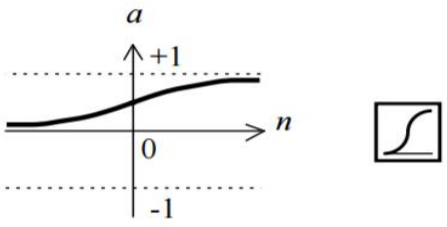
Tra i pesi ne esiste uno speciale che prende il nome di bias e viene indicato con il simbolo b_i. Esso non è legato a nessun'altra unità della rete ed è come se avesse ingresso sempre uguale a 1. Il bias serve a far compiere una traslazione sull'asse delle ascisse alla funzione di uscita.



Una delle prime e più semplici tipologie di reti neurali è stata la feedforward neural network. Nelle feedforward neural network l'informazione si muove in un'unica direzione, quella "in avanti", ossia dai nodi di input, attraverso gli strati nascosti (se presenti), ai nodi di output. Non sono presenti cicli o ripetizioni nella rete a differenza ad esempio delle reti neurali ricorsive. La nostra applicazione, infatti, si basa proprio su una rete neurale di tipo feed-forward. Poiché il dataset è costituito da 10 features e 5 classi, è stato introdotto un input layer con 10 nodi ed un output layer con 5 neuroni, inoltre tramite un flag denominato flagHidden è possibile scegliere se la rete implementi uno o due hidden layer con un numero variabile di neuroni. Infatti attraverso un flag è possibile modificare il numero di hidden units, analizzando così le prestazioni della rete in termini di classificazione dei campioni.

2.2.2 FUNZIONI DI ATTIVAZIONE

Come abbiamo accennato i neuroni, che si possono vedere come nodi di una rete orientata provvisti di capacità di elaborazione, ricevono in ingresso una combinazione dei segnali provenienti dall'esterno o dalle altre unità e ne effettuano una trasformazione tramite una funzione, detta funzione di attivazione.
Le funzioni di attivazione più comuni sono la funzione sigmoide, la funzione lineare, la funzione tangente iperbolica e corrispondono proprio alle funzioni che abbiamo utilizzato all'interno del progetto. È possibile richiamarle sempre tramite un opportuno flag.
La funzione di attivazione sigmoide mostrata nella figura sottostante prende l'input, che può avere qualsiasi valore tra più e meno infinito, e schiaccia l'uscita nell'intervallo [0, 1].



$a = logsig(n)$

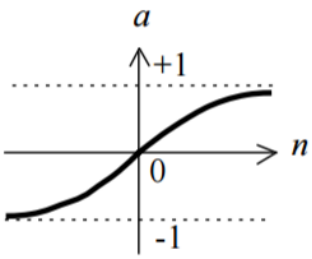
Log-Sigmoid Transfer Function

Questa funzione di trasferimento è comunemente utilizzata nelle reti con backpropagation, in quanto è una funzione continua e differenziabile.
Inoltre soddisfa la seguente proprietà:

$$\frac{d}{dt}\mathrm{sig}(t) = \mathrm{sig}(t)\,(1 - \mathrm{sig}(t))$$

Questa relazione polinomiale semplice fra la derivata e la funzione stessa è, dal punto di vista informatico, semplice da implementare.

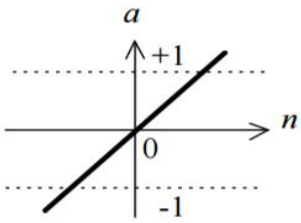
La funzione tangente iperbolica ha invece questo andamento:



$a = tansig(n)$

Tan-Sigmoid Transfer Function

Le funzioni di tipo sigmoide sono spesso utilizzate per problemi di pattern recognition, mentre la funzione di attivazione lineare viene utilizzata per problemi di fitting.
La funzione lineare è mostrata nella seguente figura:



$a = purelin(n)$

Linear Transfer Function

La funzione di attivazione lineare calcola l'output del neurone semplicemente ritornando il valore che gli era stato passato in input:

$$a = purelin(n) = purelin(\mathbf{W}\mathbf{p} + b) = \mathbf{W}\mathbf{p} + b$$

Solitamente i neuroni dell'hidden layer (o degli hidden layer nel caso ve ne siano più di uno) utilizzano la funzione sigmoide, che matematicamente, ha la seguente forma:

$$out_i = \frac{1}{1 + e^{-net_i}}$$

Nell'ultimo livello di una rete neurale di tipo feed-forward, anziché utilizzare la funzione sigmoide, si utilizza di solito la funzione *softmax*. Essa è utile tutte le volte in cui si vuole interpretare l'output di una rete come una probabilità a posteriori.
Dato

$$net_i \quad 1 \leq i \leq N$$

dove N rappresenta il numero di output della rete, la funzione softmax si scrive come:

$$out_i = \frac{e^{net_i}}{\sum_{j=1}^{N} e^{net_j}} \tag{2.4}$$

con:

$$0 \leq out_i \leq 1 \qquad \sum_{i=1}^{N} out_i = 1$$

Nella nostra applicazione il flag riguardante le funzioni di attivazione, è inizialmente impostato sulla funzione di attivazione sigmoide per quanto riguarda le hidden units in quanto è la funzione di attivazione che abbiamo più utilizzato nella fase di testing. Questo uso intenso e considerabile "privilegiato" rispetto alle altre due funzioni è dovuto ai risultati complessivi ottenuti dalla rete, risultati essere sensibilmente più performanti.
Una volta appreso il funzionamento di questi argomenti basici che caratterizzano una rete neurale, l'aspetto più importante e difficile da affrontare consiste nell'addestramento della rete: nel cercare cioè i pesi e i bias che le permettano di essere il più possibile aderente al fenomeno che deve modellare.

2.3 LA FUNZIONE COSTO

Una volta che abbiamo impostato il peso e il bias per ciascun neurone ed abbiamo stabilito qual è la funzione di attivazione per ciascuno strato che costituisce la rete, la nostra NN darà in uscita una predizione $\hat{y}$. Alla prima iterazione la nostra rete darà sicuramente un risultato che discosta molto dalla reale y, ossia l'uscita desiderata. Questo perché abbiamo inserito dei valori casuali per i pesi w_i e un valore fissato a priori per il bias b_i.
Per quantificare quanto scosta, quanto è sbagliata la predizione rispetto al valore desiderato utilizziamo la funzione costo. La funzione costo ci dice quanto è costoso il nostro modello rispetto ad un dato campione.
In realtà, come vedremo nelle sezioni successive, allenare una rete significa proprio minimizzare una funzione costo.

2.3.1 MSE V CROSS ENTROPY

Le funzioni di costo più utilizzate nelle reti neurali sono la MSE (Mean Square Error), oppure in italiano l'errore quadratico medio, e la Cross-Entropy.
Quest'ultima è una misura derivante dalla teoria dell'informazione che misura quanto è simile la probabilità in uscita alla rete neurale (ossia la predizione), rispetto alla probabilità reale dei campioni (ossia i target).
Nei problemi di classificazione risulta però essere più utilizzata la seconda in quanto il cross-entropy error risulta essere migliore dell'mse-error nella valutazione della qualità della rete.
Per comprendere il motivo di questa scelta analizziamo un esempio.
Supponiamo di avere solo tre campioni per allenare la rete e che quest'ultima utilizzi la funzione di attivazione softmax per lo strato di output in modo che ci siano tre valori di uscita che possono essere interpretati come probabilità. Ad esempio l'uscita della rete è la seguente:

```
computed          | targets              | correct?
-------------------------------------------------
0.3   0.3   0.4   | 0   0   1 (democrat)   | yes
0.3   0.4   0.3   | 0   1   0 (republican) | yes
0.1   0.2   0.7   | 1   0   0 (other)      | no
```

Questa rete neurale, che indichiamo con NN1, ha un errore di classificazione 1/3 = 0,33.
Ora consideriamo una seconda rete neurale, che denotiamo con NN2, i cui risultati sono i seguenti:

```
computed          | targets              | correct?
-------------------------------------------------
0.1   0.2   0.7   | 0   0   1 (democrat)   | yes
0.1   0.7   0.2   | 0   1   0 (republican) | yes
0.3   0.4   0.3   | 1   0   0 (other)      | no
```

Anche questa rete neurale ha un errore di classificazione di 1/3 = 0,33.
Questa seconda rete neurale però è migliore della prima perché indovina correttamente i primi due target e sbaglia di pochissimo l'ultimo dato del training set.
La prima rete neurale, invece, a malapena indovina i primi due campioni e sbaglia totalmente il terzo. Sapendo che l'errore di cross-entropy è dato dalla seguente formula:

$$H(p,q) = -\sum_x p(x)\,\log q(x).$$

Dove p(x) indica l'uscita desiderata e q(x) indica l'uscita predetta.

E che l'errore quadratico medio è data dalla seguente formula:

$$MSE = \frac{\sum_{i=1}^{n} (x_i - \hat{x}_i)^2}{n}$$

dove $\hat{x}$ rappresenta la predizione e la x rappresenta l'uscita desiderata, calcoliamo i due errori rispetto ai risultati delle due reti neurali prese in esempio.
Gli errori di cross-entropy per le due reti sono:

```
-(ln(0.4) + ln(0.4) + ln(0.1)) / 3 = 1.38
```

```
-(ln(0.7) + ln(0.7) + ln(0.3)) / 3 = 0.64
```

Invece, l'errore quadratico medio risulta essere:

```
(0.54 + 0.54 + 1.34) / 3 = 0.81
```

```
(0.14 + 0.14 + 0.74) / 3 = 0.34
```

Si nota che l'errore di cross-entropy della seconda rete neurale è molto più piccolo dell'errore di cross-entropy della prima rete neurale. La funzione logaritmo nell'errore di cross-entropy prende in considerazione la vicinanza di una previsione e un modo più granulare per calcolare l'errore. Invece, l'MSE dà una maggiore enfasi agli output non corretti.
Per riassumere, per una classificazione tramite NN durante l'allenamento utilizzare l'errore di cross-entropy risulta essere migliore rispetto all'MSE.
Se si sta utilizzando la back-propagation, la scelta di MSE o dell'errore di cross-entropy colpisce il calcolo del gradiente.
Entrambi questi ultimi due concetti verranno spiegati nella prossima sezione.

2.4 MINIMIZZAZIONE DEL COSTO

Adesso che abbiamo ben chiaro il concetto di costo, dobbiamo affrontare un aspetto fondamentale per l'efficienza della nostra rete: la minimizzazione della funzione costo (o funzione di errore).
Il nostro obiettivo è quello di avere questa relazione:

$$\hat{y} - y = 0$$

Cioè che la nostra y predetta sia uguale a quella vera e propria del training set.
L'algoritmo che viene utilizzato è il così detto "back-propagation" che consiste in una procedura iterativa per minimizzare la funzione di errore, con la determinazione dei valori dei pesi fatta in passi successivi.
Ogni neurone, che sia dell'hidden layer o dell'output layer esegue due computazioni:

- la computazione della funzione di allenamento (trainFcn in Matlab) espressa come funzione non lineare continua di un segnale di input e dei pesi sinaptici associati al neurone;
- la computazione di un vettore gradiente necessario per l'aggiornamento dei pesi sinaptici.

Per quest'ultima fase dobbiamo utilizzare un metodo per trovare il punto di minimo della funzione costo. Questo metodo è la discesa del gradiente.
In Matlab, la back-propagation non è sinonimo di discesa del gradiente perché in ogni funzione di allenamento viene calcolato il gradiente in maniera diversa. Una parte importante del nostro progetto è quella di analizzare la funzione di allenamento "traingd" (discesa del gradiente).

2.4.1 DISCESA DEL GRADIENTE

Data la nostra funzione di errore:

$$J = \sum \frac{1}{2}(y - \hat{y})^2 \ (1)$$

Il nostro obiettivo è quello di minimizzare tale funzione.
Per semplicità riportiamo qui le formule relative ad una rete neurale così strutturata:

- 2 input
- 1 hidden layer dotato di 3 neuroni
- 1 output

$$z^{(2)} = XW_1 \quad (2)$$

z(2) indica il valore di un neurone dell'hidden layer, X l'input e W1 la rispettiva matrice dei pesi.

$$a^{(2)} = f(z^{(2)}) \quad (3)$$

a(2) indica il valore del neurone una volta applicata la funzione di attivazione;

$$z^{(3)} = a^{(2)}W_2 \quad (4)$$

z(3) indica il valore del neurone dell'output layer e W2 la rispettiva matrice dei pesi.

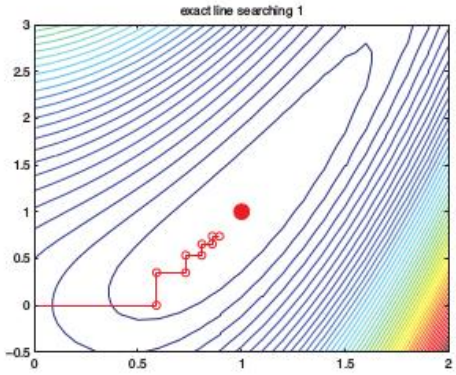
$$\hat{y} = f(z^{(3)}) \quad (5)$$

$\hat{y}$ indica il valore del neurone dell'output layer una volta applicata la funzione di attivazione.

Anche semplificando al massimo la rete, vedremo più avanti che non perderemo di generalizzazione ma riusciremo a esporre il tutto con molta più semplicità. Infatti aumentando il numero di input, di hidden layer, di neuroni per hidden layer e di output, avremo solamente più formule dotate però della stessa struttura (valore del neurone prima e dopo avergli applicato la funzione di attivazione).
Torniamo adesso alla nostra funzione di costo.
Possiamo notare che mentre y è un valore indipendente dal tipo di rete (infatti è proprio quello estrapolato dal dataset), $\hat{y}$ è strettamente legata ai pesi.
Quindi minimizzare la funzione di errore equivale a modificare opportunamente i pesi W.
Un possibile (e ingenuo) metodo per variare i pesi W potrebbe essere il perturbare m volte ogni peso con valori diversi, calcolare la funzione J, vedere se essa è diminuita o meno, salvare il risultato e andare a modificare un altro peso. Applicando questa metodologia però si avrebbe un costo computazionale O(mn) con n il numero di pesi e m il numero di perturbazioni. Pare quindi evidente che non può essere applicata neanche per reti molto piccole,basti pensare al fatto che la nostra rete (semplificata ad-hoc) possiede ben 9 pesi.
Un metodo più intelligente è proprio quello della discesa del gradiente.
Inizializzo i pesi con valori random e analizzo la funzione costo J.
Poichè la derivata parziale rispetto un solo peso mi dà "l'angolazione" di J rispetto al peso stesso, posso capire in quale direzione il costo diminuisce calcolandomi il gradiente.
Fin tanto che vale questa relazione :

$$\frac{\partial J}{\partial W} < 0$$

Significa che mi sto avvicinando sempre più al punto di minimo di J.
Quando ciò non varrà più, vorrà dire che i valori dei pesi sono tali per cui il costo J è minimo.
Bisogna prestare attenzione però al risultato ottenuto con questo metodo. Infatti la discesa del gradiente non assicura un minimo globale ma solo un minimo locale. Se la funzione costo non andasse sempre nella stessa direzione, ovvero fosse una funzione non convessa, potrebbe "ingannarci" e avere come risultato finale un valore non ottimale dei singoli pesi.
È per questo motivo che la funzione costo è la somma quadratica dei singoli errori.
La variazione del peso sinaptico nel nostro progetto viene quindi fatta facendo apprendere alla rete tutte le features insieme (Batch Gradient Descent) .



(a)

Nel nostro progetto, utilizzeremo anche il “conjugate scaled gradient descent”, una variante ottimizzata di questo metodo. L’intuizione dietro questa variante è che le direzioni verso il minimo vengono scelte in un modo tale da rendere meno probabile il “percorrere nella stessa direzione” rispetto all’iterazione successiva andando per passi ortogonali, evitando step ripetitivi a “zig-zag” (come in figura a). Così facendo il raggiungimento del minimo è più breve.
Adesso che abbiamo illustrato il metodo della discesa del gradiente, dobbiamo calcolare e propagare all’indietro gli errori per ogni singolo peso.

2.4.2 BACK-PROPAGATION

Il nostro obiettivo è adesso quello di andare a calcolare $\frac{\partial J}{\partial W}$utilizzando le formule precedentemente esposte, al fine di poter applicare la discesa del gradiente.
Il primo passo è dividere i pesi in due tipologie: W1 sono i pesi relativi all’hidden layer e W2 sono i pesi relativi all’output layer. Da questa suddivisione avrò che ciò che prima era $\frac{\partial J}{\partial W}$, adesso sarà composta da $\frac{\partial J}{\partial W_1}e\frac{\partial J}{\partial W_2}$

Per semplicità andremo prima a calcolarci la derivata relativa a W2.

$$\frac{\partial J}{\partial W_2} = \sum \frac{\partial \frac{1}{2}(y-\hat{y})^2}{\partial W_2}$$

Poichè la derivata della somma è la somma delle derivate, posso trascurare la somma per adesso e riproporla alla fine. Per la regola della catena, avrò che:

$$\frac{\partial J}{\partial W_2} = -(y-\hat{y})\frac{\partial \hat{y}}{\partial W_2} = -(y-\hat{y})\frac{\partial \hat{y}}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial W_2}$$

Con riferimento a (5)

$$\frac{\partial J}{\partial W_2} = -(y-\hat{y})f'(z^{(3)})\frac{\partial z^{(3)}}{\partial W_2}$$

A questo punto denominiamo come Back-Propagation Error :

$$\delta^{(3)} = -(y-\hat{y})f'(z^{(3)})$$

Se riprendiamo (4), notiamo che derivare z(3) rispetto al peso W2 significa calcolare la pendenza a(2) (nella singola sinapsi), ovvero la funzione di attivazione applicata al neurone del layer precedente!
É questo il cuore della Back-propagation.

Avremo quindi che:

$$\frac{\partial J}{\partial W_2} = a^{(2)T}\delta^{(3)} \text{(6)}$$

Adesso che abbiamo calcolato la derivata di J rispetto a W2, la andremo a calcolare rispetto W1 , utilizzando gli stessi principi.

$$\frac{\partial J}{\partial W_1} = -(y - \hat{y}) \frac{\partial \hat{y}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial W_1} = -(y - \hat{y}) f'(z^3) \frac{\partial z^{(3)}}{\partial W_1} = \delta^{(3)} \frac{\partial z^{(3)}}{\partial W_1}$$

Applicando la regola della catena avremo che:

$$\frac{\partial J}{\partial W_1} = \delta^{(3)} \frac{\partial z^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial W_1}$$

Data la formula (4) ora derivo non più rispetto ai pesi W ma rispetto la funzione di attivazione a(2), il ché ci porta a fare un ragionamento analogo al precedente ma leggermente diverso:

$$\frac{\partial J}{\partial W_1} = \delta^{(3)} W_2^T \frac{\partial a^{(2)}}{\partial W_1}$$

Applico un'ulteriore volta la regola della catena:

$$\frac{\partial J}{\partial W_1} = \delta^{(3)} W_2^T \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial W_1} = \delta^{(3)} W_2^T f'(z^{(2)}) \frac{\partial z^{(2)}}{\partial W_1}$$

Sta volta se osserviamo (2) ci accorgiamo che la pendenza della derivata è proprio l'input X e come passaggio conclusivo avremo:

$$\frac{\partial J}{\partial W_1} = X^T W_2^T \delta^{(3)} f'(z^{(2)}) (7)$$

Ora che abbiamo finito i passaggi matematici possiamo tranquillamente estendere il tutto anche a reti neurali più complesse di questa, e avere come unici cambiamenti la grandezza delle derivate parziali e le altre formule relative a eventuali altri hidden layer aventi però la stessa struttura di (2),(3),(4) e (5).

Infine per quanto riguarda l'arresto dell'algoritmo di back propagation, non esistono criteri ben definiti; ne citiamo alcuni:

- L'algoritmo di back-propagation può essere interrotto quando è nelle vicinanze di un minimo locale;
- L'algoritmo di back-propagation può essere interrotto quando la percentuale di variazione dell'errore tra due epoche (un interazione feed-propagation e back-propagation) consecutive è sufficientemente piccola.
- L'algoritmo di back-propagation può essere interrotto quando la capacità di generalizzazione è adeguata, ovvero non si presenta né overfitting, né underfitting.

Adesso che abbiamo illustrato il cuore della back-propagation, abbiamo le basi per poter meglio intuire l'altra funzione di allenamento utilizzata in questo progetto: trainbfg, ovvero il BFGS Quasi-Newton.

2.4.3 L'ALGORITMO BFGS QUASI NEWTON

Il BFGS (Broyden–Fletcher–Goldfarb–Shanno) è un algoritmo facente parte del gruppo dei Metodi di Newton.
In generale questi metodi usano le informazioni delle derivate seconde (Hessiane) della funzione obiettivo, cosa che la discesa del gradiente non fa.
In una dimensione, per esempio, la discesa del gradiente calcola semplicemente la tangente della funzione e in base alla sua inclinazione si dirige verso il passo successivo dove tale derivata è minore di zero.
Le informazioni fornite dalla derivata seconda riguardano la "velocità" con cui il gradiente arriva a zero ed è tramite questa informazione che il BFGS si dirige verso il passo successivo. Così facendo avrò una notevole riduzione dei passi necessari al raggiungimento del minimo. Attenzione però! È vero che il numero di passi è inferiore rispetto alla discesa del gradiente, ma non è detto che l'algoritmo sia sempre più veloce.
Infatti il calcolo approssimato della matrice Hessiana diretta o inversa (a seconda della variante dell'algoritmo) non è detto sia facile da calcolare.
Calcolo approssimato della matrice Hessiana al passaggio k+1 :

$$\begin{aligned}\mathbf{B}_{k+1} &= \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{(\mathbf{B}_k \mathbf{s}_k)(\mathbf{B}_k \mathbf{s}_k)^T}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} \\ \mathbf{s}_k &= \boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1} \\ \mathbf{y}_k &= \mathbf{g}_k - \mathbf{g}_{k-1}\end{aligned}$$

Calcolo approssimato della matrice Hessiana inversa al passaggio k+1:

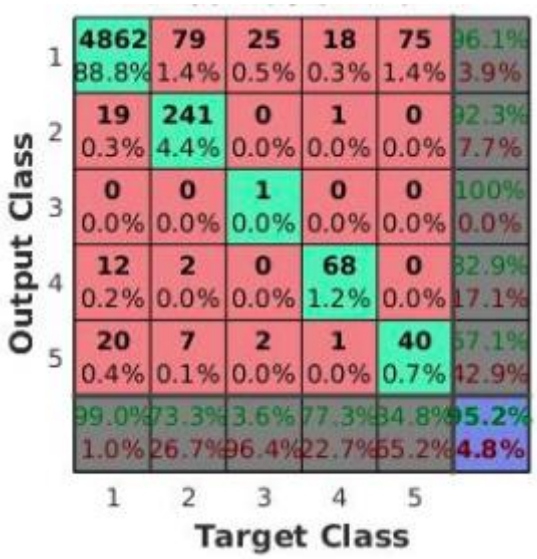
$$\mathbf{C}_{k+1} = \left(\mathbf{I} - \frac{\mathbf{s}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}\right) \mathbf{C}_k \left(\mathbf{I} - \frac{\mathbf{y}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}\right) + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}$$

Dato n il numero delle features, il costo computazionale di questo algoritmo è $O(n^2)$, mentre quello di memoria è O(mn), con m il numero di coppie più recenti <sk,yk>.
Questo costo elevato lo abbiamo sopratutto In Matlab, dove questo algoritmo richiede parecchia memoria e computazione rispetto ai metodi precedentemente analizzati. In compenso però, specificatamente al nostro progetto, Il BFGS è stato il migliore nella rete dotata di un hidden layer e avente (secondo le euristiche) valori ottimali, poiché la rete era semplice e il costo computazionale non era eccessivo.

2.5 ERRORE DI PREDIZIONE DEL MODELLO

2.5.1 COME MISURARE L'ERRORE: LA CONFUSION MATRIX

Per addentrarci nella parte conclusiva è necessario chiarire che in Matlab per misurare l'errore ottenuto al termine di una run viene fornito un mezzo di misurazione molto utile: la confusion matrix. Da essa possiamo vedere molto facilmente se il nostro scopo di avere una y predetta approssimativamente uguale alla y del dataset sia effettivamente stato raggiunto.
Se impostato a valori quasi ottimali, la nostra rete fornisce questa confusion matrix.



Essa va analizzata nel seguente modo:
Le prime 5 celle diagonali mostrano il numero e la percentuale di correttezza nella classificazione.
Per esempio nel quadrante della prima riga e prima colonna notiamo che 4862 elementi sono stati correttamente classificati nella classe n°1 con un errore del 88,8%. Allo stesso modo sono stati correttamente classificati nella classe n°2 241 elementi con un errore del 4,4%.
79 elementi che in realtà facevano parte della classe n°2, sono stati classificati erroneamente nella classe n°1 con un errore del 1,4%.
Sulla prima riga traspare il fatto che su 5059 elementi predetti come di classe n°1, solo il 3,9% di questi sono risultati essere di un'altra classe.
Sulla prima colonna invece, di 4913 elementi effettivamente di classe n°1 il 99% è stato predetto correttamente.
In totale la nostra rete ha classificato correttamente il 95.2% di tutti gli elementi.

2.5.2 OVERFITTING E UNDERFITTING

In statistica e in informatica, si parla di overfitting (in italiano: eccessivo adattamento) quando un modello statistico molto complesso si adatta ai dati osservati (il campione) perché ha un numero eccessivo di parametri rispetto al numero di osservazioni.
Un modello assurdo e sbagliato può adattarsi perfettamente se è abbastanza complesso rispetto alla quantità di dati disponibili.
Il concetto di overfitting riveste un ruolo fondamentale anche nel machine learning.
Di solito un algoritmo di apprendimento viene allenato usando un certo insieme di esempi (il training set appunto), ad esempio situazioni tipo di cui è già noto il risultato che interessa prevedere (output). Si assume che l'algoritmo di apprendimento (il learner) raggiungerà uno stato in cui sarà in grado di predire gli output per tutti gli altri esempi che ancora non ha visionato, cioè si assume che il modello di apprendimento sarà in grado di generalizzare. Tuttavia, soprattutto nei casi in cui l'apprendimento è stato effettuato troppo a lungo o dove c'era uno scarso numero di esempi di allenamento, il modello potrebbe adattarsi a caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nel resto dei casi; perciò, in presenza di overfitting, le prestazioni (cioè la capacità di adattarsi/prevedere) sui dati di allenamento aumenteranno, mentre le prestazioni sui dati non visionati saranno peggiori.
Viceversa si ha underfitting quando il modello non è abbastanza ricco di parametri per approssimare correttamente la funzione obiettivo. Sia l'overfitting sia l'underfitting implicano che il modello non sarà in grado di generalizzare correttamente, cioè di fornire risposte corrette a fronte di input diversi da quelli forniti in fase di addestramento. La capacità di generalizzazione può essere misurata attraverso i seguenti passi:

1- dividendo i dati a disposizione in tre insiemi: training set, validation set, test set.

2- Si sceglie il numero di unità facendo crescere/diminuire il numero di neuroni.

3- Per ogni numero di neuroni, si addestra la rete corrispondente usando sempre il training set.

4- Si valuta l'architettura scelta tramite il validation set e si sceglie quella che dà l'errore più basso.

5- La migliore architettura viene valutata tramite il test set.

Esistono diversi metodi che permettono di incrementare le performance della rete, tra i più utilizzati vi è la tecnica di regolarizzazione che non discutiamo in questa relazione in quanto richiede delle competenze troppo avanzate affinché venga pienamente compresa.

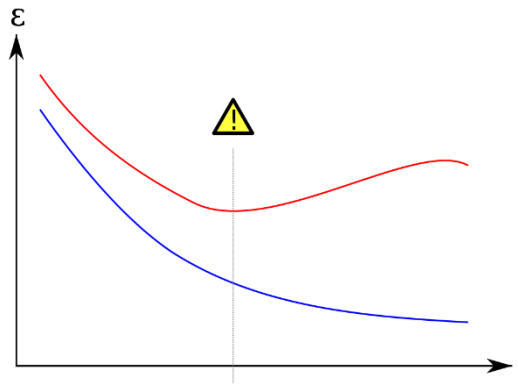
Un'alternativa alla regolarizzazione è l'early stopping (utilizzata di default da Matlab):

1- Si interrompe prematuramente la minimizzazione della funzione di errore.

2- Si valuta la rete sul validation set.

3- All'inizio l'errore sul validation set diminuisce al migliorare dell'errore sul training set.

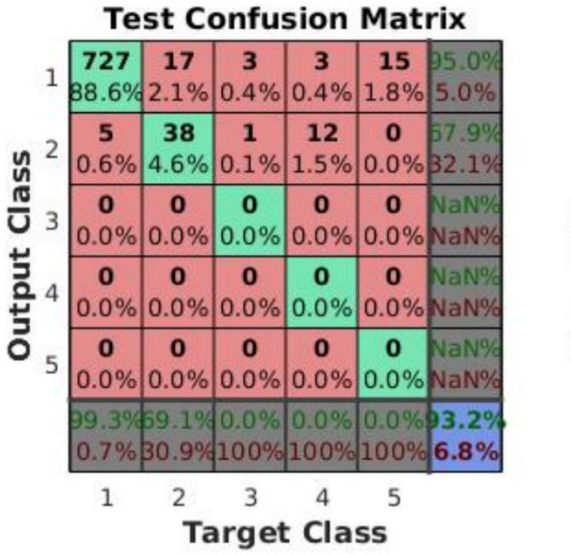
4- Se si ha il fenomeno dell'overfitting, l'errore sul validation set sale. A questo punto si interrompe il processo. Ciò è possibile notarlo dalla seguente figura:



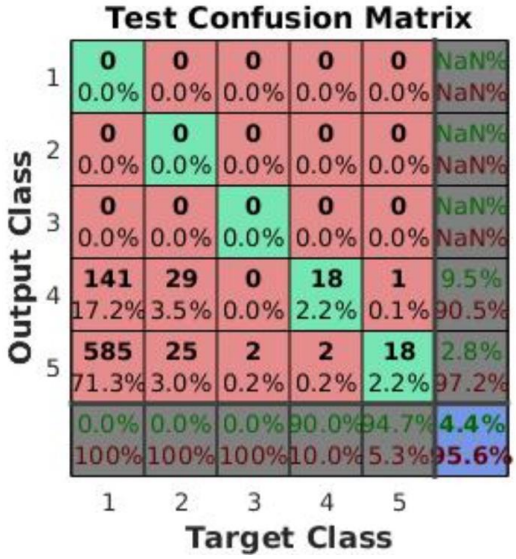
Dove la linea blu rappresenta il training set e la linea rossa rappresenta il validation set.

Un altro aspetto che può limitare la qualità del modello è la scelta dei dati: se esiste una forte correlazione tra dati di addestramento e dati di validazione si incorre in una sovrastima della capacità predittiva. L'insieme dei dati deve quindi essere il più eterogeneo possibile. Va comunque sottolineato che gli algoritmi di apprendimento hanno ottime capacità di estrazione delle caratteristiche essenziali e isolamento delle caratteristiche occasionali, sintomatiche di errore. Solitamente nelle multilayer neural network ciò che provoca underfitting ed overfitting sono il numero sottodimensionato o sovradimensionato di neuroni presenti all'interno di ciascun hidden-layer (infatti i pesi rappresentano proprio i parametri del modello) e il numero di hidden-layer che costituiscono la rete stessa.

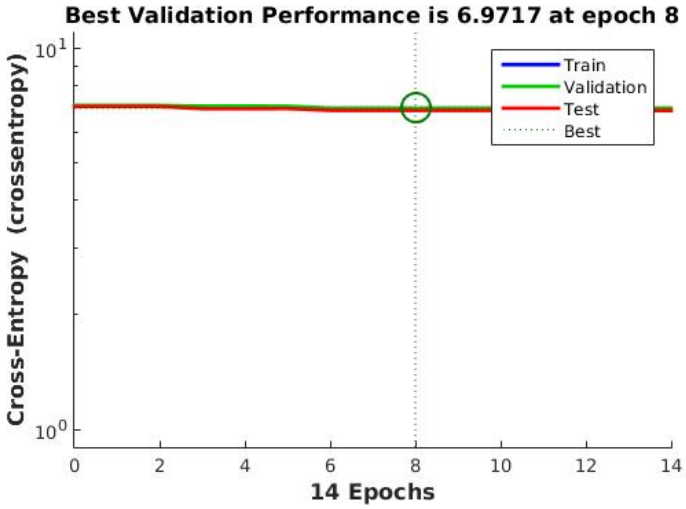
Nella fase di testing abbiamo notato che inserendo un solo hidden layer, a sua volta costituito da un neurone, si è verificato il fenomeno di underfitting. Basta visionare la confusion matrix associata al test set:



Come è possibile notare, con un solo neurone la rete "sceglie" di risolvere un problema di classificazione binaria fra la prima e la seconda classe, infatti, dalla confusion matrix, è possibile notare che la rete non cerca mai di classificare un pattern nelle classi 3, 4 e 5.
Se si prende in considerazione il fatto che il dataset è sbilanciato, si scopre, come è possibile notare dalla confusion matrix riportata, che anche se la rete non riesce a classificare perfettamente nemmeno le prime due classi, questo basta ad ottenere un errore vicino al 7%.
Nella nostra applicazione non si verifica invece overfitting, le cause che non permettono il manifestarsi di questo fenomeno possono essere molteplici.
Tra le più accreditate vi è l'utilizzo da parte del Neural Network Toolbox nella fase di minimizzazione dell'errore, del metodo di early stopping a cui viene aggiunto un termine di regolarizzazione. Infatti, aumentano le dimensioni della rete (sia del numero di neuroni sia del numero di hidden layer), l'errore di classificazione risulta assumere valori accettabili, in alcuni casi addirittura ottimali.
Dopo ripetute esecuzioni abbiamo notato che implementando la rete con un numero di neuroni pari a 5000 abbiamo ottenuto la seguenti confusion matrix:



E il seguente grafico:

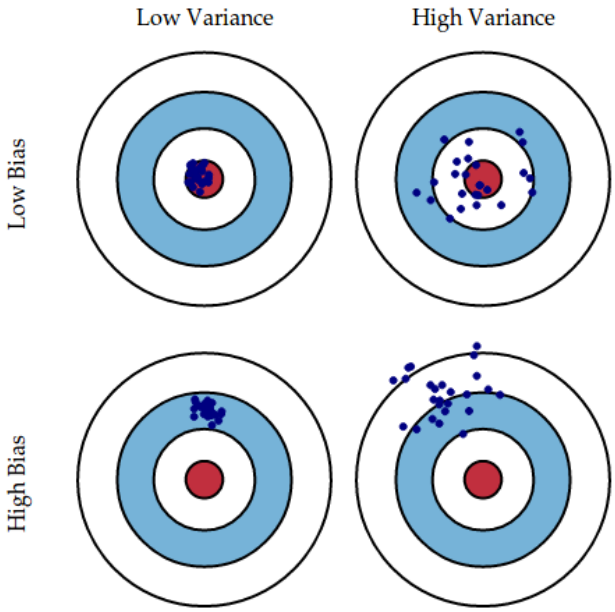


In questo caso nonostante l'errore di classificazione sia sensibilmente elevato, non si tratta di un fenomeno di overfitting.
In sostanza l'algoritmo di ottimizzazione non riesce ad allenare così tanti parametri (è evidenziabile dal fatto che il grafico di errore è "piatto") e si interrompe dopo pochissime epoche (14 nel grafico). La distribuzione dell'errore a questo punto è quasi completamente casuale.

2.5.3 BIAS-VARIANCE TRADEOFF

Prima di tutto bisogna chiarire che questo nostro progetto non riguarda il "Curve Fitting", ovvero l'approssimare un determinato set di punti in una curva, ma il vero e proprio Machine Learning.
Infatti la nostra rete viene allenata con un insieme di dati e poi testata con un altro ben differente!
Nel Curve Fitting viene fornito al modello un solo ed unico gruppo di dati. Nel Machine Learning invece si devo allenare il modello al fine di poter dare dei risultati di fitting accettabili con dati nuovi.
Ora prima di entrare specificatamente nel nostro caso, diamo un occhiata generale ai concetti di bias e variance.
Per bias si intende quanto sbagliata sia la predizione fatta dal modello. Questo è un errore dovuto alla poca risolutezza dell'algoritmo usato oppure dalla scarsità dei dati forniti. Infatti un modello affetto da un elevato bias sottoposto ad un set di dati di test mostrerà chiari segni di underfitting.
Per variance invece si intende quanto la predizione di un dato in input varia nelle diverse run. Questo invece è un errore dovuto alla sensibilità del modello alle piccole fluttuazioni nel training set. Un modello affetto da un elevata variance sottoposto ad un set di dati di test mostrerà chiari segni di overfitting.

Una spiegazione grafica e più intuitiva può essere questa:

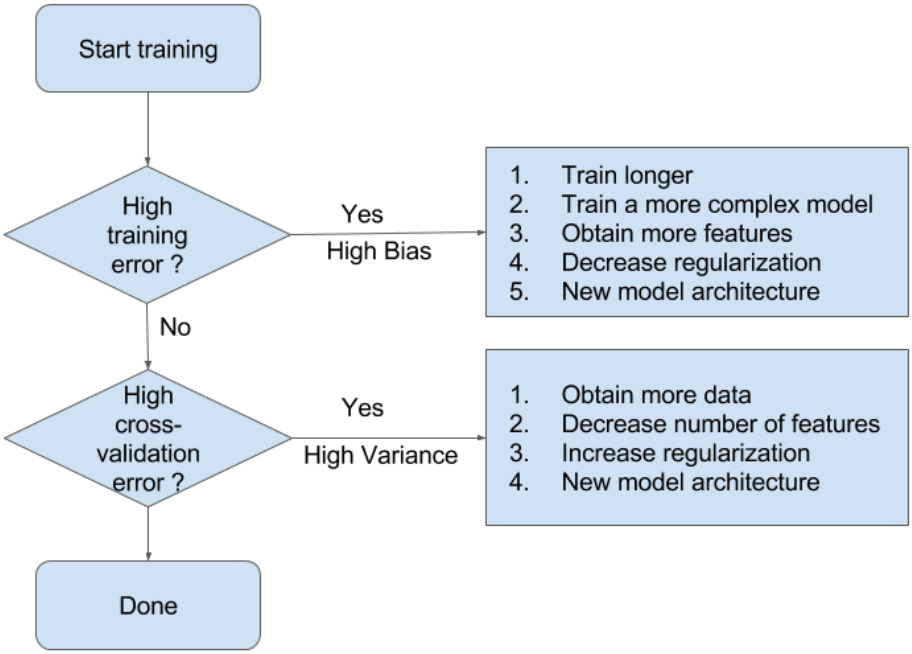


Si nota bene quindi come lo stesso punto in caso di alto bias venga predetto più o meno con le stesse caratteristiche ma completamente errate dalla realtà e come invece in caso di alta variance il punto è una volta predetto con alcune caratteristiche e la volta dopo con altre totalmente differenti.
A queste due componenti di errore bisogna aggiungerne un'altra dovuta all'affidabilità degli errori, il così detto Irreducible error.
Avremo quindi che l'errore totale sarà dato da:

$$Err = bias + variance + irreducible\ error$$

Quindi anche se si possiede un modello di predizione perfetto non avremo mai la certezza di un errore pari a zero a causa appunto dal "rumore" dei dati acquisiti.
Se il "irreducible error" non può essere diminuito dal nostro modello, bias e variance possiamo e dobbiamo gestirli al meglio.
É per questo motivo che in generale viene utilizzato il validation set.
Dividendo in giuste porzioni il nostro dataset possiamo prendere una parte dei dati necessari affinché "correggano il tiro" della predizione del modello fatta sul training set al fine di poter avere dei risultati affidabili durante la fase di test.
Un sintomo del fatto che stiamo incappando in una situazione di overfitting è l'errore molto alto dato dal validation set in rapporto a quello del training a causa di una variance molto elevata.

Ma come si gestisce il trade-off tra bias e variance?



Specificatamente alle reti neurali, molte volte se si è davanti ad un problema di alto bias, è sufficiente aumentare la complessità della rete, ad esempio aumentando il numero di hidden units o il numero di hidden layer. Un altro problema potrebbe essere il numero di features non sufficientemente alto, in questo caso basta aumentare la dimensione del dataset. La regolarizzazione non adatta alla rete può portare ad un underfitting. Diminuire la regolarizzazione vuol dire rendere la funzione costo il più semplice possibile, senza aggiungere valori in più che "regolarizzino" l'andamento della funzione.
Se al contrario invece mi trovo in una situazione di varianza elevata (overfitting) vuol dire che sono nella situazione complementare a quella appena illustrata.
Soluzioni per l'alta varianza sono quindi diminuire le features poiché forse alcune "depistano" la rete neurale, diminuire la complessità del modello usato oppure aggiungere dei termini alla funzione costo così da regolarizzarla.
Nel caso di classificazione binaria, dovuta alla carenza di parametri, vista nella sezione precedente, è possibile notare un'elevata percentuale di classificazione corretta. Questo non significa che la rete abbia ottenuto "risultati ottimi", infatti, se le classi sono sbilanciate, come in questo caso, il 7% ottenuto precedentemente può essere un pessimo risultato visto che la predizione avviene su due delle cinque classi, si ha quindi un alto bias dovuto all'underfitting.
Invece, all'aumentare del numero di hidden units o di hidden layer, in generale del numero di parametri che costituiscono il modello, si ottiene un incremento del valore della varianza, come è solito accadere in un'applicazione basata sul modello ANN, in questo caso utilizzando l'early stopping non è possibile individuare situazioni di overfitting.
Infine, come abbiamo precedentemente discusso, con un un numero di neuroni pari a 5000, la distribuzione dell'errore è quasi completamente casuale, diminuisce cioè la confidenza nella classificazione dei campioni in input.

3 CONCLUSIONI

Stando quindi ai risultati ottenuti dalla nostra rete possiamo concludere affermando che il nostro modello, se avviato con impostazioni ottimali, fornisce un errore molto basso senza distinzioni nel numero di neuroni o nel numero di hidden layer. Più in generale possiamo anche affermare che il modello in sé di Rete Neurale, per quanto possa essere complesso e soprattutto difficile da analizzare in alcune parti, fornisce un modello di machine learning ottimo per il problema di una classificazione supervisionata. Ricordiamo infine che la nostra rete utilizzata e definita "ottimale" è stata frutto di euristiche poiché il problema dell'analisi di una Neural Network è un problema tutt'oggi ancora aperto.

Di seguito riportiamo alcuni dei risultati ottenuti al variare dei parametri (numero di neuroni, numero di hidden layer, funzione di training):

NUMERO NEURONI	NUMERO HIDDEN LAYER	FUNZ. DI ATTIVAZIONE	ALGORITMO DI TRAINING	% ERRORE DI CLASSIFICAZIONE
1	1	logsig	BFGS - Quasi Newton	8,40
2	1	logsig	BFGS - Quasi Newton	8,80
3	1	logsig	BFGS - Quasi Newton	6,90
4	1	logsig	BFGS - Quasi Newton	11,70
5	1	logsig	BFGS - Quasi Newton	7,30
6	1	logsig	BFGS - Quasi Newton	6,70
7	1	logsig	BFGS - Quasi Newton	4,00
8	1	logsig	BFGS - Quasi Newton	6,90
9	1	logsig	BFGS - Quasi Newton	4,10
10	1	logsig	BFGS - Quasi Newton	4,40
11	1	logsig	BFGS - Quasi Newton	5,60
12	1	logsig	BFGS - Quasi Newton	4,80
20	1	logsig	BFGS - Quasi Newton	8,90
50	1	logsig	BFGS - Quasi Newton	5,40

NUMERO NEURONI	NUMERO HIDDEN LAYER	FUNZ. DI ATTIVAZIONE	ALGORITMO DI TRAINING	% ERRORE DI CLASSIFICAZIONE
1	1	logsig	Scaled Conjugate Gradient	6,80
2	1	logsig	Scaled Conjugate Gradient	7,20
3	1	logsig	Scaled Conjugate Gradient	6,20
4	1	logsig	Scaled Conjugate Gradient	8,60
5	1	logsig	Scaled Conjugate Gradient	4,90
6	1	logsig	Scaled Conjugate Gradient	5,00
7	1	logsig	Scaled Conjugate Gradient	5,00
8	1	logsig	Scaled Conjugate Gradient	7,20
9	1	logsig	Scaled Conjugate Gradient	5,40
10	1	logsig	Scaled Conjugate Gradient	3,40
11	1	logsig	Scaled Conjugate Gradient	4,30
12	1	logsig	Scaled Conjugate Gradient	6,50
20	1	logsig	Scaled Conjugate Gradient	5,70
50	1	logsig	Scaled Conjugate Gradient	3,70

NUMERO NEURONI	NUMERO HIDDEN LAYER	FUNZ. DI ATTIVAZIONE	ALGORITMO DI TRAINING	% ERRORE DI CLASSIFICAZIONE
1	2	logsig	BFGS - Quasi Newton	7,60
2	2	logsig	BFGS - Quasi Newton	6,80
3	2	logsig	BFGS - Quasi Newton	7,60
4	2	logsig	BFGS - Quasi Newton	8,90
5	2	logsig	BFGS - Quasi Newton	7,30
6	2	logsig	BFGS - Quasi Newton	7,70
7	2	logsig	BFGS - Quasi Newton	4,50
8	2	logsig	BFGS - Quasi Newton	5,50
9	2	logsig	BFGS - Quasi Newton	6,10
10	2	logsig	BFGS - Quasi Newton	4,50
11	2	logsig	BFGS - Quasi Newton	6,30
12	2	logsig	BFGS - Quasi Newton	7,90
20	2	logsig	BFGS - Quasi Newton	4,50
50	2	logsig	BFGS - Quasi Newton	6,30

NUMERO NEURONI	NUMERO HIDDEN LAYER	FUNZ. DI ATTIVAZIONE	ALGORITMO DI TRAINING	% ERRORE DI CLASSIFICAZIONE
1	2	logsig	Scaled Conjugate Gradient	6,00
2	2	logsig	Scaled Conjugate Gradient	7,10
3	2	logsig	Scaled Conjugate Gradient	6,20
4	2	logsig	Scaled Conjugate Gradient	8,80
5	2	logsig	Scaled Conjugate Gradient	5,70
6	2	logsig	Scaled Conjugate Gradient	7,30
7	2	logsig	Scaled Conjugate Gradient	5,40
8	2	logsig	Scaled Conjugate Gradient	3,70
9	2	logsig	Scaled Conjugate Gradient	6,30
10	2	logsig	Scaled Conjugate Gradient	4,10
11	2	logsig	Scaled Conjugate Gradient	5,00
12	2	logsig	Scaled Conjugate Gradient	5,10
20	2	logsig	Scaled Conjugate Gradient	6,80
50	2	logsig	Scaled Conjugate Gradient	3,90

4 RIFERIMENTI

1 Link al dataset: https://archive.ics.uci.edu/ml/datasets/Page+Blocks+Classification

2 Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

3 http://www.di.uniba.it/~malerba/publications/AAI94.pdf

4 https://en.wikipedia.org/wiki/Artificial_neural_network

5 Link al Neural Network Toolbox: https://www.math-works.com/help/pdf_doc/nnet/nnet_ug.pdf

6 Machine Learning: a Probabilistic Perspective (Kevin Patrick Murphy)

7 Gentle Introduction to the Bias-Variance Trade-Off in Machine Learning (Jason Brownlee): http://scott.fortmann-roe.com/docs/BiasVariance.html

8 Bias-Variance Tradeoff in Machine Learning (Satya Mallick): http://www.learnopencv.com/bias-variance-tradeoff-in-machine-learning/

9 https://en.wikipedia.org/wiki/Synapsis

10 https://it.mathworks.com/help/nnet/ug/multilayer-neural-network-architecture.html

11 https://en.wikipedia.org/wiki/Cross_entropy

12 https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural-network-classifier-training/

13 https://en.wikipedia.org/wiki/Universal_approximation_theorem

14 https://en.wikipedia.org/wiki/Overfitting

15 https://it.mathworks.com/help/stats/confusionmat.html

16 Introduction to Neural Networks for Java, 2nd Edition (Jeff Heaton)

17 https://it.mathworks.com/help/matlab/ref/bar.html

18 http://people.na.infn.it/~murano/SICSI-VIII/Labonia.pdf