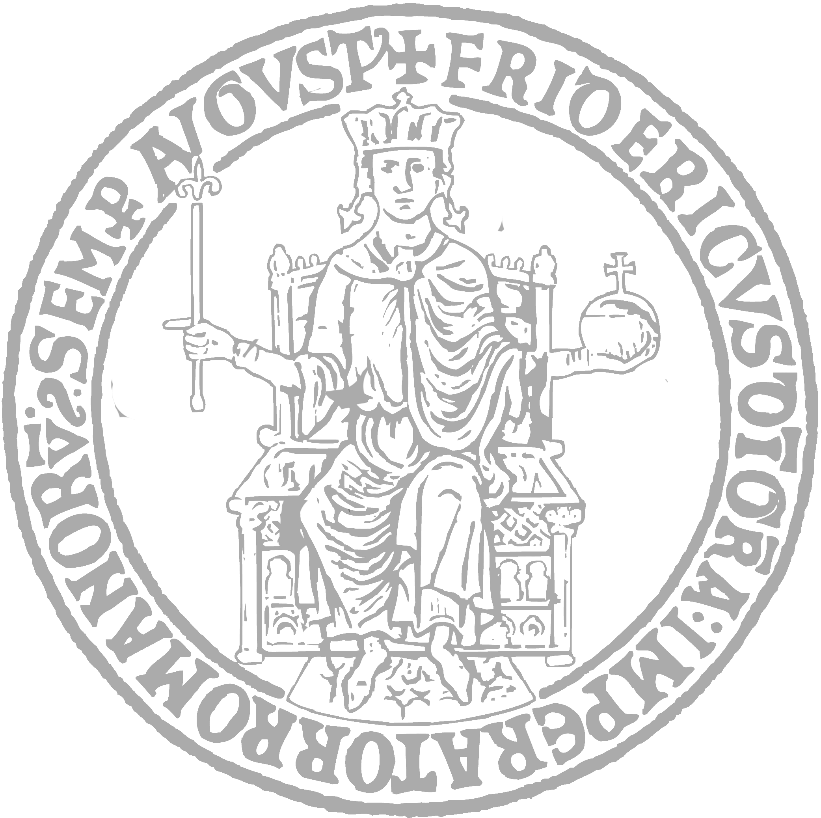
MLeA – Mod. B 2017/2018  
Progetto 5

Prof. Roberto Prevete

Djihad Boukara – N97000275

Emanuele Cioffi – N97000277



# Indice Generale

Sommario

[Indice 2](#_Toc525466973)

[Traccia 3](#_Toc525466974)

[Descrizione del problema 3](#_Toc525466975)

[Dataset 4](#_Toc525466976)

[Implementazione 4](#_Toc525466977)

[Test Plan 5](#_Toc525466978)

[Risultati K-Folding 7](#_Toc525466979)

[Test finali e conclusioni 9](#_Toc525466980)

# Indice delle figure

[Figura 1 Esempio di creazione rete neurale 4](#_Toc525466998)

[Figura 2 - In arancione le "porzioni" di training set, in blue una porzione di validation set 5](#_Toc525466999)

[Figura 3 - SSE: Sigmoide, Identità 7](#_Toc525467000)

[Figura 4 - SSE: TanH, ReLU 8](#_Toc525467001)

[Figura 5 - SSE: TanH, Sigmoide 8](#_Toc525467002)

[Figura 6 - x = Epoche, y = Accuratezza 10](#_Toc525467003)

[Figura 7 - x = epoche, y = errore totale 11](#_Toc525467004)

[Figura 8 - Output finale, 60k esempi d'addestramento, on-line learning, una epoca 12](#_Toc525467005)

# Traccia

**PARTE A.**

• Progettazione ed implementazione di funzioni per simulare la propagazione in avanti di una rete neurale multi-strato con almeno:

* due strati di pesi, con la sigmoide come funzione di output dei nodi interni e l'identità come funzione di output dei nodi di output.

(FACOLTATIVO: permettere all'utente di implementare reti con più di uno strato di nodi interni e con qualsiasi funzione di output per ciascun strato).

• Progettazione ed implementazione di funzioni per la realizzazione della back-propagation per reti neurali multi-strato con almeno:

* due strati di pesi, con la sigmoide come funzione dioutput dei nodi interni e l'identità come funzione di output dei nodi di output, con la somma dei quadrati come funzione di errore.

(FACOLTATIVO: permettere all'utente di realizzare la back-propagation con più di uno strato di nodi interni, con qualsiasi funzione di output per ciascun strato e con qualsiasi funzione di errore derivabile rispetto all'output).

**PARTE B.**

Si consideri come input le immagini raw del dataset mnist. Si ha, allora,un problema di classificazione a C classi, con C=10. Si estragga opportunamente un dataset globale di N coppie (ad esempio, N=200). Si fissi la discesa del gradiente come algoritmo di aggiornamento dei pesi, ed una rete neurale con un unico strato di nodi interni. Si scelgano gli iper-parametri del modello, cioè eta della regola di aggiornamento ed il numerodi nodi interni, sulla base di un approccio di cross-validation k-fold (ad esempio k=5).

Scegliere e mantenere invariati tutti gli altri "parametri" come, ad esempio, le funzioni di output e la funzione di errore. Se è necessario, per questioni di tempi computazionali espazio in memoria, si possono ridurre (ad esempio dimezzarle) le dimensioni delle immagini raw del dataset mnist (ad esempio utilizzando in matlab la funzione imresize).

# Descrizione del problema

Per risolvere il problema di classificazione con apprendimento supervisionato richiesto, si vuole implementare un modello di rete neurale artificiale feed forward, multistrato e full connected, che permetta all’utente di specificare gli iperparametri e le funzioni di attivazione ed errore desiderate, per addestrare così la rete sul dataset Mnist per il riconoscimento di caratteri. L’ambiente di sviluppo scelto per implementare tale modello è Matlab. Si vuole inoltre studiare tramite la tecnica di cross validation K-Fold, il miglioramento o il peggioramento delle performance di riconoscimento della rete neurale al variare degli iperparametri, in particolare learning rate e numero di nodi dell’unico hidden layer, tenendo fissati gli altri parametri, quali numero di strati, funzioni di attivazione ed errore.

Si vuole quindi affrontare un problema di classificazione, dove i target appartengono a 10 classi distinte, utilizzando la backpropagation e la discesa del gradiente per l’addestramento della rete.

# Dataset

Il dataset impiegato per l’addestramento della rete è il database di caratteri scritti a mano Mnist. Tale dataset è composto da 60000 esempi di caratteri numerici (da 0 a 9) corredati di label compilate a mano e di 10000 esempi ulteriori per il testing. Il dataset è pubblicamente disponibile, e per leggerne i dati ed acquisire in campioni in formato leggibile per Matlab, ci si è serviti delle funzioni sviluppate all’università di Stanford (allegate al progetto, si veda il file *loadMNISTImages.m*). Tali funzioni, una volta letto l’intero dataset, restituiscono una matrice di dimensioni 784\*60000 (per il training set, 784\*10000 per il test set), in cui le colonne rappresentano i singoli campioni, e le righe rappresentano le feature dei campioni. Le 784 feature rappresentano il totale dei pixel per singola immagine grande 28x28, e descrivono tramite un valore numerico compreso tra 0 (il bianco) e 255 (il nero) il contenuto del pixel (in scala di grigi). Un singolo campione del training/test test può quindi essere visto come un vettore colonna della matrice generata. Tale vettore viene poi diviso valore per valore, per 255, per normalizzare i valori di input da consegnare alla rete.

# Implementazione

Il cuore della rete neurale è osservabile nella funzione *feedforwardnet.m*, che se chiamata, restituisce la rete neurale con i parametri specificati:

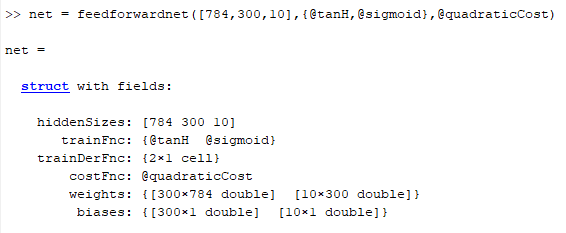


Figura 1 Esempio di creazione rete neurale

In Figura 1 un esempio di rete neurale con un solo strato hidden, specificando come funzione di attivazione per tale strato la Tangente Iperbolica, la Sigmoide come funzione d’attivazione per quello di output e la somma dei quadrati come funzione d’errore (il numero di input e output scelti sono in questo caso quelli necessari al riconoscimento dei dati Mnist).

La rete è stata progettata in modo da permettere un numero arbitrario di strati hidden e di specificare per ciascuno di questi la funzione di attivazione sui nodi interessati. Data la necessità di dover parametrizzare il calcolo della derivata (diverso per ogni funzione di attivazione) necessario nella backpropagation, si era pensato in un primo momento di adoperare uno dei toolbox di Matlab per la valutazione delle espressioni e il calcolo delle derivate su queste; tale soluzione si è rivelata però troppo dispendiosa in termini di tempo, in un contesto dove i tempi di esecuzione sono già lunghi di per sé. Pertanto, si è scelto di adottare una soluzione diversa, ossia di fornire in allegato al codice di base alcune funzioni di attivazione (scelte fra le più comuni: Sigmoide, Identità, tanH, ReLU), e le relative implementazioni delle derivate, che l’algoritmo potrà poi chiamare a seconda dei casi. I codici relativi alle funzioni d’attivazione sono stati allegati al progetto (es: *sigmoid.m)*, insieme alle rispettive funzioni per la derivata (es: *sigmoidDerivative.m)*. L’utente dovrà quindi solo specificare le funzioni d’attivazione scelte per ogni strato tramite l’handle matlab (@nomeFunzione), e il sistema provvederà ad accoppiargli le relative funzioni per il calcolo della derivata.

Per gestire i pesi della rete, è stato creato un array cell di matrici *weights*, che comprende una matrice di pesi per ogni layer della rete. Ognuna di queste matrici conta una riga per ogni nodo del layer relativo alla matrice, ed una colonna per ogni nodo del layer precedente. Osservando quindi tale matrice riga per riga, si otterrà l’insieme dei pesi entranti nel nodo relativo alla riga, se invece la si osserva per colonna, si otterranno tutti gli archi uscenti del nodo relativo alla colonna della matrice (ed entranti nei rispettivi nodi dello strato corrente). I bias sono stati rappresentati come vettori associati ai layer.

Il codice per la forward propagation è visibile nel file *propagate.m*. Il calcolo dei valori d’uscita dei nodi negli strati è stato effettuato strato per strato, tramite prodotti matrice per vettore, recuperando i pesi dalla matrice weights, e i valori dei nodi di output dal vettore corrispettivo nel cell array *out*, sommando il vettore bias relativo al layer.

[Training online – Batch, spiegare cosa si è implementato]

Spiegare questione della funzione d’errore

# Test Plan

Per valutare quali iperparametri e quali combinazioni di funzioni di attivazione risultino più performanti in fase di classificazione, si è realizzata una procedura di cross validation su un sottoinsieme ristretto del training set, per testare la capacità di generalizzare le valutazioni da parte della rete. In particolare, si è utilizzata una strategia di K-Folding, visibile nello script *kfold.m*. Il training set è stato limitato a 500 esempi, e per i test effettuati si è scelto *cinque* come valore per il parametro K, formando così 5 raggruppamenti contenenti 100 esempi ciascuno, di cui 4 utilizzati per l’addestramento (in arancio in Figura 2), ed uno per la validation (in blue in Figura 2), ruotando a turno su tutte e 5 le posizioni.

Figura 2 – K-fold

Il test plan programmato prevede di far spaziare i seguenti parametri, tra i valori di fianco riportati:

* Numero di **nodi** per lo strato interno:
  + 200
  + 300
  + 600
  + 800
* Coppia di **funzioni** **d’attivazione** per strato interno ed output:
  + Tangente Iperbolica (tanH) – Unità Lineare Rettificata (ReLU)
  + Sigmoide – Identità
  + Tangente Iperbolica – Sigmoide
* **Eta** (Learning Rate)
  + 0.7
  + 0.1
  + 0.05
  + 0.01
  + 0.008
  + 0.004

Tali parametri sono stati scelti a seguito di una breve scrematura preventiva, effettuata osservando euristicamente il funzionamento delle funzioni su determinati parametri nelle fasi iniziali di sviluppo della rete; sono quindi stati scelti i parametri che forniscono un quadro d’insieme più descrittivo e che abbracci una variegata serie di casistiche.

Le 4 fasi di addestramento e la singola di validation che contraddistinguono un k-folding sono quindi eseguite per un totale di 3x3x6 volte, incrociando le possibili combinazioni.

Sono poi stati effettuati a scopo dimostrativo, un addestramento di tipo batch ed un online utilizzante questa volta due strati di nodi interni.

# Risultati K-Folding

Di seguito vengono mostrati i grafici che descrivono i dati ottenuti estrapolati dalla cross validation per la ricerca degli iperparametri migliori. I colori delle barre rappresentano il learning rate utilizzato mentre i raggruppamenti sull’asse delle ascisse indicano il numero di hidden nodes utilizzato. Il valore numerico ottenuto rappresenta la deviazione standard dell’errore dato dalla somma dei quadrati. Tali valori rappresentano quindi un valore “medio” di errore sulle rotazioni del k-folding eseguito con quei determinati parametri. I grafici corrispondono alle 3 coppie di funzioni di output utilizzate come campione d’esame.

Figura 3 - SSE: Sigmoide, Identità

Figura 4 - SSE: TanH, ReLU

Figura 5 - SSE: TanH, Sigmoide

Come è possibile osservare dai dati ad un primo sguardo, salta all’occhio innanzitutto come un numero di nodi interno maggiore non si traduca automaticamente in un miglioramento delle performance, ma anzi in alcuni casi viene osservata una generale tendenza positiva (errore più basso), intorno al valore di 200 nodi interni (sebbene questa non sia una regola generale, si veda la Figura 5 dove il minimo viene trovato ad 800 nodi).

Il parametro eta è invece risultato essere molto più delicato da gestire: la rete neurale risulta profondamente sensibile ai cambiamenti di questo, anche in prossimità di valori piuttosto simili. Salta subito all’occhio il grafico in Figura 4 dove valori di eta troppo grandi risultino nel fallimento (colonne segnate con il valore -1) dell’addestramento della rete quando si usano le funzioni Tangente Iperbolica e ReLU. Tale comportamento è risultato evidente anche in altri test effettuati ma non mostrati in questa relazione, specie quando utilizzato ReLU e Identità, ossia le funzioni che hanno codominio fino a più infinito e che inoltrano in attivazione l’input in arrivo; tale comportamento porta gli output dei nodi ed i corrispettivi delta a far crescere i gradienti molto velocemente quando il learning rate risulta essere troppo grande, risultando in aggiornamenti di pesi troppo veloci e grandi, tanto da poter portare all’overflow tali valori. Una funzione come la sigmoide (con codominio tra 0 ed 1) o la Tangente Iperbolica (-1,1), risultano invece utilizzabili con valori di learning rate più alto (0.1) diventando conseguentemente più rapide nell’apprendere.

Il valore migliore rispetto ai dati ottenuti è quindi risultato essere quello della rete addestrata sulla tripla:

* Funzioni: TanH/Sigmoide:
* Eta: 0.1
* Numero di nodi interni: 800

Che ha prodotto un punteggio d’errore medio di 85,360115.

# Test finali e conclusioni

Una volta ottenuti i migliori iperparametri (fra quelli testati), si è provveduto ad effettuare dei test finali per valutare l’effettiva bontà ed efficacia della rete nel riconoscere i caratteri del dataset MNIST.

Una prima misurazione è stata effettuata per mettere a confronto il tasso crescita dell’accuratezza, intesa come numero di esempi del test set classificati correttamente, e il tasso di decrescita dell’errore totale accumulato (sulla quantità di errore ottenuta dopo ogni classificazione di un esempio nel test set), quando si confrontano le modalità di apprendimento On-line e Batch. Sui grafici che seguono sono stati inclusi come valori iniziali di errore e accuratezza (epoca 0), valori d’esempio di rete non ancora addestrata. L’addestramento di tipo Batch è stato effettuato utilizzando una strategia micro-batch, utilizzando come batch size il valore di 10 (un aggiornamento di pesi ogni 10 forward/backpropagation).

Figura - x = Epoche, y = Accuratezza

In Figura 6 è stata valutata l’accuratezza della rete nel riconoscere correttamente gli esempi del test set, utilizzando la rete i cui iperparametri sono stati ottenuti nel paragrafo precedente. Sull’asse delle ascisse sono visibili il numero di epoche d’addestramento, mentre sulle ordinate il valore in percentuale del tasso di accuratezza. Escludendo il caso banale di rete non addestrata (epoca 0), dall’andamento evidenziato dai dati, si evince come l’apprendimento on-line risulti sensibilmente più veloce nel raggiungere un grado di accuratezza maggiore, ma anche come i valori risultino pressoché identici al crescere delle epoche.

Figura - x = epoche, y = errore totale

In Figura 7 sono mostrati invece i valori dell’errore totale, accumulato calcolando l’errore parziale dopo ogni classificazione del test set facendo riferimento alla funzione d’errore somma dei quadrati).

Al netto di qualche oscillazione che colpisce la misura dell’errore, è possibile notare come l’apprendimento online raggiunga valori migliori in anticipo rispetto a quello batch.

In fine, è stato sviluppato un ulteriore test conclusivo, addestrando la rete in modalità on-line, con 1 sola epoca ed utilizzando l’intero training set offerto dal dataset mnist, ossia 60000 esempi. Il test è stato effettuato al solito con l’intero test set offerto, ossia 10000 esempi.



Figura - Output finale, 60k esempi d'addestramento, on-line learning, una epoca

In Figura 8 è possibile osservare l’output ottenuto dal test finale, utilizzando la rete con i parametri mostrati (quelli migliori ottenuti dalle valutazioni del k-fold). L’accuratezza finale ottenuta classificando i diecimila esempi del test set ammonta a 93.83%, ottenuta appunto da un numero di esempi classificati correttamente corrispondente a 9383 su 1000. Il tempo di esecuzione per un addestramento completo ed un test completo ha richiesto circa 7 ore e 20 minuti su un notebook equipaggiato con un processore i7-7700HQ.