MLeA – Mod. B 2017/2018  
Progetto 5

Prof. Roberto Prevete

Djihad Boukara – N97000275

Emanuele Cioffi – N97000277

# Indice

Sommario

[Indice 2](#_Toc524617395)

[Traccia 3](#_Toc524617396)

[Descrizione del problema 3](#_Toc524617397)

[Dataset 4](#_Toc524617398)

[Implementazione 4](#_Toc524617399)

[Test Plan 5](#_Toc524617400)

[Risultati K-Folding 6](#_Toc524617401)

[Test finale e conclusioni 8](#_Toc524617402)

# 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 tale problema di apprendimento supervisionato, 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 tale 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 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. 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 originale. Tale vettore viene poi diviso cella per cella, per 255, per normalizzare i valori di input da dare alla rete.

# Implementazione

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

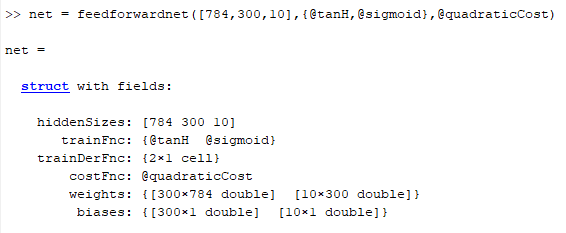


Figura 1 Esempio di creazione rete neurale

In Figura 1 un esempio di rete neurale con un solo strato hidden, specificando come funzione di attivazione per tale strato la tanH, la sigmoide come funzione d’attivazione per quello di output e la somma dei quadrati come funzione d’errore (il numero di input ed 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 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. 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]

# 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 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, che vanno a formare 5 raggruppamenti contenenti 100 esempi ciascuno, di cui 4 utilizzati per l’addestramento, ed uno per la validation, ruotando su tutte e 5 le posizioni.

Figura 2 - In arancione le "porzioni" di training set, in blue la porzione di validation set

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  
   • Coppia di **funzioni d’attivazione** per strato interno ed output  
       ◦ Tangente Iperbolica (tanH) – Unità Lineare Rettificata (ReLU)  
       ◦ Sigmoide – Identità  
       ◦ Sigmoide – Sigmoide  
   • **Eta**  
       ◦ 0.7  
       ◦ 0.1  
       ◦ 0.05  
       ◦ 0.01  
       ◦ 0.008  
       ◦ 0.004

Una volta ottenuti i migliori iperparametri dalla cross validation, si è provveduto ad effettuare un addestramento sul training set completo, ed un test sull’intero test set, utilizzando appunto tali iperparametri (e funzioni d’attivazione). 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 - Errore utilizzando Sigmoide e Identità

Figura 4 - Errore utilizzando Tangente Iperbolica e Unità Lineare Rettificata

Figura 5 - Errore utilizzando Sigmoide per lo strato interno e quell di output

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 si è osservato una generale tendenza positiva (errore più basso), intorno al valore di 200 nodi interni.

Il parametro eta è invece risultato essere molto più delicato, e la rete neurale ne risulta profondamente sensibile ai cambiamenti, anche in prossimità di valori piuttosto vicini. 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 questo documento, specie quando utilizzate ReLU e Identità, ossia le funzioni che hanno dominio 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 se 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 dominio tra 0 ed 1) o la Tangente Iperbolica (-1,1), risultano quindi più veloci nell’appendere, potendo operare con learning rate maggiori.

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

* Funzioni: Sigmoide/Sigmoide:
* Eta: 0.1
* Numero di nodi interni: 200

Essendo i risultati molto vicini sulla stessa terna ma variando i nodi interni da 200 a 300, un ulteriore test ha mostrato come un valore di 250 nodi interni fosse effettivamente il migliore, ottenendo un valore di errore pari a 110,627749. Per cui il test finale verrà effettuato con i parametri di sopra e con 250 hidden nodes.

# Test finale e conclusioni