Relazione Ottimizzazione di Big Data

Costruzione di una Rete neurale

Antonini Davide 0316030

Di Vincenzo Fabio 0310272

Menichelli Alberto 0308559

**README**

Nel file allegato ‘ProgettoOBDReteNeurale’ sono presenti il main file (main.m), la classe dove è descritta la rete Neurale (NeuralNetwork.m), due funzioni di supporto per l’elaborazione dei dataset (createMatrix.m e divideMatrix.m) e cinque dataset elaborati in file ‘.mat’.

Per eseguire il codice bisogna aprire il file ‘main.m’ e cliccare ‘RUN’ oppure scrivere sul Command Window ‘main’. Esiste inoltre una sezione nel main file dove è possibile modificare i valori dei parametri (riga 32). Dopo svariate prove si è comunque arrivati alla conclusione che i parametri inseriti sono i migliori per questo tipo di dataset e rete neurale.

**INTRODUZIONE**

L’obiettivo del progetto è di creare una rete neurale senza l’utilizzo di librerie già esistenti nell’ambiente di programmazione scelto, Matlab, e farlo funzionare con un dataset a scelta. Abbiamo scelto di utilizzare un’architettura, che affronta problemi di Classificazione, con due strati nascosti, costituiti dal numero di neuroni che dipendono dal dataset e dai parametri risultati maggiormente funzionali. I pesi W e i bias B sono stati inizializzati generando valori casuali da una distribuzione normale standard; il learning rate, con cui vengono aggiornati, viene preso come un valore costante compreso tra .

In base alla generalizzazione della rete neurale, a fronte di dataset binari e multiclasse, viene fatta una distinzione sulla funzione di attivazione sull’ultimo strato, nella quale, per dataset binari viene utilizzata la sigmoide, mentre, per dataset multiclasse, viene utilizzata la ReLU (Rectified Linear Unit). Questa distinzione viene fatta perché la sigmoide fornisce una previsione nell’intervallo [0, 1], che può essere interpretata come la probabilità di appartenenza a una delle due classi, mentre la ReLU si adatta perfettamente a problemi multiclasse grazie alla sua non linearità. Per la funzione sigmoidea si è preferito usare la funzione logistica rispetto alla tangente iperbolica per un livello migliore di accuratezza della rete.

Il dataset viene prima ordinato in maniera casuale, dopodiché viene diviso in training set (75%) e test set (25%). I dati delle features vengono normalizzati secondo una normalizzazione standardizzata, per portare la media delle features a zero e la deviazione standard a uno, ed evitare che alcune caratteristiche abbiano un peso sproporzionato durante l'addestramento. Se la deviazione standard risulta pari a zero, caso probabile in dataset con molte features, viene posta pari ad uno, perché vuol significare che la deviazione standard non si discosta dalla media, in modo da evitare la divisione 0/0 ed avere valori NaN all’interno della matrice delle features.

Viene usato il metodo stocastico invece del metodo batch, quindi ad ogni iterazione si usano solo alcuni campioni del training set invece di usarli tutti. Questo perché il metodo stocastico, anche mettendoci quattro volte il tempo del metodo batch, restituisce valori nettamente migliori per l’accuratezza ed è più stabile. Perciò per ognuna delle epoche vengono studiati tot campioni per volta, ripetendo i passaggi di forward\_propagation, calcolo della funzione di perdita e back\_propagation:

* Nel forward\_propagation utilizziamo i pesi per calcolare con quale probabilità, prodotta da una softmax sull’output finale, i campioni presi in esame appartengono alle etichette.
* La funzione di perdita scelta è la Cross Entropy, che misura invece la differenza tra la y vera e la y predetta.
* La back\_propagation viene infine utilizzata per calcolare i gradienti dell'errore rispetto ai pesi e successivamente tali pesi vengono aggiornati tramite questi gradienti. Viene utilizzato il metodo del gradiente stocastico con direzione di discesa l’antigradiente.

Alla fine di ogni epoca viene calcolata l’accuracy del training set, cioè i risultati degli output in confronto alle etichette vere; in seguito, viene valutata la rete neurale tramite il test set, cioè lo si sottopone alla forward\_propagation e infine si calcola l’accuracy dell’intera rete.

**Scelta dei Parametri e Discussione dei risultati**

La scelta dei parametri dipende molto dal dataset adottato. Nel nostro caso abbiamo scelto il dataset binario ‘cod-rna’, per la rilevazione degli RNA non codificanti, basato sulla variazione dell'energia libera di formazione della struttura secondaria prevista, composto da 8 features e 236.901 campioni. Attraverso l’uso della funzione createMatrix(filename) abbiamo estratto due matrici numeriche: una contenente le labels e l’altra le features. Essendo un calcolo computazionalmente oneroso, le matrici numeriche vengono salvate sulla stessa cartella del main come file ‘.mat’ e richiamate in un’altra funzione divideMatrix(). In quest’ultima vengono eseguite tutte le operazioni di caricamento, riordinamento e normalizzazione del dataset e vengono restituite le matrici già divise in training e test set.

Dopo la creazione della rete neurale, esiste una sezione dove è possibile modificare i parametri, tra i quali i neuroni dei due strati nascosti, il learning rate, il numero di epoche totali e il numero di campioni da scansionare in ogni ciclo. Infine, tali parametri vengono inseriti nella rete neurale come delle properties pubbliche.

Nella scelta del numero di neuroni è emersa una regola empirica: il numero di neuroni è legato al numero di features e labels; pertanto, tale numero di norma deve essere proporzionato e adeguato al dataset scelto.

Un learning rate troppo alto implica un aggiornamento più veloce, ma questo potrebbe portare a pesi instabili; con un learning rate più basso invece si raggiunge comunque la stabilità, ma in un numero di epoche maggiore. Dopo vari tentativi la scelta migliore per il learning rate si è rivelata essere pari a . Lo stesso tipo di procedimento è stato adottato per la scelta del numero dei neuroni per gli strati nascosti: non troppo elevato, per non rendere la rete neurale troppo complessa, e nemmeno troppo limitato per non renderla troppo semplice ed evitare in entrambi i casi il fenomeno di overfitting. La condizione ottimale, quindi, è stata scegliere un numero di neuroni pari a 10 per il primo e pari a 2 per il secondo.

Anche la grandezza del batch size deve essere una via di mezzo, grande abbastanza per accelerare l'addestramento ma non troppo per evitare un rallentamento nell'apprendimento di dettagli fini nei dati. Di conseguenza è stato scelto un batch di 16 campioni e si raggiunge la stabilità della rete neurale in meno di 20 epoche.

Per le prove è stato utilizzato un portatile con processore Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz; ogni epoca impiega all’incirca dieci secondi e durante il tempo del processamento della rete neurale, compare una waitbar che indica in percentuale l’andamento del processo. Al completamento del processo la waitbar viene sostituita dal Training Progress che mostra in due grafici l’andamento dell’accuracy e la Loss Function rispetto al numero di epoche.

Durante le prove è successo che l’accuracy e la loss function avessero un andamento piatto, questo perché, avendo generato dei pesi casuali, si è trovato un punto con derivata pari a zero (come ad esempio un punto di minimo o di flesso); perciò questo rende l’aggiornamento impossibile e vengono restituiti valori costanti.

Il Training Progress rileva che l’accuracy, con la scelta dei parametri compiuta, sale intorno al 95%.

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, linea, software

Descrizione generata automaticamente

a) Immagine dell’avanzamento del processo sulla waitbar

b) Immagine del Training Progress con verifica dell’accuracy