*Relazione esame di*

*Neural networks and Deep Learning*

Giuseppe Porcaro N97000362

a.a. 2020/2021

Definizione delle parti del progetto:

* Parte A: Progettazione e implementazione di funzioni per simulare una rete neurale feed forward multistrato con possibilità di implementare reti con più di uno strato interno con qualunque funzione di attivazione. Implementazione dell’algoritmo di *back propagation* per qualunque funzione di attivazione e la possibilità di usare la *somma dei quadrati* o la *cross-entropy* con o senza softmax come funzione di errore.
* Parte B (traccia 3): Considerando come input il dataset mnist, si ha un problema di classificazione a C = 10 classi, per il quale si deve estrarre opportunamente un dataset di N coppie. Si fissi come algoritmo di aggiornamento la *resilient back propagation (RPROP)* e una rete neurale con un unico strato di nodi interni. Si scelgano gli iper-parametri del modello su un approccio di *k-fold*, scegliendo e tenendo fissi tutti gli altri “parametri” della rete.

Tecnologie usate

Entrambe le parti del progetto sono state realizzate nel linguaggio Python in modo tale da poter sfruttare principalmente la libreria Numpy, la quale fornisce supporto ed efficienza computazionale a tutte le principali operazioni su array e matrici di grandi dimensioni, incluse operazioni matematiche, di manipolazione, selezione, trasformazione e algebra lineare.

Per quanto riguarda l’implementazione delle funzioni per la simulazione delle reti, così come per l’algoritmo di back propagation, RPROP, funzioni di attivazione ed errore con relative derivate, questa è stata effettuata partendo da zero seguendo principalmente le metodologie viste a lezione. Altre librerie, le quali sono state usate per compiti più specifici, come ad esempio *Matplotlib* per il plot, verranno discusse nei seguenti paragrafi.

Parte A

Implementazione della rete neurale

La rete è una struttura dati per la quale, alla creazione, bisogna specificare:

* Il numero di valori in input per un singolo dato del dataset.
* Il numero di strati interni (minimo 1).
* Il numero di neuroni per ogni strato hidden (tramite array dato in input: **arrNumNeuroni**).
* La funzione di attivazione per ogni strato hidden (tramite array dato in input: **arrFunAttivazioneHidden**)
* La funzione di attivazione per lo strato di output.

Pesi e bias sono memorizzate in matrici Numpy e l’implementazione è la seguente:

* Primo layer hidden:

Dove m1 è il numero nodi dello strato, d è il numero di dati in input

* Dal secondo all’ultimo layer hidden:

Dove *i* = , *mi* è il numero di nodi dell’i-esimo strato hidden.

* Strato output:

Dove *c* è il numero di neuroni dello strato di output e *mk* è il numero di neuroni dello strato hidden precedente a quello output

Tutti i valori sono generati randomicamente nell’intervallo [-1,1]. Infine nella rete sono state memorizzate in semplici variabili tutte le informazioni base della rete quali numero di layer hidden, il numero di neuroni per ogni layer hidden, il numero di neuroni per lo strato di output e tutte le funzioni di attivazione per ogni strato.

Immagine che contiene testo

Descrizione generata automaticamente

Figura . Funzione costruttrice di una rete neurale multi strato full connected.

Implementazione della simulazione in avanti

La simulazione in avanti della rete è stata implementata nelle funzioni *simulaRete* e *forwardStep* presenti nel file *backPropagation.py.* Entrambe le funzioni hanno lo stesso comportamento tranne per il fatto che la prima restituisce solo l’output della rete, mentre la seconda memorizza e restituisce in array tutti gli output di ogni strato della rete, insieme alle somme pesate dell’input. Nel dettaglio il comportamento della rete segue quello visto a lezione:

1. La prima fase del calcolo riguarda le somme pesate:

Ossia il prodotto scalare tra i pesi della j-esima connessione input dell’i-esimo neurone dell’h-esimo strato con il j-esimo output dello strato precedente h-1. Infine si somma il bias del corrispondente neurone.

1. La seconda fase è la sola applicazione della funzione di attivazione *f* dell’h-esimo strato:

Nel codice questo è tutto ciò e stato implementato verificando se la rete possiede un singolo o più di uno strato hidden. Nel primo caso si calcola direttamente in maniera hard-coded il comportamento per il singolo layer hidden e il layer output, mentre nel secondo caso si itera sul numero di strati della rete e si memorizzano, in array con dimensione pari al numero di strati hidden, tutte le componenti usate per il calcolo del comportamento. Queste serviranno per calolare poi il gradiente della funzione di errore nella back propagation.

Immagine che contiene testo

Descrizione generata automaticamente

Figura . Forward step di una rete neurale. Calcola il comportamento della rete e memorizza gli output di ogni strato della rete, insieme alle somme pesate degli input, per il calcolo del gradiente nella backPropagation.

Come già specificato la funzione che simula soltanto la rete, restituendo l’output calcolato dalla rete, effettua lo stesso calcolo della funzione forwardStep in figura, tranne per il fatto che non memorizza output e somme pesate per usarli nella backpropagation.

Algoritmo back propagation e aggiornamento pesi

Ciclo di learning

Learning di una rete neurale

-Dataset usato

-Preprocessing effettuato

-Valutazione errore e precisione della rete con e senza uso di softmax e relu, magari variando anche il numero di neuroni, il learning rate e il numero di strati.