Advanced Machine Learning: Assignment #2

Fabrizio D'Intinosante

Università degli Studi di Milano Bicocca — October 27, 2019

Introduzione

Il *dataset* contiene immagini rappresentanti le lettere P-Z in formato 28x28. Queste immagini sono suddivise in *training set* e *test set* in quantità **14000** e **8800**. L'obiettivo dell'elaborato è quello di realizzare un classificatore avvalendosi solo di *layers fully connected* evitando l'uso di *layers convolutionals*, un *autoencoder* (e di verificare le sue capacità di ricostruzione delle immagini) e per ultimo, come procedimento opzionale, utilizzare la parte di *encoder* come prima parte di un classificatore sempre costruito utilizzando solo *layers fully connected*.

1 Esplorazione e preprocessing

Come detto, il dataset è composto da immagini 28x28 rappresentanti le lettere P-Z in scala di grigi; le 11 classi presenti sono piuttosto bilanciate.

Come prima azione di *preprocessing* si è proceduto a riscalare i pixel dell'immagine dall'intervallo [0, 255] a quello [0, 1] per ridurre così il loro campo di escursione lasciando inalterata la sfumatura di colore generata dai pixel. In secondo luogo si è proceduto ad effettuare un *reshape* della immagini, convertendone il formato dall'iniziale 28x28 ad un più semplice 1x784, una procedura standard e necessaria per poter sottomettere le immagini ai *layers fully connected* della **rete neurale**.

Successivamente si è proceduto a suddividere il *training set* originale in un ulteriore partizione, il *validation set* con una proporzione 70% e 30%.

NB: La tecnica di *sampling* utilizzata è quella del *sampling* stratificato così da garantire la proporzionalità delle diverse lettere nelle due partizioni.

Infine, con lo scopo di migliorare le capacità di generalizzazione del classificatore si è proceduto ad applicare una tecnica di *data augmentation*, nello specifico una *elastic transformation*. Questa tecnica consiste nel prendere in input un'immagine e nel distorcerla come si vede dall'esempio in fig. 1, così da creare ancora più versioni di una stessa lettera. Con questa applicazione si è triplicato il *training set*, oltre a renderlo ancora più vario, mentre il *validation set* è rimasto inalterato.

Per quanto riguarda la variabile y, ovvero le etichette riferite alle diverse lettere, si è proceduto a ricodificarle dal loro formato originale a quello *one-hot encoding*, una procedura standard per permettere la creazione di un classificatore multiclasse.

(a) Formato originale (b) Formato distorto

Figure 1: Differenza tra lettera originale e distorta da *elastic transformation*

NB: Il fatto che le lettere presenti nel dataset siano P-Z porta le etichette originali ad andare da 16 a 26, provocando quindi nella creazione del vettore *one-hot* la formalizzazione di 15 zeri precedenti alle 11 classi poi effettivamente presenti. Per ovviare a questo problema si è proceduto, prima della ricodifica, a sottrarre 16 a tutte le classi e a risommarlo successivamente in fase di valutazione con il *validation set*.

2 Modelli

Come detto in precedenza si è proceduto innanzitutto a realizzare un classificatore composto semplicemente di *layers fully connected*, un *autoencoder* e un classificatore composto per la prima parte dall'*encoder* trainato in precedenza e da una serie di *layers fully connected* in coda così da permettere la classificazione.

2.1 Classificatore

Il classificatore realizzato per assolvere al task di classificazione multiclasse è stato così strutturato:

- 20 epoche si sono dimostrate sufficienti per ottenere buone *performances* anche per evitare il rischio di *overfitting*;
- 3 *hidden layers fully connected* da 512, 256 e 128 neuroni. Questa dimensionalità ha permesso di gestire efficacemente la mole di dati di *train* e ottenere un apprendimento piuttosto rapido anche considerando l'impatto sulla dimensionalità del *training set* della *data augmentation*;
- funzione di attivazione ReLU sugli *hidden layers*, uno standard che dopo diverse prove si è rivelato essere il migliore;
- 3 *dropout layers* con un *rate* di 0.4 l'uno tra ogni *hidden layer*, più una *regularization* L1 sul primo *hidden layer* di 1e-05 per penalizzare leggermente così i pesi più piccoli ed evitare *overfitting*;
- funzione di attivazione *softmax* sull'*output layer*, necessaria per la classificazione multiclasse;
- ottimizzatore Adam, piuttosto rapido e con buone prestazioni essendo un ottimizzatore adattivo;
- *loss function categorical crossentropy*, anche questa scelta standard per quanto riguarda la classificazione multiclasse;
- *batch size* di 1024, computazionalmente più onerosa ma più rapida in realizzazione così da velocizzare processo di *training*.

2.2 Autoencoder

Avendo sottoposto il *training set* ad una procedura di *data augmentation* si è notato un certo aumento di difficoltà dell'*autoencoder* nel riprodurre l'input iniziale. Per evitare il rischio di *underfitting* è stato quindi necessario aumentare il numero di epoche ma anche il numero di parametri del modello. La struttura dell'*autoencoder* è presentata di seguito:

- 500 epoche, necessarie come detto affinchè ci fosse convergenza;
- 5 *hidden layers fully connected* in *encoding* e 5 in *decoding* con dimensionalità 512, 512, 256, 128 fino a raggiungere la rappresentazione più densa a 64 neuroni. Si è deciso di fermare l'elaborazione a 64 neuroni proprio per evitare l'*underfitting* poichè dopo numerose prove di elaborazione fino a 32 neuroni ci si è resi conto che la convergenza non sarebbe stata raggiunta a parità degli altri parametri;
- funzione di attivare ReLU su tutti gli *hidden layers* fatta eccezione per il *layer* finale di decodifica per cui è stata utilizzata una funzione Sigmoide, questo per ricondurre l'*output*, ovvero i pixel delle nostre immagini, ad un formato [0, 1];
- ottimizzatore Adam, anche per l'autoencoder rivelatosi il migliore dopo diverse prove;
- *loss function binary crossentropy*, siccome il valore dei nostri pixel, essendo stati stati scalati, si muovono nell'intervallo [0,1];
- *batch size* di 1024, per le stesse ragioni indicate sopra.

2.3 Classificatore con encoder

La creazione del classificatore preceduto dall'*encoder* è consistita semplicemente "nell'attaccare" l'*encoder*, addestrato all'interno dell'*autoencoder*, fissandone quindi i pesi e rendendoli non più "addestrabili", al

	precision	recall	f1-score	support
16	0.97	0.93	0.95	388
17	0.92	0.94	0.93	379
18	0.94	0.93	0.93	404
19	0.96	0.98	0.97	399
20	0.97	0.97	0.97	401
21	0.95	0.92	0.93	389
22	0.90	0.92	0.91	381
23	0.97	0.98	0.98	398
24	0.98	0.95	0.96	397
25	0.91	0.92	0.92	396
26	0.94	0.98	0.96	268
accuracy			0.95	4200
macro avg	0.95	0.95	0.95	4200
weighted avg	0.95	0.95	0.95	4200

Figure 2: Report classificatore semplice

Figure 3: Risultato ricostruzione autoencoder

classificatore semplice presentato come primo modello. Le specifiche delle due componenti sono state pertanto già illustrate.

2.4 Valutazione dei modelli

I modelli di classificazione, quello semplice e quello inizializzato tramite *encoder*, performano entrambi piuttosto bene come si può vedere per il primo in fig. 2 e per il secondo in fig. 4. Le prestazioni dei due modelli per quanto riguarda *accuracy* ed *average f1-score* differiscono di appena un punto percentuale in favore del secondo, ma probabilmente non si tratta di una differenza statisticamente significativa. Trattandosi di immagini, comunque, le prestazioni raggiunte sono piuttosto buone anche considerando il fatto che non ci si è avvalsi di *layers convolutionals* ovvero i più adatti all'elaborazione di immagini. Per quanto riguarda l'*autoencoder* invece, presenta capacità di ricostruzione delle immagini piuttosto buone. Come si può infatti vedere in fig. 3 il modello, anche grazie al lavoro di *data augmentation* sembrerebbe generalizzare bene durante la ricostruzione, correggendo le imperfezioni presenti nelle lettere e tendendo a ricostruirle mediando tra i diversi stili di scrittura appresi.

3 Conclusioni

In conclusione, si può affermare che il modello vincitore risulti essere quello presentato in fig. 4, ovvero il modello che unisce *encoder* e classificatore semplice poichè raggiunge le prestazioni più elevate in *accuracy* ed *average f1-score*.

	precision	recall	f1-score	support
16	0.99	0.96	0.98	388
17	0.94	0.94	0.94	379
18	0.91	0.97	0.94	404
19	0.98	0.98	0.98	399
20	0.98	0.97	0.97	401
21	0.94	0.94	0.94	389
22	0.93	0.92	0.92	381
23	0.98	0.98	0.98	398
24	0.99	0.95	0.97	397
25	0.94	0.94	0.94	396
26	0.95	0.98	0.97	268
accuracy			0.96	4200
macro avg	0.96	0.96	0.96	4200
weighted avg	0.96	0.96	0.96	4200

Figure 4: Report classificatore con encoder