Università degli Studi dell'Aquila
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

---

Corso di Machine Learning

Homework 2

Angelo Damiani

Anno Accademico 2017/2018

Indice

Capitolo 1

Strumenti utilizzati

1.1 Linguaggio e librerie utilizzate

Per la realizzazione di questo homework è stato utilizzato il Python e sono state utilizzate le seguenti librerie:

- Tensorflow: una libreria open source originariamente sviluppata dai ricercatori di Google per la ricerca sul deep learning. Fornisce delle API per diversi linguaggi di programmazione (Python, C++, Java, ...). Nel contesto del homework è stata utilizzata per la compatibilità con la libreria Keras;
- Keras: è una high level API scritta in Python per diverse piattaforme (TensorFlow,CNTK e Theano). Originariamente è stata sviluppata per testare e sperimentare reti neurali nel più breve tempo possibile. In questo homework è stato il core dell'applicativo, mi ha permesso di descrivere una rete neurale in meno di una decina di righe di codice;
- Numpy: è una libreria matematica principalmente utilizzata per calcoli vettoriali e matriciali. Come per tensorflow, anche questa libreria è stata utilizzata per la compatibilità con Keras.

1.2 Repository

Come da consegna è stata utilizzata la repositori UCI denominata "Letters". Consta di 20000 campioni di lettere (da cui sono state estratte 16 caratteristiche fondamentali) con le proprie feature e il corrispettivo valore letterale. Come si vedrà la repository è stata divisa in 10 parti per consentire la k-fold validation (con k = 10).

1.3 [Extra] EMNIST e python-mnist

Avendo iniziato a sviluppare l'homework prima dell'effettiva consegna avevo ricercato da me una repository e dei campioni. Mi sono imbattuto nella repository del NIST (National Institute of standards and Technologies) per i caratteri scritti a mano. La repository constava in più di 140000 campioni (120800 categorizzati come training set e 20800 categorizzati come test set) in un formato bit stream che è stato letto con l'ausilio della libreria **python-mnist**. Ciascun campione consta di 784 features (una feature per pixel, per un'immagine 28x28, codificato in bianco e nero, da 0 a 255) e di un valore numerico da 1 a 26 (da "A" a "Z"). Ho deciso di aggiungere i file (main in Python e file di repository) al restante materiale relativo a questo homework.

Capitolo 2

Rete Neurale e preprocessing

2.1 Struttura della rete neurale

Come detto nel capitolo precedente, era stato iniziato l'homework su dei dati del NIST. Dopo diversi test per ottenere un risultato decente in termini di errore con dei dati grezzi come i pixel, era stata sviluppata una rete neurale composta da 1 input layer, 4 hidden layer e 1 output layer. Ciascun layer, dall'input all'output, aveva rispettivamente le seguenti unità: 100, 200, 300, 200, 100, 26. Avendo avuto un risultato relativamente buono con i dati grezzi, ho deciso di "riciclare" la rete neurale per dei dati raffinati come quelli della repository UCI.

2.1.1 Activation functions e ricerca dell'ottimo

Per ciascuna unità dell'input layer e degli hidden layer della rete neurale è stata scelta come funzione di attivazione una rectifier ($f(x) = max(0, x)$) mentre per l'output layer è stato scelto un sigmoide ($f(x) = \frac{1}{1+e^{-x}}$). Infine è stato scelto come metodo di ottimizzazione un metodo di stochastic gradient descent (nel particolare Adaptive Moment Estimation, ADAM). Queste scelte sono state prese dopo differenti test, tutti eseguiti per la repository EMNIST.

2.2 Preprocessing

A parte la lettura dei dati con la rimozione delle virgole, l'unico preprocessing effettuato per i dati della repository UCI è stato il cambiamento della lista delle feature in un array numpy (semplicemente realizzato dal costruttore degli array di numpy) e il cambiamento delle label di ogni campione in un "one hot vector". Per effettuare ciò, ciascuna lettera è stata convertita in un intero da 1 a 26 (dove "A" sta per 1 e "Z" sta per 26). Una volta convertito in un intero, il valore identifica (dopo aver sottratto 1) la posizione del one hot vector da porre a 1.
(Extra) Lo stesso, tranne la conversione a intero, è stato effettuato per il cambiamento dei label della repository EMNIST.

Capitolo 3

Codice e salvataggio

3.1 k-Fold validation

La k-Fold validation, con k = 10, è stata realizzata mediante un ciclo. Tale ciclo, indicizzato con "i", ha permesso di creare una sorta di sliding window all'interno del data set selezionando il test set e considerando tutto ciò non selezionato dalla finestra come parte del training set. Il core principale della k-Fold validation è presentata nella figura 3.1.

```
k = 10
lenTestSet = int(len(dataSet)/k)

for i in range (k):

    testSet = dataSet[i*lenTestSet:(i+1)*lenTestSet]
    labelTestSet = labelDataSet[i*lenTestSet:(i+1)*lenTestSet]

    trainingSet = dataSet[:i*lenTestSet] + dataSet[(i+1)*lenTestSet:]
    labelTrainingSet = labelDataSet[:i*lenTestSet] + labelDataSet[(i+1)*lenTestSet:]
```

Figura 3.1

3.2 Costruzione della rete neurale, training e testing

La creazione della rete, il training e il testing è stato effettuato per mezzo di keras. Per prima cosa è stato definito il modello sequenziale della rete neurale; in seguito è

stato scelto il numero di percettroni per layer nella rete e le funzioni di attivazioni di quest'ultimo. A questo punto è stata costruita la rete mediante le funzioni in figura 3.2. A questo punto è stata "compilata" la rete neurale selezionando il metodo di

```
# Number of units (perceptrons) per layer
n0 = 100
n1 = 200
n2 = 300
n3 = 200
n4 = 100
n5 = nclasses

# Network definition.
# Have been defined 6 layers (1 input, 1 output, 4 hidden)
# First 5 with activation function ReLU (rectifier linear unit): f(x) = max(0,x)
# Last one with activation function Sigmoid: f(x) = 1/(1+e^(-x))
model = Sequential()
model.add(Dense(n0, input_dim=images_dim, activation='relu', kernel_initializer="uniform"))
model.add(Dense(n1, activation='relu', kernel_initializer="uniform"))
model.add(Dense(n2, activation='relu', kernel_initializer="uniform"))
model.add(Dense(n3, activation='relu', kernel_initializer="uniform"))
model.add(Dense(n4, activation='relu', kernel_initializer="uniform"))
model.add(Dense(n5, activation='sigmoid', kernel_initializer="uniform"))
```

Figura 3.2

ottimizzazione e la funzione da ottimizzare (figura 3.4). A questo punto è stata

```
# Configure network to use as:
# Optimizing function : cross entropy
# Optimizer : Stochastic Gradient Descent (Adaptive Moment Estimation, ADAM)
# What I want to optimize: accuracy of predictions
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figura 3.3

allenata la rete (con 220 epochs). Una volta allenata è stata testata e i risultati salvati in un array per poi essere valutati come predizioni errate o predizioni corrette, figura **??**. La percentuale di errore per ogni ciclo viene appesa in una lista e, alla fine della k-Fold validation, viene fatta una media delle percentuali di errore.

```
# Training of the network.
model.fit(images, labels, epochs=220, batch_size=100,  verbose=2)


# Here start the testing of the network
test_images = np.array(test_images)
test_labels = test_labels

# Testing
predictions = model.predict(test_images)
prd = []
for pred in predictions:
    prd.append(label_deconverter(pred))

# Wrong and right categorization counter
err_counter = 0
right_counter = 0
for j in range(len(prd)):
    #print("Prediction "+str(i)+": '"+map_character[prd[i]]+"' instead of '"+test_labels[i]+"'.")
    if(prd[j] == map_character[test_labels[j]]):
        right_counter = right_counter + 1
    else:
        err_counter = err_counter + 1
```

Figura 3.4: fig:nnfitpred

3.3 Salvataggio del modello in JSON

Uno dei modelli allenati nelle k iterazioni effettuate è stato salvato in un file `.json` per mezzo di una funzione built-in di keras illustrata in figura 3.5. In questo modo

```
if i==0:
    # Saving model
    model_json = model.to_json()
    with open("model.json", "w") as json_file:
        json_file.write(model_json)
```

Figura 3.5

è stato possibile salvare la rete per usi futuri

Maggiori dettagli

Per maggiori dettagli sono stati aggiunti dei commenti sul codice per una maggiore comprensione.

Capitolo 4

Risultati e link utili

4.1 Risultati e confronto con i raw data di EMNIST

Dopo aver costruito, allenato e testato la rete è giunto il momento di tirare le somme. La rete, come detto in precedenza, è stata utilizzata per entrambe le repository, l'unica differenza era il numero di input nella rete. Per quanto riguarda la repository UCI, l'errore medio dopo la k-Fold validation è del **3.4%**; il massimo errore riscontrato è stato del 5.1% mentre il minimo è stato il 2.1%. I risultati peggiorano pesantemente con la repository NIST. Con i dati grezzi il risultato migliore è stato un **10%** d'errore (anche se in questo caso non è stata effettuata k-Fold validation). Questo ha instillato in me la voglia di approfondire i metodi di estrazione di feature fondamentali a partire da dati grezzi come possono essere i pixel.

4.2 Link utili

- `keras.io`: link alla libreria Keras;
- `www.numpy.org/`: link alla libreria numpy;
- `archive.ics.uci.edu/ml/datasets.html`: link alla repository UCI;

- `www.nist.gov/itl/iad/image-group/emnist-dataset`: sito internet del NIST (in particolare la repository EMNIST).