UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA
MACROAREA DI INGEGNERIA

Metodi di Ottimizzazione per Big Data

Relazione Progetto Finale

0350048	0350111	0350110
Marco Aragona	Sara Malaspina	Silvia Perelli

Anno Accademico 2023/2024

Indice

Introduzione

Il nostro progetto si pone l'obiettivo di realizzare una rete neurale in grado di risolvere un problema di regressione a partire da tre diversi dataset:

- Air Quality
  - Campioni: 9358
  - Features: 15
  - Descrizione: contiene la media oraria delle risposte di un dispositivo multisensore di gas installato in una città italiana
  - Target: quantità di benzene nell'aria

- Seoul Bike Sharing Demand
  - Campioni: 8760
  - Features: 13
  - Descrizione: contiene il numero di biciclette pubbliche affittate ogni ora a Seoul, nei giorni feriali e festivi, con le corrispondenti condizioni atmosferiche
  - Target: numero di biciclette affittate all'ora

- Housing
  - Campioni: 20641
  - Features: 10
  - Descrizione: contiene le caratteristiche delle case in California provenienti da un censimento svolto nel 1990
  - Target: prezzi delle case

Ogni dataset è stato preprocessato prima di effettuare l'addestramento della rete neurale per eseguire una pulizia e una codifica dei dati. Successivamente sono state valutate le performance della rete mediante due metriche: Root Mean Squared Error (RMSE) e Mean Absolute Errore (MAE). Abbiamo scelto di osservare entrambe le metriche in quanto RMSE permette di penalizzare maggiormente errori più grandi, ma è sensibile agli outlier, a differenza di MAE che fornisce una stima più stabile e meno influenzata da questi ultimi.

Rete Neurale

La rete neurale realizzata si compone di un livello di input con tanti neuroni quante sono le features del dataset, uno o più livelli nascosti con la relativa funzione di attivazione e un livello di output con un solo neurone e una funzione di attivazione lineare, al fine di predire un valore reale.

Cross Validation

L'architettura della rete neurale è stata determinata mediante una tecnica di cross validation, implementata nell'omonima funzione python, al fine di valutare la migliore combinazione di:

- Configurazione dei livelli nascosti ([input_layer, 128, 1], [input_layer, 128, 64, 1], [input_layer, 64, 32, 1], [input_layer, 128, 64, 64, 1], [input_layer, 128, 64, 32, 1])
- Funzione di attivazione nei livelli nascosti (ReLU, Tanh)
- Funzione di regolarizzazione (nessuna, L1, L2)
- Parametro di regolarizzazione (0.1, 0.01, 0.001)
- Numero di epoche (80)

Ogni modello è stato addestrato sul training set ed è stata utilizzata la metrica RMSE sul validation set per valutare le performance delle varie configurazioni e trovare la migliore. Infine, è stata ricalcolata la metrica del modello migliore sul test set.

Per ottimizzare la fase di cross validation, abbiamo scelto di utilizzare dei thread tramite ThreadPoolExecutor per valutare in parallelo la scelta della funzione di regolarizzazione, per ogni combinazione di tutti gli altri parametri.

Addestramento e Backpropagation

La rete è stata addestrata usando il metodo del gradiente stocastico, suddividendo il training set in minibatch di dimensione variabile in base al dataset e usando un diminishing stepsize con learning rate iniziale pari a 0.01. La funzione di perdita che abbiamo utilizzato è l'errore quadratico medio (MSE) a cui è stato aggiunto un termine di regolarizzazione, se previsto. Abbiamo inizializzato i parametri con il metodo He nel caso in cui è stata usata la ReLU come funzione di attivazione dei livelli nascosti, mentre nel caso della tangente iperbolica abbiamo utilizzato il metodo Xavier/Glorot. Le due inizializzazioni prevedono che i pesi vengano estratti da una distribuzione Gaussiana con $\mu = 0$ e $\sigma^2 = \frac{2}{n_{in}}$ nel caso di He e con $\mu = 0$ e $\sigma^2 = \frac{2}{n_{in}+n_{out}}$ nel caso di Xavier, dove n_{in} e n_{out} sono rispettivamente il numero di input e output del livello. L'aggiornamento dei parametri è stato effettuato con la tecnica del Momentum con parametro $\beta = 0.9$, al fine di accelerare il metodo del gradiente stocastico. È stata inoltre usata la tecnica del gradient clipping per evitare il fenomeno dell'exploding gradient. Per il calcolo del gradiente è stata implementato l'algoritmo di Backpropagation, attraverso due funzioni python: *forward_propagation* e *backward_propagation*.

Datasets e Risultati Finali

Preprocessamento

La fase di preprocessamento consiste nei seguenti passi:

1. Lettura del file csv contenente il dataset
2. Gestione di valori nulli o mancanti
3. Eliminazione dei duplicati
4. Suddivisione del dataset in training (65%), validation (15%) e test set (20%)
5. Analisi e trasformazione degli attributi
6. Codifica degli attributi categorici con one-hot encoding
7. Normalizzazione degli attributi numerici

AirQuality

In Fig. 1 sono elencate le caratteristiche del dataset originario. Al suo interno i dati mancanti erano rappresentati dal valore -200; per gestirli più facilmente siamo andati inizialmente a sostituirli con valori nulli. Abbiamo poi rimosso gli attributi NMHC(GT), Unnamed: 15, Unnamed: 16, in quanto composti principalmente da entry nulle, e i campioni che presentavano valori nulli in ogni feature. Mentre i valori nulli di *CO(GT)*, *NOx(GT)* e *NO2(GT)*, essendo in quantità minore, sono stati sostituiti con la mediana. Osservando che i dati erano stati raccolti nell'arco dello stesso anno, dell'attributo *Date* abbiamo mantenuto solamente il mese, mentre dell'attributo *Time* abbiamo mantenuto l'ora, in quanto non erano presenti entry in cui i secondi e i minuti fossero diversi da zero. In questo dataset non erano presenti attributi categorici, infatti è stata effettuata soltanto la normalizzazione degli attributi numerici. L'attributo target è *C6H6(GT)*.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9471 entries, 0 to 9470
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           9357 non-null   object
 1   Time           9357 non-null   object
 2   CO(GT)         7674 non-null   float64
 3   PT08.S1(CO)    8991 non-null   float64
 4   NMHC(GT)       914 non-null    float64
 5   C6H6(GT)       8991 non-null   float64
 6   PT08.S2(NMHC)  8991 non-null   float64
 7   NOx(GT)        7718 non-null   float64
 8   PT08.S3(NOx)   8991 non-null   float64
 9   NO2(GT)        7715 non-null   float64
 10  PT08.S4(NO2)   8991 non-null   float64
 11  PT08.S5(O3)    8991 non-null   float64
 12  T              8991 non-null   float64
 13  RH             8991 non-null   float64
 14  AH             8991 non-null   float64
 15  Unnamed: 15    0 non-null      float64
 16  Unnamed: 16    0 non-null      float64
dtypes: float64(15), object(2)
memory usage: 1.2+ MB
```

Fig. 1 – Air Quality

Per l'addestramento abbiamo utilizzato dei mini-batch di taglia 32. La configurazione migliore della rete neurale restituita dalla cross validation ha tre livelli nascosti, il primo con 128 neuroni, il secondo e il terzo con 64 neuroni [11, 128, 64, 64, 1], regolarizzazione L2 con lambda pari a 0.1, funzione di attivazione dei livelli nascosti tanh e 80 epoche. I valori di RMSE e di MAE sul validation set e sul test set sono mostrati in Fig. 2, mentre in Fig. 3 è rappresentato il valore della funzione di perdita, durante l'addestramento, al crescere delle epoche.

```
The RMSE on validation set is: 0.1282374402336333
The MAE on validation set is: 0.060236500382265495

The RMSE on test set is: 0.12673047352159547
The MAE on test set is: 0.06080819530720003
```

Fig. 2 – Metriche Air Quality

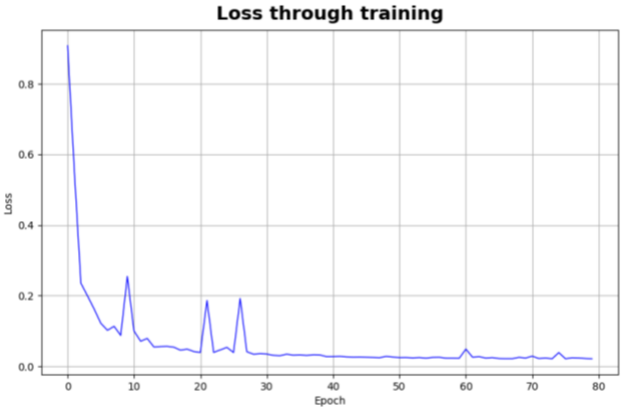

Fig. 3 – Loss Air Quality

Seoul Bike Sharing Demand

In Fig. 4 sono elencate le caratteristiche del dataset originario. Osservando che i dati erano stati raccolti nell'arco di un anno, dell'attributo *Date* abbiamo mantenuto solamente il mese. In questo dataset non erano presenti valori nulli. L'attributo target è *Rented Bike Count*.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   8760 non-null   object
 1   Rented Bike Count      8760 non-null   int64
 2   Hour                   8760 non-null   int64
 3   Temperature            8760 non-null   float64
 4   Humidity               8760 non-null   int64
 5   Wind speed             8760 non-null   float64
 6   Visibility             8760 non-null   int64
 7   Dew point temperature  8760 non-null   float64
 8   Solar Radiation        8760 non-null   float64
 9   Rainfall               8760 non-null   float64
 10  Snowfall               8760 non-null   float64
 11  Seasons                8760 non-null   object
 12  Holiday                8760 non-null   object
 13  Functioning Day        8760 non-null   object
dtypes: float64(6), int64(4), object(4)
memory usage: 958.2+ KB
```

Fig. 4 – Seoul Bike Sharing Demand

Per l'addestramento abbiamo utilizzato dei mini-batch di taglia 64. La configurazione migliore della rete neurale restituita dalla cross validation ha un livello nascosto con 128 neuroni [18, 128, 1], regolarizzazione L1 con lambda pari a 0.01, funzione di attivazione del livello nascosto tanh e 80 epoche. I valori di RMSE e di MAE sul validation set e sul test set sono mostrati in Fig. 5, mentre in Fig. 6 è rappresentato il valore della funzione di perdita, durante l'addestramento, al crescere delle epoche.

```
The RMSE on validation set is: 232.46123605784555
The MAE on validation set is: 150.170578605746

The RMSE on test set is: 251.36615236177371
The MAE on test set is: 158.55385442561456
```

Fig. 5 – Metriche Seoul Bike Sharing Demand

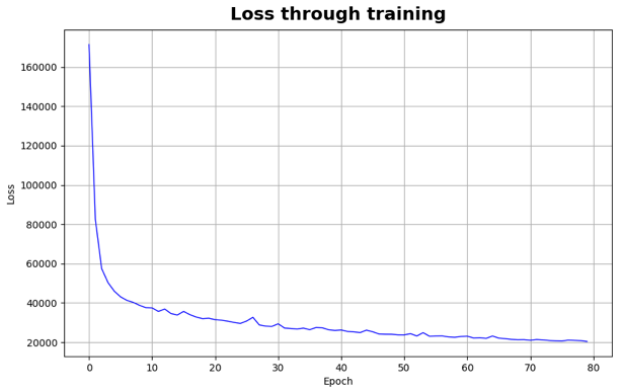

Fig. 6 – Loss Seoul Bike Sharing Demand

Housing

In Fig. 7 sono elencate le caratteristiche del dataset originario. In questo dataset abbiamo rimosso alcuni campioni a causa della presenza di valori nulli nell'attributo *total_bedrooms*. L'attributo target è *median_house_value*.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Fig. 7 – Housing

Per l'addestramento abbiamo utilizzato dei minibatch di taglia 512. La configurazione migliore della rete neurale restituita dalla cross validation ha due livelli nascosti, il primo con 128 neuroni e il secondo con 64 neuroni [13, 128, 64, 1], regolarizzazione L2 con lambda pari a 0.001, funzione di attivazione dei livelli nascosti ReLU e 80 epoche. I valori di RMSE e di MAE sul validation set e sul test set sono mostrati in Fig. 8, mentre in Fig. 9 è rappresentato il valore della funzione di perdita, durante l'addestramento, al crescere delle epoche.

```
The RMSE on validation set is: 58169.00480555704
The MAE on validation set is: 41731.033356689884

The RMSE on test set is: 60502.148478240684
The MAE on test set is: 42624.91305278082
```

Fig. 8 – Metriche Housing

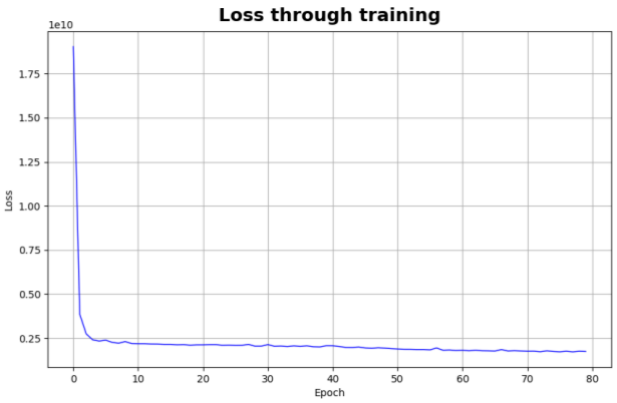

Fig. 9 – Loss Housing

Istruzioni per l'uso

Per l'esecuzione del programma è necessario preventivamente installare le librerie numpy, pandas, matplotlib e scikit-learn. Per verificare se tali pacchetti sono già presenti, ed eventualmente installarli, abbiamo realizzato il file *Packages.py*, che utilizza il comando *pip* per l'installazione. Per avviare il programma è sufficiente eseguire il file Main.py, in cui è possibile scegliere su quale dataset svolgere l'addestramento e la predizione.

Da terminale, i comandi per l'esecuzione sono:

1. *cd path_directory_progetto* (per posizionarsi nella cartella del progetto)
2. *python3 Packages.py* (per installare le librerie, se necessario)
3. *python3 Main.py* (per eseguire il programma)