ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Campus di Cesena
Corso di Laurea in Ingegneria e Scienze Informatiche

ANALISI DATI INQUINAMENTO ATMOSFERICO MEDIANTE MACHINE LEARNING

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Lorenzo Mondani

Prima sessione
Anno accademico 2017/2018

Ringraziamenti

Giunto alla fine di questo impegnativo percorso, sento il dovere di ringraziare tutti gli insegnanti che hanno arricchito le mie competenze e le mie conoscenze, ma soprattutto un ringraziamento particolare va al Prof. Vittorio Ghini che è stato un valido aiuto e supporto in questo mio lavoro a conclusione del corso di laurea triennale. Un doveroso e sentito ringraziamento è riservato anche alla mia famiglia, che mi ha supportato e sopportato in questi anni e un grazie di cuore anche agli amici che mi sono stati vicini e mi hanno aiutato a sdrammatizzare le tensioni.

Lorenzo Mondani

Indice

Introduzione

Le problematiche derivanti dall'inquinamento atmosferico sono numerose, sia per quanto riguarda la salute umana sia per l'ambiente. Per questo, i valori di inquinamento atmosferico sono costantemente monitorati da stazioni sparse sul territorio. Dato che la qualità dell'aria potrebbe incidere seriamente sulla salute del cittadino, oltre ai valori attuali forniti dalle varie stazioni, sarebbe molto utile avere a disposizione delle previsioni su tali agenti inquinanti; l'obiettivo principale consisterà quindi nella generazione di previsioni mediante le principali tecniche di apprendimento automatico. In questa trattazione, previa descrizione dello scenario, verranno individuate le varie tipologie di fonti dati disponibili, sia sul territorio regionale (Emilia-Romagna), sia su territorio estero; si procederà quindi con la descrizione delle operazioni necessarie al reperimento ed all'importazione dei dati provenienti dalle suddette fonti. Per la memorizzazione delle varie misurazioni sarà inoltre definita una base dati comune, che permetterà di accedere ai dati mediante un'interfaccia unificata. Subito dopo il completamento della procedura di raccolta dati, si potrà procedere ad una fase di analisi di essi, per poter individuare le loro caratteristiche e, conseguentemente, la migliore strategia per effettuare qualche tipo di previsione. Sarà inoltre necessario conoscere, a livello teorico, le principali tecniche di apprendimento automatico, oltre che i principali strumenti disponibili per l'implementazione delle suddette tecniche di apprendimento. Combinando tale conoscenza con il risultato della fase di analisi dei dati, sarà possibile definire opportuni modelli, capaci di effettuare, partendo dai dati del passato, una previsione più o meno corretta. In questa trattazione le tecniche di apprendimento automatico si baseranno esclusivamente sulla teoria delle reti neurali artificiali. Tali reti dovranno essere addestrate in modo supervisionato e, per fare ciò, saranno utilizzati i dati raccolti nelle fasi precedenti, estraendo da essi una serie di sottosequenze, che, fornite alla rete, produrranno una previsione; durante il processo di addestramento e di valutazione tale previsione sarà confrontata con la rispettiva misurazione presente fra i dati reali, determinando un valore corrispondente all'errore di previsione. I vari modelli di rete saranno in questo modo valutati e successivamente confrontati, paragonando i valori ottenuti dal calcolo di opportune metriche per la determinazione dell'errore medio fra le previsioni effettuate e le misurazioni reali. Successivamente all'individuazione del modello migliore, si

procederà con alcune sperimentazioni su di esso, adottando particolari strategie note in letteratura o modificando i dati di input tramite apposite tecniche, esaminando poi la ripercussione di tali cambiamenti sui risultati. L'implementazione delle varie reti neurali artificiali avverrà mediante alcune delle librerie più diffuse in questo ambito: TensorFlow, Keras ed alcune funzioni di Scikit-learn. Esse sono state utilizzate attraverso il linguaggio di programmazione Python, nonostante esistano alcune varianti, anche non ufficiali, che si appoggiano su altri linguaggi. I risultati derivanti dalle varie sperimentazioni dei modelli saranno infine presentati e discussi, per determinare se sia opportuno utilizzare tecniche di questo tipo per effettuare previsioni sufficientemente attendibili. In conclusione saranno evidenziati i principali pregi e difetti di tali tecniche, e si cercherà di chiarire quali siano le strategie migliori e quali problematiche sia necessario tenere in considerazione per ottenere buoni risultati da questi tipi di modelli. Questa trattazione non coprirà completamente ogni aspetto di questo argomento né fornirà, per ogni problematica affrontata, una soluzione ottima: in questi casi si cercherà di definire un modo di procedere probabilmente efficace, basandosi sia sulle conoscenze teoriche che sulle prove sperimentali.

Il documento sarà strutturato in questo modo:

L'inquinamento atmosferico Introduzione alla problematica dell'inquinamento atmosferico, descrizione dello scenario del problema, definizione dei vari tipi di inquinanti e delle loro caratteristiche, individuazione delle principali fonti dati.

Reperimento dati Descrizione delle modalità di reperimento dei dati e delle tecniche di conversione e memorizzazione.

Tecniche di apprendimento automatico Introduzione alle principali tecniche di apprendimento automatico: le reti neurali. Descrizione dei principali argomenti teorici e dei vari modelli di rete neurale.

Tecnologie disponibili Individuazione delle principali librerie e piattaforme di sviluppo per reti neurali artificiali.

Analisi dei dati e definizione obiettivi Descrizione nel dettaglio del problema della previsione, analisi delle caratteristiche dei dati raccolti e definizione degli obiettivi in funzione delle tecniche di apprendimento automatico e tecnologie disponibili individuate nei capitoli precedenti.

Progettazione Descrizione delle scelte progettuali adottate per il conseguimento degli obiettivi definiti nel capitolo precedente.

Prove sperimentali per la scelta del modello Descrizione delle prove sperimentali effettuate per l'individuazione del modello maggiormente adatto ad effettuare

previsioni sui dati disponibili, tenendo in considerazione le conoscenze teoriche presentate nel capitolo "Tecniche di apprendimento automatico".

Prove sperimentali sul modello scelto Descrizione dei risultati ottenuti, sul modello scelto nel capitolo precedente, al variare di alcune condizioni; definizione di eventuali prove aggiuntive per approfondire le questioni non completamente trattate.

Conclusioni Riassunto e discussione dei risultati ottenuti.

Capitolo 1

L'inquinamento atmosferico

L'inquinamento atmosferico, cioè l'alterazione della composizione dell'aria causata dalla diffusione di particolari sostanze nocive, è una problematica assai grave e diffusa in ogni parte del mondo; non solo, come si potrebbe pensare, nelle maggiori città o nei paesi maggiormente industrializzati, ma anche e soprattutto nei paesi in cui non sono state adottate particolari contromisure, come, ad esempio, in Africa e India[48]. Questi agenti inquinanti possono avere effetti anche molto gravi sulla salute umana, causando ad esempio, problemi all'apparato respiratorio o all'apparato cardiocircolatorio[43]. Ovviamente, gli effetti sulla salute si aggravano all'aumentare della concentrazione nell'aria di tali sostanze. Per questo vari Stati del mondo, fra cui l'Italia e gli altri paesi dell'Unione Europea, hanno dovuto imporre limiti di legge sulla concentrazione di alcuni inquinanti, per salvaguardare la salute dei cittadini. In Italia ed in altri paesi, per garantire il rispetto di tali limiti, sono state dislocate sul territorio una serie di stazioni di monitoraggio ufficiali, la cui validità dei dati è costantemente garantita.

1.1 Principali inquinanti

Le fonti fisiche che diffondono i vari agenti inquinanti sono numerose: autoveicoli, sistemi di riscaldamento, processi industriali. Alcune delle principali sostanze inquinanti prodotte da questi sono:

Biossido di Azoto (NO_2) Composto prodotto principalmente da processi di combustione, che avvengono, ad esempio, negli autoveicoli o negli impianti di riscaldamento domestico; è un gas irritante e può quindi causare gravi problemi all'apparato respiratorio. Nei casi più gravi può portare alla morte.

Polveri sottili Le polveri sottili sono classificate in base al diametro delle particelle; le due classificazioni principali sono PM_{10} (diametro inferiore a 10 μm) e $PM_{2.5}$ (diametro inferiore a 2.5 μm). Queste sono prodotte principalmente da processi

di combustione, che avvengono, ad esempio, negli autoveicoli e negli impianti di riscaldamento domestico (soprattutto a legna) o da processi secondari come l'usura dei freni e degli pneumatici[14]. Gli effetti di queste polveri possono essere estremamente nocivi per la salute umana, soprattutto quelli causati dalle più fini ($PM_{2.5}$); queste, infatti, date le loro dimensioni, potrebbero raggiungere anche gli alveoli polmonari, causando gravi patologie sia all'apparato respiratorio che all'apparato cardio-circolatorio.

Monossido di carbonio (CO) Composto prodotto principalmente da processi di combustione in carenza di ossigeno, come nei motori degli autoveicoli; è un gas tossico, che respirato, si lega all'emoglobina più saldamente rispetto a quanto farebbe l'ossigeno. Ciò riduce la capacità del sangue di trasportare ossigeno, portando, nel caso di un'alta concentrazione nell'aria, alla morte.

Biossido di zolfo (SO_2) Composto prodotto principalmente da processi di combustione, in presenza di zolfo sotto forma di impurità; è un gas irritante che può comportare gravi patologie all'apparto respiratorio, e, nei casi più gravi, può portare alla morte. Inoltre può causare gravi danni all'ambiente, acidificando le precipitazioni e contaminando, di conseguenza, i corpi idrici.

1.2 Limiti di legge

In Italia sono stati fissati, per la concentrazione di questi inquinanti, alcuni limiti da rispettare. Di seguito sono riportati alcuni esempi:

Biossido di Azoto (NO_2) Valore limite annuale: 40 µg/m^3, valore limite orario: 200 µg/m^3 da non superare più di 18 volte/anno.[23]

Polveri sottili (PM_{10}) Valore limite annuale 40 µg/m^3, valore limite giornaliero: 50 µg/m^3 da non superare più di 35 volte/anno.[26]

Monossido di carbonio (CO) Valore limite, come massimo della media mobile su 8 ore, di 10 µg/m^3.[25]

Biossido di zolfo (SO_2) Valore limite giornaliero 125 µg/m^3 da non superare più di 3 volte/anno, valore limite orario 350 µg/m^3, da non superare più di 24 volte/anno.[24]

1.3 Principali fonti dati

Per permettere il rispetto dei limiti di legge, sono state installate, sul territorio nazionale italiano, una serie di stazioni ufficiali, che monitorano costantemente la qualità dell'aria. Le misurazioni di queste stazioni sono affidabili, poiché i dati sono costantemente validati dal personale, verificando il corretto funzionamento dei sensori. Le misurazioni di queste stazioni sono disponibili pubblicamente, messe a disposizione dai vari enti regionali ARPA, per ogni Regione italiana; in questo caso specifico ci si è esclusivamente concentrati su dati dell'Emilia-Romagna, offerti da Arpae Emilia-Romagna. Esistono anche altre iniziative non ufficiali per il monitoraggio della qualità dell'aria, ma in Italia nessuna di esse mette a disposizione un archivio storico. In altre parti del mondo, invece, esistono progetti di monitoraggio non ufficiali che consistono nella condivisione di misurazioni sulla qualità dell'aria, fornite da stazioni installate nelle abitazioni dei singoli cittadini.

ARPA

In Italia la principale fonte risulta essere il portale dell'ente ARPA, relativo ad una specifica Regione. Nel caso dell'Emilia-Romagna, l'ente ufficiale che si occupa della pubblicazione dei dati relativi al meteo ed alla qualità dell'aria è Arpae Emilia-Romagna[5]. Sono offerti sia dati in tempo reale sia archivi storici; entrambi sono accessibili attraverso due canali principali: sito web[16] e portale OpenData[31]. Il sito web mette a disposizione una maschera per effettuare richieste di dati; la risposta perviene attraverso una tabella in formato HTML. Il portale OpenData permette invece di ottenere i dati direttamente in formato CSV o JSON. Non è però chiaro se i due canali offrano esattamente le stesse misurazioni. Il punto di forza dei dati ARPA è la completezza degli archivi storici: sono infatti forniti, per ogni stazione, le misurazioni di molti inquinanti con frequenza oraria, dal 2010 in avanti. L'unica carenza consiste nella densità di stazioni: in tutta l'Emilia-Romagna sono presenti solamente circa quaranta stazioni, comportando una distribuzione spaziale molto sparsa, con una copertura che stenta a raggiungere l'accettabilità anche nelle maggiori città.

Altre fonti su territorio estero

In Italia, come già affermato, non esistono valide alternative ai dati provenienti dagli enti regionali ARPA. Esistono, in realtà, alcuni portali che mettono a disposizione i dati sulla qualità dell'aria nella maggior parte dei paesi, fra cui l'Italia, come World Air Quality Index[49], OpenWeatherMap[29] o BreezoMeter[7]. I dati che questi mettono a disposizione derivano comunque dalle stazioni ARPA e, inoltre, vengono imposte delle limitazioni sul numero di richieste effettuabili tramite le API messe a disposizione. Esistono poi altre iniziative interessanti, che si basano sulla condivisione

delle misurazioni effettuate da stazioni installate dai singoli cittadini nelle proprie abitazioni; i dati provenienti da tali stazioni sono messi a disposizione sui rispettivi portali dei singoli progetti. Alcuni di questi sono:

PurpleAir [30] è un'iniziativa che consiste nel monitorare la qualità dell'aria tramite centraline installate da utenti privati, nelle loro abitazioni o in luoghi di particolare interesse. Le centraline, appena configurate e connesse alla rete, pubblicano sul web i dati letti dai propri sensori. I dati di tutte le centraline sono accessibili tramite la mappa messa a disposizione sul sito. L'intero sistema si basa sui servizi offerti da Thingspeak[42], ed è possibile utilizzare le loro API per reperire i dati delle singole stazioni. Non è possibile ottenere dati storici con alta precisione oltre date antecedenti i 10 giorni circa. I dati sono però completi, con i valori per ogni inquinante.

AirVisual [2] è un'azienda che mette a disposizione dati in tempo reale e previsioni sulla qualità dell'aria per molte località del mondo. I dati provengono sia da fonti ufficiali sia da dispositivi di monitoraggio acquistabili dal loro sito web. Per la Cina, nel rispetto dei suoi regolamenti, sono pubblicati solamente i dati provenienti da fonti ufficiali[18]. Le varie informazioni sulla qualità dell'aria sono consultabili attraverso una mappa interattiva, ma i dati non sono completi (solo indice AQI). Per ottenere i dati completi con i valori per ogni inquinante è possibile interrogare l'API a pagamento.

uRADMonitor [46] è un progetto finanziato attraverso un processo di *crowdfunding*, che consiste nel monitoraggio di alcuni parametri ambientali (tra cui inquinanti e radiazioni) attraverso piccole stazioni. Queste possono essere acquistate da utenti privati ed installate nelle loro abitazioni. I dati raccolti da ogni stazione sono caricati on-line e consultabili da una mappa interattiva presente sul sito web del progetto. I dati sono completi ma le stazioni non sono in numero molto elevato; inoltre potrebbe rivelarsi difficile reperire i dati in modo automatico.

LOOKO2 [27] è un progetto, attuato in Polonia, che, similmente a PurpleAir, consiste nel monitorare la qualità dell'aria tramite centraline private. Le varie centraline, dopo essere state configurate dall'utente, caricano on-line le proprie misurazioni, che sono consultabili sul sito web. Sono disponibili i dati attuali delle varie stazioni, ma non quelli storici.

Air quality egg [1] è un progetto simile a PurpleAir, che consiste nell'installazione di minuscole stazioni private, soprannominate *egg*. Ognuna di queste incorpora i sensori necessari al monitoraggio della qualità dell'aria e, le misurazioni effettuate, vengono condivise on-line con altri utenti registrati.

Questo tipo di fonti sembra però non essere adeguato al tipo di operazioni che si vorranno effettuare in futuro. Infatti, nonostante vengano offerte misurazioni con una densità spaziale molto più elevata rispetto ai dati ARPA, non sono messi a disposizione archivi storici con un numero sufficiente di dati per effettuare analisi su essi. Inoltre, le limitazioni sul numero di richieste effettuabili attraverso le API offerte, rende difficile ottenere una buona quantità di dati.

Dati meteorologici

Sebbene ci si sia concentrati maggiormente sull'individuazione di dati relativi all'inquinamento atmosferico, si è rivelato necessario individuare anche alcune sorgenti di dati meteorologici. Le misurazioni inerenti alle condizioni meteo potranno infatti rendersi utili per migliorare la qualità delle previsioni sulla qualità dell'aria. Anche in questo caso, i dati meteorologici più completi e con un buon archivio storico sembrano essere quelli offerti dagli enti ARPA. Si è deciso di concentrarsi esclusivamente sull'Emilia-Romagna e quindi sui dati offerti da Arpae Emilia-Romagna. Sono disponibili sia misurazioni in tempo reale sia archivi storici; entrambe sono accessibili attraverso il portale OpenData[28], dal quale è possibile ottenere i dati direttamente in formato JSON.

Capitolo 2

Reperimento dati

Successivamente all'individuazione delle principali fonti dati ci si è posti il problema di raccogliere tali misurazioni in una base di dati adeguata. Si è rivelato quindi necessario definire la struttura di tale banca dati e, inoltre, sviluppare un applicativo capace di importare in modo automatico le misurazioni dalle diverse fonti dati. L'applicativo è stato sviluppato interamente in C#, mentre il motore di database scelto è stato PostgreSQL.

2.1 Sviluppo applicazione per importazione dati

Per le fonti considerate più adeguate, in questo caso per Arpae Emilia-Romagna, sono stati sviluppati dei moduli applicativi per la raccolta e l'interpretazione dei dati in modo automatico. Come sarà precisato successivamente in ogni sottosezione, i dati caricati da ogni sorgente sono memorizzati in specifiche classi, la cui struttura dipende, ancora parzialmente, dal formato dei dati originale. Sono state quindi definite, per ognuna, delle specifiche procedure di conversione, atte ad ottenere i record sotto forma di classi universali, che rispecchiano la struttura utilizzata nel database.

Come già precisato il portale Arpae Emilia-Romagna mette a disposizione i dati attraverso due canali:

- Pagina web.
- Portale OpenData.

Importazione dati ARPAE da pagina web

Per operare in modo semplice e veloce tramite codice *C#* su pagine HTML è stata utilizzata la libreria *open source* Html Agility Pack. Il portale web[16] di Arpae

permette di ispezionare le stazioni presenti, raggruppate per provincia. Successivamente, selezionando la stazione prescelta da una combo box, è possibile definire, nella maschera fornita, il tipo di inquinante e l'intervallo di date, ottenendo i record relativi in formato tabellare HTML. Il rispettivo modulo dell'applicazione deve quindi occuparsi di automatizzare tale operazione. Inizialmente vengono individuate le sigle corrispondenti alle varie province; ognuna di queste viene quindi inserita come parametro nell'URL, individuando, per ogni provincia, la relativa pagina web contenente le informazioni sulle rispettive stazioni. Successivamente, dal codice HTML delle *combo box* presenti in ogni pagina individuata, si ottengono gli identificativi delle varie stazioni e degli inquinanti che quella particolare stazione può misurare. Tali identificativi sono utilizzati come parametro di una richiesta *GET* HTTP, comprendente anche l'intervallo di date, ad una pagina che fornisce i dati richiesti in formato tabellare (la stessa pagina a cui si viene reindirizzati quando si confermano i parametri della maschera di selezione menzionata precedentemente). Ottenuta la pagina con la tabella HTML si estraggono i dati, che vengono memorizzati in classi specifiche definite all'interno dell'applicazione. La pagina che fornisce i dati non accetta però richieste con un intervallo di date superiore ad un anno, per cui è stato necessario effettuare dall'applicativo una serie di richieste separate.

Importazione dati ARPAE sulla qualità dell'aria (OpenData)

Sul portale OpenData[31] di Arpae Emilia-Romagna è possibile reperire file in formato CSV contenenti i dati storici relativi alla qualità dell'aria. Sono presenti inoltre i file contenenti gli identificativi delle diverse stazioni e dei vari parametri, necessari per interpretare correttamente i record. Il rispettivo modulo dell'applicazione è in grado di leggere tali file CSV ed interpretarli, memorizzandoli in classi definite internamente. In questo caso la strutturazione dei record è particolarmente semplice e per questo non si è rivelata la necessità di adottare particolari accorgimenti.

Importazione dati meteorologici ARPAE (OpenData)

Sul portale OpenData[28] di Arpae Emilia-Romagna è inoltre possibile reperire file in formato JSON contenenti i dati storici relativi alle condizioni meteo. I file JSON sono però caratterizzati da una strutturazione molto complessa, definita da un particolare RFC[15]. Il rispettivo modulo dell'applicazione è in grado di leggere tali file JSON ed interpretarli, memorizzandoli in classi definite internamente. In questo caso la strutturazione dei record è particolarmente complessa e, di conseguenza, lo è anche la procedura che si occupa dell'interpretazione e conversione.

2.2 Definizione del database

L'utilizzo di una base di dati consente di raggruppare le misurazioni in un unico contenitore e di mantenerle secondo una struttura universale. La soluzione più ovvia, anche se forse non la più agevole, è stata quella di utilizzare un DBMS (PostgreSQL), definendo le opportune tabelle. La scelta di questo particolare motore di database è stata giustificata dalla maggiore libertà che offre, rispetto ai concorrenti, in termini di ottimizzazione e tipi di dati utilizzabili; tale libertà potrebbe rivelarsi molto utile, specialmente in questo caso, in cui è necessario operare con una mole di dati molto elevata. Inizialmente è stata effettuata un'attività di analisi che ha permesso poi di definire, attraverso uno schema E/R, il modello concettuale del problema. Al momento della traduzione in tabelle si è però deciso di discostarsi leggermente dal modello concettuale, prevalentemente per una questione di performance e di semplicità d'uso. La struttura delle varie tabelle può essere esaminata in dettaglio in Figura 2.1 a pagina 15.

Come si può notare la tabella principale è *aq_values*, che mantiene gli attributi dei singoli record, quali informazioni temporali, spaziali, tipo di dato e aggregazione. Le tabelle *aq_aggregation_type* e *aq_value_type* hanno invece un contenuto informativo minore; sono infatti necessarie per attribuire un significato agli identificativi presenti in *aq_values* ma, poiché tali informazioni sono presenti anche all'interno dell'applicazione, potevano anche essere omesse. Gli attributi di *aq_station*, ad eccezione del nome, potrebbero essere considerati ridondanti, poiché mantengono le stesse coordinate spaziali relative ai rispettivi record; in effetti lo sono (a meno che una stazione venga spostata), ma si rivelano comunque necessari per individuare velocemente le stazioni su una mappa, senza eseguire *query* sulla tabella *aq_values*. Si sarebbe anche potuto pensare di rimuovere le coordinate spaziali dai singoli record, recuperandole dalla relativa stazione; ciò risulterebbe però poco ragionevole, poiché le coordinate sono relative alla singola misurazione, indipendentemente dalla stazione che l'ha misurata (si pensi ad esempio a stazioni mobili). Anche la tabella *aq_station_value_type*, che mantiene quali parametri una stazione può misurare, potrebbe considerarsi superflua, poiché tali informazioni sono quasi totalmente ricavabili dalla tabella aq_values. Ovviamente, questo è possibile se si dispone di un numero adeguato di record. Si deve inoltre tenere presente che non sempre le sorgenti dati forniscono le informazioni sulle stazioni, costringendo ad effettuare comunque un'operazione di questo tipo. Anche in questo caso però tale tabella si rivela utile per filtrare le stazioni in base ai parametri che misurano, senza ricorrere a gravose *query* sulla tabella *aq_values*. Come effetto collaterale sarà però necessario aggiornare periodicamente, o, almeno, dopo gli inserimenti in *aq_values*, le tabelle *aq_station* e *aq_station_value_type*, attraverso apposite *query*.

Per migliorare le performance e la facilità di inserimento, si è deciso di non definire vincoli di integrità referenziale: la garanzia che gli identificativi siano corretti è fornita

esclusivamente dall'applicazione.
Per ogni tabella sul database è presente una corrispondente classe nell'applicazione; definendo un'istanza di una di tali classi, è possibile inserirla direttamente sul database in modo automatico. Stessa cosa avviene per l'estrazione dei dati dal database: invocando un'opportuna procedura è possibile ottenere il risultato di una *query* come collezione di istanze di una di tali classi.
Inoltre, all'avvio dell'applicazione, se le tabelle non sono presenti sul database vengono create automaticamente.

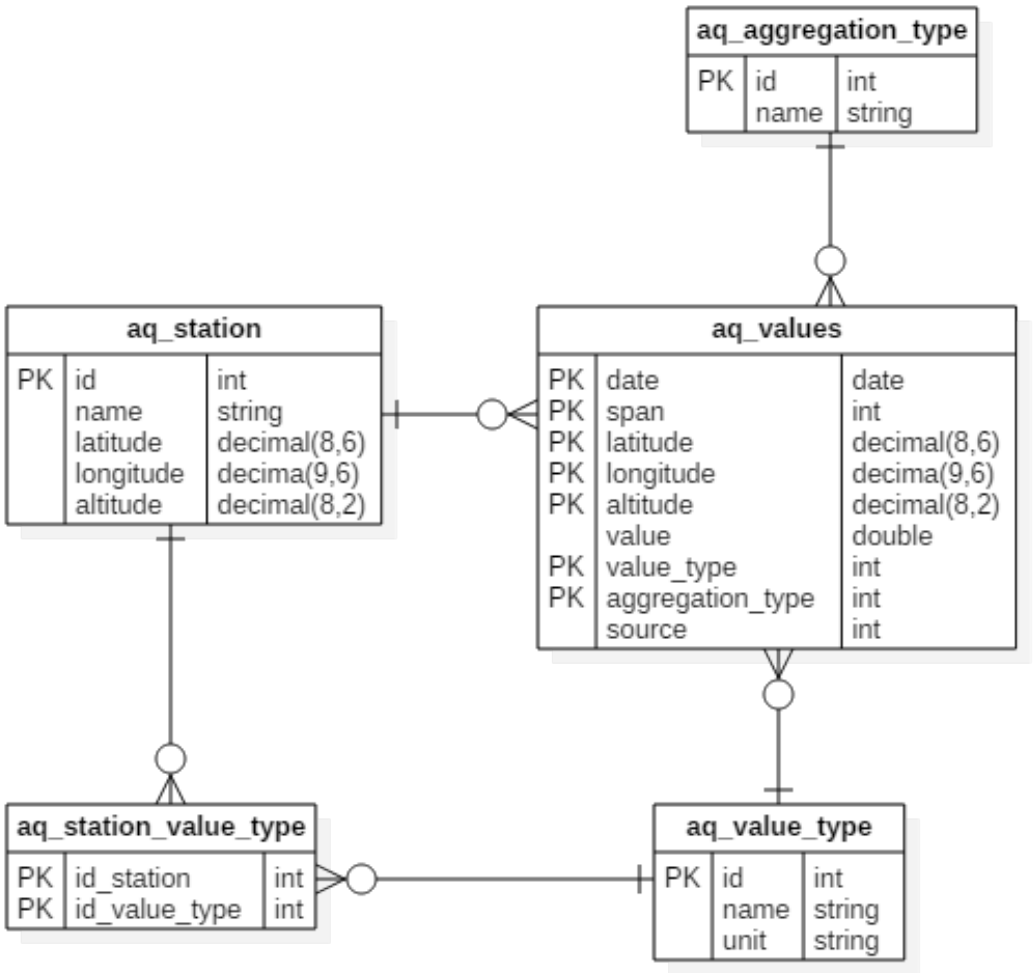

Figura 2.1: schema delle tabelle definite sul database.

Capitolo 3

Tecniche di apprendimento automatico

Successivamente alla raccolta dei dati ed alla loro memorizzazione su una banca dati, dalla quale potranno essere estratti in conformità di una struttura ben definita, si procederà con la ricerca delle principali tecniche di apprendimento automatico; queste saranno successivamente adottate sfruttando le misurazioni raccolte nella fase precedente.

3.1 Introduzione al machine learning

L'obiettivo principale del *machine learning* è riuscire ad addestrare una macchina in modo che questa impari a compiere determinate operazioni, per ottenere i risultati voluti, senza fornirle esplicitamente l'algoritmo da eseguire. La macchina deve essere in grado di apprendere i legami presenti in un insieme di dati, sviluppando uno specifico comportamento; tale comportamento appreso potrà poi essere sfruttato per effettuare previsioni su nuovi dati. L'apprendimento può avvenire in tre modi possibili[17, p. 2][41, p. 25]:

Supervised learning L'addestramento avviene fornendo i dati di input e, per ognuno, il corrispondente risultato atteso[6, p. 10]. L'obiettivo è quello di individuare una funzione che trasformi i dati di input nei risultati attesi, minimizzando l'errore. Ad ogni iterazione di addestramento si confronta la previsione effettuata dal modello con il risultato atteso, calcolando poi i nuovi parametri per ridurre il divario fra i due valori.

Unsupervised learning Questo tipo di addestramento avviene fornendo solamente i dati di input, senza i risultati attesi[6, p. 12]. Sarà compito del modello cercare di estrarre qualche tipo di conoscenza dai dati che gli sono stati somministrati.

Reinforcement learning In questo tipo di addestramento il modello viene inserito in un particolare ambiente[6, p. 14]. L'obiettivo del modello sarà quello di

massimizzare una sorta di punteggio, attraverso delle ricompense fornite dall'ambiente, come effetto di particolari interazioni con esso.

Una delle principali tecniche per effettuare apprendimento automatico risulta essere l'utilizzo delle cosiddette reti neurali artificiali. Tali reti sono ispirate alla struttura del cervello umano: una delle componenti principali di tali reti sono infatti i neuroni artificiali. Un neurone artificiale, in sintesi, riceve un insieme di input, li somma fra loro moltiplicandoli per dei pesi specifici ed infine vi applica una particolare funzione di attivazione, restituendo il valore calcolato come output. Collegando fra loro un certo numero di questi neuroni è possibile ottenere una rete molto complessa, simile, appunto, al cervello umano. In generale, il comportamento di tale rete è definito dai pesi degli input di ogni singolo neurone, che dovranno essere determinati attraverso il processo di addestramento. I neuroni sono raggruppati in *layer*, ricevendo gli input dal *layer* precedente e fornendo gli output al *layer* successivo. Il *layer* iniziale, connesso ai dati di input, è detto *input layer*, mentre l'ultimo, che fornisce l'output della rete è detto *output layer*; fra questi possono esserci vari *hidden layer*, che permettono di incrementare la complessità e le potenzialità della rete. Poiché gli *hidden layer* possono essere presenti in numero molto elevato, è nato il termine *Deep Learning*, che allude appunto alla profondità molto elevata di *layer* che queste reti possono avere. Il *Deep Learning* ha avuto una grande diffusione negli ultimi anni, sia per gli ottimi risultati raggiunti in numerosi campi, sia per la crescente disponibilità di hardware molto performante, capace di addestrare questi tipi di modelli in tempi ridotti (impossibile fino a qualche decennio fa).[17, p. 3]

3.2 Terminologia

Di seguito sono riportati alcuni termini, molto utilizzati in ambito *machine learning*[12].

Features e Labels

Le cosiddette *features* sono i valori che contiene ogni singolo record dei dati di input. Ogni record, o esempio, sarà fornito alla rete uno per volta. Ad esempio, considerando la Tabella 3.1, ogni record possiede quattro *features*.

Data	Temperatura	Umidità	Precipitazioni
01/01/18	275	60	2
02/01/18	277	50	0

Tabella 3.1: esempio di alcuni record con quattro features.

Le *labels* invece sono i valori che il modello dovrà cercare di predire. Durante la fase di addestramento (*training*), saranno fornite al modello anche le *label* corrispondenti ai dati di input (*labeled example*), come possibile osservare in Tabella 3.2; in questo modo potrà calcolare l'errore fra la *label* voluta e la previsione, correggendosi di conseguenza.

Data	Temperatura	Umidità	Precipitazioni	NO2 (*target label*)
01/01/18	275	60	2	20
02/01/18	277	50	0	30

Tabella 3.2: *labeled example*.

Durante la fase di valutazione/utilizzo del modello (*inference*), sono invece forniti al modello solo i dati di input (*unlabeled example*) ed esso dovrà produrre le *label* corrispondenti sfruttando la conoscenza accumulata durante la fase di *training*.

Regressione e Classificazione

Le *label* che il modello dovrà predire potrebbero assumere solamente valori ben specifici oppure un qualsiasi valore reale. Nel primo caso è necessario risolvere un problema di classificazione, mentre nel secondo un problema di regressione. In base al tipo di problema da risolvere sarà necessario adottare gli opportuni accorgimenti nella definizione del modello.

3.3 Regressione lineare

Si supponga di avere un insieme di record con una singola *feature*, e di dover determinare la *label* corrispondente, come mostrato in Figura 3.1.

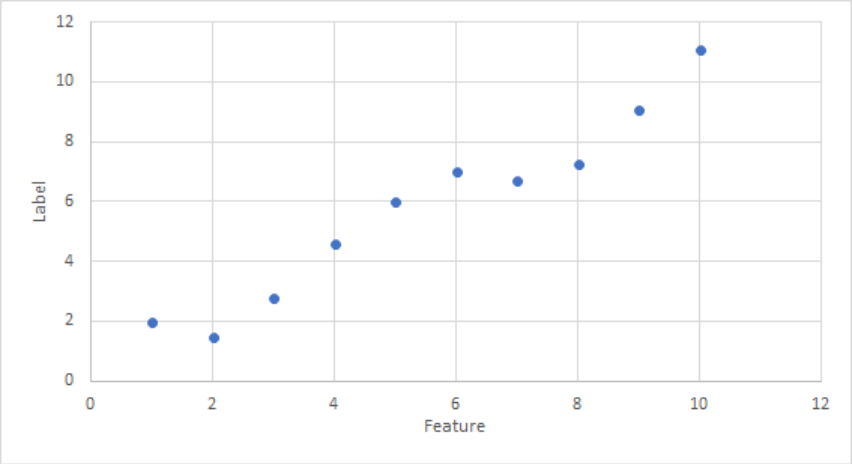

Figura 3.1: grafico di esempio con valore della *feature* sull'asse x e valore della *label* sull'asse y.

Se, fra la *feature* e la *label*, esiste una relazione pressoché lineare, allora sarà sufficiente definire una retta. La funzione associata a tale retta sarà capace di prevedere, approssimativamente, le *label* da associare ad un qualsiasi valore della *feature* in input[6, p. 72], come mostrato in Figura 3.2.

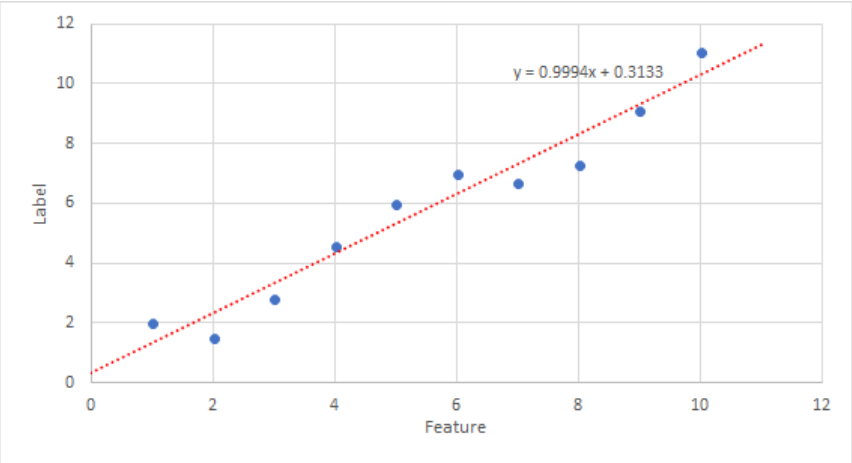

Figura 3.2: grafico di esempio con retta di previsione.

L'equazione della retta mostrata in Figura 3.2, sarà, ovviamente, del tipo[8]:

$$y = b + wx$$

dove:

y è il valore che si vuole prevedere (*label*).

b è il punto di intersezione con l'asse delle ordinate; in ambito *machine learning* corrisponde al *bias*.

w è il coefficiente angolare della retta; in ambito *machine learning* corrispondente al peso associato alla *feature x*.

x è il valore della *feature* in input.

Per poter individuare tale retta sarà quindi necessario determinare i valori corrispondenti a b e w. Questo è esattamente l'obiettivo che ci si pone durante la fase di training della rete. Una semplicissima rete neurale, senza funzione di attivazione, capace di rappresentare la retta appena descritta è mostrata in Figura 3.3.

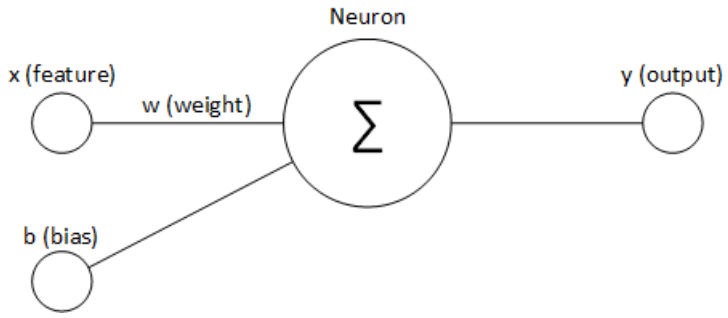

Figura 3.3: semplice rete neurale di esempio con un singolo neurone, una singola *feature* in input e senza alcuna funzione di attivazione applicata all'output.

Nel caso fossero presenti più *feature* l'equazione diventerebbe del tipo:

$$y = b + w_1x_1 + w_2x_2 + ... + w_nx_n$$

3.4 Calcolo e minimizzazione dell'errore

L'obiettivo della fase di addestramento del modello è quello di individuare i pesi più adeguati, cioè quelli che permettono di ottenere le previsioni migliori[9]. La fase di training avviene quindi confrontando l'output della rete (previsione della *label*) con il valore della *label* atteso, per ogni record fra i dati di input; i vari confronti vengono quindi ridotti ad un singolo valore (in base ad un'opportuna metrica), che corrisponde all'errore globale che la rete dimostra sull'insieme dei dati di input[6, p. 28]. Individuato tale valore di errore, la rete dovrà correggere i vari pesi in modo da poterlo minimizzare.

Mean square error (MSE)

Una metrica molto comune per il calcolo dell'errore è il *mean square error* (errore quadratico medio), definito come segue:

$$\frac{1}{N}\sum(y-y')^2$$

dove:

N è il numero di record presenti fra i dati di input.

y è la previsione attesa.

y' è la previsione della rete.

In altri termini, si tratta della media degli errori di ogni singolo record, calcolati come il quadrato della differenza fra il valore atteso e il valore previsto. Il vantaggio di questa funzione di errore (la parte all'interno della sommatoria) è che non ha punti di discontinuità, facilitando alcune operazioni che saranno descritte successivamente.

Mean absolute error (MAE)

Un'altra metrica spesso utilizzata per il calcolo dell'errore è il *mean absolute error*, definito come segue:

$$\frac{1}{N}\sum|y-y'|$$

dove:

N è il numero di record presenti fra i dati di input.

y è la previsione attesa.

y' è la previsione della rete.

In altri termini, si tratta della media degli errori di ogni singolo record, calcolati come il modulo della differenza fra il valore atteso e il valore previsto. Il vantaggio di questa funzione (la parte all'interno della sommatoria) è che l'errore è più facilmente quantificabile rispetto ad MSE, ma bisogna considerare che presenta un punto di discontinuità.

Calcolo del gradiente

Successivamente alla determinazione dell'errore che la rete dimostra sui dati, è necessario individuare una tecnica per aggiornare i pesi in modo che l'errore diminuisca. Si può realizzare ciò attraverso il calcolo del gradiente, per individuare in quale direzione l'errore diminuisce (*gradient descend*)[32]. Si supponga che la rete abbia una sola *feature* in input e, di conseguenza, un solo peso; si supponga inoltre di inizializzare tale peso *w1* casualmente, e, calcolando l'errore (*loss*), di trovarsi nel punto *p* di Figura 3.4.

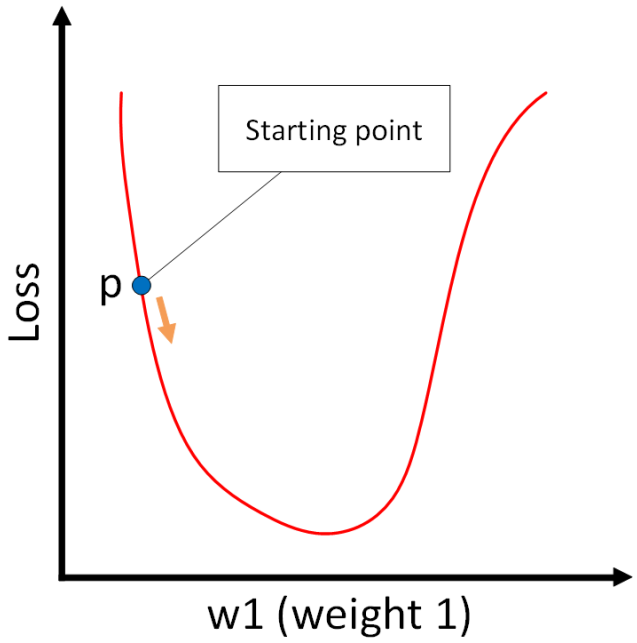

Figura 3.4: grafico che descrive la variazione dell'errore al variare del peso w1.

Si vuole ora individuare un nuovo punto *p'*, corrispondente ad un determinato valore di *w1*, tale per cui l'errore associato è minimo. Il grafico di Figura 3.4, in un problema reale, si potrebbe ottenere calcolando l'errore per ogni possibile valore del

peso *w1*, e, di conseguenza, si potrebbe trovare direttamente il valore per cui l'errore è minimo. Un approccio di questo tipo, per reti neurali complesse, sarebbe però improponibile. Un metodo invece utilizzabile per individuare il punto di minimo è il calcolo del gradiente, cioè la derivata, in questo caso rispetto a *w1* e nel punto *p*. La derivata serve principalmente per individuare in quale direzione muoversi, cioè se aumentare o diminuire *w1*. Nell'esempio, poiché nel punto *p* la funzione decresce, la derivata sarà negativa e *w1* dovrà essere incrementato. In generale se la derivata è negativa *w1* deve essere incrementato mentre se positiva deve essere ridotto; questo permette di muovere *w1* verso il minimo della funzione. Bisogna però stabilire anche quanto incrementare o decrementare *w1*; ciò è definito attraverso un parametro, chiamato *learning rate*[33]. Ad ogni iterazione, il valore di *w1* sarà quindi modificato in base alla seguente formula:

$$w_1 = w_1 - \alpha \frac{\partial f(w_1)}{\partial w_1}$$

dove α è il *learning rate*. Un *learning rate* troppo piccolo aumenta i tempi di *training*, poiché ad ogni passo i pesi variano troppo lentamente; un *learning rate* troppo grande, invece, potrebbe impedire al modello di convergere, poiché il punto di minimo viene costantemente "saltato".

Nell'esempio era presente un singolo peso; lo stesso ragionamento può però essere adottato anche nel caso in cui siano presenti più *feature* in input, e quindi più pesi, considerando le varie derivate parziali rispetto ad ogni singolo peso $w_1, w_2, ..., w_n$.

Inoltre, nell'esempio, si supponeva di calcolare l'errore su tutti i record dei dati di input; in un caso reale, in cui i record di input potrebbero essere presenti in numero molto elevato, risulterebbe poco conveniente calcolare l'errore per ognuno di essi. Solitamente quindi si dividono i record di input in *batch*, spesso in modo casuale; per ogni *batch* si calcola quindi l'errore e si aggiornano i pesi. La dimensione del *batch* è quindi un altro parametro regolabile, che potrebbe influire sull'efficienza processo di *training*: un *batch* troppo piccolo potrebbe causare un errato aggiornamento dei pesi, un *batch* troppo grande potrebbe rendere intollerabili i tempi di *training*. La dimensione del *batch* caratterizza alcune tecniche di *training*[34]:

Stochastic gradient descent (SGD) La dimensione del *batch* è 1.

Mini-batch stochastic gradient descent (mini-batch SGD) La dimensione del *batch* è solitamente compresa fra 10 e 1000.

3.5 Overfitting, insiemi di training, validation e test

Nell'esempio citato nelle sezioni precedenti si assumeva di eseguire la fase di *training* su tutti i dati di input disponibili. In realtà, questo approccio può causare problemi, poiché il modello potrebbe adattarsi eccessivamente all'insieme dei dati fornitogli, comportandosi negativamente con nuovi dati. Questo fenomeno è chiamato *overfitting*[13][6, p. 27]: la rete si adatta eccessivamente alle caratteristiche dei dati su cui è stata fatta la fase di *training*, perdendo la capacità di generalizzare correttamente su nuovi esempi. Una soluzione per risolvere tale problematica potrebbe essere quella di dividere i dati di input in due insiemi distinti: *training set* e *validation set*[44]. Il *training set* sarà quello che avrà il maggior numero di record (ad esempio si potrebbero dividere rispettivamente in 80%-20%). Gli esempi contenuti nel *training set* non devono comparire nel *validation set* e viceversa; inoltre le caratteristiche dei dati presenti nei due insiemi dovrebbero essere simili. Ad ogni iterazione del processo di *training* si calcola quindi l'errore sul *training set*, aggiornando i pesi di conseguenza. Al termine dell'aggiornamento dei pesi si valuta il modello sul *validation set* e si reitera il procedimento. La configurazione dei pesi migliore sarà quella che produce l'errore minore sul *validation set*. Procedendo in questo modo si rischia però di fare *overfitting* sul *validation set*. Per ridurre ulteriormente la probabilità che questo accada, si possono suddividere ulteriormente i dati di input in tre insiemi distinti: *training set*, *validation set* e *test set*[47]. In questo caso si procede esattamente come già descritto, ma, alla fine del processo di *training*, la rete viene collaudata sul *test set*, per verificarne le prestazioni. Utilizzando questa tecnica è facile determinare se il modello è affetto da *overfitting* sui dati di *training*; in tal caso, infatti sarà possibile notare, durante la fase di addestramento, una diminuzione dell'errore sul *training set*, ma un continuo incremento dell'errore sul *validation set*, come mostrato in Figura 3.5.

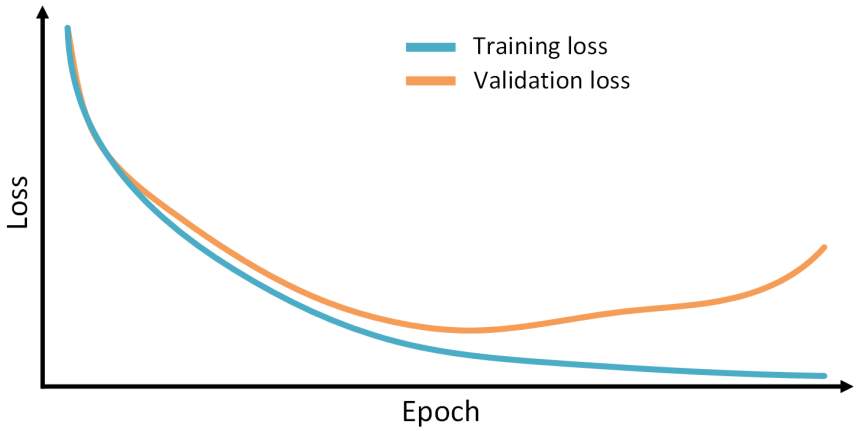

Figura 3.5: esempio di *overfitting* durante la fase di addestramento.

Risulta però necessario specificare che l'*overfitting* può manifestarsi solamente se il modello è sufficientemente complesso da poter adattarsi completamente alle caratteristiche del *training set*. Un altro buon metodo per combattere questo tipo di fenomeno è quindi quello di ridurre la complessità del modello; questo argomento verrà trattato più ampiamente nella sezione *Regolarizzazione*.

3.6 Formattazione dei dati di input

Per ottenere i risultati migliori, è necessario fornire alla rete i dati nel giusto formato[21, p. 32]. Questo implica la definizione di una procedura di adattamento dei dati, che permetta, partendo dai dati nel loro formato originale, di ottenere i record con le *feature* da fornire al modello; tale procedura, chiamata *feature engineering*[39][6, p. 44], potrebbe rivelarsi anche molto complessa e laboriosa.

Scelta delle feature di input

Una delle operazioni iniziali da compiere è la scelta delle *input feature*. Si potrebbe essere tentati di utilizzare tutte le *feature* presenti nei dati originali, e lasciare al modello l'onere di capire quali fra esse siano quelle da cui dipende maggiormente l'output. Questo non costituisce però il miglior modo di procedere; bisognerebbe infatti cercare di minimizzare il numero di *feature* da cui deve dipendere il modello, per ridurre la probabilità di doverlo modificare a fronte di una variazione delle caratteristiche dei dati di input. Le varie *feature*, inoltre, dovrebbero essere poco dipendenti fra loro, per evitare che un sottoinsieme di queste fornisca, essenzialmente, la stessa informazione. I valori di una qualsiasi *feature* scelta dovrebbero comparire, singolarmente, almeno, ad esempio, cinque volte, per permettere alla rete di comprendere il rapporto che tale valore ha con l'output[40]. I *magic number* sono da evitare: al loro posto è necessario inserire una *feature* aggiuntiva che indichi se il dato è presente o meno[40].

Feature numeriche

Le *feature* numeriche presenti nei dati originali possono essere convertite direttamente nelle corrispettive *feature* da fornire alla rete, convertendo ogni valore numerico in un dato di tipo *float*[39]. Potrebbe essere conveniente però, soprattutto se tali *feature* assumono valori molto elevati, scalarle opportunamente a valori più bassi, ad esempio fra zero e uno. Questo permetterebbe alla rete di convergere più velocemente al valore dei pesi ottimale, di evitare che qualche peso diventi *NaN* e di evitare che le *feature* con i valori più elevati prendano il sopravvento sulle altre[38].

One hot encoding

Alcune *feature* potrebbero essere di tipo categorico. In questo caso si potrebbe semplicemente assegnare ad ogni categoria uno specifico valore. Spesso però questa non è la soluzione che può portare ai risultati migliori. La tecnica che si applica viene detta *one hot encoding*; essa consiste nella definizione di un vettore, di lunghezza pari al numero di categorie da rappresentare. Per ogni record, tale vettore avrà un 1 in posizione corrispondente alla categoria a cui appartiene e 0 in tutte le altre. Questo permette alla rete di imparare pesi diversi per ogni singola categoria, senza alcuna relazione di linearità (come invece potrebbe avvenire assegnando un valore diverso per ogni categoria)[39]. Il *one hot encoding* può essere applicato anche nel caso di *feature* numeriche; per fare ciò si divide il dominio dei possibili valori che la *feature* esaminata può assumere in gruppi, chiamati *bucket*, e ad ognuno di essi sarà associata una posizione nel vettore[38]. In altri termini si trasforma una *feature* numerica, che assume valori continui, in una *feature* categorica, che assume valori discreti, ed infine vi si applica il *one hot encoding*. Tale operazione, solitamente, si effettua se quella particolare *feature* non ha una qualche relazione di linearità con l'output della rete.

3.7 Regolarizzazione

Come già descritto, durante il *training* della rete si potrebbe incorrere nel problema dell'*overfitting*. La suddivisione dei dati di input nei tre insiemi (*training set*, *validation set* e *test set*) permette di individuare l'*overfitting* sul *training set* e di mitigarlo, considerando come configurazione dei pesi migliore quella che produce l'errore minore sul *validation set*. Esistono però altre metodologie per evitare l'*overfitting*: una di queste è la cosiddetta regolarizzazione. L'*overfitting* sull'insieme di *training* è infatti possibile solamente se il modello è sufficientemente complesso da riuscire ad adattarsi completamente alle caratteristiche dei dati di *training*. L'idea è quindi quella di ridurre, durante il *training*, la complessità del modello, andando a regolarizzarlo[35]. L'obiettivo principale della fase di *training* è quello di minimizzare l'errore:

$$minimize(loss)$$

Adottando invece una tecnica di regolarizzazione, non si andrà solamente a minimizzare l'errore, ma a minimizzare anche la complessità del modello:

$$minimize(loss + complexity)$$

Si rivela inoltre molto utile la possibilità di controllare l'aggressività della regolarizzazione. Tale controllo avviene attraverso un parametro λ (*regularization*

rate)[36], che viene moltiplicato per la complessità; esso definisce l'influenza che la complessità del modello avrà nel processo di minimizzazione:

$$minimize(loss + \lambda complexity)$$

Il valore λ può influire pesantemente sulla qualità dei risultati: un valore troppo piccolo potrebbe causare *overfitting*, mentre un valore troppo grande potrebbe causare *underfitting*, cioè il modello viene reso troppo semplice per il tipo di previsione da effettuare[36]. Le varie tecniche di regolarizzazione si differenziano, principalmente, per le modalità con cui rappresentano la complessità del modello.

Regolarizzazione L2

La *regolarizzazione L2* rappresenta la complessità del modello come la somma dei quadrati di ogni singolo peso w della rete[35].

$$complexity = w_1^2 + w_2^2 + w_3^2 + ... + w_n^2$$

Minimizzando la complessità attraverso la fase di *training* si otterranno valori, dei singoli pesi, tendenti a zero, ma non esattamente zero (per questioni legate al calcolo del gradiente)[37]. Per questo il numero di operazioni aritmetiche da effettuare per ottenere l'output della rete rimane costante.

Regolarizzazione L1

La *regolarizzazione L1* rappresenta invece la complessità del modello come il modulo della somma di ogni singolo peso *w* della rete[37].

$$complexity = |w_1 + w_2 + w_3 + ... + w_n|$$

Minimizzando la complessità attraverso la fase di *training*, per questioni legate al calcolo del gradiente, alcuni dei pesi si azzereranno completamente[37]. Questo permette di ridurre il numero di operazioni aritmetiche da effettuare per ottenere l'output della rete.

Dropout

Una tecnica di regolarizzazione diversa da quelle descritte precedentemente è la cosiddetta *Dropout Regularization*. L'idea è quella di azzerare temporaneamente e in modo casuale, durante il processo di training, l'output di alcuni nodi interni della rete[45]. Questo introduce una sorta di rumore all'interno della rete, che evita di eseguire *overfitting* sui dati. Anche in questo caso è presente un parametro, compreso tra 0 e 1, che indica la percentuale di nodi da azzerare ad ogni iterazione del processo di *training*.

3.8 Reti neurali semplici

Nell'esempio considerato precedentemente di regressione lineare era presente un singolo neurone, che operava moltiplicando gli input per un determinato peso e sommandoli tra loro. L'output era quindi una semplice combinazione lineare degli input. Si potrebbe pensare di aumentare la complessità di tale modello, aggiungendo altri neuroni, che, solitamente, come già accennato nella sezione introduttiva, sono raggruppati in *layer*. Ogni *layer* riceve gli input dal *layer* precedente e fornisce gli output al *layer* successivo. Il *layer* iniziale, connesso ai dati di input, è detto *input layer*, mentre l'ultimo, che fornisce l'output della rete è detto *output layer*; fra questi possono esserci vari *hidden layer*, che permettono di incrementare la complessità e le potenzialità della rete. Un esempio di rete a più livelli è mostrato in Figura 3.6.

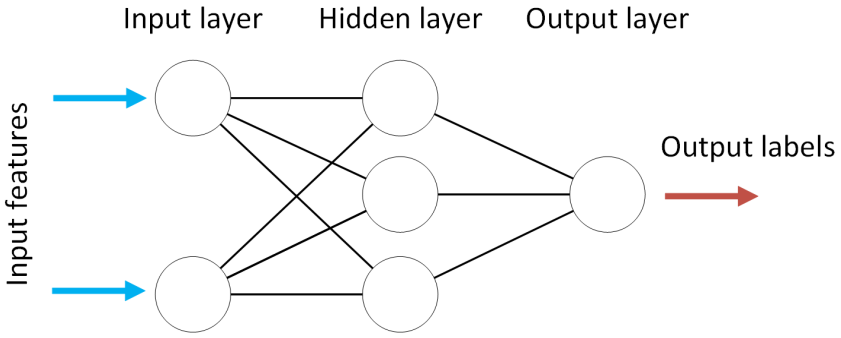

Figura 3.6: esempio di rete neurale a più livelli.

Anche nel caso di Figura 3.6 però, utilizzando i semplici neuroni finora descritti, si ottiene comunque, come output, una combinazione lineare dei dati di input[20]. Se l'obiettivo fosse quindi quello di descrivere una relazione complessa fra input e output, non lineare, sarebbe necessario aggiungere un qualche componente che modelli tale non linearità. Una soluzione adottabile è il *feature crossing*[10]: si aggiungono agli input della rete altre *feature* artificiali, ottenute applicando funzioni non lineari ad alcune delle *feature* originali. Questa tecnica può produrre buoni risultati, ma in casi complessi potrebbe essere non banale determinare quali *feature* artificiali aggiungere. Una soluzione più generale è quella di definire delle funzioni di attivazione, non lineari, per ogni neurone[20]; tale funzione sarà applicata all'output del neurone, prima che questo venga passato al neurone successivo. Solitamente, ai neuroni dello stesso *layer*, viene applicata la stessa funzione di attivazione. Tali funzioni vengono scelte in base alle necessità e permettono di modellare una non linearità all'interno della rete. Sovrapponendo quindi una serie di *layer* non lineari è possibile ottenere un modello estremamente complesso, capace di comprendere relazioni anche molto complicate. Durante la fase di *training*, per correggere i pesi, viene utilizzata una tecnica di

back-propagation[17, p. 40]: partendo dai nodi di output si calcolano i vari gradienti per ogni *layer*, aggiornando i pesi di conseguenza, fino a raggiungere l'*input layer*.

Funzioni di attivazione

Alcune delle funzioni di attivazione comunemente utilizzate[20], ma non le uniche, sono, ad esempio, la funzione *sigmoid* e la funzione *ReLU*. La funzione *sigmoid* genera valori sempre compresi fra 0 e 1; essa è così definita:

$$f(x) = \frac{1}{1+e^{-x}}$$

Il suo grafico è mostrato in Figura 3.7.

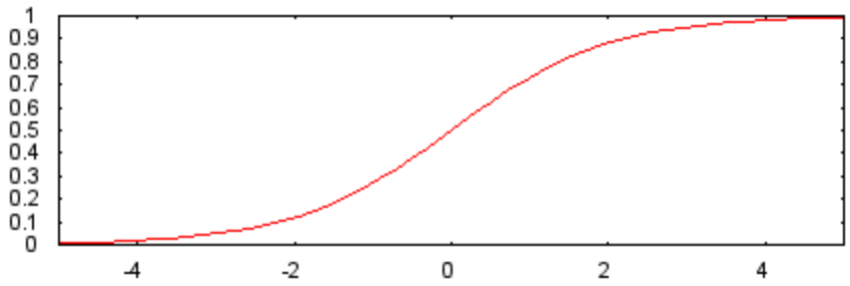

Figura 3.7: grafico della funzione *sigmoid* (sorgente: [20]).

La funzione *ReLU* è così definita:

$$f(x) = max(0,x)$$

Il suo grafico è mostrato in Figura 3.8.

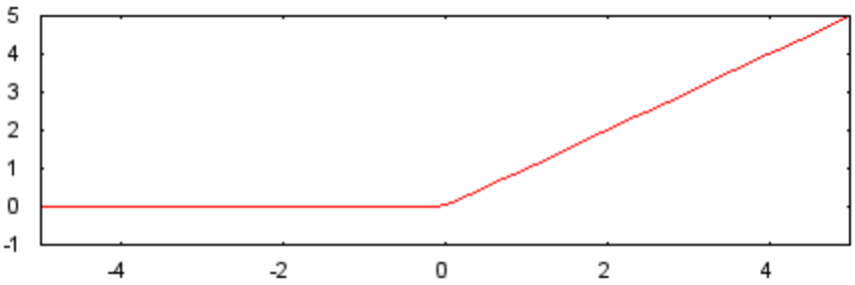

Figura 3.8: grafico della funzione *ReLU* (sorgente: [20]).

3.9 Reti neurali ricorrenti

Le reti considerate in precedenza associano un determinato input ad un determinato output. L'ordine con cui gli input sono presentati alla rete non influiscono sul risultato: ogni record è indipendente e le informazioni per calcolare l'output devono essere contenute completamente in esso. Esistono però problemi in cui il risultato può dipendere dalla particolare sequenza con cui si presentano i dati; in questo caso quindi l'output attuale non dipenderà solamente dal record di input attuale, ma anche dalla storia passata[17, p. 175]. Un problema di esempio potrebbe essere quello di voler prevedere la parola successiva ad una particolare frase. Ovviamente, in questo caso, non sarebbe del tutto sufficiente conoscere solamente l'ultima parola per poter prevedere la successiva: è necessario avere qualche informazione in più sulle parole che si sono presentate di recente, per determinare il contesto della frase. Una soluzione potrebbe essere quella di passare l'intera frase ad una rete di struttura simile a quelle viste precedentemente. Sorgono però due problemi: il primo è che le frasi potrebbero essere di dimensione variabile; il secondo è che una frase troppo lunga potrebbe comportare la definizione di un modello con un numero elevatissimo di *input feature*. Sorge quindi la necessità di una rete capace di operare su una sequenza di dati arbitraria, e di astrarre e memorizzare informazioni sull'ordine con cui i dati si presentano. Tali reti sono soprannominate reti neurali ricorrenti. Esse sono molto utili per effettuare previsioni in cui la sequenza dei dati è importante, ad esempio sequenze temporali, come previsioni di borsa o previsioni meteo. Una rete neurale ricorrente è formata da una serie di celle ricorrenti: una cella è paragonabile ad un neurone delle reti neurali viste in precedenza, ma ha una complessità più elevata (al suo interno sono infatti presenti altre piccole reti neurali capaci di mantenere una memoria).

Semplice cella ricorrente

Una cella ricorrente può essere pensata come una rete neurale, che fornito un input restituisce un output. Fra gli input non vi sono però solo le *input feature*, ma anche lo stato precedente della rete. Questo permette alla rete di mantenere una memoria interna, che memorizza alcune informazioni inerenti ai dati esaminati in precedenza[17, p. 176]. Lo schema di una semplice cella ricorrente è mostrato in Figura 3.9.

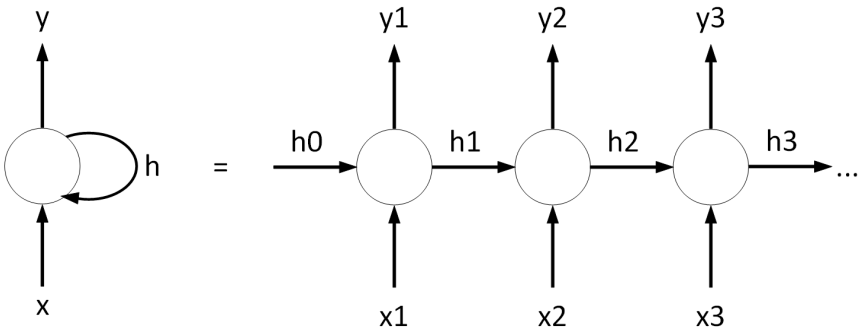

Figura 3.9: cella di una rete neurale ricorrente semplice, a sinistra nella notazione compatta a destra nella notazione estesa.

Nonostante una rete formata da celle di questo tipo abbia molte potenzialità, soffre di alcuni problemi, legati alla persistenza della memoria ed al calcolo del gradiente. Come per le reti neurali viste in precedenza, per l'aggiornamento dei pesi viene utilizzata una tecnica di *back-propagation*. In questo caso però l'output dipende anche dallo stato precedente della rete, per cui il calcolo del gradiente si propaga anche agli stati precedenti; questo processo è detto *backpropagation through time (BPTT)*[17, p. 184]. Poiché gli stati precedenti potrebbero essere numerosi, si potrebbe verificare un problema di *vanishing gradient* o *exploding gradient*: il primo caso si verifica se il gradiente raggiunge valori troppo piccoli, il secondo se raggiunge valori troppo elevati, diventando *NaN* (not a number). Questo può capitare poiché il calcolo del gradiente può essere scomposto in un prodotto di altri gradienti inerenti allo stato precedente, che a loro volta possono essere scomposti in prodotti di gradienti inerenti agli stati precedenti[17, p. 186]. Se i valori da moltiplicare sono minori di zero, bastano poche moltiplicazioni per raggiungere un valore estremamente piccolo; al contrario, moltiplicando molti valori maggiori di zero, si potrebbero eccedere facilmente i limiti di un *floating-point*. Il problema del *vanishing gradient* causa una limitata persistenza della memoria della rete, poiché i dati visti in un periodo lontano, avendo un gradiente molto basso, contribuiranno minimamente al training della rete.

Celle LSTM e GRU

I problemi relativi alle celle ricorrenti semplici sono risolti da celle più complesse: LSTM (Long Short Term Memory)[17, p. 187] e GRU (Gated Recurrent Unit)[17, p. 196]. Queste celle utilizzano, al loro interno, dei cosiddetti *gate*, che permettono di selezionare quali segnali di input o output tenere in considerazione, regolandone l'intensità. Le celle GRU sono leggermente più semplici di LSTM, ma ottengono risultati molto simili con costi di *training* inferiori[17, p. 196]. Le celle LSTM e GRU sono interscambiabili senza problemi; la scelta della prima o della seconda può essere determinata in base alle prestazioni sullo specifico problema[17, p. 176].

Capitolo 4

Tecnologie disponibili

Le reti neurali considerate precedentemente, potrebbero essere implementate direttamente tramite un qualsiasi linguaggio di programmazione, definendo opportune classi o strutture per modellare i neuroni e le loro connessioni. Questo modo di procedere, utilizzando il linguaggio Java, è approfonditamente descritto nel libro *Neural Network Programming with Java*[41]. Un approccio di questo genere permetterebbe sicuramente di comprendere meglio la teoria e le caratteristiche delle reti neurali artificiali, ma, dall'altro lato, sarebbe un'operazione molto dispendiosa; inoltre, probabilmente, si otterrebbe un'implementazione poco efficiente e difficilmente adattabile. La soluzione più comoda ed efficace consisterebbe quindi nell'individuare librerie già implementate, che permettano di definire, in modo semplice e veloce, una qualsiasi rete neurale. Dopo aver effettuato alcune ricerche è stato possibile individuare numerose librerie e *framework* di questo tipo, alcune sviluppate dalle maggiori aziende informatiche, altre da piccoli team o singoli sviluppatori indipendenti. Analizzando lo scenario attuale inerente a tali librerie, il *framework* predominante, in termini di vastità della documentazione e popolarità fra i programmatori, risulta essere TensorFlow[4], sviluppato da Google. Nonostante TensorFlow sia molto efficiente nella gestione di reti neurali, non mette a disposizione delle modalità intuitive ed immediate per la loro definizione. La modalità più semplice per definire tali reti risulta essere quella offerta da Keras, un'API che, appoggiandosi su TensorFlow, offre una serie di procedure semplici ed intuitive per la loro gestione. Anche il *framework* Scikit-learn, come TensorFlow, permette di gestire reti neurali, ma si è scelto di non utilizzarlo a tale scopo; ciononostante mette a disposizione una serie di funzioni molto comode per l'analisi e la manipolazione dei dati. Inoltre, poiché risulterebbe molto scomodo rappresentare i dati di input o output come semplici vettori multidimensionali, si rivela molto utile servirsi di librerie specifiche per la rappresentazione e manipolazione di dati di questo tipo. Una delle più popolari, consigliata anche dalla documentazione di TensorFlow, risulta essere Pandas. Queste librerie sono utilizzabili attraverso il linguaggio di programmazione Python.

4.1 TensorFlow

TensorFlow è una libreria *open source* sviluppata originariamente dal team di Google *Google Brain*; inizialmente doveva essere utilizzata per scopi interni all'azienda ma, successivamente, nel Novembre 2015, venne pubblicata come libreria *open source*. La peculiarità di TensorFlow è la capacità di eseguire computazioni numeriche ad alte prestazioni, indipendentemente dall'hardware sottostante: le stesse operazioni possono essere infatti eseguite, senza adottare particolari accorgimenti, sulla CPU, sulla GPU, su *cluster* di server o su dispositivi mobili. Tale capacità di eseguire computazioni numeriche ad alte prestazioni rende questo *framework* particolarmente adatto non solo per la gestione di reti neurali, ma anche per applicazioni di calcolo scientifico di natura differente. L'elevata efficienza di calcolo di questa libreria è determinata dalla previa definizione di un grafo, che, una volta compilato, verrà mandato in esecuzione. TensorFlow mette a disposizione una serie di API, che permettono di operare a diversi livelli di astrazione: le API di livello più basso sono più vicine all'hardware e permettono di operare direttamente sul grafo, mentre le API di livello più alto permettono di effettuare operazioni complesse, come la gestione di reti neurali, in maniera molto più semplice, ma con meno libertà di azione[11]. La gerarchia delle varie componenti di TensorFlow è riportata in Figura 4.1.

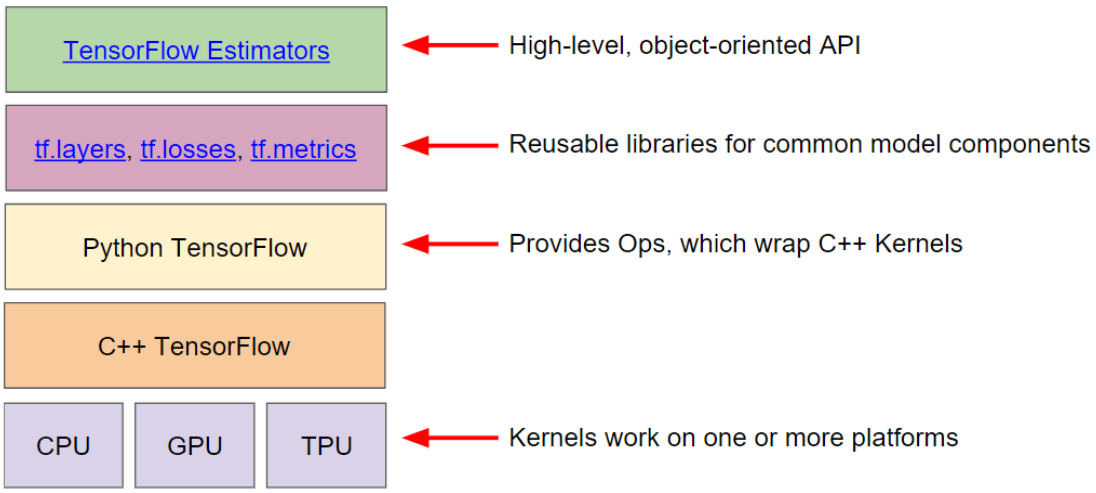

Figura 4.1: gerarchia delle componenti di TensorFlow (sorgente: [11]).

Il principale linguaggio attraverso cui utilizzare TensorFlow è Python, che, di per sé, non è molto efficiente per effettuare calcoli scientifici. Infatti il livello di TensorFlow più

vicino all'hardware è stato implementato in C++, utilizzando Python solo per esporre le API. TensorFlow può eseguire le operazioni richieste in modo trasparente su hardware di tipo differente, come CPU, GPU, o architetture più complesse. Al livello più alto di astrazione sono disponibili diverse API (come gli *estimators* mostrati in Figura 4.1), che offrono modalità per effettuare pressoché le stesse operazioni per la gestione di reti neurali, ma in maniera differente. Lo scenario di queste API non è però ben definito, poiché ne esistono varie e nessuna di esse è riconosciuta ufficialmente come principale. Analizzando però le varie caratteristiche di tali API, sembrerebbe che quella più facile da utilizzare e con le maggiori potenzialità sia Keras.

4.2 Keras

Keras è un'API che offre un'interfaccia di alto livello, per la gestione di vari tipi di rete neurale. Il punto di forza di questa libreria è la possibilità di definire, addestrare e valutare una qualsiasi rete neurale, in maniera semplice e veloce, facilitando al massimo l'utilizzo da parte del programmatore. Keras è inoltre altamente modulare, permettendo di combinare ed aggiungere ad una rete, in maniera intuitiva, i diversi moduli necessari, come nuovi *layer* di neuroni, diverse funzioni di attivazione o diversi ottimizzatori. Tutte le operazioni ripetitive inerenti alla gestione della rete neurale sono automaticamente gestite dalla libreria, permettendo al programmatore di scrivere meno codice e di concentrarsi sulle caratteristiche del modello più importanti. La libreria Keras è stata scritta in linguaggio Python, e, può operare, in maniera trasparente, sopra diverse librerie di calcolo; fra queste vi sono TensorFlow, CNTK o Theano. Attraverso poche righe di codice è possibile definire, addestrare e valutare una qualsiasi rete neurale, come mostrato in Elenco 4.1.

Elenco 4.1: esempio di definizione di una rete neurale tramite Keras.

```
# model definition
model = Sequential()
model.add(layers.Dense(3, input_dim=2, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer=Adam(), loss='mae')
# training on training set
model.fit(x_train, y_train, epochs=5, batch_size=32)
# evaluation on test set
loss = model.evaluate(x_test, y_test)
# prediction on test set
previsions = model.predict(x_test)
```

4.3 Scikit-learn

Scikit-learn è una libreria che mette a disposizione vari strumenti per l'applicazione di operazioni inerenti al *machine learning*, fra cui quelle per la gestione di semplici reti neurali. Essa offre quindi un'interfaccia di programmazione simile, anche se per molti aspetti differente, a TensorFlow, ma è orientata più sulla semplicità di apprendimento da parte del programmatore, piuttosto che sull'efficienza, estensibilità e varietà delle operazioni disponibili. Scikit-learn è adatto per compiere i primi passi verso il *machine learning*, ma non mette a disposizione una gamma di operazioni completa per la gestione di reti neurali complesse. Per questo si è scelto di non adottare tale libreria per la gestione dei vari modelli di rete; sono state però utilizzate alcune funzioni utili di questo *framework*, ad esempio per scalare fra zero e uno i dati di input o per calcolare l'errore medio.

4.4 Pandas

Pandas è una libreria che mette a disposizione procedure per analizzare e modellare un insieme di dati in formato tabellare. Molti *framework* di *machine learning*, come TensorFlow, supportano come input strutture dati relative a questa libreria. Pandas permette di definire tabelle (*DataFrame*) o serie di dati (*Series*); una tabella è un insieme di *Series*, una per ogni colonna. Un *DataFrame* può essere creato, ad esempio, importando un file CSV o JSON; una volta creata l'istanza del *DataFrame*, è possibile estrarre informazioni o effettuare manipolazione dei dati in maniera molto semplice ed intuitiva. Ad esempio, attraverso il metodo *DataFrame.describe()* è possibile ottenere alcune informazioni utili sui dati contenuti nella tabella, come valore medio, massimo, minimo o deviazione standard; attraverso *DataFrame.hist()* è possibile stampare a video grafici relativi ai dati, oppure attraverso *DataFrame.interpolate()* è possibile riempire i dati mancanti attraverso un'interpolazione. Pandas si rivela quindi uno strumento molto utile, soprattutto in ambito *machine learning*, dove le operazioni di analisi e manipolazione dei dati ricoprono un ruolo fondamentale nell'ottenimento di buoni risultati.

Capitolo 5

Analisi dei dati e definizione obiettivi

Nel Capitolo 1 e nel Capitolo 2 sono state descritte le principali fonti dati e le modalità con cui essi sono stati estratti. Sarà ora necessario, tenendo in considerazione le nozioni apprese nel Capitolo 3 e nel Capitolo 4, analizzare i dati raccolti, per determinare quali tipi di previsione sia possibile effettuare su essi, e quali strategie adottare.

5.1 Analisi dei dati

I dati che si andranno ad analizzare, determinandone la qualità e le principali caratteristiche, provengono esclusivamente dai file CSV e JSON messi a disposizione sul portale OpenData da parte di Arpa Emilia-Romagna. Tali dati sono inerenti alle misurazioni, prodotte da stazioni ufficiali, dei vari inquinanti, ed alle condizioni meteo degli ultimi anni. Una prima analisi è consistita nella localizzazione delle stazioni su una mappa, analizzando le coordinate presenti nella tabella *aq_station* del database. Per fare ciò, tale tabella è stata esportata in formato CSV, importando poi ogni record in un applicativo capace di visualizzare tali *marker* su una mappa (Google Earth). La prima caratteristica osservata è che ogni stazione presente in tale tabella può effettuare solamente alcuni tipi di misurazione: alcune non sono provviste dei sensori necessari per la misurazione di determinati inquinanti, ed inoltre, nessuna stazione di monitoraggio della qualità dell'aria corrisponde con una stazione meteorologica.

Dati sulla qualità dell'aria

Gli inquinanti possibili fra le varie misurazioni sono: SO_2 (Biossido di zolfo), CO (Monossido di carbonio), $PM_{2.5}$, PM_{10}, C_6H_6 (Benzene), $C_6H_5CH_3$ (Toluene), NO (Monossido di azoto), O_3 (Ozono), NO_2 (Biossido di azoto), o-xylene; ognuno misurato in $\mu g/m^3$. Ovviamente nessuna delle stazioni è capace di monitorare tutti questi composti, ma solamente un sottoinsieme di essi. Analizzando i dati, si è stabilito

che quasi la totalità delle stazioni è capace di effettuare misurazioni di PM_{10} e NO_2, mentre molte, ma non tutte, sono capaci di misurare C_6H_6, CO, $PM_{2.5}$ e O_3. La maggior parte delle misurazioni hanno inoltre una distanza temporale di un'ora l'una dall'altra; solamente alcune, come PM_{10} e $PM_{2.5}$ hanno invece solamente misurazioni giornaliere. Prendendo come esempio la concentrazione di NO_2, la collocazione delle stazioni in Emilia-Romagna capaci di effettuare tale misurazione è riportata in Figura 5.1.

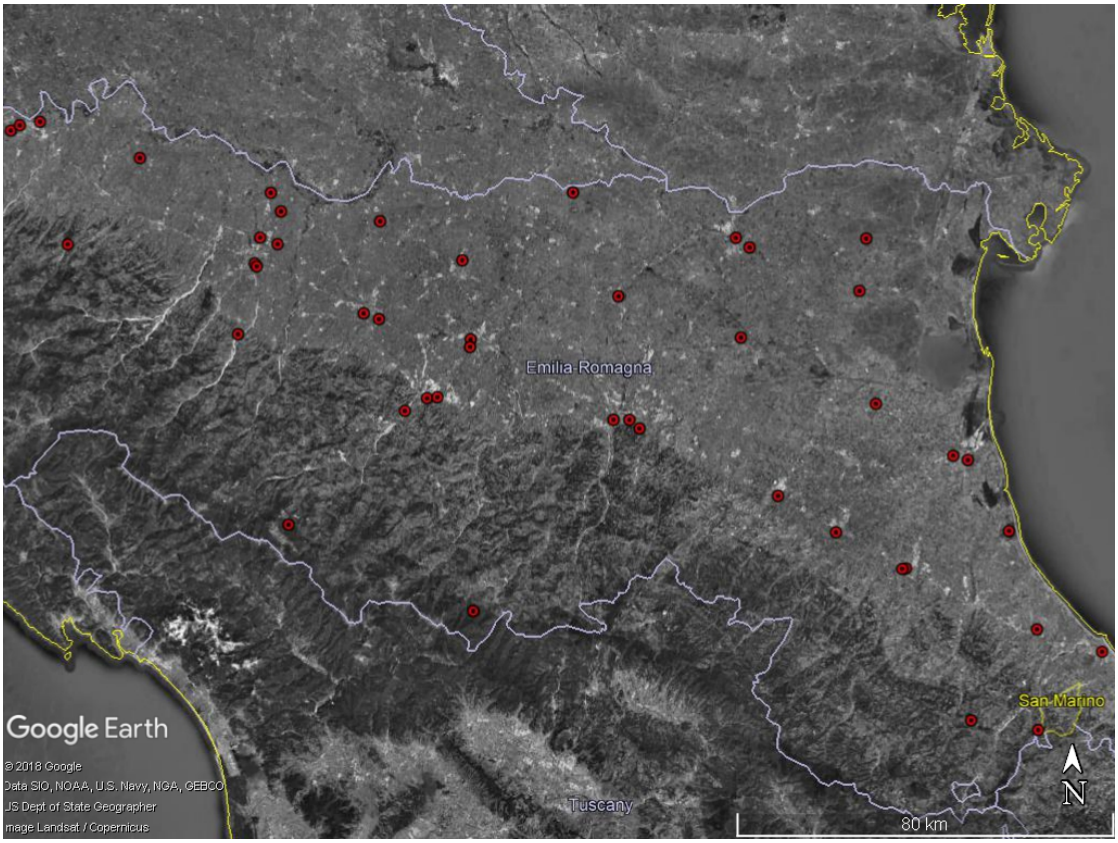

Figura 5.1: distribuzione delle stazioni di monitoraggio NO_2 (marker rosso) in Emilia-Romagna.

Come già descritto, le stazioni capaci di misurare altri tipi di inquinante non corrisponderanno esattamente a quelle riportate in Figura 5.1 (probabilmente saranno in numero inferiore, dato che la quasi totalità delle stazioni di monitoraggio della qualità dell'aria è capace di misurare la concentrazione di NO_2). Come è possibile notare dalla Figura 5.1, le stazioni di monitoraggio della qualità dell'aria non sono presenti in numero molto elevato e questo sarà un fattore da tenere presente nella definizione degli obiettivi.

Dati meteorologici

I tipi di misurazione inerenti alle condizioni meteo sono molteplici; le più significative sono temperatura (kelvin), precipitazioni (mm), pressione (hPa), umidità (percentuale), velocità del vento (m/s), direzione del vento (gradi), precipitazione nevosa (mm), radiazione visibile (W/m^2), radiazione solare globale (J/m^2). Le misurazioni inerenti a queste grandezze sono disponibili a varie frequenze temporali e con tipo di aggregazione differente (valore massimo, minimo, medio, istantaneo). Prendendo come esempio la temperatura, la collocazione delle stazioni in Emilia-Romagna capaci di effettuare tale misurazione è riportata in Figura 5.2.

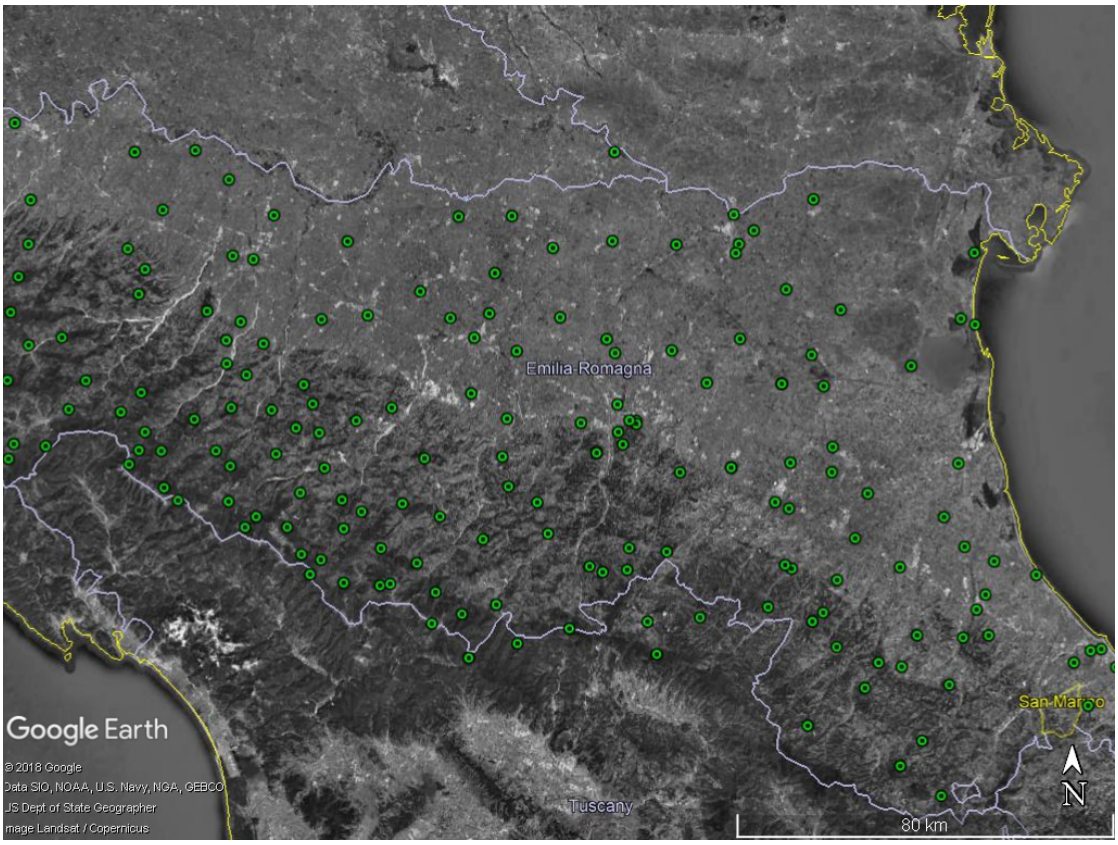

Figura 5.2: distribuzione delle stazioni di monitoraggio temperatura (marker verde) in Emilia-Romagna.

Anche in questo caso, le stazioni capaci di misurare altri tipi di grandezze non corrisponderanno esattamente a quelle riportate in Figura 5.2; inoltre, molto probabilmente, quasi nessuna stazione meteorologica corrisponderà esattamente ad una stazione di monitoraggio della qualità dell'aria. In questo caso però, come è possibile notare dalla Figura 5.2, le stazioni meteorologiche sono presenti in numero superiore.

Anche questi fattori saranno da tenere in considerazione durante la definizione degli obiettivi.

Dati mancanti

Analizzando accuratamente i dati e visualizzandoli su un grafico è possibile notare facilmente che, per alcuni intervalli temporali, che possono variare da poche ore a varie settimane, le misurazioni risultano assenti; un esempio di ciò è mostrato in Figura 5.3, in cui mancano le misurazioni per l'inquinante NO_2 per circa un mese.

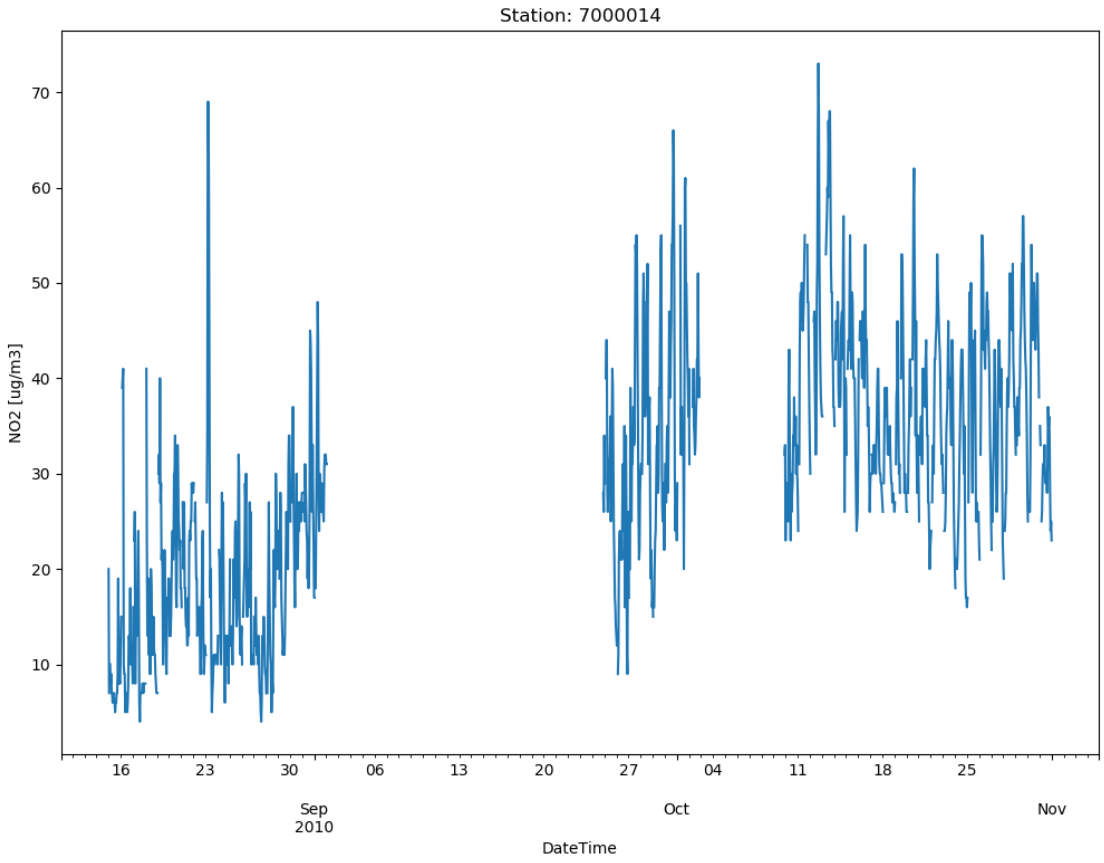

Figura 5.3: esempio di intervallo temporale in cui i dati risultano incompleti (in questo caso per la concentrazione di NO_2).

Ciò non accade solo per le misurazioni relative agli inquinanti, ma anche per le misurazioni meteorologiche. Questo sarà un altro fattore da tenere presente durante la definizione degli obiettivi.

5.2 Obiettivi da raggiungere

La fase di analisi dei dati ha fatto sorgere alcune problematiche, che dovranno essere risolte. Tenendo quindi presenti le caratteristiche dei dati a disposizione e le tecnologie utilizzabili, si andrà a delineare un percorso che porterà, auspicabilmente, al raggiungimento di risultati apprezzabili. L'obiettivo principale sarà quello di definire un modello capace di effettuare previsioni su un particolare inquinante atmosferico.

Tecniche e tecnologie adottate

Per effettuare le previsioni saranno adottate le tecniche e tecnologie descritte, rispettivamente, nel Capitolo 3 e nel Capitolo 4. La previsione verrà quindi effettuata attraverso una qualche rete neurale artificiale, che sarà definita attraverso una serie di prove sperimentali. L'addestramento della rete avverrà fornendole una serie di misurazioni dell'inquinante, più qualche altra *feature* corrispondente ai dati meteorologici, inerenti ad un determinato intervallo temporale; essa, partendo da tali dati, dovrà quindi cercare di prevedere una misurazione futura di quel particolare inquinante, confrontandola poi con quella reale. L'addestramento potrà avvenire poiché il valore reale con cui confrontare la previsione sarà presente fra le misurazioni memorizzate nel *database*.

Scelta inquinante da prevedere

L'inquinante più adatto su cui effettuare tale previsione sembrerebbe essere l'NO_2 (biossido di azoto), principalmente perché è il composto a cui è associato il maggior numero di stazioni all'interno del *database* e che ha il maggior numero di record. Esistono molte stazioni anche per il PM_{10}, ma la frequenza con cui le misurazioni di tale inquinante vengono effettuate è molto inferiore (giornaliera), e, per questo, il numero di record ad esso associati è molto più scarso; tale scarsità di dati potrebbe rendere difficoltoso l'addestramento della rete neurale.

Distanza temporale della previsione

Le misurazioni inerenti al biossido di azoto sono presenti con frequenza oraria. Si potrebbe quindi pensare di prevedere, partendo da un insieme di record con misurazioni fino ad una determinata ora, il valore di inquinamento nell'ora successiva. Una previsione di una singola ora non sarebbe però molto interessante, e, probabilmente, sarebbe anche abbastanza semplice da prevedere. Una previsione più difficile e rilevante sarebbe invece quella a distanza di 24 ore; in questo caso si potrebbero mettere fortemente alla prova le capacità di previsione di una rete neurale artificiale.

Luogo della previsione

Il luogo o i luoghi sui quali si effettueranno le previsioni corrisponderanno alle locazioni delle stazioni di monitoraggio del biossido di azoto. In Emilia-Romagna tali stazioni sono circa quaranta; poiché, almeno inizialmente nella fase di sperimentazione, risulterebbe poco pratico lavorare con un modello capace di effettuare previsioni in circa quaranta luoghi diversi, ci si concentrerà esclusivamente nella zona del centro di Bologna, dove sono presenti tre stazioni di monitoraggio del biossido di azoto. La rete restituirà quindi esattamente tre output, ognuno corrispondente al valore di inquinamento previsto per ogni stazione di Bologna. Si è scelto questo approccio poiché la densità spaziale delle varie stazioni è molto scarsa; questa caratteristica rende poco adeguata l'adozione di altre strategie, come ad esempio la suddivisione dello spazio in una griglia, poiché non si avrebbero abbastanza luoghi di misurazione per riempire tutte le caselle (considerando una dimensione di queste non esageratamente ampia). Le coordinate geografiche relative ad ogni stazione di monitoraggio non saranno fornite come input alla rete, poiché esse non forniscono alcuna informazione significativa che possa migliorare la previsione; la rete, infatti, può benissimo comprendere le relazioni che sussistono fra le tre stazioni, senza conoscere i punti esatti dove queste sono localizzate.

Dati meteo di supporto alla previsione (input features)

Come già descritto nel Capitolo 3, è necessario individuare le *feature* da fornire in input al modello. La *feature* di input ovvia è sicuramente il valore del biossido di azoto; probabilmente, però, utilizzare solamente questa *feature* non porterebbe a buoni risultati, poiché il valore di inquinamento non varia in modo regolare, ma dipende da molti altri fattori; alcuni di essi potrebbero essere, ad esempio, data e ora, oppure le condizioni meteorologiche. Fra le *feature* di input serviranno quindi sicuramente anche delle informazioni temporali, più qualche informazione sulle condizioni meteorologiche per migliorare la qualità della previsione. Saranno quindi adottate come *feature* di input le più significative, da cui l'inquinamento atmosferico probabilmente dipende; esse sono:

Temperatura La temperatura altera le proprietà dell'atmosfera, influendo sulla velocità di dispersione degli inquinanti atmosferici.

Umidità Come per la temperatura, l'umidità altera le proprietà dell'atmosfera, influendo sulla velocità di dispersione degli inquinanti atmosferici.

Precipitazioni Le precipitazioni riescono ad intrappolare gli inquinanti atmosferici, portandoli al suolo e riducendo quindi la loro concentrazione nell'aria.

Velocità e direzione del vento Il vento, spostando le masse d'aria, muove o disperde gli inquinanti atmosferici.

Le *feature* di input saranno relative alle stazioni di monitoraggio del biossido di azoto; poiché le stazioni meteorologiche si trovano in luoghi differenti, sarà necessario definire delle procedure per associare alle misurazioni del biossido di azoto le misurazioni meteorologiche più appropriate (ad esempio le più vicine spazialmente).

Riempimento dati mancanti

Come osservato nella sezione precedente, per alcuni intervalli temporali non è presente una serie di misurazioni completa. Dato che non è possibile fornire in input alla rete un valore non presente (*NaN*), sarà necessario definire una procedura per completare i dati mancanti; se però, per un determinato periodo, i record incompleti fossero troppi, sarà necessario scartare completamente quell'intervallo temporale.

Obiettivo finale

Ricapitolando, l'obiettivo finale sarà quello di ottenere un modello, sotto forma di rete neurale, capace di prevedere il livello di concentrazione del biossido di azoto fra 24 ore, per ognuna delle tre stazioni presenti a Bologna. I dati di input da fornire alla rete per effettuare tale previsione saranno un insieme di record, inerenti ad un determinato intervallo temporale di lunghezza arbitraria, contenenti i valori di concentrazione di biossido di azoto, temperatura, umidità, precipitazioni, vento e direzione del vento per ogni stazione. La rete dovrà quindi prevedere la concentrazione di biossido di azoto, per ogni stazione, 24 ore dopo all'ultimo record fornitole (il più recente). Per fornire alla rete i dati strutturati in questo modo sarà necessario definire un'adeguata interfaccia e procedura di conversione. Il tipo e la struttura della rete neurale da utilizzare non sono ben noti; sarà quindi necessario effettuare una serie di prove sperimentali, per individuare un modello sufficientemente adeguato. Successivamente alla sua individuazione, si effettueranno altre prove, adottando particolari strategie, per osservare e discutere la risposta del modello a tali cambiamenti.

Capitolo 6

Progettazione

Tenendo in considerazione gli obbiettivi e le condizioni definite nel Capitolo 5, si andranno a definire le tecniche adottate per predisporre i dati in modo che possano essere utilizzati come input di una rete neurale artificiale.

6.1 Nuova struttura dei dati

I record memorizzati sul database hanno una struttura simile a quella mostrata in Tabella 6.1; sono stati omessi, per semplicità, alcuni attributi.

DateTime	Value	ValueType	Latitude	Longitude	Station
01/01/18 01:00:00	16	2	44.48	11.36	7000014
01/01/18 01:00:00	274	1000	44.60	11.20	250014
01/01/18 01:00:00	70	1003	44.55	11.26	8500324
01/01/18 01:00:00	19	2	44.50	11.33	7000015
01/01/18 02:00:00	15	2	44.48	11.36	7000014
01/01/18 02:00:00	273	1000	44.60	11.20	250014
01/01/18 02:00:00	71	1003	44.55	11.26	8500324
01/01/18 02:00:00	16	2	44.50	11.33	7000015

Tabella 6.1: struttura dei record nel database.

Come possibile notare in Tabella 6.1, è presente un record in tabella per ogni misurazione; ognuna di esse ha una data, un valore, un tipo valore (ad esempio 2=Biossido di azoto, 1000=temperatura, 1003=umidità) e le coordinate geografiche, oltre che al riferimento alla stazione che ha effettuato tale misurazione. I dati strutturati in questo modo non sono però adatti per essere forniti ad una rete neurale: è necessario individuare una modalità per poterli raggruppare, in modo che ogni record, come

specificato nel Capitolo 5, contenga la concentrazione dell'inquinante e le informazioni meteo, per ogni stazione di monitoraggio del biossido di azoto presa in considerazione. In Tabella 6.1, le stazioni di monitoraggio del biossido di azoto erano due: 7000014 e 7000015. Si andrà quindi a generare una nuova serie di record, in cui una particolare data comparirà una sola volta, che conterranno le varie misurazioni per ogni stazione individuata (in questo esempio 7000014 e 7000015). Un esempio di tale conversione è mostrato in Tabella 6.2.

Station	7000014			7000015		
Parameter	2	1000	1003	2	1000	1003
DateTime						
01/01/18 01:00:00	16	274	70	19	274	70
01/01/18 02:00:00	15	273	71	16	273	71

Tabella 6.2: nuova struttura dei record adatta per essere utilizzata come input di una rete neurale.

Come è possibile notare in Tabella 6.2, alle stazioni di monitoraggio del biossido di azoto sono state associate le misurazioni meteorologiche, provenienti da altre stazioni. Questa procedura di assegnamento sarà meglio definita nella prossima sezione. Per ottenere il risultato mostrato in Tabella 6.2, è stata eseguita, sul *database*, una *query* di raggruppamento per data con frequenza oraria; i dati raggruppati sono stati quindi strutturati in formato JSON, attraverso le funzioni di PostgreSQL *json_agg* e *json_build_object*. La struttura del risultato della *query* descritta è mostrata in Tabella 6.3.

DateTime	JSON
01/01/18 01:00:00	*Records...*
01/01/18 02:00:00	*Records...*
01/01/18 03:00:00	*Records...*

Tabella 6.3: risultato *query* di raggruppamento per data con frequenza oraria.

Dal risultato di tale *query* sono state quindi estratte le misurazioni necessarie, strutturandole nel modo precedentemente descritto. Tale operazione è stata implementata in Python usufruendo degli strumenti messi a disposizione dalla libreria Pandas, descritta nel Capitolo 4; si è quindi ottenuto come risultato un'istanza di *DataFrame*, avente una struttura simile alla Tabella 6.2 (a differenza dell'esempio, che omette qualche parametro, l'istanza di *DataFrame* reale avrà gli attributi definiti nel Capitolo 5).

6.2 Associazione delle misurazioni meteorologiche

Per effettuare le operazioni descritte nella sezione precedente, in particolare nel processo di trasformazione del risultato della *query* (Tabella 6.3) nella Tabella 6.2, è stato necessario adottare una particolare strategia per assegnare le misurazioni meteorologiche più adatte alle varie stazioni di monitoraggio del biossido di azoto. Si sono presentate due alternative principali:

Record meteorologico più vicino In questo caso, per ogni record presente nel JSON di Tabella 6.3, si associa alla stazione di monitoraggio il record meteorologico con coordinate più vicine ad essa; questo comporta che, in una particolare colonna di Tabella 6.2, potrebbero essere presenti record appartenenti a stazioni meteorologiche diverse, poiché, per ogni ora, si seleziona sempre la misurazione disponibile più vicina.

Stazione meteorologica più vicina In questo caso si determina a priori, per ogni stazione di monitoraggio e per ogni grandezza meteorologica, la stazione più vicina dalla quale reperire la misurazione; tale associazione viene salvata, ad esempio, su un dizionario. Per ogni record presente nel JSON di Tabella 6.3, si va a ricercare, per il parametro preso in considerazione, il record corrispondente alla stazione definita nel dizionario. Se la misurazione è presente viene inserita nella nuova tabella, altrimenti si inserisce un valore nullo. In questo caso, i record meteorologici di una particolare colonna di Tabella 6.2, apparterranno sempre alla stessa stazione (definita nel dizionario).

Fra le due alternative si è preferita la seconda, cioè la determinazione a priori della stazione meteorologica più vicina. La prima infatti crea problematiche nel caso in cui, in determinate ore, manchino i record relativi alla stazione meteorologica più vicina. In tal caso, la procedura selezionerebbe, se presente, il record relativo alla seconda stazione più vicina, procedendo in questo modo fino ad individuare la misurazione presente più prossima. Se le stazioni fossero molto distanti fra loro, o se si verificasse un caso sfortunato in cui tutte le misurazioni più vicine non fossero presenti, vi sarebbe il rischio di associare delle misurazioni spazialmente molto distanti, che probabilmente si discosterebbero notevolmente dal valore reale di quella zona. In questo caso, probabilmente, si sarebbero ottenuti migliori risultati generando i valori mancanti attraverso un'interpolazione sul tempo, piuttosto che recuperarli da misurazioni reali ma distanti. Questo è esattamente il motivo per cui si è preferito adottare la seconda soluzione: si prelevano tutte le misurazioni dalla stazione più vicina che abbia un numero di record sufficiente, e si interpolano i dati mancanti. Ciò consente di ottenere dati di qualità superiore (ovviamente se i dati mancanti non sono troppi), permettendo probabilmente alla rete neurale di raggiungere migliori risultati.

6.3 Eliminazione o completamento dei dati mancanti

Le colonne della nuova tabella generata, potrebbero non essere complete, a causa di una mancanza dei dati sorgente in un particolare intervallo temporale, come già descritto nel Capitolo 5. Dato che la tabella da fornire alla rete deve essere necessariamente completa, è necessario adottare una qualche strategia per generare i dati mancanti. Sia le misurazioni del biossido di azoto, sia le misurazioni meteorologiche potrebbero presentare questo problema; nel caso l'intervallo temporale caratterizzato da tale mancanza fosse molto breve, si può procedere con una semplice interpolazione lineare rispetto al tempo. Per fare ciò è sufficiente utilizzare il metodo *DataFrame.interpolate(method='time')*, reso disponibile dalla libreria Pandas; l'interpolazione avverrà su tutti i dati mancanti della tabella (*DataFrame*). Nel caso invece in cui l'intervallo caratterizzato dalla mancanza sia lungo, è necessario adottare altre strategie: se i dati mancanti sono relativi alle condizioni meteo, si potrebbero semplicemente prelevare le misurazioni da un'altra stazione vicina; se invece, nel caso più grave, fossero mancanti i dati sul biossido di azoto, si potrebbe essere costretti ad eliminare interamente quel particolare intervallo dai dati di input. L'eliminazione di un intervallo potrebbe però non essere banale, poiché si dovrà lavorare su delle sequenze contigue di dati; è quindi preferibile non frammentare l'intervallo relativo ai dati di input.

6.4 Aggiunta colonne con informazioni temporali

Per aiutare la rete nell'operazione di previsione, saranno aggiunte alcune *feature* artificiali, ricavate dalle informazioni temporali; da queste dovrebbe dipendere, almeno in parte, il valore dell'inquinamento. Le *feature* che saranno aggiunte sono:

Mese Valore intero fra 1 e 12. Dal mese è possibile risalire alla stagione, e la stagione può determinare condizioni meteo differenti, permettendo la dispersione degli inquinanti con maggiore o minore velocità; inoltre, i mesi più freddi saranno caratterizzati da un inquinamento maggiore, a causa dei sistemi di riscaldamento domestici.

Giorno della settimana Valore intero fra 1 e 7. Il giorno della settimana potrebbe fornire una stima dell'inquinamento; infatti i giorni lavorativi potrebbero essere caratterizzati da un maggiore inquinamento atmosferico.

Ora Valore intero fra 0 e 23. Anche l'ora potrebbe fornire una stima dell'inquinamento: probabilmente, nelle ore di punta, l'aria sarà molto più inquinata rispetto ad orari notturni.

Era stato anche considerato, come *feature* aggiuntiva, il giorno dell'anno (fra 1 e 365), ma si è deciso di non aggiungerlo, vista la scarsa frequenza con cui i singoli valori comparirebbero fra i dati di input. I dati a disposizione coprono infatti solamente gli ultimi 4-5 anni; per poter adottare tale *feature* sarebbe stato necessario un insieme di dati che coprisse un intervallo più lungo.

6.5 Estrazione delle label da prevedere

Nel Capitolo 5 ci si è posti come obiettivo di effettuare una previsione 24 ore dopo al record più recente della sequenza fornita alla rete. Per poter effettuare la fase di *training*, la rete deve quindi avere il valore reale con cui confrontare la previsione. Sarà quindi necessario generare una nuova tabella (*DataFrame*), che contenga, per ogni record della tabella generata precedentemente (Tabella 6.2), la corrispondente *label* da prevedere. La *label* per ogni record consisterà in tre valori distinti, poiché la rete deve prevedere la concentrazione del biossido di azoto per ognuna delle tre stazioni di Bologna. Per generare la tabella contenente le *label* è sufficiente estrarre dalla tabella generata precedentemente (Tabella 6.2) le colonne relative alle misurazioni del biossido di azoto e, successivamente, effettuare su di esse uno *shift* di meno 24 ore; ciò equivale ad eliminare i primi 24 record dalla tabella delle *label* e gli ultimi 24 record dalla tabella dei dati di input (che non avrebbero *label* associate). Un esempio di tale procedura, considerando una previsione di due ore nel futuro e due stazioni di monitoraggio, è mostrata in Tabella 6.4 e Tabella 6.5.

Station	7000014		7000015	
Parameter	2	1000	2	1000
DateTime				
01/01/18 01	16	273	19	274
01/01/18 02	15	274	18	273
01/01/18 03	14	273	17	274
01/01/18 04	13	274	16	273
01/01/18 05	12	273	15	274
01/01/18 06	11	274	14	273

Tabella 6.4: dati di input da fornire alla rete (molti parametri sono stati omessi per semplicità).

Station	7000014	7000015
Parameter	2	2
DateTime		
01/01/18 03	14	17
01/01/18 04	13	16
01/01/18 05	12	15
01/01/18 06	11	14
01/01/18 07	*null*	*null*
01/01/18 08	*null*	*null*

Tabella 6.5: *label* relative ai dati di input da prevedere, in questo esempio due ore nel futuro.

La corrispondenza fra i valori di input in Tabella 6.4 e le *label* in Tabella 6.5 è determinata dal numero di riga; quando tali tabelle saranno convertite in vettori multidimensionali, la corrispondenza sarà determinata dalla posizione nel vettore (indice).

6.6 Generazione sequenze

In realtà, l'input della rete non deve essere un singolo record, ma una sequenza contigua di essi relativa ad un determinato intervallo temporale. Partendo quindi dalla tabella dei dati di input (Tabella 6.4) e della tabella con le *label* (Tabella 6.5), si andranno ad estrarre delle sequenze di record e, ad ognuna, si assocerà la rispettiva *label* da prevedere; essa corrisponderà alla *label* relativa all'ultimo record presente nella sequenza. Ad esempio, considerando la Tabella 6.4 e sequenze di lunghezza tre, è possibile estrarre da essa due sequenze, mostrate in Tabella 6.6 e in Tabella 6.8; le relative *label* sono mostrate, rispettivamente, in Tabella 6.7 e in Tabella 6.9.

Station	7000014		7000015	
Parameter	2	1000	2	1000
DateTime				
01/01/18 01	16	273	19	274
01/01/18 02	15	274	18	273
01/01/18 03	14	273	17	274

Tabella 6.6: prima sequenza estratta dalla Tabella 6.4.

Station	7000014	7000015
Parameter	2	2
DateTime		
01/01/18 05	12	15

Tabella 6.7: label associata alla prima sequenza, procurata dalla Tabella 6.5; corrisponde alla *label* associata all'ultimo record di Tabella 6.6.

Station	7000014		7000015	
Parameter	2	1000	2	1000
DateTime				
01/01/18 02	15	274	18	273
01/01/18 03	14	273	17	274
01/01/18 04	13	274	16	273

Tabella 6.8: seconda sequenza estratta dalla Tabella 6.4.

Station	7000014	7000015
Parameter	2	2
DateTime		
01/01/18 06	11	14

Tabella 6.9: label associata alla seconda sequenza, procurata dalla Tabella 6.5; corrisponde alla *label* associata all'ultimo record di Tabella 6.8.

Come si può notare le sequenze sono parzialmente sovrapposte, poiché alcuni record sono gli stessi; nonostante ciò sono comunque considerati due input diversi ed indipendenti. Nell'esempio sono state estratte sequenze di lunghezza tre, ma tale valore è regolabile a piacere, tenendo presente che potrebbe influire sulla qualità dei risultati. Un valore ragionevole, che sarà utilizzato nelle sperimentazioni è 72 (24 * 3); questo significa che saranno utilizzate le misurazioni degli ultimi tre giorni per prevedere l'inquinamento fra 24 ore. Da ora in avanti, una sequenza sarà considerata un singolo esempio da fornire alla rete, durante la fase di *training* o valutazione. Per ottenere prestazioni migliori, come già descritto nel Capitolo 3, tutti i dati saranno inoltre scalati fra 0 e 1, attraverso la classe *MinMaxScaler* messa a disposizione da Scikit-learn.

6.7 Suddivisione dei dati negli insiemi di training, validation e test

Prima di procedere con la fase di addestramento della rete, è necessario suddividere i dati di input nei set di *training*, *validation* e *test*. Si presentano due principali alternative per effettuare la suddivisione:

Suddivisione della tabella di input In questo caso si suddivide la tabella con i record, prendendo, ad esempio, il primo 80% dei record per il *training set*, il successivo 10% per il *validation set* ed il restante 10% per il *test set*. Successivamente, per ogni insieme, si estrarranno le rispettive sequenze da fornire in input alla rete. I record di input non possono essere rimescolati, poiché in tal caso si perderebbe la sequenzialità temporale dei dati.

Suddivisione delle sequenze generate In questo caso invece si estraggono prima tutte le possibili sequenze dai dati di input. Successivamente, tali sequenze, verranno spartite fra i tre insiemi. Questo approccio permette invece di rimescolare in modo casuale le sequenze, poiché la sequenzialità temporale è contenuta all'interno della singola sequenza.

La scelta di questi insiemi influisce notevolmente sui risultati[22, p. 16]; in particolare, ad esempio, nel caso sfortunato in cui il *test set* contenesse esempi di bassa qualità (esempi molto più facili o molto più difficili da prevedere rispetto a quelli del *training set*), si giungerebbe erroneamente alla conclusione che la rete non sia capace di generalizzare, oppure il contrario. Un problema di questo tipo potrebbe essere causato da una bassa qualità dei dati: le caratteristiche di questi potrebbero non essere sempre uniformi. Per evitare problemi di questo tipo si è deciso di adottare la seconda alternativa, cioè la suddivisione delle sequenze generate. Questo approccio permette di ottenere degli insiemi con caratteristiche più simili fra loro, perché permette di distribuire le sequenze in modo casuale. Nella prima alternativa si era invece costretti ad una suddivisione più rigida e dipendente dalla distribuzione della qualità dei dati di input. In realtà, nelle prove effettuate nel capitolo successivo, sono state distribuite casualmente solamente le sequenze del *training* e del *validation set*, poiché, altrimenti, non sarebbe stato possibile generare un grafico visualizzabile delle previsioni sul *test set*. Per la fase di addestramento è stato inoltre necessario definire il *batch size*, cioè quante sequenze esaminare per l'aggiornamento dei pesi. Anche questo è un valore regolabile che può influire sulla qualità dei risultati; un *batch size* sufficientemente grande, potrebbe infatti permettere di aggiornare i pesi della rete in modo migliore. Bisogna anche tenere in considerazione però che un *batch size* troppo grande, oltre a prolungare notevolmente i tempi di *training*, potrebbe saturare la memoria del calcolatore, rendendo, di fatto, impossibile la fase di addestramento. Nelle varie sperimentazioni è stato utilizzato un *batch size* pari a 128.

Capitolo 7

Prove sperimentali per la scelta del modello

Successivamente alla manipolazione della struttura dei dati (per renderli utilizzabili da una rete neurale) si potrà procedere con la definizione del modello che dovrà effettuare l'operazione di previsione. La struttura più adatta per tale modello non è però ben nota a priori, e, per questo, si procederà con una serie di prove sperimentali delle tipologie di rete definite nel Capitolo 3. Solitamente, si parte con un modello molto semplice, e si aggiunge complessità ad esso solo se realmente necessario. Non sarebbe infatti conveniente utilizzare un modello troppo complesso per un problema semplice, poiché, oltre ad aumentare la quantità di risorse necessarie per effettuare la fase di *training*, si rischierebbe di cadere nel problema dell'*overfitting*. Per confrontare i vari modelli si prenderà in considerazione l'errore MAE (*mean absolute error*) dimostrato sul *test set*. Per questo tipo di sperimentazioni MAE risulta più adatto di MSE (*mean square error*), perché genera valori facilmente confrontabili ed interpretabili. In realtà, se un particolare modello presenta un MAE inferiore, non significa sicuramente che esso sia migliore, poiché il *mean absolute error* calcolato dipende anche dalle caratteristiche del *test set*. In ogni caso, nelle varie sperimentazioni, si considererà migliore, per semplicità, il modello con MAE inferiore. Ogni sperimentazione sarà caratterizzata da una fase di *training* di 20 epoche; ogni epoca è composta da 100 fasi di *training* (*step*). Ogni fase di *training* aggiornerà i pesi analizzato un numero pari a *block size* di sequenze, che in questo caso è 128. Alla fine di ogni epoca si valuterà il modello sul *validation set*, mantenendo, di volta in volta, la configurazione che dimostra il *mean absolute error* inferiore su di esso. Alla fine delle 20 epoche, si valuterà il modello sul *test set*, calcolando il *mean absolute error* su di esso. Nella fase di *training*, inoltre, sarà utilizzato un ottimizzatore di tipo *Adam*; il *learning rate* sarà regolato automaticamente durante il *training* mediante la definizione di opportune *callback*. Nelle varie sperimentazioni, come primo approccio, si è adottata la seguente suddivisione dei vari insiemi: si sono prima estratti i record di input contigui da

assegnare al *test set*; dai restanti sono state generate tutte le possibili sequenze, suddividendole poi casualmente fra *training* e *validation set*. Questo approccio, come si vedrà nel capitolo successivo, porterà però a problemi, che influiranno sulla qualità dei risultati.

7.1 Funzione di previsione semplice

Il calcolo del *mean absolute error* sul *test set* permette di confrontare le prestazioni dei vari modelli. Serve però una strategia per determinare come si comporta il modello in generale; una tecnica spesso utilizzata è quella di confrontare il modello con una funzione di previsione molto semplice, che un uomo potrebbe pensare a "buon senso", senza utilizzare tecniche di *machine learning*[3]. Si andrà quindi ad analizzare come il modello si comporta rispetto a tale funzione: se il *mean absolute error* sul *test set* dimostrato dal nuovo modello è maggiore, significa che la semplice funzione a "buon senso" riesce a generare previsioni migliori, rendendolo del tutto inutile. La funzione di previsione semplice utilizzata in queste prove, è così definita:

$$concentrazione\_NO2\_fra\_24\_ore = concentrazione\_di\_NO2\_attuale$$

La funzione assume che la concentrazione di biossido di azoto fra 24 ore corrisponda alla concentrazione di biossido di azoto attuale. Questa funzione risulta alquanto banale, ma, in realtà, riesce comunque ad effettuare previsioni accettabili, poiché è probabile, che domani a questa stessa ora, vi sia un inquinamento analogo. Essa coglie una proprietà dei dati che all'uomo pare ovvia, ma una rete neurale potrebbe non individuarla immediatamente. L'errore di tale funzione sul *test set* è mostrato in Elenco 7.1.

Elenco 7.1: MAE della funzione di previsione semplice.

```
MAE loss (test-set): 0.05289441290882698
MAE loss rescaled (test-set): 8.145739587959355
```

Il valore mostrato in Elenco 7.1 sarà utilizzato come riferimento per la valutazione dei modelli definiti successivamente. Il valore *MAE loss* corrisponde al *mean absolute error* calcolato sul *test set* scalato fra 0 e 1, mentre il valore *MAE loss rescaled* corrisponde all'errore rispetto ai valori reali delle *label* (non scalate fra 0 e 1): questo significa che la previsione sbaglia, in media, di 8.15 $\mu g/m^3$.

7.2 Rete neurale lineare

Il modello di rete neurale più semplice è caratterizzato da soli tre neuroni, uno per ogni output e senza funzione di attivazione (lineare). Prima di tale livello è stato necessario aggiungere un *layer* di tipo *flatten*, che si occupa di concatenare fra loro i vari record della sequenza; non sarebbe infatti stato possibile fornire ad una rete di questo tipo una sequenza di record. In questo caso, ogni record della sequenza contiene 21 *feature*, e la sequenza è formata da 72 record. Il *layer flattern* concatena tali record, generando un input formato da $21 * 72 = 1512$ *feature*. Questo significa che ognuno dei tre neuroni avrà 1512 connessioni di input, ognuna con un peso indipendente. I parametri regolabili (pesi) sono quindi $1512 * 3 = 4536$ più il *bias* relativo ad ogni neurone ($1 * 3 = 3$), per un totale di 4539 parametri regolabili. La struttura del modello è mostrata in Elenco 7.2.

Elenco 7.2: struttura della rete neurale lineare messa alla prova.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 1512)              0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 4539
=================================================================
Total params: 4,539
Trainable params: 4,539
Non-trainable params: 0
_________________________________________________________________
```

Il risultato della fase di *training* è mostrato in Elenco 7.3.

Elenco 7.3: valutazione del modello (errore).

```
Last training epoch: 5ms/step - loss: 0.0658 - val_loss: 0.0683
Best MAE on validation: 0.06640

MAE loss (test-set): 0.05574148546584521
MAE loss rescaled (test-set): 8.584188761740162
```

Come si può notare, l'errore è maggiore rispetto alla funzione di riferimento; ciò significa che questo modello si comporta in modo peggiore di essa, probabilmente a causa della scarsa complessità (solo tre neuroni). A pagina 54 è mostrato il grafico che descrive la variazione del MAE durante la fase di *training* (Figura 7.1) ed un esempio di previsione sul *test set* (Figura 7.2). Come si può notare dalla Figura 7.1 non si presenta un problema di *overfitting* e le previsioni di Figura 7.2 non sempre sono corrette; ciò denota, probabilmente, una insufficiente complessità del modello.

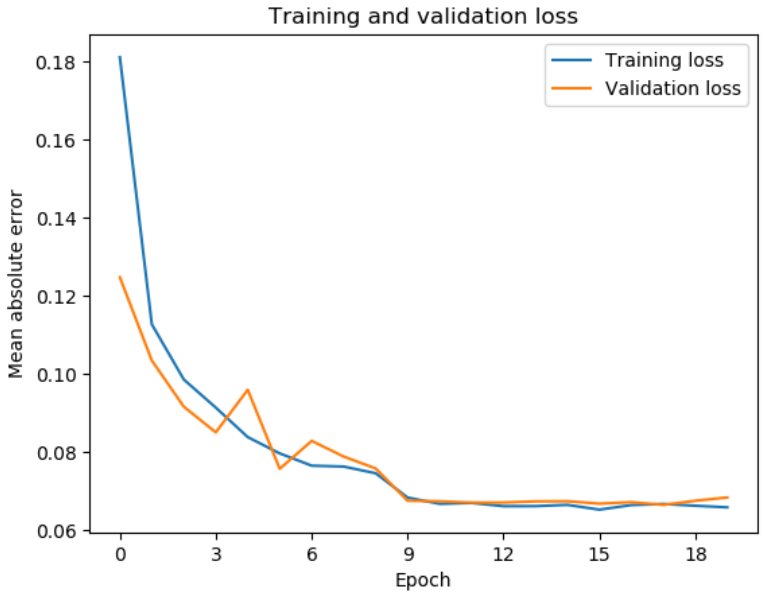

Figura 7.1: errore MAE durante le epoche di *training*.

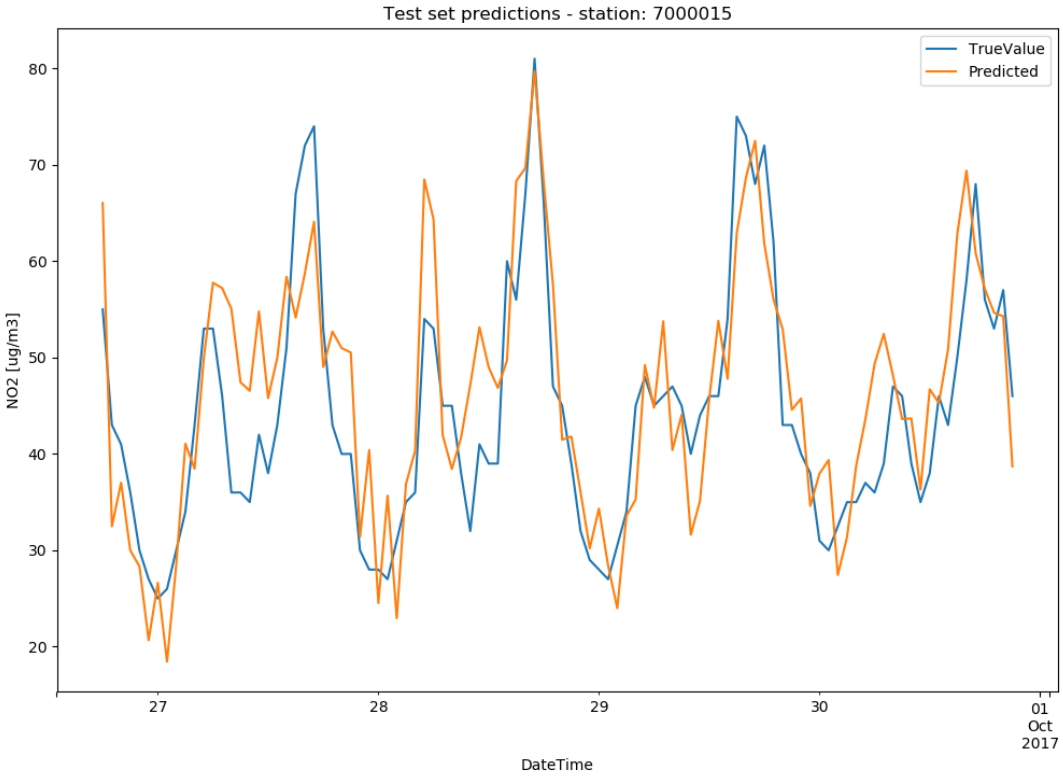

Figura 7.2: previsione su una parte del *test set* per una delle tre stazioni.

7.3 Rete neurale non lineare multilivello

Successivamente si è quindi definita una rete più complessa, composta da due *hidden layer*, ognuno con 32 neuroni, più i tre neuroni dell'*output layer*. Agli *hidden layer* è stata associata, come funzione di attivazione, una *ReLu*, mentre all'*output layer* una *sigmoid*. La funzione di attivazione dell'*output layer* produce valori valori compresi fra 0 e 1; questo aiuta la rete ad ottenere previsioni più accurate, poiché, sia le *feature* di input, sia le *label*, erano state scalate fra 0 e 1. Anche in questo caso è stato necessario un *layer* iniziale *flatten*, per le motivazioni già descritte. La struttura del modello è mostrata in Elenco 7.4.

Elenco 7.4: struttura della rete neurale non lineare multilivello messa alla prova.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 1512)              0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                48416
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 99
=================================================================
Total params: 49,571
Trainable params: 49,571
Non-trainable params: 0
_________________________________________________________________
```

Come si può notare dall' Elenco 7.4, questo modello è molto più complesso rispetto a quello mostrato nella sezione precedente: circa 10 volte di più (quasi 50000 parametri). Il risultato della fase di *training* è mostrato in Elenco 7.5.

Elenco 7.5: valutazione del modello (errore).

```
Last training epoch: 6ms/step - loss: 0.0583 - val_loss: 0.0587
Best MAE on validation: 0.05868

MAE loss (test-set): 0.047521420827528714
MAE loss rescaled (test-set): 7.3182988074394215
```

In questo caso, l'errore è minore rispetto sia alla funzione di riferimento che al modello lineare; questo significa che una rete di questo tipo può effettuare previsioni migliori. A pagina 56 è mostrato il grafico che descrive la variazione del MAE durante la fase di *training* (Figura 7.3) ed un esempio di previsione sul *test set* (Figura 7.4). Questo modello produce buone previsioni, ma è limitato dal fatto che necessita di un *layer* di *flatten* iniziale. Ciò limita la lunghezza massima della sequenza di input, poiché la complessità del modello aumenta proporzionalmente con essa.

Figura 7.3: errore MAE durante le epoche di *training*.

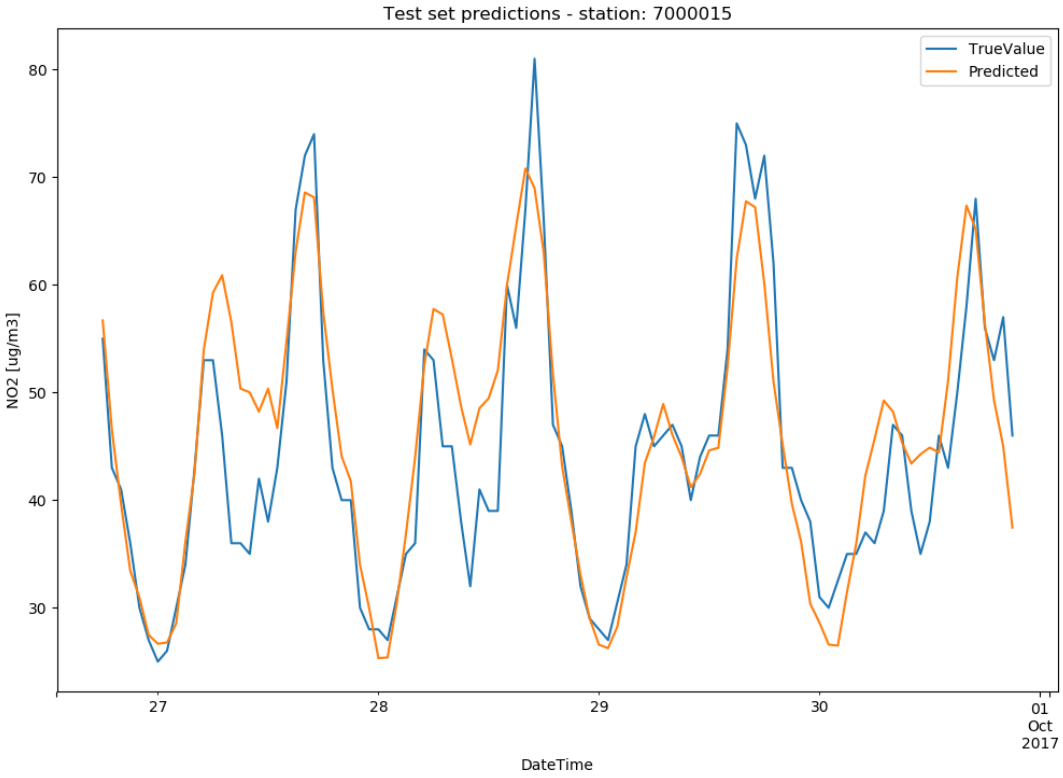

Figura 7.4: previsione su una parte del *test set* per una delle tre stazioni.

7.4 Rete neurale ricorrente

La tipologia di rete più adatta per questo problema dovrebbe essere una rete neurale ricorrente. Per questa prova è stato quindi definito un modello con un *hidden layer* composto da 32 celle GRU ed un normale *output layer* con tre neuroni per generare i tre valori di output. In questo caso il *layer flatten* non è necessario, poiché le celle GRU accettano come input sequenze di record. La struttura del modello è mostrata in Elenco 7.6.

Elenco 7.6: struttura della rete neurale ricorrente messa alla prova.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_1 (GRU)                  (None, 32)                5184
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 99
=================================================================
Total params: 5,283
Trainable params: 5,283
Non-trainable params: 0
_________________________________________________________________
```

Come si può notare dall' Elenco 7.6, questo modello ha una complessità pari al primo esaminato (circa 5000 parametri). Il risultato della fase di *training* è mostrato in Elenco 7.7.

Elenco 7.7: valutazione del modello (errore).

```
Last training epoch: 85ms/step - loss: 0.0601 - val_loss: 0.0596
Best MAE on validation: 0.05939

MAE loss (test-set): 0.05077240879187175
MAE loss rescaled (test-set): 7.8189509539482485
```

In questo caso, l'errore è maggiore rispetto alla rete multilivello esaminata precedentemente, ma è comunque minore della funzione di riferimento e della rete neurale lineare. Inoltre, nonostante avesse lo stesso numero di parametri della prima rete neurale lineare messa alla prova, ha ottenuto risultati migliori. Considerando poi, che un modello di questo tipo potrebbe gestire senza problemi sequenze di lunghezza arbitraria, è facile affermare la sua superiorità rispetto ai modelli visti precedentemente, almeno per questo tipo di problema. Bisogna però notare che i tempi necessari alla fase di *training* sono stati maggiori: 85 millisecondi per *step*, contro i 5 millisecondi per *step* del modello iniziale. A pagina 58 è mostrato il grafico che descrive la variazione del MAE durante la fase di *training* (Figura 7.5) ed un esempio di previsione sul *test set* (Figura 7.6). Anche in questo caso, sembrano non esserci problemi di *overfitting*; si potrebbe quindi pensare di aumentare ulteriormente la complessità del modello.

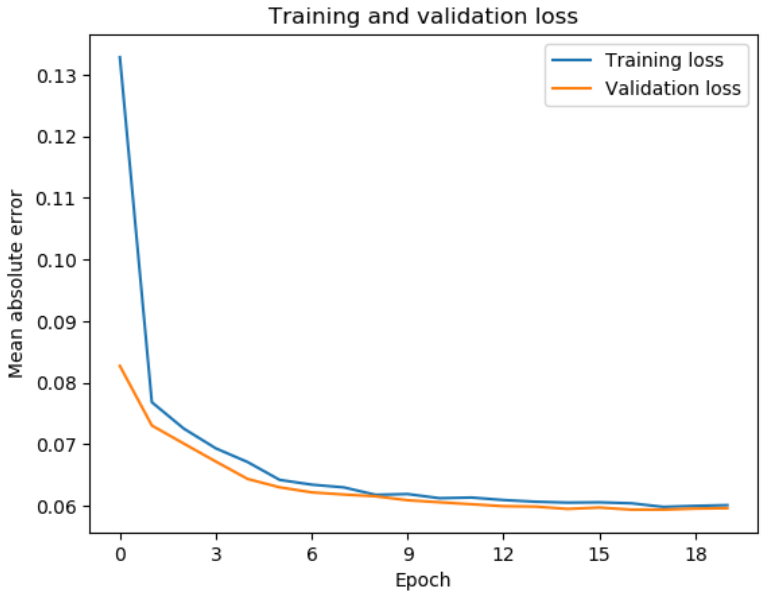

Figura 7.5: errore MAE durante le epoche di *training*.

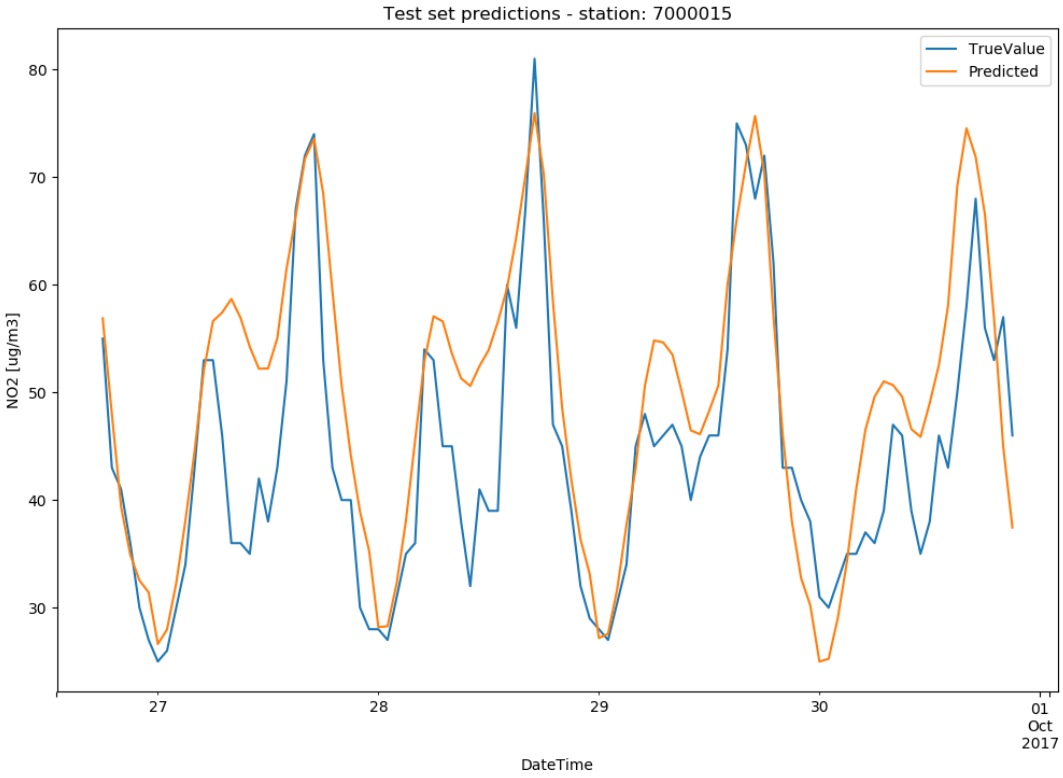

Figura 7.6: previsione su una parte del *test set* per una delle tre stazioni.

7.5 Rete neurale ricorrente multilivello

Fino a quando non si riscontrano problemi di *overfitting*, è ragionevole aumentare la complessità del modello. Sovrapponendo più livelli GRU, è possibile ottenere un modello ricorrente più complesso. In questa prova, è stata costruita una rete formata da un primo *hidden layer* con 64 celle GRU ed un secondo con 32, oltre che al normale *output layer* con tre neuroni. Come per il caso precedente, non è necessario alcun *layer flatten*. La struttura del modello è mostrata in Elenco 7.8.

Elenco 7.8: struttura della rete neurale ricorrente multilivello messa alla prova.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_1 (GRU)                  (None, 72, 64)            16512
_________________________________________________________________
gru_2 (GRU)                  (None, 32)                9312
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 99
=================================================================
Total params: 25,923
Trainable params: 25,923
Non-trainable params: 0
_________________________________________________________________
```

Come si può notare dall'Elenco 7.8, questo modello ha una complessità pari alla metà della rete neurale multilivello vista in precedenza. Il risultato della fase di *training* è mostrato in Elenco 7.9.

Elenco 7.9: valutazione del modello (errore).

```
Last training epoch: 165ms/step - loss: 0.0552 - val_loss: 0.0551
Best MAE on validation: 0.05487

MAE loss (test-set): 0.043882974702216 63
MAE loss rescaled (test-set): 6.757978104141361
```

In questo caso, l'errore dimostrato dal modello sul *test set* è il minore fra quelli ottenuti finora. Bisogna comunque notare, che l'incremento della complessità ha allungato ulteriormente i tempi di *training*, raddoppiandoli rispetto alla prova precedente: da 85 millisecondi per *step* si è passati a 165 millisecondi per *step*. A pagina 60 è mostrato il grafico che descrive la variazione del MAE durante la fase di *training* (Figura 7.7) ed un esempio di previsione sul *test set* (Figura 7.8). Anche in questo caso sembrano non esserci problemi di *overfitting*; si sarebbe quindi potuto continuare ad aumentare la complessità del modello. Per evitare però di dover gestire le varie problematiche relative ad un modello troppo pesante (come ad esempio gli estesi tempi di *training*), si è deciso di considerare questa configurazione come miglior compromesso fra semplicità del modello e qualità delle previsioni.

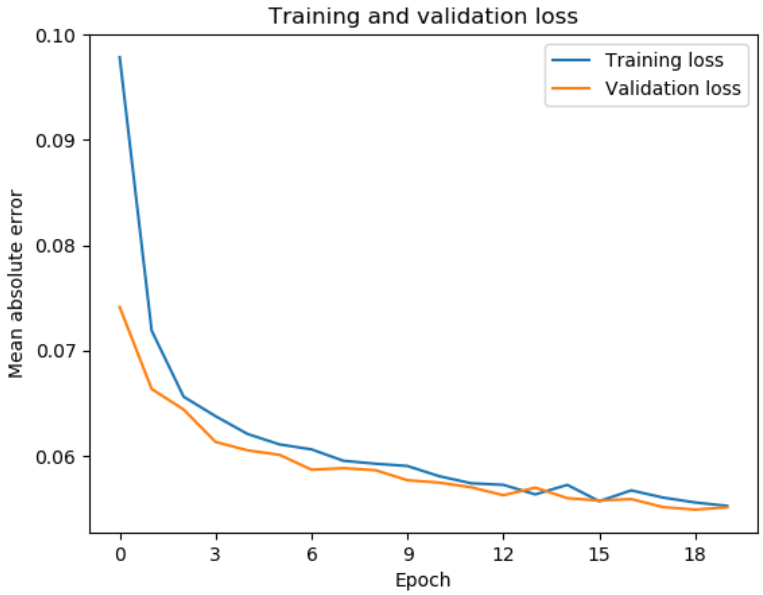

Figura 7.7: errore MAE durante le epoche di *training*.

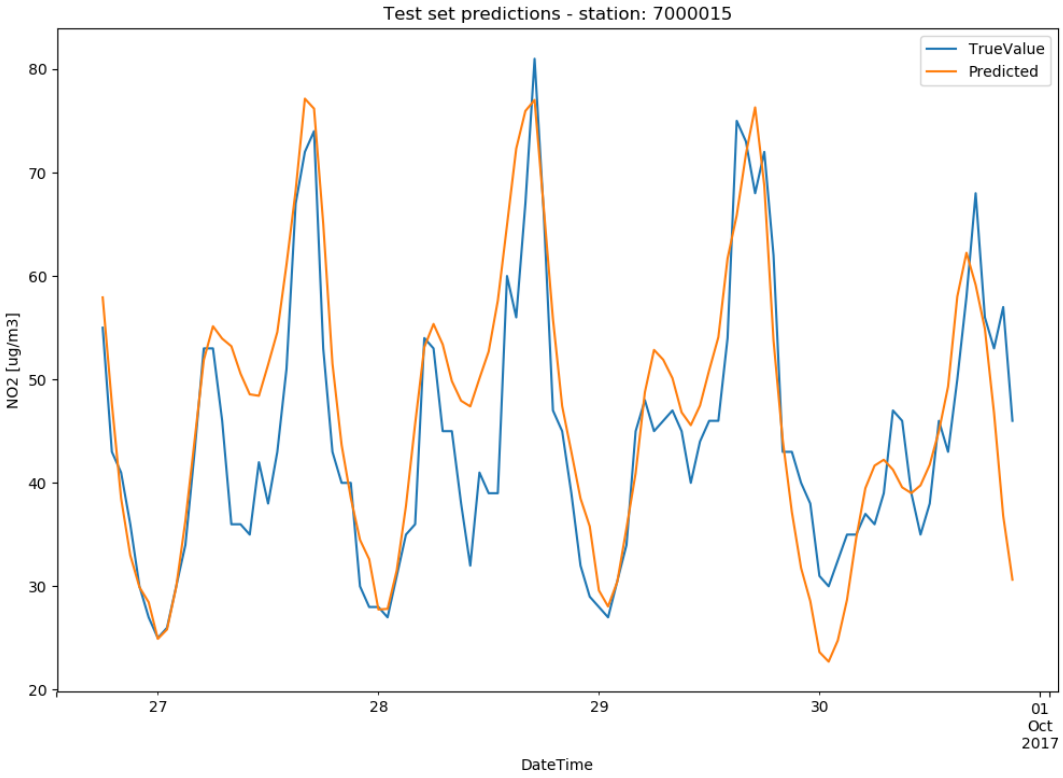

Figura 7.8: previsione su una parte del *test set* per una delle tre stazioni.

Capitolo 8

Prove sperimentali sul modello scelto

Utilizzando il modello individuato nel capitolo precedente (rete ricorrente multilivello), saranno modificate alcune condizioni, per esaminare le ripercussioni sui risultati. Saranno effettuate alcune prove suddividendo in modo diverso gli insiemi di *training*, *validation* e *test*, oppure variando alcune caratteristiche delle *feature* di input. Infine, si determinerà se una rete neurale di questo tipo possa essere utilizzata per effettuare previsioni su un numero più elevato di stazioni.

8.1 Influenza della suddivisione dei dati sui risultati

La metodologia utilizzata per suddividere gli insiemi di *training*, *validation* e *test*, ha fatto sorgere qualche problematica. La suddivisione adottata nelle sperimentazioni precedenti, consisteva nel selezionare preventivamente i record contigui da assegnare al *test set* (estraendone poi le rispettive sequenze); successivamente, dai rimanenti, venivano estratte tutte le sequenze, assegnandole casualmente fra il *training set* ed il *validation set*. Questo approccio, permetteva di ottenere un *training* ed un *validation set* con caratteristiche uniformi; non si può però affermare lo stesso per il *test set*, poiché le sue sequenze non venivano selezionate casualmente, ma estratte da record contigui dai dati di input. Effettuando varie sperimentazioni, ci si è resi conto che le caratteristiche del *validation set* e del *test set* così costruiti, erano differenti fra loro. Ciò comportava alcune problematiche durante l'addestramento, poiché veniva selezionato il modello che dimostrava l'errore minore sul *vadation set*; quando esso veniva però messo alla prova sul *test set*, si ottenevano risultati deludenti, poiché esso aveva caratteristiche diverse dall'insieme di validazione. Si è quindi provato a definire un *validation set* con caratteristiche simili al *test set*, effettuando la suddivisione preventiva dei record di input fra gli insiemi. Il modello individuato nel capitolo precedente (Elenco 7.8) è stato quindi addestrato sui nuovi insiemi, analizzando i risultati. Si vuole far notare che il *test set* è identico a quello utilizzato

precedentemente; sono cambiati solamente gli insiemi di *training* e *validation*. Il risultato della fase di *training* è mostrato in Elenco 8.1.

Elenco 8.1: valutazione del modello (errore).

```
Last training epoch: 162ms/step - loss: 0.0577 - val_loss: 0.0550
Best MAE on validation: 0.05313

MAE loss (test-set): 0.046545147043577206
MAE loss rescaled (test-set): 7.16795264471089
```

Come si può notare, si ottiene un errore sul *test set* leggermente maggiore; le anomalie più rilevanti sono però osservabili in Figura 8.1, che mostra la variazione dell'errore sugli insiemi di *training* e *validation* durante l'addestramento.

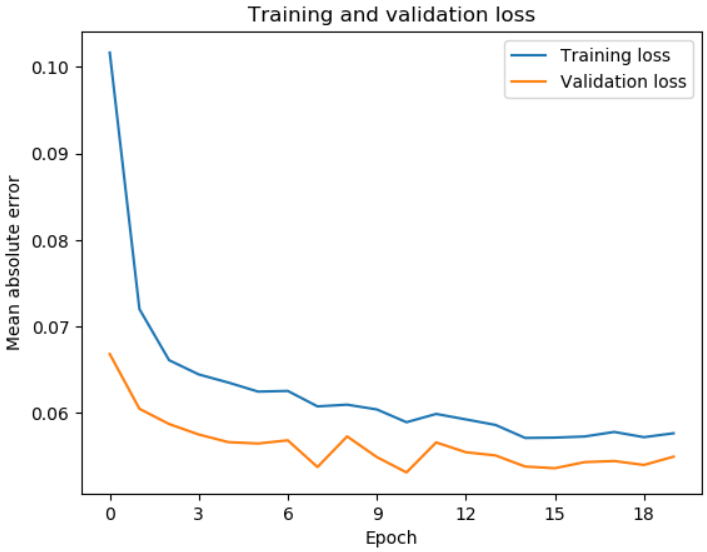

Figura 8.1: errore MAE durante le epoche di *training*.

Come si può notare da Figura 8.1, l'errore sul *validation set* rimane sempre minore di quello sul *training set*; solitamente ciò non dovrebbe accadere, poiché gli esempi contenuti nel *validation set* non sono mai stati esaminati dal modello durante il *training*. Da questo si può concludere che, probabilmente, i dati di input presentano delle anomalie, poiché gli esempi del *validation set* risultano più facili da prevedere rispetto a quelli dell'insieme di *training*. L'insieme di *training* contiene infatti le sequenze relative agli anni 2013-2016, mentre il *validation* ed il *test* contengono sequenze relative al 2017; forse, fra quegli anni, sono cambiate alcune caratteristiche dei dati, rendendoli poco omogenei. Questo è un punto debole di queste tecniche di *machine learning*, poiché i risultati dipendono severamente dalla qualità dei dati di input, e, quindi, anche dalla scelta di suddivisione dei dati nei tre insiemi di *training*, *validation* e *test*.

8.2 One hot encoding e bucketing

Un'altra prova effettuata è consistita nella modifica di alcune *feature* di input, applicando tecniche di *one hot encoding* e *bucketing*. Solitamente, come già descritto nel Capitolo 3, questo tipo di codifica permette alla rete di adattarsi meglio ai dati che non presentano relazioni di linearità con l'output della rete. Si è deciso di applicare una tecnica di *bucketing* alla direzione del vento, e, successivamente, una tecnica di *one hot encoding* sulla direzione del vento così modificata e sulle informazioni temporali. La direzione del vento originale era un numero intero, corrispondente ai gradi, compreso fra 0 e 360. Una informazione così precisa non è però molto utile alla rete, poiché essa serve solo per determinare approssimativamente verso quale direzione la massa d'aria inquinata o pulita si sposterà; sarebbe quindi meglio non considerare la direzione come un valore continuo, ma come un valore discreto (categorico). Per fare ciò, sono stati individuati otto intervalli, ognuno corrispondente ad una determinata direzione discreta; il nuovo valore corrispondente alla direzione sarà quindi un numero intero fra 0 e 7. Successivamente, alle *feature* corrispondenti alla nuova direzione del vento, al mese, al giorno della settimana ed all'ora, è stata applicata una tecnica di *one hot encoding*, utilizzando la funzione *get_dummies()*, messa a disposizione dalla libreria Pandas. Essa permette di estrarre i vettori relativi al *one hot encoding* per una particolare serie (colonna del *DataFrame*) fornita come input. Sono state quindi aggiunte, sostituendole alle *feature* originali, 12 *feature* relative al mese, 7 *feature* relative al giorno della settimana, 24 *feature* relative all'ora e 8 *feature* relative alla direzione del vento per ogni stazione. La suddivisione dei vari *set* adottata è la stessa utilizzata per le prove del Capitolo 7, cioè una distribuzione casuale delle sequenze fra *training* e *validation set*. Anche il modello utilizzato è stato quello individuato nel Capitolo 7 (rete ricorrente multilivello). Il risultato della fase di training è mostrato in Elenco 8.2.

Elenco 8.2: valutazione del modello (errore).

```
Last training epoch: 169ms/step - loss: 0.0490 - val_loss: 0.0502
Best MAE on validation: 0.05024

MAE loss (test-set): 0.05309275713016705
MAE loss rescaled (test-set): 8.176284598045726
```

I risultati ottenuti non si discostano molto da quelli attesi: il modello riesce ad adattarsi meglio ai dati di *training* e *validation*; infatti l'errore su tali insiemi è minore rispetto a quelli dimostrati nella prova del Capitolo 7 (0.0490 contro 0.0552 per il *training set* e 0.05024 contro 0.05487 per il *validation set*). Nonostante ciò, però, le prestazioni sul *test set* risultano peggiori, anche della funzione di riferimento, probabilmente per le problematiche descritte nella sezione precedente. Disponendo di dati di input più omogenei, queste tecniche avrebbero forse permesso di ottenere risultati migliori. I relativi grafici (Figura 8.2 e Figura 8.3) sono mostrati a pagina 64.

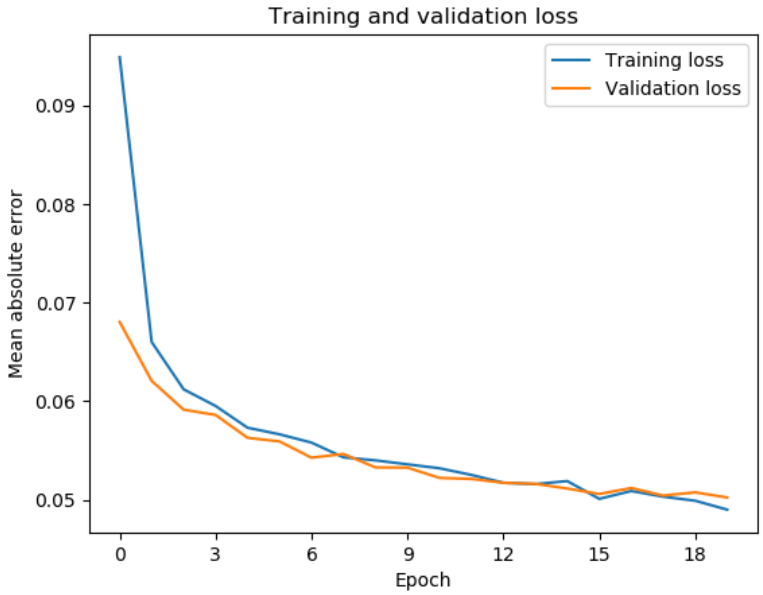

Figura 8.2: errore MAE durante le epoche di *training*.

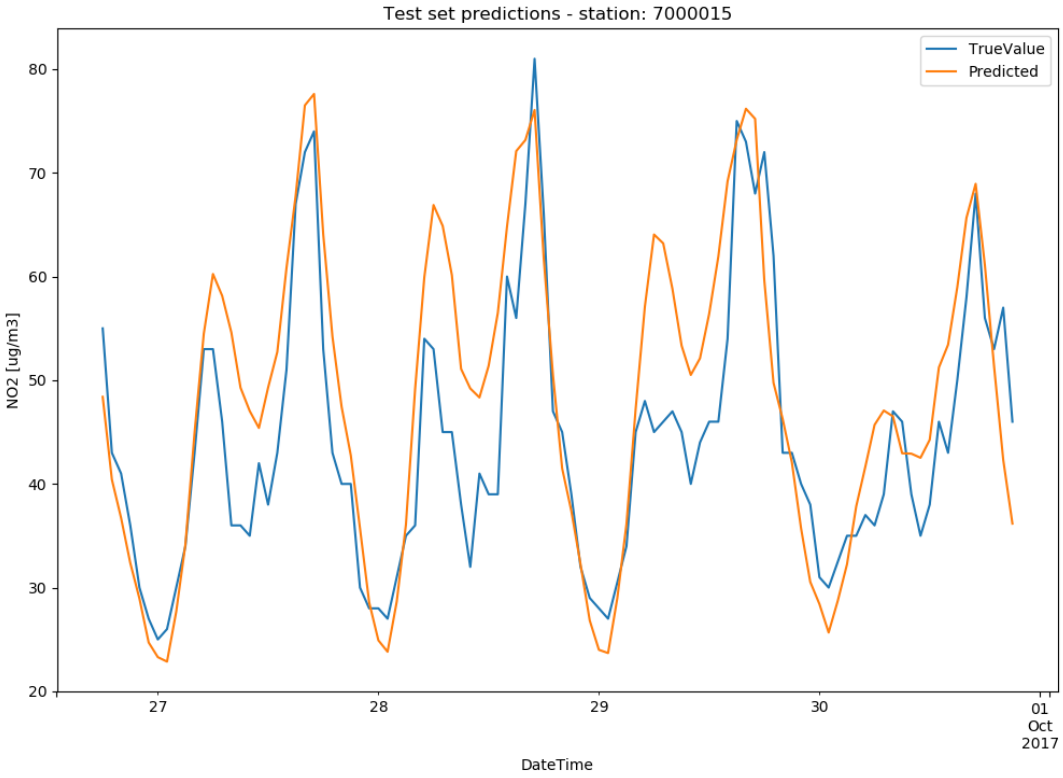

Figura 8.3: previsione su una parte del *test set* per una delle tre stazioni.

8.3 Scalabilità

Tutte le prove effettuate fino ad ora, in linea con gli obiettivi preposti, sono state effettuate considerando un caso semplice: prevedere i valori di inquinamento relativi alle sole tre stazioni di monitoraggio presenti a Bologna. Sarebbe però interessante esaminare un caso più complesso, con un numero più elevato di stazioni, per concludere se un approccio di questo tipo possa essere adottato efficacemente anche per problemi su larga scala. I dati di ARPA Emilia-Romagna contengono le misurazioni, relative al biossido di azoto, di 42 stazioni, distribuite sul territorio regionale. Si proverà quindi ad effettuare una previsione di 24 ore nel futuro per ognuna delle 42 stazioni, utilizzando, come primo approccio, il modello definito nel Capitolo 7. Saranno utilizzate le stesse tecniche e la stessa struttura dei dati definite nei capitoli precedenti, con l'unica differenza di considerare 42 stazioni invece di 3. Si è riscontrato però un problema durante la fase di assegnamento dei record meteorologici alle varie stazioni: con 42 stazioni, la procedura in Python definita originariamente per convertire i record presenti sul *database* in record utilizzabili dalla rete neurale è risultata troppo lenta. Infatti, considerando l'intera Emilia-Romagna, aumenta notevolmente anche il numero di stazioni meteorologiche da considerare, incrementando considerevolmente il numero di operazioni da effettuare per la conversione. È stato quindi necessario effettuare tale procedura attraverso un applicativo in C#, che generasse un file CSV con i record già strutturati adeguatamente per essere forniti alla rete neurale. Dall'applicativo in Python ci si è quindi limitati ad importare tale file, generando il *DataFrame* associato. I record di input, rispetto alle prove precedenti, risultano, molto più voluminosi: con 6 *feature* per ogni stazione, più 3 *feature* relative alle informazioni temporali, si ottiene un totale di 255 *feature* di input. La stessa considerazione va fatta anche per le *label* di output: in questo caso una *label* sarà composta da 42 valori distinti, uno per ogni stazione. Rispetto al modello originale è stato modificato solamente l'*output layer*, che ora sarà composto da 42 neuroni. La struttura del modello è mostrata in Elenco 8.3.

Elenco 8.3: struttura della rete neurale utilizzata per la prova di scalabilità.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_1 (GRU)                  (None, 72, 64)            61440
_________________________________________________________________
gru_2 (GRU)                  (None, 32)                9312
_________________________________________________________________
dense_1 (Dense)              (None, 42)                1386
=================================================================
Total params: 72,138
Trainable params: 72,138
Non-trainable params: 0
```

Il modello, come si può notare da Elenco 8.3, nonostante la struttura sia praticamente identica all'originale, contiene quasi il triplo dei parametri addestrabili; ciò dipende dal fatto che ogni cella del *layer* di input deve avere una connessione con ogni *feature* di input. Aumentando quindi il numero di *feature* di input, è aumentata, di conseguenza, anche la complessità del modello. Per ottenere un valore di riferimento, è stata applicata ai dati di input la funzione di previsione semplice definita nel Capitolo 7, calcolando quindi il MAE relativo ad essa. Il risultato è mostrato in Elenco 8.4.

Elenco 8.4: MAE della funzione di previsione semplice sui dati di test relativi alla prova di scalabilità.

```
MAE loss (test-set): 0.06469441194254698
MAE loss rescaled (test-set ): 9.704161791382045
```

Il risultato della fase di *training* è mostrato in Elenco 8.5.

Elenco 8.5: valutazione del modello (errore).

```
Last training epoch: 180ms/step - loss: 0.0561 - val_loss: 0.0558
Best MAE on validation: 0.05578

MAE loss (test-set): 0.0583130720009464
MAE loss rescaled (test-set): 8.74696080014196
```

Come si può notare dall' Elenco 8.5, il modello riesce ad ottenere risultati migliori della funzione di previsione semplice (0.0583 contro 0.0647). A pagina 67 è mostrato il grafico che descrive la variazione del MAE durante la fase di *training* (Figura 8.4) ed un esempio di previsione sul *test set* (Figura 8.5). Per permettere un confronto con i modelli definiti precedentemente, è stato riportato il grafico (Figura 8.5) con le previsioni relative alla solita stazione. Come si può facilmente notare da tale grafico, le previsioni sono alquanto peggiori rispetto alle prove precedenti; ciò è però un risultato atteso e prevedibile, poiché lo stesso modello deve ora prevedere 42 valori invece di 3. Probabilmente, quindi, aumentando la complessità del modello, si sarebbero potuti ottenere risultati migliori, ma si è deciso di non proseguire, poiché ciò non era parte degli obiettivi preposti. Questa prova è stata effettuata solamente per stabilire se un approccio caratterizzato dalla struttura dei dati e dalle reti definite nei capitoli precedenti, fosse applicabile su scala più larga. Ne è risultato che ciò è possibile, ma bisogna tenere in considerazione che per ottenere una qualità dei risultati adeguata sarebbe necessario incrementare la complessità del modello in relazione al numero di stazioni considerate. Se le stazioni da considerare fossero in numero molto elevato, si potrebbe quindi essere costretti a dover definire un modello estremamente complesso, difficile da gestire; in tal caso si potrebbe preferire un approccio diverso, definendo, ad esempio, vari modelli più semplici, ed assegnando ad ognuno un qualche sottoinsieme delle stazioni da considerare.

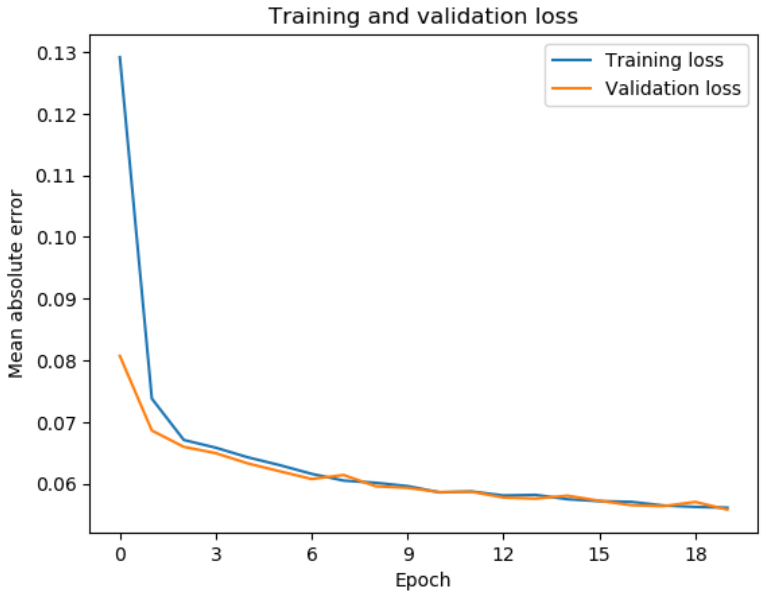

Figura 8.4: errore MAE durante le epoche di training.

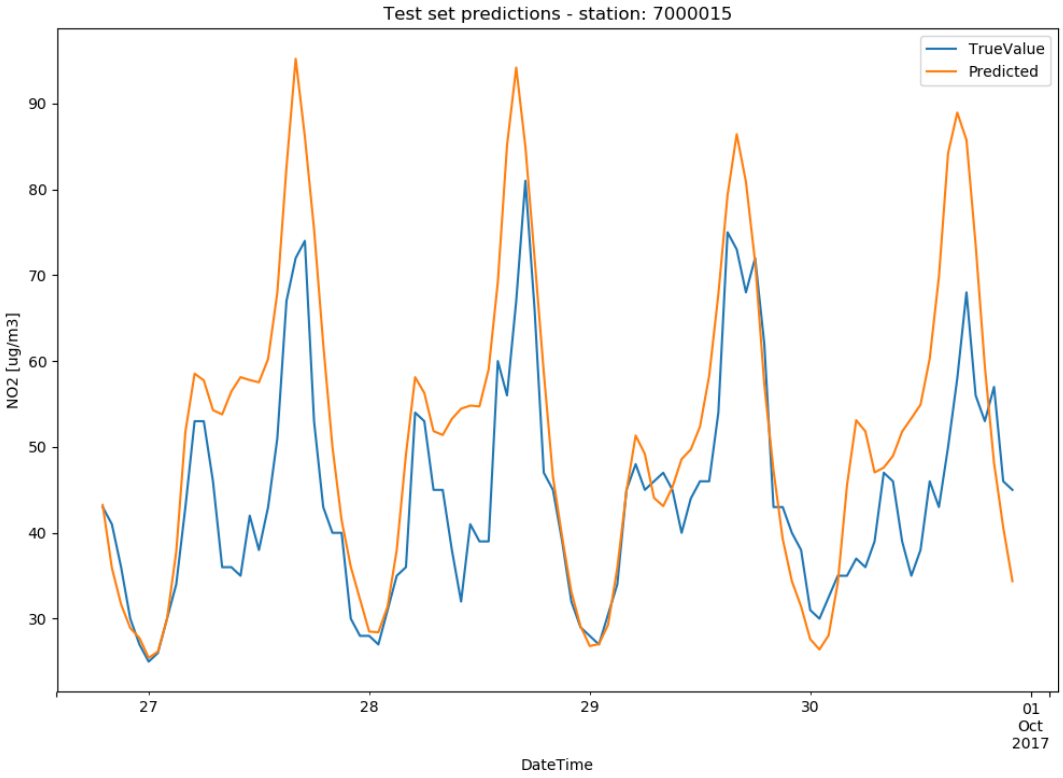

Figura 8.5: previsione su una parte del test set per una delle 42 stazioni.

8.4 Possibili prove ulteriori

Sicuramente, il modello individuato nel Capitolo 7 non è il migliore fra tutti quelli possibili. Sono state infatti definite alcune configurazioni arbitrarie, e, tramite le relative sperimentazioni, è stata determinata quella capace di ottenere i risultati migliori. Il processo di ricerca di tale configurazione non è però stato condotto in maniera eccessivamente approfondita e, per questo, si potrebbe proseguire ulteriormente con altre sperimentazioni. Le possibili configurazioni ed i parametri regolabili sono però molto numerosi, rendendo, tale operazione di ricerca, potenzialmente difficoltosa ed onerosa in termini di tempo. Ad esempio, si potrebbe variare il numero di celle per ogni livello, oppure si potrebbero aggiungere o rimuovere determinati *layer*. La cella ricorrente utilizzata nelle varie prove dei capitoli precedenti è stata esclusivamente una GRU, ma si potrebbero fare altre sperimentazioni utilizzando celle ricorrenti semplici o LSTM; si potrebbero inoltre aggiungere, all'interno della rete ricorrente, dei *layer* non ricorrenti, composti da normali neuroni con una qualche funzione di attivazione. I *layer* GRU utilizzati hanno, come funzione di attivazione predefinita, una *hyperbolic tangent*; da alcune prove effettuate sembrerebbe che questa funzione di attivazione offra i risultati migliori, ma si potrebbe tentare con altre, ad esempio *linear*, *ReLu* o *sigmoid*, utilizzando anche diverse di queste fra *layer* diversi. Le varie celle ricorrenti, inoltre, hanno un parametro denominato *return_sequences*, che indica se il *layer* debba fornire un output per ogni record della sequenza o solamente per l'ultimo. Nell'ultima prova del capitolo precedente, tale parametro era impostato a vero solamente per il *layer* ricorrente più vicino all'input, mentre per l'altro era impostato a falso; in questo modo la rete generava, per ogni sequenza, un singolo output composto da tre valori, confrontabile con la rispettiva *label*. Si potrebbe quindi pensare di definire una rete che fornisca un output per ogni record della sequenza; in tal caso sarebbe necessario modificare la procedura per la definizione delle *label*, poiché, ad ogni sequenza, sarebbe da assegnare un numero di *label* pari alla sua lunghezza. Un approccio di questo tipo potrebbe generare risultati con caratteristiche differenti[19]. Un'altra alternativa potrebbe essere quella di utilizzare reti ricorrenti bidirezionali, o reti ricorrenti *stateful*. Vi sono poi altri parametri, che potrebbero influire sulla qualità delle previsioni. Ad esempio, si potrebbero effettuare delle prove variando la lunghezza delle sequenze, variando il *batch size*, o utilizzando un ottimizzatore diverso, regolando i relativi parametri come il *learning rate* o scegliendo un differente metodo di calcolo dell'errore (MAE o MSE). Nelle varie prove è stato utilizzato un ottimizzatore di tipo *Adam*, che sembrerebbe offrire buone prestazioni, ma ne esistono altri, ognuno con le proprie peculiarità: fra questi *RMSprop* (spesso adatto per reti ricorrenti), SGD o *Adagrad*. Un altro parametro molto importante è il tempo di *training*: solitamente modelli più complessi, o soggetti a regolarizzazione, necessitano di un tempo di *training* più lungo per convergere. Per questo, in alcuni casi, sarebbe opportuno

effettuare alcune prove allungando i tempi di *training*, fino a quando non inizia a verificarsi il fenomeno dell'*overfitting*. In caso di *overfitting*, si potrebbe tentare di applicare un qualche tipo di regolarizzazione, ad esempio un *dropout*, definendo le relative percentuali di regolarizzazione, anch'esse determinabili attraverso prove sperimentali. Si potrebbero anche effettuare ulteriori manipolazioni dei dati di input, ad esempio scegliendo dati meteorologici diversi, generando gli intervalli mancanti attraverso altre reti neurali definite per tale scopo, o aggiungendo particolari *feature* artificiali. Si potrebbe inoltre esaminare il comportamento di un modello simile su un altro inquinante atmosferico. Gli effetti di tali variazioni, applicabili ai parametri o alla struttura della rete, non sono stati determinati esaurientemente, anche perché ciò non era compreso fra gli obiettivi prefissati. In ogni caso, sono stati illustrati i principali approcci che potrebbero condurre a risultati migliori, utili per eventuali sviluppi successivi.

Conclusioni

In questa trattazione, sono state inizialmente affrontate le principali problematiche ed i rischi inerenti all'inquinamento atmosferico. Tali rischi, che potrebbero compromettere seriamente la salute dei cittadini, giustificano la presenza sul territorio di una serie di stazioni ufficiali, che monitorano continuamente la concentrazione dei principali inquinanti. Il monitoraggio continuo permette di adottare tempestivamente contromisure in caso di superamento dei limiti di legge previsti per ogni inquinante. In Emilia-Romagna, le misurazioni effettuate da tali stazioni, sia attuali che storiche, sono disponibili pubblicamente sul portale di ARPA Emilia-Romagna. Data la gravità del problema, sarebbe però molto utile avere a disposizione, oltre alle misurazioni attuali, anche delle previsioni, determinate attraverso l'analisi dei dati disponibili. In vista di ottenere previsioni di qualità, sono stati raccolti anche dati storici relativi alle condizioni meteorologiche, poiché la concentrazione degli inquinanti atmosferici dipende, molto probabilmente, anche da queste. Sono state, quindi, descritte le modalità adottate per recuperare tali misurazioni ed è stata definita la struttura della base dati utilizzata per memorizzarle e per fornire un'interfaccia di accesso unificata ai dati raccolti. Partendo da tali dati, è stato delineato un percorso in grado di portare, mediante l'utilizzo di tecniche e tecnologie specifiche, alla definizione di un modello capace di effettuare previsioni sulla qualità dell'aria. Sono state quindi descritte, a livello teorico, le principali tecniche di apprendimento automatico: le reti neurali. Tali tecniche si basano sulla definizione di un modello, composto da una serie di livelli di neuroni interconnessi tra loro, capace di apprendere, attraverso la somministrazione di un insieme di esempi, le relazioni fra gli input e gli output attesi (nel caso di *supervised learning*). Tale processo si basa su un'operazione di minimizzazione dell'errore fra i valori predetti ed i valori reali, attraverso il calcolo del gradiente. Inoltre, poiché la rete deve apprendere da un insieme di esempi significativi, è necessaria anche un'operazione preliminare di selezione e manipolazione dei dati. Le reti neurali possono avere caratteristiche e strutture differenti. Fra queste vi sono le cosiddette reti neurali ricorrenti, che, solitamente, utilizzano celle GRU o LSTM; esse sono utilizzate quando è necessario operare con dati sequenziali, come in questo caso. Successivamente sono state individuate le principali tecnologie disponibili per definire ed utilizzare, in modo semplice e veloce, tali reti neurali. Si è scelto di utilizzare, per

questo progetto, esclusivamente TensorFlow e Keras, poiché risultano essere quelli più popolari e con più documentazione disponibile. Analizzando i dati, ci si è resi conto che il numero di stazioni di monitoraggio della qualità dell'aria non è molto elevato, mentre le stazioni meteorologiche sono presenti in numero maggiore. Le misurazioni sono presenti con frequenza oraria e l'inquinante con la maggior mole di dati risulta essere il biossido di azoto. Per non complicare eccessivamente il problema, almeno inizialmente, si è deciso di operare su un'area ristretta, corrispondente alla zona centrale di Bologna, che contiene tre stazioni di monitoraggio. Tenendo presenti tali considerazioni e le nozioni apprese relative alle tecniche e tecnologie disponibili, ci si è posti come obiettivo finale quello di ottenere un modello, sotto forma di rete neurale, capace di prevedere il livello di concentrazione del biossido di azoto 24 ore nel futuro, per ognuna delle tre stazioni di monitoraggio presenti a Bologna. Una porzione di considerevole importanza per il raggiungimento di tale obiettivo è consistita nella manipolazione dei dati raccolti, per renderli fruibili da parte di una rete neurale. È stato infatti necessario definire una nuova struttura dei dati, in modo che ogni record, relativo ad una determinata ora, contenesse, per ogni stazione, le informazioni relative alla concentrazione di biossido di azoto, temperatura, umidità, precipitazioni, velocità del vento e direzione del vento; tali informazioni infatti, potrebbero determinare la velocità di dispersione degli inquinanti atmosferici, permettendo di ottenere previsioni migliori. Per l'associazione delle informazioni meteorologiche, si è scelto di considerare la stazione meteorologica più vicina alla stazione di monitoraggio della qualità dell'aria presa in esame; sono stati inoltre completati gli eventuali dati mancanti attraverso un'interpolazione lineare sul tempo. Da tali dati sono state quindi estratte delle sottosequenze e, ad ognuna, è stata assegnata la *label* da prevedere. Manipolando i dati in questo modo, è stato possibile fornirli ad una rete neurale. Effettuando varie sperimentazioni, si è giunti alla conclusione che, il modello più adatto a questo tipo di problema consista, come prevedibile, in una rete neurale ricorrente con più livelli. Si è dimostrato che tale rete genera buone previsioni, poiché ottiene risultati notevolmente migliori rispetto ad una funzione di previsione di riferimento, definita a "buon senso", che supponeva la concentrazione di biossido di azoto a distanza di 24 ore identica a quella attuale. Questo giustifica quindi l'adozione di queste tecniche di *machine learning*, poiché riescono, effettivamente, ad ottenere risultati apprezzabili. Con la definizione del modello finale e con l'analisi delle relative previsioni, è stato quindi pienamente raggiunto l'obiettivo preposto. Malgrado ciò, durante le sperimentazioni, sono sorte alcune problematiche, che hanno messo in luce il principale punto debole di queste tecniche. Infatti, il modello viene addestrato sul *training set*, viene validato alla fine di ogni epoca sul *validation set*, per comprendere se il modello sia effettivamente migliorato rispetto all'epoca precedente, ed infine verificato sul *test set*, per valutarne le prestazioni finali. Questo significa che il risultato finale della fase di addestramento ed il verdetto sulla valutazione delle prestazioni, dipendono, quasi esclusivamente,

dalle caratteristiche dei tre insiemi. Se i dati hanno qualche tipo di anomalia, come probabilmente in questo caso, e se i tre insiemi non hanno, di conseguenza, caratteristiche omogenee, si potrebbero ottenere risultati discutibili. In un problema in cui non sono coinvolte sequenze, ciò si potrebbe semplicemente risolvere distribuendo casualmente i record fra i tre insiemi. In questo particolare caso però, in cui è necessario trattare delle sequenze di dati, tale problematica è più difficile da risolvere, poiché non si può effettuare una semplice redistribuzione casuale dei dati di input, dato che, in tal caso, si perderebbero le relazioni di sequenzialità fra i record. Comunque, nonostante i dati fossero affetti da qualche anomalia, si è riusciti ad ottenere previsioni accettabili. L'obiettivo infatti non consisteva nell'identificare un metodo o una configurazione per ottenere previsioni ottime, ma, piuttosto, nell'individuare le principali tecniche di *machine learning*, applicandole ad un problema reale, per ottenere risultati rispettabili ma, certamente, non perfetti. Sono state poi effettuate, sul modello individuato, alcune prove aggiuntive, analizzandone le ripercussioni sui risultati. Si è tentato di migliorare i risultati applicando tecniche di *one hot encoding* e di *bucketing* ad alcune delle *feature* di input, ma non si è riusciti ad ottenere miglioramenti apprezzabili, probabilmente a causa delle anomalie dei dati citate precedentemente. Infine è stata effettuata una prova di scalabilità, considerando 42 stazioni di monitoraggio ed utilizzando lo stesso modello individuato precedentemente, senza dilungarsi nella ricerca di un modello più prestante. Anche in questo caso si sono ottenuti buoni risultati, dimostrando che un approccio di questo tipo è adottabile anche per un problema a più larga scala. Bisogna però tenere in considerazione che per ottenere una qualità dei risultati adeguata sarebbe necessario incrementare la complessità del modello in relazione al numero di stazioni considerate; ciò potrebbe rendere preferibile un approccio diverso, definendo, ad esempio, vari modelli più semplici, ed assegnando ad ognuno un qualche sottoinsieme delle stazioni da considerare. Le reti neurali sono strumenti molto potenti; infatti è stato possibile ottenere buone previsioni senza conoscere minimamente alcuna proprietà fisica o chimica, inerente ad esempio alle proprietà di dispersione, di tali composti, ma avendo a disposizione solamente una buona collezione di dati. In generale, il maggior sforzo richiesto nell'applicare queste tecniche consiste nell'analisi e nella manipolazione iniziale dei dati, selezionando le *feature* più importanti e definendo la nuova struttura dei dati da fornire alla rete. Un'altra parte significativa, che tuttavia in questa trattazione non è stata approfondita attraverso sperimentazioni dedicate, consiste nel cosiddetto *tuning* della rete, cioè nell'individuazione della configurazione e dei parametri che consentono di ottenere i risultati migliori. Tale ricerca avviene attraverso una serie di prove sperimentali. In questo documento sono state descritte le principali nozioni relative alle reti neurali e le principali tecnologie disponibili, mettendole poi in pratica su un caso reale in modo semplice, per raggiungere gli obiettivi preposti, senza dilungarsi su tutti i possibili approfondimenti o sperimentazioni attuabili. Durante le

fasi necessarie al conseguimento degli obiettivi sono state adottate alcune strategie e sono state operate determinate scelte, basandosi, a volte, sulle conoscenze teoriche, a volte, sui risultati di prove sperimentali o, altre volte, sull'intuito. Non si può quindi garantire che le soluzioni proposte in questa trattazione siano state sempre le scelte ottimali. Inoltre, visto l'elevatissimo numero di configurazioni ottenibili, è stata tralasciata l'analisi di alcune condizioni, che avrebbero potuto influire sui risultati: ad esempio l'utilizzo di diverse celle ricorrenti (semplici, LSTM), l'utilizzo di diverse funzioni di attivazione, la variazione della lunghezza delle sequenze o del *batch size*, l'utilizzo di altri ottimizzatori e la regolazione del *learning rate*, l'adozione di tecniche di regolarizzazione o l'utilizzo di *input feature* diverse. La ripercussione della variazione di tali condizioni sui risultati potrebbe essere argomento di studi e sperimentazioni ulteriori. In ogni caso, i risultati ottenuti, considerando che la base di partenza era una semplice collezione di dati, sono sicuramente ragguardevoli e rispecchiano appieno le potenzialità di queste tecniche.

Bibliografia

[6] Giuseppe Bonaccorso. *Machine Learning Algorithms*. Packt Publishing, 2017. 360 pp. ISBN: 9781785889622.

[14] Jacopo Giliberto. «Smog: in Lombardia le stufe a pellet la prima causa di Pm10». In: *Il Sole 24 Ore* (15 giu. 2016). URL: http://www.ilsole24ore.com/art/impresa-e-territori/2016-06-14/smog-lombardia-stufe-pellet-prima-causa-pm10-120910.shtml?uuid=ADVdBnb (visitato il 16/06/2018).

[17] Antonio Gulli e Sujit Pal. *Deep Learning with Keras*. Packt Publishing, 2017. 318 pp. ISBN: 9781787128422.

[21] Prateek Joshi. *Artificial Intelligence with Python*. Packt Publishing, 2017. 446 pp. ISBN: 9781786464392.

[22] Uday Kamath e Krishna Choppella. *Mastering Java Machine Learning*. Packt Publishing, 2017. 556 pp. ISBN: 9781785880513.

[41] Fabio M. Soares e Alan M. F. Souza. *Neural Network Programming with Java - Second Edition*. Packt Publishing, 2017. 269 pp. ISBN: 9781787126053.

[43] Paul Tinker e Tom Levitt. «How air pollution affects your health - infographic». In: *The Guardian* (5 lug. 2016). URL: https://www.theguardian.com/sustainable-business/2016/jul/05/how-air-pollution-affects-your-health-infographic (visitato il 04/07/2018).

[48] John Vidal. «Air pollution more deadly in Africa than malnutrition or dirty water, study warns». In: *The Guardian* (20 ott. 2016). URL: https://www.theguardian.com/global-development/2016/oct/20/air-pollution-deadlier-africa-than-dirty-water-or-malnutrition-oecd (visitato il 04/07/2018).

Sitografia

[1] *Air quality egg*. URL: https://airqualityegg.wickeddevice.com/ (visitato il 05/07/2018).

[2] *AirVisual*. URL: https://www.airvisual.com/ (visitato il 05/07/2018).

[3] Denis Akhiyarov. *Advanced usage of recurrent neural networks*. URL: https://urldatebooks.azure.com/denfromufa/libraries/pmlc/html/Lectures/Lecture_11_LSTM/6.3-advanced-usage-of-recurrent-neural-networks.ipynb (visitato il 04/07/2018).

[4] Rachel Allen e Michael Li. *Ranking Popular Deep Learning Libraries for Data Science*. URL: https://blog.thedataincubator.com/2017/10/ranking-popular-deep-learning-libraries-for-data-science/ (visitato il 26/06/2018).

[5] *Arpae Emilia-Romagna*. URL: https://www.arpae.it/ (visitato il 05/07/2018).

[7] *BreezoMeter*. URL: https://breezometer.com/ (visitato il 05/07/2018).

[8] *Descending into ML: Linear Regression*. URL: https://developers.google.com/machine-learning/crash-course/descending-into-ml/linear-regression (visitato il 19/06/2018).

[9] *Descending into ML: Training and Loss*. URL: https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss (visitato il 21/06/2018).

[10] *Feature Crosses: Encoding Nonlinearity*. URL: https://developers.google.com/machine-learning/crash-course/feature-crosses/encoding-nonlinearity (visitato il 21/06/2018).

[11] *First Steps with TensorFlow: Toolkit*. URL: https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit (visitato il 27/06/2018).

[12] *Framing: Key ML Terminology*. URL: https://developers.google.com/machine-learning/crash-course/framing/ml-terminology (visitato il 18/06/2018).

[13] *Generalization: Peril of Overfitting*. URL: https://developers.google.com/machine-learning/crash-course/generalization/peril-of-overfitting (visitato il 21/06/2018).

[15] *Gruppo Meteo/RFC-rmap*. URL: http://www.raspibo.org/wiki/index.php/Gruppo_Meteo/RFC-rmap (visitato il 05/07/2018).

[16] *Guida per scaricare i dati di qualità dell'aria*. URL: https://www.arpae.it/dettaglio_generale.asp?id=1603&idlivello=1637 (visitato il 05/07/2018).

[18] *How can I make my outdoor Node/Pro's data public?* URL: http://support.airvisual.com/knowledgebase/articles/948895-how-can-i-make-my-outdoor-node-pro-s-data-public (visitato il 05/07/2018).

[19] Hvass-Labs. *Time-Series Prediction*. URL: https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/23_Time-Series-Prediction.ipynb (visitato il 04/07/2018).

[20] *Introduction to Neural Networks: Anatomy*. URL: https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/anatomy (visitato il 21/06/2018).

[23] *Livelli di concentrazione di biossido di azoto (NO2)*. URL: http://www.arpa.veneto.it/arpavinforma/indicatori-ambientali/indicatori_ambientali/atmosfera/qualita-dellaria/livelli-di-concentrazione-di-biossido-di-azoto-no2 (visitato il 16/06/2018).

[24] *Livelli di concentrazione di biossido di zolfo (SO2)*. URL: http://www.arpa.veneto.it/arpavinforma/indicatori-ambientali/indicatori_ambientali/atmosfera/qualita-dellaria/livelli-di-concentrazione-di-biossido-di-zolfo-so2 (visitato il 16/06/2018).

[25] *Livelli di concentrazione di monossido di carbonio (CO)*. URL: http://www.arpa.veneto.it/arpavinforma/indicatori-ambientali/indicatori_ambientali/atmosfera/qualita-dellaria/livelli-di-concentrazione-di-carbonio-co (visitato il 16/06/2018).

[26] *Livelli di concentrazione di polveri fini (PM10)*. URL: http://www.arpa.veneto.it/arpavinforma/indicatori-ambientali/indicatori_ambientali/atmosfera/qualita-dellaria/livelli-di-concentrazione-di-polveri-fini-pm10 (visitato il 16/06/2018).

[27] *LOOKO2*. URL: http://looko2.com/ (visitato il 05/07/2018).

[28] *Meteo - dati osservati*. URL: https://dati.arpae.it/dataset/dati-dalle-stazioni-meteo-locali-della-rete-idrometeorologica-regionale (visitato il 05/07/2018).

[29] *OpenWeatherMap*. URL: https://openweathermap.org/ (visitato il 05/07/2018).

[30] *PurpleAir*. URL: https://www.purpleair.com/ (visitato il 05/07/2018).

[31] *Qualità dell'Aria - Dati di monitoraggio*. URL: https://dati.arpae.it/dataset/qualita-dell-aria-rete-di-monitoraggio (visitato il 05/07/2018).

[32] *Reducing Loss: Gradient Descent*. URL: https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent (visitato il 21/06/2018).

[33] *Reducing Loss: Learning Rate*. URL: https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate (visitato il 21/06/2018).

[34] *Reducing Loss: Stochastic Gradient Descent*. URL: https://developers.google.com/machine-learning/crash-course/reducing-loss/stochastic-gradient-descent (visitato il 21/06/2018).

[35] *Regularization for Simplicity: L2 Regularization*. URL: https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/l2-regularization (visitato il 21/06/2018).

[36] *Regularization for Simplicity: Lambda*. URL: https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/lambda (visitato il 21/06/2018).

[37] *Regularization for Sparsity: L1 Regularization*. URL: https://developers.google.com/machine-learning/crash-course/regularization-for-sparsity/l1-regularization (visitato il 21/06/2018).

[38] *Representation: Cleaning Data*. URL: https://developers.google.com/machine-learning/crash-course/representation/cleaning-data (visitato il 21/06/2018).

[39] *Representation: Feature Engineering*. URL: https://developers.google.com/machine-learning/crash-course/representation/feature-engineering (visitato il 21/06/2018).

[40] *Representation: Qualities of Good Features*. URL: https://developers.google.com/machine-learning/crash-course/representation/qualities-of-good-features (visitato il 21/06/2018).

[42] *Thingspeak*. URL: https://thingspeak.com/ (visitato il 05/07/2018).

[44] *Training and Test Sets: Splitting Data*. URL: https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data (visitato il 21/06/2018).

[45] *Training Neural Networks: Best Practices*. URL: https://developers.google.com/machine-learning/crash-course/training-neural-networks/best-practices (visitato il 21/06/2018).

[46] *uRADMonitor*. URL: https://www.uradmonitor.com/ (visitato il 05/07/2018).

[47] *Validation: Another Partition*. URL: https://developers.google.com/machine-learning/crash-course/validation/another-partition (visitato il 21/06/2018).

[49] *World Air Quality Index*. URL: http://aqicn.org (visitato il 05/07/2018).