Agenzia nazionale per le nuove tecnologie,
l'energia e lo sviluppo economico sostenibile

MINISTERO DELLO SVILUPPO ECONOMICO

Deep Learning per la disaggregazione dei consumi elettrici di edifici residenziali

C. La Riccia, C. Liti, V. Piccialli A. Pomente

Università di Roma
Tor Vergata

Report RdS/PAR2017/048

DEEP LEARNING PER LA DISAGGREGRAZIONE DI CONSUMI ELETTRICI DI EDIFICI RESIDENZIALI

C. La Riccia, C. Liti, V. Piccialli, A. Pomente (DICII "Università degli Studi di Roma “Tor Vergata”)

Settembre 2018

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dello Sviluppo Economico - ENEA
Piano Annuale di Realizzazione 2018
Area: Efficienza energetica e risparmio di energia negli usi finali elettrici e interazione con altri vettori energetici
Progetto: D.6 Sviluppo di un modello integrato di smart district urbano
Obiettivo: B
Responsabile del Progetto: Claudia Meloni, ENEA

Il presente documento descrive le attività di ricerca svolte all’interno dell’Accordo di collaborazione “*Sviluppo di modelli e di soluzioni black box per la disaggregazione dei dati di consumo elettrico domestico residenziale generale nelle componenti attribuibili ai singoli elettrodomestici*”
Responsabile scientifico ENEA: Claudia Snels
Responsabile scientifico DICII Università degli Studi di Roma Tor Vergata: Veronica Piccialli

Indice

Sommario

Il problema della disaggregazione dei consumi elettrici è noto in letteratura con l'acronimo NILM (dall'inglese Non-intrusive Load Monitoring). Con il termine NILM si fa riferimento ad un processo volto a stimare, sulla base delle variazioni del consumo generale, il consumo energetico di un dato elettrodomestico. I sistemi NILM permettono quindi di monitorare il consumo elettrico all'interno di una abitazione senza richiedere l'utilizzo di smart meter dedicati ai singoli elettrodomestici.

In letteratura, sono stati proposti diversi approcci atti alla risoluzione del problema della disaggregazione del consumo elettrico. Alcuni di essi si basano sull'utilizzo di modelli Markoviani, mentre altri prediligono tecniche di processamento di segnali mediante grafi. Negli ultimi anni, con l'avanzare della capacità computazionale delle macchine e grazie al forte interesse per le tecniche di Machine Learning, il problema della disaggregazione dei consumi elettrici è stato affrontato ricorrendo agli algoritmi di Deep Learning.

Alla luce dei promettenti risultati ottenuti nell'ambito dei problemi NILM utilizzando delle architetture di tipo deep, abbiamo deciso di ricorrere a questi algoritmi come strumento per risolvere il problema della disaggregazione del consumo energetico oggetto di questo progetto.

Seguendo la letteratura, sono state sviluppate due diverse architetture deep:

1. una rete neurale ricorrente, detta LSTM (Long Short-Term Memory);
2. una rete deep di tipo feed -forward basata sui DAE (Denoising Auto-Encoder).

La fase di addestramento dei modelli si è articolata in due fasi. In un primo momento le architetture sviluppate sono state testate sul dataset UKDALE considerato come il dataset benchmark per questa tipologia di problemi. Il dataset UKDALE contiene i dati dei consumi elettrici di alcune abitazioni situate nel Regno Unito.

Una volta valutata la bontà e la robustezza del modello di rete deep da noi individuata sui dati di benchmark, questa è stata testata utilizzando i dati delle case messi a disposizione da Enea nell'ambito di questo progetto sulla disaggregazione del consumo energetico.

Data la mole molto ridotta dei dati raccolti per le case ENEA abbiamo utilizzato i modelli addestrati sul dataset di letteratura per predire i consumi di frigorifero e lavatrice di alcune case ottenendo risultati molto interessanti. Si ha infatti un errore di predizione comparabile con quelli presentati in letteratura e ancora più interessante sui dati aggregati al giorno si riesce a predire la percentuale dovuta all'elettrodomestico con una buona precisione.

1 Introduzione

Il problema della disaggregazione dei consumi elettrici è noto in letteratura con l'acronimo NILM (dall'inglese Non-intrusive Load Monitoring). Con il termine NILM si fa riferimento ad un processo volto a stimare, sulla base delle variazioni del consumo generale, il consumo energetico di un dato elettrodomestico. I sistemi NILM permettono quindi di monitorare il consumo elettrico all'interno di una abitazione senza richiedere l'utilizzo di smart meter dedicati ai singoli elettrodomestici. Questo tipo di studio consente di fornire informazioni dettagliate sui consumi agli utenti, così da indurli a modificare le loro abitudini verso un uso più saggio dell'energia elettrica.

Obiettivo del presente progetto è quello di inferire, mediante l'utilizzo di tecniche di deep learning applicate ai dati relativi al consumo generale, il consumo elettrico di ogni elettrodomestico su cui fosse installata una smart plug. Lo studio ha coinvolto otto case situate a Roma, per le quali sono stati registrati dati sul consumo elettrico sia generale che per singolo dispositivo nel corso dell'anno 2017. In particolare ci sono stati forniti dati per una durata massima di otto mesi del 2017.

Prima di procedere ad esaminare questo insieme di dati si è studiata la letteratura e si è considerato il dataset UKDALE descritto in [4], dove sono raccolti i dati dei consumi di 5 case raccolti per al massimo 3 anni, con diversi elettrodomestici.

Si è scelto di considerare prima il dataset proposto dalla letteratura per validare la bontà del modello prodotto in quanto erano disponibili in letteratura risultati ottenuti su questo dataset che sono stati considerati come benchmark.

Come vedremo in dettaglio più avanti all'interno del report, la nostra implementazione dei modelli proposti in letteratura fornisce non solo dei risultati comparabili con quanto presentato in passato, ma riesce, talvolta, a migliorarli, restituendo predizioni e stime più accurate da un punto di vista dell'errore tra segnale stimato e segnale originale. Questi risultati ci hanno permesso quindi di inferire non solo una corretta implementazione del modello della letteratura, ma anche di concludere che la nostra implementazione supera in termini di accuratezza quella presentata in [5].

2 Descrizione delle attività svolte e risultati

In questa sezione verranno descritte le attività che hanno caratterizzato il progetto oggetto di questo report. La prima fase ha riguardato un'attenta analisi dei lavori pubblicati nell'ambito della risoluzione dei problemi di disaggregazione del consumo energetico. Una volta individuata la strategia più idonea a risolvere il problema in esame si è passata alla fase di pulizia e preparazione dei dati fornitici. I modelli di deep learning individuati sono stati testati dapprima considerando un dataset ampiamente utilizzato nella letteratura NILM per poi essere utilizzati sui dati fornitici.

2.1 Analisi della Letteratura

Il problema della disaggregazione dei consumi elettrici è noto in letteratura con l'acronimo NILM. Con il termine NILM si fa riferimento ad un processo volto a stimare, sulla base delle variazioni del consumo generale, il consumo energetico di un dato elettrodomestico. I sistemi NILM permettono quindi di monitorare il consumo elettrico all'interno di una abitazione senza richiedere l'utilizzo di smart meter dedicati ai singoli elettrodomestici.
In letteratura, sono stati proposti diversi approcci atti alla risoluzione del problema della disaggregazione del consumo elettrico [1-8]. Alcuni di essi si basano sull'utilizzo di modelli Markoviani [10, 11, 12, 13] mentre altri prediligono tecniche di processamento di segnali mediante grafi [14, 15]. Per quanto concerne invece l'utilizzo di strumenti di machine learning, il problema della disaggregazione del consumo elettrico è stato

affrontato mediante l'utilizzo di architetture di tipo "deep" ovvero mediante delle reti neurali costituite da molti strati [9].
Alla luce dei promettenti risultati presentati in [5] abbiamo deciso di ricorrere alle architetture deep come strumento per risolvere il problema della disaggregazione del consumo energetico oggetto di questo progetto. In [5] viene dettagliata e motivata in modo chiaro e coinciso la scelta dell'utilizzo di reti neurali deep per la risoluzione del problema della disaggregazione. In particolare, vengono mostrati i promettenti risultati ottenuti da due modelli particolari che sono poi stati implementati per la realizzazione del presente report.
Da questo articolo sono state estrapolate le principali informazioni che permettessero di riprodurre i medesimi esperimenti sia sul dataset UKDALE, sia sul dataset oggetto di questo lavoro; lo sviluppo del modello, la parametrizzazione e la scelta della dimensione delle finestre temporali sono tutti aspetti seguiti pedissequamente in partenza, nella realizzazione del modello qui presentato, fornendo una base per lo sviluppo di un modello leggermente differente e talvolta migliore.
L'autore dell'articolo mostra anche dei risultati relativi ad altri modelli, non appartenenti al mondo del deep learning, sottolineando quanto questi siano molto spesso peggiori e poco raccomandabili da un punto di vista dell'accuratezza. Grazie a queste informazioni siamo riusciti ad orientarci subito su quali modelli fare affidamento e quali scartare fin dall'inizio.

2.2 Modelli di Machine Learning utilizzati:

Prima di procedere alla descrizione delle architetture deep utilizzate nell'ambito di questo progetto, procederemo con il descrivere brevemente che cos'è una rete neurale artificiale, dal momento che una rete deep altro non è che una particolare tipologia di rete neurale.

2.2.1 Rete Neurale Artificiale

Una rete neurale artificiale può essere definita come un processore distribuito, ispirato ai principi di funzionamento del sistema nervoso degli organismi evoluti. È costituita da una serie di unità computazionali interconnesse fra di loro e in grado di immagazzinare le informazioni apprese durante un processo di addestramento. In altri termini, una rete neurale può essere descritta mediante un grafo orientato, i cui nodi, detti neuroni, rappresentano le unità computazionali. Gli archi del grafo rappresentano invece l'equivalente delle sinapsi di cui è costituito il cervello umano. La conoscenza viene immagazzinata all'interno di questi collegamenti sotto forma di pesi, motivo per il quale il grafo è diretto e pesato. Il valore dei pesi di ogni arco deve essere dinamico e aggiornato durante la fase di addestramento, permettendo alla rete neurale di imparare ogni volta che riceve un nuovo esempio.
Ogni rete neurale artificiale è suddivisa in strati, il cui numero insieme al numero di neuroni per ogni strato definisce l'architettura della rete. In particolare una rete neurale caratterizzata dalla presenza di un numero elevato di strati si dice *deep,* mentre a fronte di un numero ridotto di livelli si parla di architettura *shallow.*
Ogni rete neurale possiede uno *strato di input (o input layer)*, uno *strato di output (output layer)* ed un numero arbitrario di strati intermedi, detti *strati nascosti (o hidden layer).* In Figura 1 vengono presentati due esempi di architetture: nel primo caso la rete è costituita da un solo strato nascosto, mentre nella seconda rete sono connessi in cascata due strati nascosti.

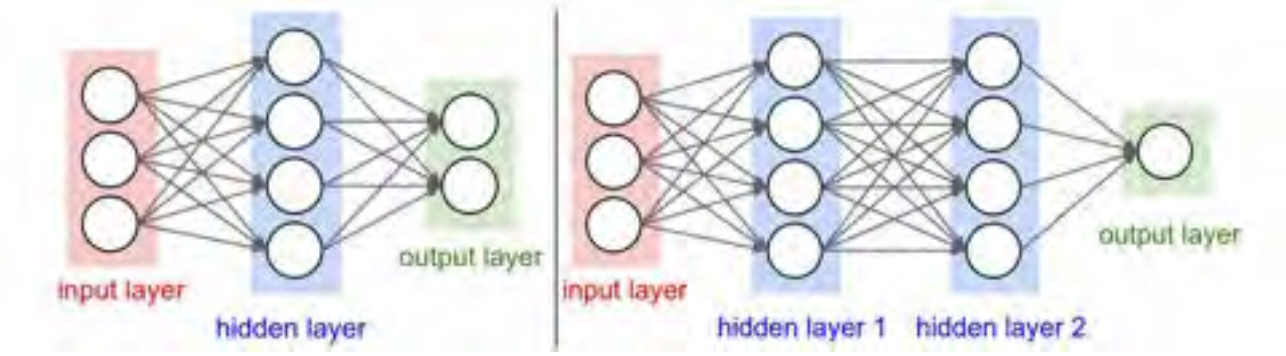

Figura 1: Esempi di reti neurali artificiali

Lo strato di input si occupa di ricevere segnali dall'esterno ed inoltrarli all'interno della rete, quindi verso gli strati nascosti (se presenti) o direttamente verso lo strato di uscita.
I neuroni in ogni strato nascosto applicano una particolare funzione (detta *funzione di attivazione*) al segnale ricevuto dallo strato precedente, che può essere lo strato di input o un altro strato nascosto, in modo tale da estrarre ed imparare relazioni complesse e altamente non lineari che esistono tra i dati. Formalmente dato un insieme di addestramento

$$T = \{(x^k, y^k) : x^k \in R^n, y^k \in R, k = 1, \dots, P\}$$

costituito da P coppie ingresso-uscita dove all'ingresso x^k viene associata un'uscita y^k, il problema dell'addestramento di una rete neurale si riduce a risolvere un problema di ottimizzazione le cui variabili altro non sono che i pesi associati alle connessioni tra i neuroni della rete. Una volta determinato il vettore dei pesi w, per ogni elemento del training set sarà possibile calcolare l'uscita prodotta dalla rete. Un esempio di neurone con funzione di attivazione f è disponibile in Figura 2. Come è possibile osservare, l'uscita del neurone y dipende da tutti gli input forniti x e dal vettore di pesi w, tramite la formula:

$$y = f\left(\sum_i w_i x_i\right)$$

Solitamente, lo scopo della funzione f è quello di introdurre un certo fattore di non linearità all'interno della rete.
Esempi di funzione di attivazione sono la funzione lineare, la sigmoide, la tangente iperbolica, la ReLU ecc.
Infine, lo strato di output si occupa di fornire uno o più risultati in uscita, basandosi su quanto appreso fino a quel momento; nel caso specifico relativo al problema di disaggregazione dell'energia, l'uscita della rete neurale, a prescindere dal modello implementato, deve essere un numero reale rappresentante il consumo dell'elettrodomestico in esame ad un certo istante temporale. Questo significa che la rete neurale deve risolvere un problema di *regressione supervisionato*, in cui il modello effettua una stima del consumo energetico, dopo aver effettuato una fase di training che si avvale della conoscenza dell'uscita sugli esempi del training set.
Sotto opportune ipotesi sulle funzioni di attivazione si riesce a dimostrare che una rete neurale con un solo strato nascosto è in grado di approssimare con precisione arbitraria qualunque funzione continua su un insieme compatto. Si dice quindi che le reti neurali sono approssimatori universali.

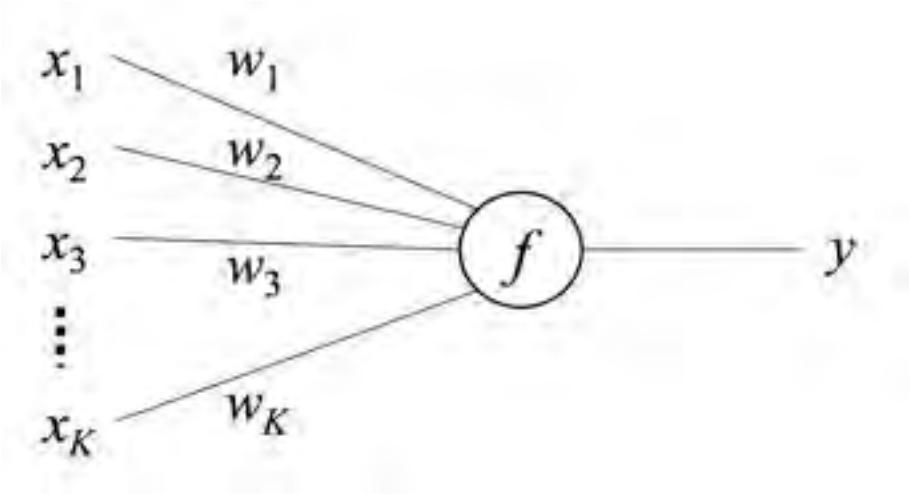

Figura 2: Esempio di neurone con funzione di attivazione f

L'addestramento di una rete neurale può avvenire secondo tre modalità differenti:

- *Supervisionato*: in questo caso la rete viene addestrata attraverso l'utilizzo di un training set e la minimizzazione di una funzione di errore che tiene conto degli esempi di uscita a disposizione sul training set;
- *Non Supervisionato*: in questo caso la rete viene addestrata tramite una serie di input non etichettati e che verranno classificati ed organizzati sulla base di caratteristiche comuni, in modo da poter effettuare stime su dati futuri mai visti in precedenza;
- *Apprendimento per Rinforzo*: in questo caso la rete è in grado di apprendere ed adattarsi a mutazioni dell'ambiente in cui vive, tramite l'utilizzo di un sistema di ricompense (positive o negative).

Nel caso supervisionato, l'addestramento del modello implementato viene effettuato in genere per mezzo dell'algoritmo *backpropagation*, che corrisponde ad applicare il metodo del gradiente a passo costante sulla funzione di errore complessiva della rete. In particolare, secondo l'algoritmo inizialmente l'input viene propagato all'interno della rete e quindi verso gli strati centrali di cui essa è composta, fino al raggiungimento dello strato di output, ovvero del risultato della predizione (forward pass); successivamente viene valutato l'errore tra la stima effettuata ed il caso reale aggiornando, di conseguenza, i pesi della rete (feedback pass). La funzione da minimizzare viene scelta in fase di progettazione della rete neurale; le funzioni tipiche utilizzate per problemi di regressione sono: Mean Absolute Error – MAE, Mean Square Error - MSE, Root Mean Square Error – RMSE:

$$MAE = \frac{1}{T}\sum_{t=1}^{T}|\hat{y}_t - y_t|\,, MSE = \frac{1}{T}\sum_{t=1}^{T}(\hat{y}_t - y_t)^2\,, RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}$$

Nelle nostre implementazioni abbiamo usato come funzione da minimizzare la funzione MSE.

In letteratura si distinguono due tipi principali di architetture:

1. Reti feed-forward
2. Reti ricorrenti

L'architettura Feed-Forward può essere rappresentata mediante un grafo diretto aciclico, in cui non sono ammessi archi all'indietro o archi trasversali, cioè tra nodi appartenenti al medesimo strato della rete.

Le reti ricorrenti, al contrario, ammetto la presenza di cicli e quindi è possibile l'esistenza di archi all'indietro e tra nodi appartenenti al medesimo strato. Le reti ricorrenti possiedono la capacità di imparare ed individuare le relazioni temporali tra i dati, in quanto durante l'addestramento i dati attuali vengono influenzati dai dati passati (o addirittura futuri se gli strati sono bidirezionali), grazie alla presenza di archi tra strati differenti. In altre parole, ogni neurone può ricevere in input non solo dati provenienti dall'esterno, ma anche gli output provenienti dagli strati successivi così che una parte della rete, detta *cella*, possieda una memoria preservando degli stati nel tempo.
In Figura 3 sono riportati due esempi inerenti alle due tipologie di architetture di rete neurale citate; possiamo notare, come detto in precedenza, che la rete neurale ricorrente possiede archi all'indietro o trasversali, mentre la rete feed-forward è totalmente aciclica.

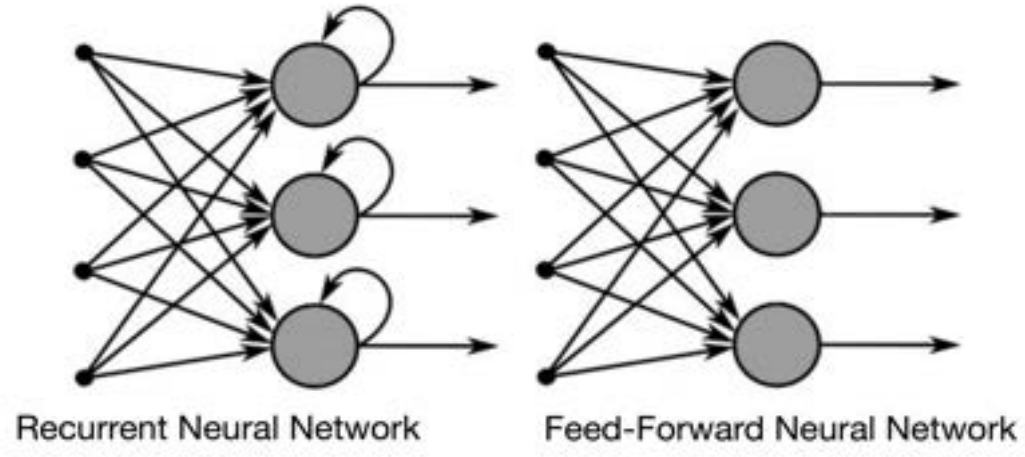

Figura 3: esempi di reti neurali ricorrenti e feed forward

Per il problema specifico, seguendo l'approccio proposto in [5] sono state implementati due modelli differenti, una rete ricorrente detta LSTM ed una rete Feed-Forward che fa uso di Denoising Auto-Encoder. Entrambe le reti sono state prima valutate utilizzando il dataset UKDALE, in modo da poter prendere dalla letteratura dei risultati di benchmark che permettessero di valutare la bontà dell'approccio scelto.

2.2.2 Long Short-Term Memory

La Long Short-Term Memory è un tipo di rete neurale ricorrente profonda, in cui le celle di memoria sono dette LSTM ed hanno la particolare caratteristica di riuscire ad individuare le dipendenze tra dati contenuti in un ampio intervallo temporale.
In Figura 4 è mostrata l'architettura di una cella LSTM; lo stato è suddiviso in due vettori h_t e c_t, rappresentanti rispettivamente lo stato a breve termine e lo stato a lungo termine.

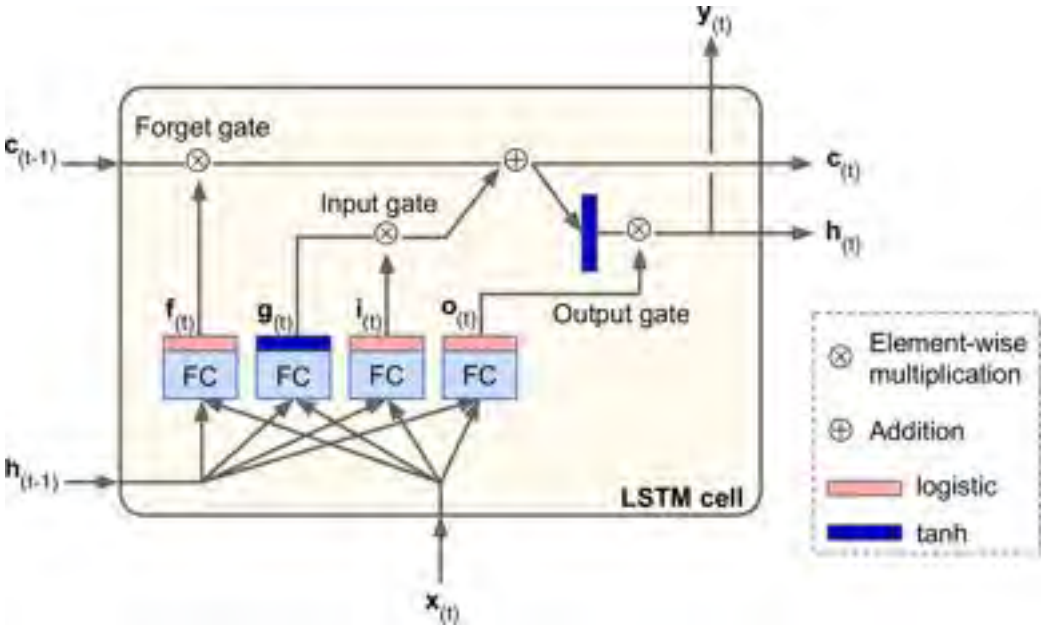

Figura 4: struttura della cella LSTM

L'idea fondamentale è che la rete ha la facoltà di imparare quali dati memorizzare nello stato a lungo termine, quali invece non devono essere memorizzati e quali dati leggere da esso. Infatti, come mostrato in Figura 4, lo stato a lungo termine c_{t-1} viene prima processato dal forget gate che si occupa di eliminare alcune memorie e successivamente viene sommato con le nuove memorie che sono state selezionate all'interno dell'input gate. A questo punto lo stato a lungo termine viene copiato; la prima copia viene inoltrata direttamente in uscita senza applicare ulteriori trasformazioni, mentre alla seconda copia viene applicata la tangente iperbolica ed il risultato filtrato attraverso l'output gate. In questo modo vengono prodotti i due stati h_t e c_t. Notiamo che lo stato a breve termine prodotto coincide anche con l'output y_t della cella in quel preciso istante temporale.
Inoltre, per generare le nuove memorie da aggiungere allo stato a lungo termine, l'input x_t e lo stato a breve termine nell'istante precedente h_{t-1} vengono processati da quattro strati paralleli totalmente connessi:

- Strato f_t: controlla quali dati devono essere eliminati dallo stato a lungo termine, tramite l'input gate;
- Strato g_t: analizza l'output corrente e lo stato a breve termine relativo all'istante di tempo precedente;
- Strato i_t: controlla quali informazioni prodotte dallo strato g_t devono essere aggiunte allo stato a lungo termine;
- Strato o_t: controlla quali dati appartenenti allo stato a lungo termine devono essere letti. In questo modo controlla quale deve essere il contenuto dello stato a breve termine all'istante successivo e quindi l'output della cella.

Solitamente le connessioni della rete LSTM vengono rese *peephole*, cioè si permette ad ogni gate di poter ispezionare lo stato della cella anche quando il gate di output risulta essere chiuso. Questo permette alla rete di mantenere le dipendenze temporali a lungo termine e mitiga il problema dell' *Exploding Gradient* o del *Vanishing Gradient*. Questi problemi si verificano rispettivamente quando, durante l'applicazione dell'algoritmo backpropagation, il gradiente diventa troppo grande o troppo piccolo, provocando nel primo caso aggiornamenti troppo frequenti, mentre nel secondo caso quasi assenti, bloccando il processo di ottimizzazione.

2.2.3 Auto Encoder

Con il termine Auto-Encoder (AE) si fa riferimento ad una particolare tipologia di rete neurale addestrata con l'obiettivo di riprodurre una copia dei dati forniti come input al modello [9]. Queste architetture sono state introdotte negli anni 80 per risolvere il problema dell'addestramento non supervisionato, utilizzando gli stessi dati di input come benchmark per valutare la bontà del modello.
Un AE si costituisce di due moduli: l'encoder e il decoder. I dati di input vengono inviati all'encoder che, sfruttando trasformazioni tipicamente non-lineari, individua una rappresentazione compatta dei dati in ingresso. I dati così codificati vengono inviati al decoder il cui compito è quello di ricostruire, a partire dalla versione codificata dei dati di input, l'input stesso. In Figura 5 viene presentato un esempio di auto-encoder.

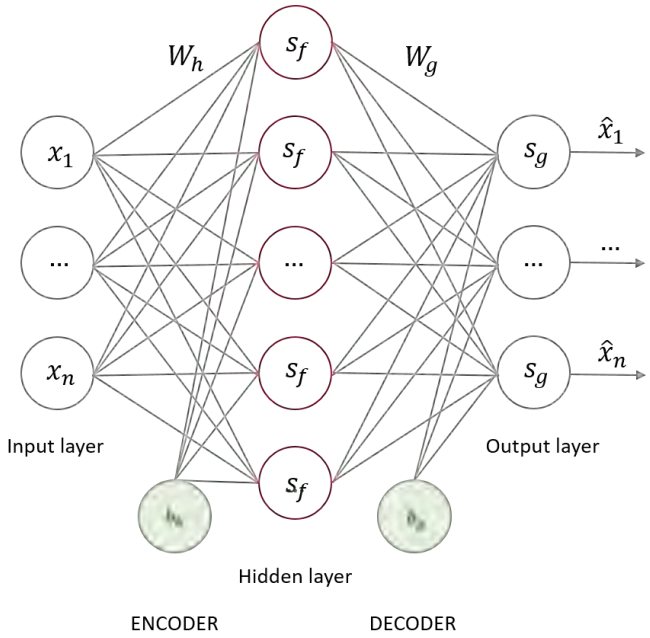

Figura 5: Architettura di un Auto-Encoder

Dato un campione x appartenente al training set durante le fase di addestramento si cerca di minimizzare una funzione di errore o di ricostruzione $r(x) = g(f(x))$ che è data dalla composizione della funzione di encoding $h = f(x) = s_f(W_h x - b_h)$ e quella di decoding $r = g(h) = s_g(W_g h - b_g)$. Dato il generico campione x del training set quello che si vuole minimizzare è la differenza tra l'input x e la decodifica di x calcolata dalla rete.
È opportuno sottolineare che l'obiettivo di un auto-encoder non è quello di copiare i dati in ingresso, ma di individuare quali sono le caratteristiche che meglio rappresentano il fenomeno di interesse. La capacità di ricostruire i dati in ingresso a partire da una loro versione codificata deve essere interpretata come la capacità del modello di individuare tutte e sole quelle componenti che sono rappresentative del fenomeno di interesse.
Per evitare che la rete si limiti semplicemente a realizzare una copia dei dati in ingresso possono essere utilizzati termini di regolarizzazione, come nel caso dei contractive o degli sparse auto-encoder oppure si può corrompere il dato in ingresso così da incoraggiare l'encoder ad individuare una rappresentazione dei dati di ingresso che sia quanto più robusta possibile rispetto al rumore precedentemente aggiunto ai dati. Gli auto-encoder che si avvalgono dell'aggiunta di un rumore ai dati di input sono noti in letteratura con il termine di denoising auto-encoder (DAE).
Indipendentemente dalla strategia impiegata per evitare che il modello copi banalmente i dati che è chiamato ad analizzare, gli auto-encoder risultano essere ampiamente utilizzati in moltissimi ambiti come ad esempio: anomaly detection, data denoising, image inpainting, manifold learning, transfer learning, etc.

Per quanto riguarda il progetto oggetto di questo report ci si è concentrati sull'utilizzo dei denoising auto-encoder. In un denoising auto-encoder il dato in input viene corrotto aggiungendo del rumore. In letteratura sono state introdotte diverse strategie atte ad aggiungere rumore nei dati, le più utilizzate sono il *Zero Masking Noise (MN),* il *Salt-and- Pepper Noise (SPN)* e il *isotropic Gaussian Noise (GS).* Nel zero masking noise vengono selezionate randomicamente una serie di componenti dal campione e il loro valore viene forzato a zero. Nel caso del salt-and-pepper-noise le componenti selezionate vengono forzate al loro valore minimo o massimo (tipicamente 0 o 1), infine nel caso dell'isotropic gaussian noise il processo di corruzione prevede di aggiungere alle componenti di x una quantità ϵ calcolata ricorrendo ad una distribuzione Gaussiana. Una volta corrotto il dato di input la rete verrà addestrata con l'obiettivo di minimizzare l'errore tra la versione originale del dato in input e quella calcolata dalla rete a partire dalla versione corrotta del dato.
Gli auto encoder vengono utilizzati come strati all'interno di reti deep per estrarre delle caratteristiche di interesse del dataset in modo da massimizzare l'informazione che attraversa gli altri strati della rete. Un esempio è l'architettura da noi implementata dove l'auto encoder è connesso a strati convoluzionali e strati completamente connessi (fully connected).

2.3 Preprocessing dei dati delle case ENEA:

Una volta completata la fase di analisi della letteratura si è passati a quella di pre-processing. In questa fase sono state attuate tutta una serie di procedure volte alla pulizia e trasformazione dei dati a nostra disposizione. È importante sottolineare che la bontà dei risultati ottenibili mediante l'addestramento di un qualsiasi algoritmo di apprendimento dipende fortemente dalla qualità dei dati che costituiscono il set di addestramento (o training set). Tutti gli algoritmi di apprendimento, per quanto complessi e articolati, infatti risentono del cosiddetto fenomeno del *garbage in garbage out.* La fase di pre-processing rappresenta pertanto una delle fasi più critiche all'interno di un processo di predictive data analytics.

Le attività di pre-processing realizzate nell'ambito di questo progetto di disaggregazione del consumo energetico possono essere raggruppate in tre categorie: i dati a nostra disposizione sono stati dapprima analizzati in termini di numerosità e significatività, per poi essere valutati in termini di consistenza e qualità. I problemi riscontrati durante queste due analisi preliminari sono stati risolti ricorrendo a soluzioni custom scritte in Python e R. Una volta risolti i problemi dovuti alla numerosità e alla qualità dei dati si è passati alla fase di trasformazione e conversione, necessaria per rendere i dati compatibili con le specifiche degli algoritmi di apprendimento utilizzati. In quest'ultima fase abbiamo deciso di ricorrere alla libreria nilmtk (dall'inglese Non-Intrusive Load Monitoring Toolkit), ampiamente utilizzata nell'ambito dei problemi di disaggregazione del consumo energetico.

Il convertitore nilmtk offre un'interfaccia unificata, grazie alla quale è possibile analizzare statisticamente i dati sotto forma di serie temporali, applicare filtri, utilizzare algoritmi di disaggregazione pre-implementati ed eseguire operazioni di conversione del dataset (spesso spezzettato in più file) in un unico file. Sfruttando le funzionalità offerte da questa libreria è pertanto possibile lavorare su una versione compatta del dataset che risulta essere facilmente utilizzabile. Inoltre, tale libreria è stata sviluppata con un approccio general purpose, che permette agli sviluppatori di arricchire la stessa con ulteriori funzionalità e dataset.

Prima di approfondire l'utilizzo del convertitore, e quindi della terza fase delle operazioni di pre-processing, procederemo con il descrivere le criticità riscontrate durante le fasi di analisi preliminare dei dati. È stato sviluppato un piccolo script in R in grado di mostrare, giorno per giorno, la distribuzione dei dati all'interno di ogni singola casa, in modo tale da capire quali fossero le case con il maggior numero di informazioni utili da utilizzare per lo studio del miglior modello.
Questa fase preliminare è stata necessaria in quanto non tutte le case possiedono gli stessi elettrodomestici ed in particolare non è detto che tutti i giorni siano attivi gli stessi dispositivi. Informazioni di questo genere sono fondamentali per valutare la disponibilità dei dati che si è chiamati ad analizzare. L'assenza di registrazioni da parte di un certo elettrodomestico può essere imputabile a cause diverse: il dispositivo potrebbe essere semplicemente spento oppure la mancanza di dati potrebbe dipendere da malfunzionamenti del sensore o ancora dall'assenza di una connessione wi-fi, mancanza che impedisce al

sistema di trasmettere i segnali registrati. Una volta individuati i dati mancanti è possibile, in base alla tipologia del problema riscontrato, definire delle opportune operazioni di sostituzione dei valori mancanti.
Per quanto riguarda le case situate in zona Casaccia, di seguito Enea0, Enea1, Enea2, …, Enea9 ci sono stati forniti dati per un totale di 365 giorni. Tutte le 365 cartelle (una per ogni giorno dell'anno) sono state analizzate utilizzando uno script R, in grado di controllare giorno per giorno la presenza o l'assenza di dati per tutte le case. Le informazioni così ottenute sono state rappresentate mediante calendar plot. In Figura 6 e in Figura 7 vengono mostrati i calendar plot delle case di Casaccia. Questi grafici mostrano per ogni casa e relativamente all'anno 2017 i dati disponibili: se il giorno della settimana è in verde allora esistono dei file excel contenenti delle rilevazioni per quella casa altrimenti le cartelle corrispondenti risultano essere vuote. Notiamo che questo dà solo una prima informazione, in quanto un giorno può essere verde anche se c'è un'unica numerazione per quel giorno.

Figura 6: per ogni casa sono mostrati i dati disponibili relativamente all'anno 2017 (in verde i giorni in cui sono presenti delle rilevazioni)

Come si può notare, le cartelle corrispondenti ad alcune case sono pressoché vuote. Di conseguenza tali abitazioni sono state escluse da qualsiasi studio successivo perché del tutto inutilizzabili. Nello specifico le abitazioni escluse per assenza di un campione di dati significativo sono state: Enea0, Enea1, Enea5, Enea6 ed Enea10.

Figura 7: grafici sulla presenza di dati delle case rimanenti

Una volta individuate le case con un numero sufficiente di dati, sfruttando i report tecnici fornitici, abbiamo provveduto a stilare una lista contenente, per ogni abitazione, gli elettrodomestici connessi a smart plug. In Tabella 1 presentiamo l'elenco dei dispositivi per abitazione.

Per ogni abitazione la lista dei dispositivi è stata confrontata con il contenuto delle varie cartelle. Il risultato di questa seconda analisi è sintetizzato nella Figura 8.

	Smart Plug								
Casa	Lavatrice	Lavastoviglie	Frigorifero	Forno	Microonde	TV	Lampada	Termoventilatore	Scaldabagno

Enea3	✓	✗	✓	✗	✓	✗	✓	✗	✗
Enea4	✗	✗	✗	✓	✗	✓	✗	✗	✗
Enea7	✓	✗	✓	✗	✗	✓	✗	✗	✗
Enea8	✗	✓	✓	✓	✓	✓	✗	✗	✓
Enea9	✓	✓	✓	✗	✗	✗	✗	✓	✗

Tabella 1 :Elenco dei dispositivi rilevati per abitazione. Il simbolo ✓ indica la presenza di almeno un elettrodomestico per tipologia.

Figura 8: Analisi del contenuto delle cartelle in termini di numero di dispositivi. Se il giorno della settimana è in verde allora esistono dei file excel che contengono delle rilevazioni per tutti i dispositivi individuati per quella casa.

Come si può notare in alcuni casi, come ad esempio in Enea3 relativamente al mese di novembre, in nessun giorno sono presenti rilevazioni per tutti i dispositivi connessi a smart plug. Prima di escludere dallo studio altre abitazioni, a causa della mancanza di un campione significativo di dati, abbiamo deciso di controllare giorno per giorno il numero di dispositivi mancanti e la loro natura. L'assenza dei dati del consumo generale rendono completamente inutilizzabile il resto dei dati, viceversa se a mancare sono i dati di un altro dispositivo e se tale assenza non è eccessivamente prolungata nel tempo è possibile intervenire sostituendo i valori mancanti con opportune grandezze statistiche (media o mediana delle registrazioni del periodo

precedente e successivo etc.). In Figura 9 viene riportato il numero di dispositivi assente per ogni giorno della settimana per le abitazioni Enea3, Enea4, Enea7, Enea8 e Enea9. La scala cromatica presente in questo calendar plot va interpretata come segue: a colori più scuri corrisponde un maggior numero di dispositivi per i quali non si hanno dati a disposizione. Come si può notare in alcuni casi non è possibile intervenire con operazioni di sostituzione dei dati mancanti dal momento che i buchi sono anche di settimane. Si consideri ad esempio la casa Enea8 relativamente al mese di Novembre, in questo caso per nessuno dei sei dispositivi connessi a smart meter si hanno dati disponibili. In situazioni come questo non è possibile rimpiazzare i valori mancanti dal momento che la loro percentuale è troppo elevata e si rischierebbe di introdurre ulteriore rumore nei dati deteriorandone la qualità. In altri casi, come ad esempio in Enea7 relativamente alle prime due settimane di marzo, il numero di dispositivi mancanti è più contenuto ed è quindi possibile intervenire sostituendo i valori mancanti.

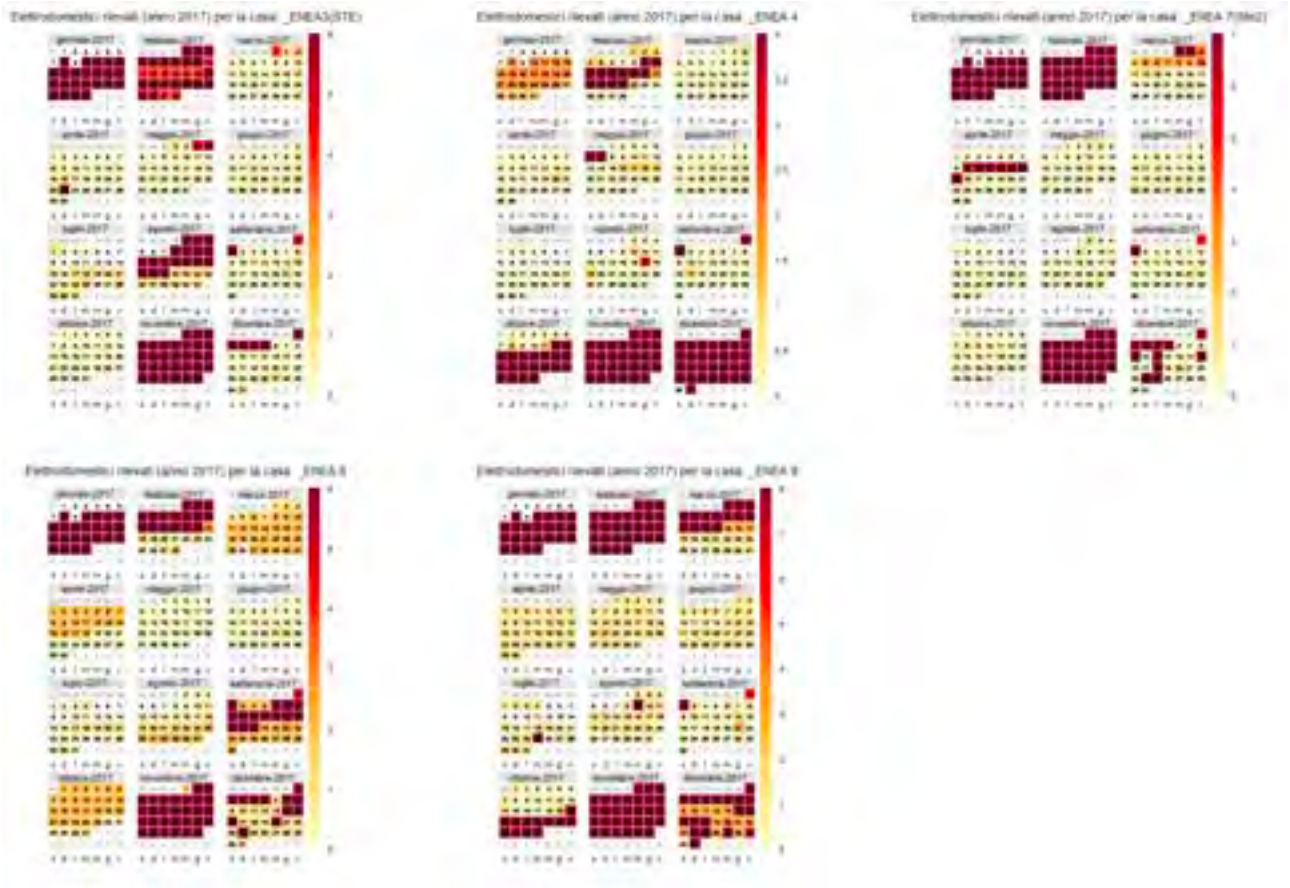

Figura 9: Numero di dispositivi mancanti per giorno della settimana

Per ogni abitazione sono stati inoltre analizzati il numero di meter presenti. Alcune case infatti sono caratterizzate dalla presenza di un impianto fotovoltaico. Questa informazione è utile per due ragioni: in primo luogo in fase di conversione è necessario, al fine di individuare il reale consumo energetico, sommare i due contributi. In secondo luogo la presenza di uno solo dei due meter è sufficiente per poter stimare il consumo degli altri elettrodomestici.

In Tabella 2, per ogni abitazione, vengono presentati il numero e il tipo di meter.

In tre delle cinque case considerate sono presenti entrambi i tipi di meter.

Abitazione	Home Energy Meter	Home Energy Meter Fotovoltaico
Enea3	✓	X
Enea4	✓	X
Enea7	✓	✓
Enea8	✓	✓
Enea9	✓	✓

Tabella 2 :Elenco dei smart meter rilevati per abitazione. Il simbolo ✓ indica la presenza dello smart meter.

Alla luce dei risultati ottenuti durante questa prima fase di analisi, abbiamo deciso di concentrarci sulle abitazioni Enea7 e Enea8. La percentuale di valori mancanti in queste abitazioni non è elevata. I periodi di prolungata assenza di rilevazioni si trovano o all'inizio o alla fine dell'anno e di conseguenza è possibile isolare almeno una sequenza temporale che copra un orizzonte temporale superiore a 6 mesi (cosa che non accade ad esempio per l'abitazione Enea3). Il numero di dispositivi connessi a smart meter è sufficientemente elevato, nel caso dell'abitazione Enea4, anch'essa caratterizzata da una buona numerosità dei dati, si hanno a disposizione i consumi soltanto di due elettrodomestici: forno e televisore. Un campione così ridotto di dispositivi da analizzare rende il problema della disaggregazione del consumo energetico molto più difficile da risolvere, dal momento che la percentuale del consumo imputabile ai dispositivi per i quali si hanno dati disponibili è del tutto irrisoria rispetto al consumo generale.

Le Figure 10 e 11 mostrano rispettivamente la distribuzione di elettrodomestici per le due case selezionate. In rosso sono evidenziati i giorni in cui sono presenti almeno un dispositivo per famiglia di elettrodomestici. Nell'abitazione Enea7 sono presenti tre televisori, ma per poter portare a termine il processo di disaggregazione del consumo e individuare l'andamento di un generico televisore all'interno di una abitazione è sufficiente che ci siano almeno i dati uno di questi tre televisori che possono essere considerati a tutti gli effetti come un'unica entità.

Figura 10: Distribuzione elettrodomestici Enea 7

Figura 11: Distribuzione elettrodomestici Enea 8

Come si evince dalle figure mostrate finora, le case con il maggior numero di dati sono Enea7 ed Enea8, le quali rappresentano quindi l'insieme dei dati che sono stati presi in considerazione, in quanto risultano essere più ricchi e completi. Tuttavia, anche in questo caso ci sono numerosi dati mancanti che sono stati però manipolati e gestiti tramite l'utilizzo di opportune tecniche statistiche, come descritto più dettagliatamente nel prosieguo del paragrafo.

Un altro problema riscontrato nei dati è stata la presenza di numerosi outliers, ovvero misurazioni troppo elevate rispetto all'effettivo consumo possibile dell'elettrodomestico considerato.

Questo tipo di situazioni devono essere necessariamente gestite nel campo del machine learning, in quanto possono compromettere il corretto funzionamento del modello, inducendolo a considerare veritieri questi dati e che verranno quindi riproposti in output dalla rete quando si dovrà ricostruire il segnale disaggregato. In ogni caso, a prescindere dal campo di applicazioni, tali situazioni devono essere gestite opportunamente al fine di evitare interpretazioni errate dei risultati.

In questo caso si è proceduto alla sostituzione di questi valori anomali considerando dei valori di soglia (*threshold)* per ogni singolo elettrodomestico; in particolare, questi valori di soglia sono stati fissati dopo aver valutato, sia tramite l'esperienza di tutti i giorni sia tramite ricerche mirate in letteratura, il massimo consumo di un determinato elettrodomestico. Prima di fornire il dettaglio dei valori di soglia utilizzati, risulta opportuno sottolineare il fatto che le istanze di televisore non sono state valutate singolarmente, ma sono state aggregate come se in ogni casa fosse presente un'unica TV con un consumo più elevato; questa scelta è stata dettata principalmente da due motivi: in primo luogo si ha una maggiore semplificazione nello studio ed il trattamento del dataset; in secondo luogo, ai fini del problema di disaggregazione, non è importante capire quale specifico televisore ha consumato una determinata parte dell'energia aggregata, ma riuscire a determinare la percentuale di consumo per la tipologia di elettrodomestico. Lo stesso ragionamento è stato utilizzato per le situazioni in cui si ha sia un contatore generale standard che un contatore derivante dal

fotovoltaico, perché in questo caso non siamo interessati a quale dei due supporti ci sta realmente offrendo la potenza necessaria, ma siamo interessati piuttosto ad un consumo aggregato complessivo.

A questo punto possiamo quindi mostrare i valori di threshold utilizzati per le case Enea7 ed Enea8:

1. Enea7:
    a. Consumo aggregato (Standard e fotovoltaico): 5000 Watt;
    b. Lavatrice: 2000 Watt;
    c. Televisori (3 TV aggregate): 900 Watt;
    d. Frigorifero: 300 Watt.
2. Enea8:
    a. Consumo aggregato (unico contatore): 3000 Watt;
    b. Lavatrice: 2000 Watt;
    c. Lavastoviglie: 2500 Watt;
    d. Frigorifero: 300 Watt.
    e.

Prima di esplicitare la tecnica utilizzata per la sostituzione di questi valori anomali, è opportuno mostrare graficamente, tramite dei *box plot*, gli outliers rinvenuti durante lo studio dei dati nella casa di Enea7. Si può notare in Figura 12 come i box plot di ogni elettrodomestico, ma anche del contatore, siano molto dilatati; questo significa che all'interno del dataset non solo sono presenti dei valori anomali che potrebbero compromettere, statisticamente, le performance del modello, ma anche che questi outliers si discostano notevolmente dalla media, talvolta in negativo come accade per il contatore generale in cui si è rilevata una misurazione istantanea di circa -2000000 Watt, oppure in positivo come invece accade per gli altri elettrodomestici, con registrazioni puntuali anomale pari a milioni di watt.

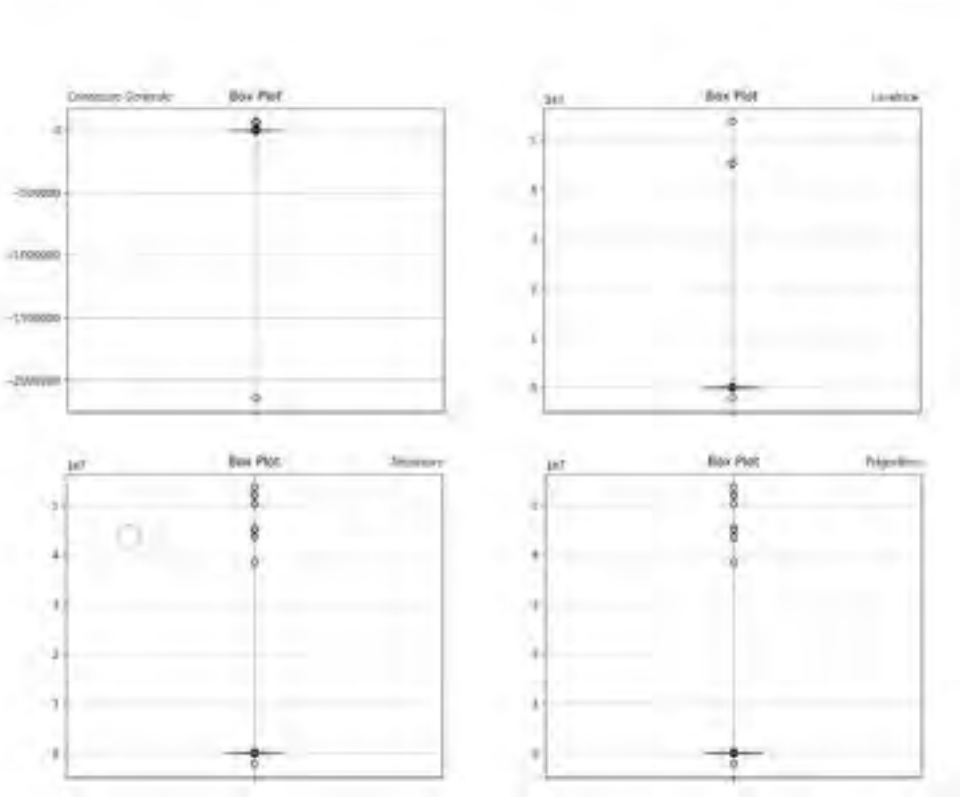

Figura 12: Box plot del contatore e degli elettrodomestici di Enea7

Notiamo che all'interno dei grafici la barra verde indica il valore medio per quella specifica distribuzione; ovviamente i valori della media non sono pari a zero come può sembrare dal grafico, ma semplicemente l'ampiezza della scala di riferimento non permette di apprezzare dettagliatamente valori decisamente minori delle centinaia di migliaia di Watt. I cerchi in nero presenti nel grafico rappresentano invece i valori anomali che sono stati individuati durante la generazione dei plot specifici.

A questo punto è possibile dettagliare la modalità di sostituzione dei valori anomali; in particolare è stato utilizzato il metodo dell'*interpolazione lineare* descritto dalla seguente formula:

$$f(x) = \frac{f(a) + f(b)}{2}$$

Questa formula ci dice che $f(x)$ è posto come la media tra i punti a e b. Allora una misurazione anomala nel dataset viene rimpiazzata con la media tra la misurazione precedente e la successiva, commettendo un errore relativamente basso in questo caso, in quanto essendo continue, in quanto i dati come vedremo sono stati ricampionati con una granularità fine al secondo, è poco probabile che due misurazioni prese in istanti di tempo molto ravvicinato si discostino molto tra loro.

Per essere più precisi, l'errore dovuto all'utilizzo dell'interpolazione lineare può essere descritto mediante la seguente formula:

$$| f(x) - g(x) | \leq C(b - a)^2$$

Nella formula precedente $g(x)$ rappresenta la funzione interpolante, mentre C è un termine costante che dipende fortemente dall'interpolazione prodotta (supponendo f due volte differenziabile) all'interno dell'intervallo $[a, b]$.

La formula ci dice che l'errore prodotto dall'interpolazione lineare è proporzionale al quadrato della distanza tra i punti di interesse. Questo significa che in molti casi l'interpolazione lineare risulta essere molto debole e troppo approssimativa; tuttavia, vista la forte densità di valori di cui è composta una serie temporale con granularità al secondo, questo errore risulta essere molto mitigato, permettendo una gestione efficace e semplice degli outliers.

Per dettagliare in modo pratico la quantità di valori anomali all'interno del dataset, è stato scritta un'apposita funzione in grado di contare tutti i valori fuori specifica (rispetto ai threshold impostati) e calcolarne una percentuale rispetto al totale.

Nella tabella seguente è possibile apprezzare in dettaglio, per Enea7, la percentuale di outliers rilevati in ogni elettrodomestico:

Contatore	Frigorifero	Lavatrice	Televisore
0.03%	0.03%	0.0011%	0.0011%

Tabella 3: Percentuale di outliers per ogni elettrodomestico

Dalla Tabella 3 è possibile notare come la presenza di valori anomali rispetto ai thresholds impostati risulti essere molto scarsa. Questo ci permette di considerare comunque attendibili i dati in nostro possesso e velocizza molto le operazioni di sostituzione.

Inoltre, un secondo importante problema è costituito, come già accennato in precedenza, dalla presenza di dati mancanti. Questi dati sono stati gestiti in due modi differenti, in quanto occorre distinguere tra dati mancanti relativi a singole giornate e dati mancanti relativi a giornate multiple consecutive, come settimane o più giorni.

I dati mancanti relativi a giornate singole all'interno del dataset sono stati sostituiti con la media puntuale, dove possibile, tra il giorno precedente ed il giorno successivo; per media puntuale si intende che ogni singolo secondo di campionamento, se mancante, viene rimpiazzato con il valore medio tra il campionamento nel giorno precedente e successivo al medesimo istante temporale, rispettando così le abitudini giornaliere delle utenze all'interno dell'abitazione.

Nel caso particolare in cui siano più giorni consecutivi ad essere mancanti, ad esempio una settimana, i dati vengono sostituiti con una media aritmetica tra i valori della settimana precedente e quella successiva, sempre in maniera puntuale.

Inoltre, in tutti quei rari casi in cui il contatore generale, seppur aggregato con il fotovoltaico, ha registrato un consumo nullo, abbiamo proceduto con la sostituzione di tali valori con la media rispetto l'interna serie temporale, in quanto il consumo totale non può mai essere nullo, soprattutto se nel frattempo gli altri elettrodomestici mostrano una potenza istantanea maggiore di zero.

In questo modo si riesce ad ottenere un dataset completo con una granularità molto fine (al secondo), pronto per essere fornito al modello di machine learning implementato. Inoltre, si ottiene una forte semplicità dal punto di vista della gestione grazie al formato strutturato scelto per la memorizzazione delle informazioni, permettendo, nella fase successiva, di concentrarsi esclusivamente sullo sviluppo e sull'addestramento del modello.

Successivamente, dopo aver determinato quali dati potessero essere utilizzati per l'addestramento del modello di machine learning e dopo aver pulito accuratamente gli stessi, è stato implementato un convertitore all'interno della libreria nilmtk, integrato con codici python ad-hoc con l'obiettivo di esportare l'intero dataset in un formato specifico strutturato, arricchito con i relativi metadati. La prima fase produce quindi in output un file con estensione ".h5", contenente tutti i dati raccolti dai contatori ed elettrodomestici di ogni casa, ai quali vengono associati tutti i metadati necessari al loro riconoscimento (tipo di contatore, casa di appartenenza, nome, ecc...). In questo modo viene generato un unico file strutturato partendo da tanti fogli di calcolo e/o di testo difficili da mantenere e da riutilizzare nel momento in cui il modello di machine learning deve essere addestrato e valutato.

Tuttavia, a monte di tutte queste operazioni, è stato necessario un intervento manuale, in quanti i metadati sono stati preparati all'interno di vari file yaml, ovvero un formato simile al json e che consente di strutturare le informazioni mediante una rappresentazione gerarchica, che viene poi mantenuta tale all'interno del file di output finale.
Al termine delle operazioni di conversione si ottiene quindi un file altamente strutturato, contenente sia i metadati che i dati ottenuti dal campionamento sul campo delle prese intelligenti, opportunamente manipolati. Il file prodotto può essere poi letto mediante delle procedure specifiche offerte dalla libreria niltmtk; in particolare, il convertitore mette a disposizione delle funzionalità che consentono all'utilizzatore di richiamare la serie temporale di uno specifico elettrodomestico all'interno di una determinata casa. In questo modo è possibile effettuare:

1. visualizzazione completa dei dati tramite la funzione *power series all data* oppure una versione downsampled degli stessi tramite una semplice funzione plot;
2. passaggio dei dati in modo corretto al modello di machine learning implementato.

In Figura 13 è presentato un esempio di serie temporale, su singolo giorno, relativo alla casa Enea7, a seguito della lettura del file ".h5" tramite le funzionalità di cui sopra; in questo caso, come detto in precedenza, il contatore generale di casa ed il contatore appartenente all'impianto fotovoltaico sono stati aggregati, in modo da visualizzare correttamente l'andamento dei vari dispositivi rispetto a consumo totale, tralasciando quale dei due impianti sta effettivamente erogando energia, essendo questa un'informazione poco rilevante ai fini della disaggregazione. I televisori sono stati mantenuti separati esclusivamente per fini grafici e per coglierne l'andamento, mentre al modello di machine learning implementato viene fornito il dato aggregato, come già discusso in precedenza.

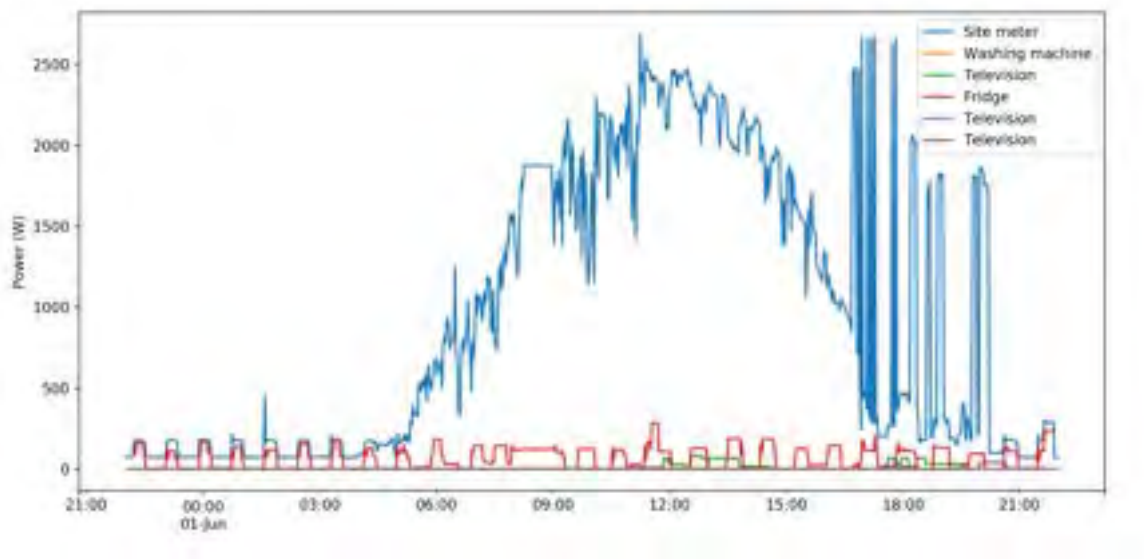

Figura 13: esempio di serie temporale Enea 7

Dopo aver completato la fase di conversione, il file può essere fornito in input ad una serie di script Python, con l'obiettivo di estrarne i dati strutturati e fornirli al modello implementato.
In primo luogo, occorre scegliere quali case e quali finestre temporali devono essere utilizzate per la fase di training, in modo da addestrare la rete su un insieme di dati sostanzioso e completo. I dati rimanenti vengono invece utilizzati per valutare il modello e costituiscono il test set, in modo da poter valutare la capacità di generalizzazione del modello.
In particolare, tramite delle funzionalità implementate all'interno della libreria nilmtk, le serie temporali vengono suddivise in porzioni più piccole dette *chunk*, le quali vengono fornite in input, sequenzialmente, alla rete neurale. In questo modo è possibile addestrare la rete senza dover caricare l'intera serie temporale in memoria centrale. Questi chunck non sono altro che finestre temporali estrapolate dalla serie, permettendo così un processamento di tipo *windowed*, cioè basato su finestre.
Occorre sottolineare che ogni elettrodomestico possiede una particolare lunghezza di queste finestre, la quale dipende dalla tipologia di lavoro che compie lo stesso. Ad esempio, il frigorifero è un elettrodomestico che opera periodicamente, cioè possiede dei cicli abbastanza regolari di lavoro come è possibile verificare in Figura 13. Questo significa che la dimensione della finestra deve permettere di intercettare le variazioni d'onda sia in salita che in discesa, in modo tale che la rete possa cogliere ed imparare il pattern corretto.

2.4 Implementazione dei modelli di Machine Learning

Entrambi i modelli di machine learning introdotti nei capitoli precedenti sono stati implementati in Python utilizzando la libreria Keras. La scelta di questa libreria è dettata in primo luogo dal fatto che permette la generazione di modelli, anche complessi, quasi senza sforzi, offrendo un insieme di classi e funzionalità che permettono un'implementazione rapida e modulare.

In particolare, Keras offre una grande quantità di strati già implementati e che è possibile connettere tra loro per costruire il modello desiderato. Questo è un grande vantaggio perché permette al programmatore di concentrarsi esclusivamente sulla scelta del modello più appropriato da implementare, astraendo le difficoltà legate alla struttura di ogni singolo layer.

Inoltre, Keras offre delle funzionalità fondamentali per eseguire l'addestramento ed il testing del modello, organizzando e gestendo automaticamente la suddivisione in batch e in epoche, informando il programmatore sull'andamento dell'addestramento tramite aggiornamenti real-time sulla funzione di errore stabilita in fase di modellazione.

2.4.1 Implementazione Long Short-Term Memory

Il primo modello che è stato implementato è stata la rete neurale ricorrente LSTM. Come già descritto in precedenza, questa rete ricorrente si basa sull'utilizzo di un particolare tipo di cella detta appunto LSTM, che mantiene in memoria un determinato stato raggiunto fino a quel preciso istante.

Keras offre una classe omonima che si occupa proprio di istanziare queste celle LSTM, inizializzandole in modo completamente automatico. Nel caso specifico le connessioni tra i neuroni della rete sono state modellate come bidirezionali, permettendo ad ogni istante l'influenza sia di dati passati che di dati futuri; in questo modo la stima prodotta dalla rete risulta essere certamente più accurata, ma si ha un significativo aumento dei tempi computazionali di addestramento.

In Figura 14 è possibile apprezzare l'architettura di una rete neurale con celle LSTM e connessioni bidirezionali.

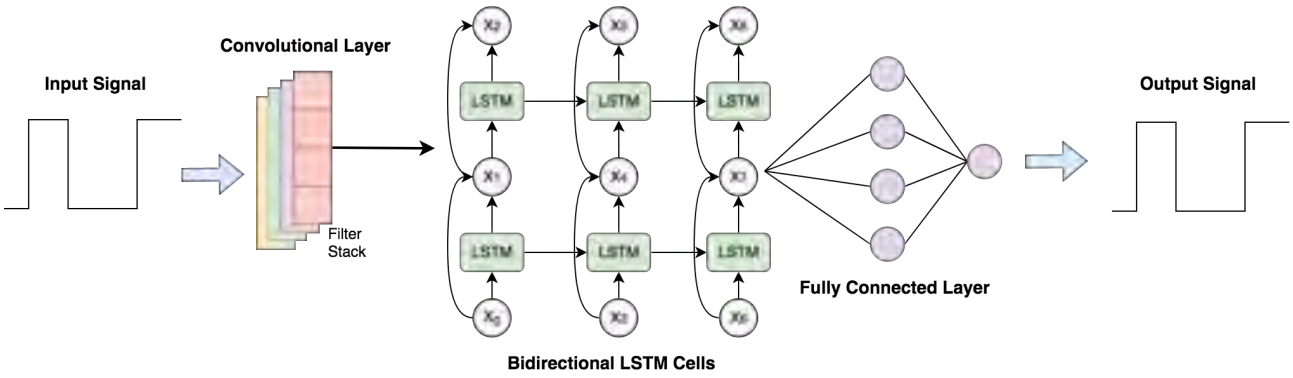

Figura 14: Rete realizzata con celle LSTM bidirezionali

La rete implementata è mostrata di seguito:

1. Strato di Input;
2. 1D conv (filter size=4, strides=1, number of filters=16, activation function=linear, padding=same)
3. Bidirectional LSTM (N=128, with peepholes)
4. Bidirectional LSTM (N=256, with peepholes)
5. Fully Connected (N=128, activation function = TanH)
6. Fully Connected (N=1, activation function = linear)

Vediamo più in dettaglio il funzionamento di ogni singolo layer della rete.

Ad ogni istante temporale, la rete riceve in input uno specifico dato di segnale aggregato e restituisce in output un dato disaggregato, relativo all'elettrodomestico scelto per l'addestramento. Lo strato di input ha proprio il compito di ricevere in ingresso i dati e di inoltrarli verso gli strati più interni della rete.
Lo strato convoluzionale è detto 1D in quanto lavora ricevendo in ingresso un vettore di elementi, ai quali applica la convoluzione. Lo strato qui implementato produce in uscita sedici vettori della medesima dimensione, ad ognuno dei quali viene applicato uno specifico filtro di dimensione pari a quattro e stride pari

a uno; questo significa che viene spostata una finestra di quattro elementi all'interno del vettore, applicando a questi ultimi i filtri accennati in precedenza. Successivamente, a questi elementi viene applicata la funzione di attivazione, che in questo caso è una funzione lineare del tipo $f(x) = x$.
L'utilizzo dello strato convoluzionale permette al modello di estrarre delle features nascoste all'interno dei dati e che il modello utilizza per riconoscere pattern ricorrenti.
Successivamente, l'uscita dello strato convoluzionale viene processata dal primo strato bidirezionale LSTM; questo layer è stato implementato utilizzando la maggior parte dei parametri di default, tra cui:

- Funzione di attivazione: Tangente Iperbolica
- Attivazione ricorrente: Hard Sigmoid (Logistic)

L'utilizzo delle connessioni peepholes permette di ottenere una stima più accurata del timing degli eventi e quindi nel nostro caso delle attivazioni di un determinato elettrodomestico. In questo modo la rete risulta essere più precisa nello stimare il momento preciso in cui si ottiene l'attivazione del dispositivo.
Infine la rete è costituita da uno strato denso, totalmente connesso, costituito da 128 neuroni e con funzione di attivazione a tangente iperbolica, e da uno strato denso costituito da un solo neurone in quanto rappresenta la singola uscita della rete. In questi due ultimi strati avvengono effettivamente la stima del segnale disaggregato e la presentazione del risultato per mezzo del layer di uscita con attivazione lineare.

Come detto in precedenza, prima di procedere con l'addestramento sui dati reali di Enea, la rete è stata valutata sul dataset UKDALE per verificarne la corretta implementazione e fornire una metrica di confronto. In Figura 15 viene mostrato un esempio di addestramento e valutazione su una porzione del test set (quindi su dati non utilizzati per l'addestramento).

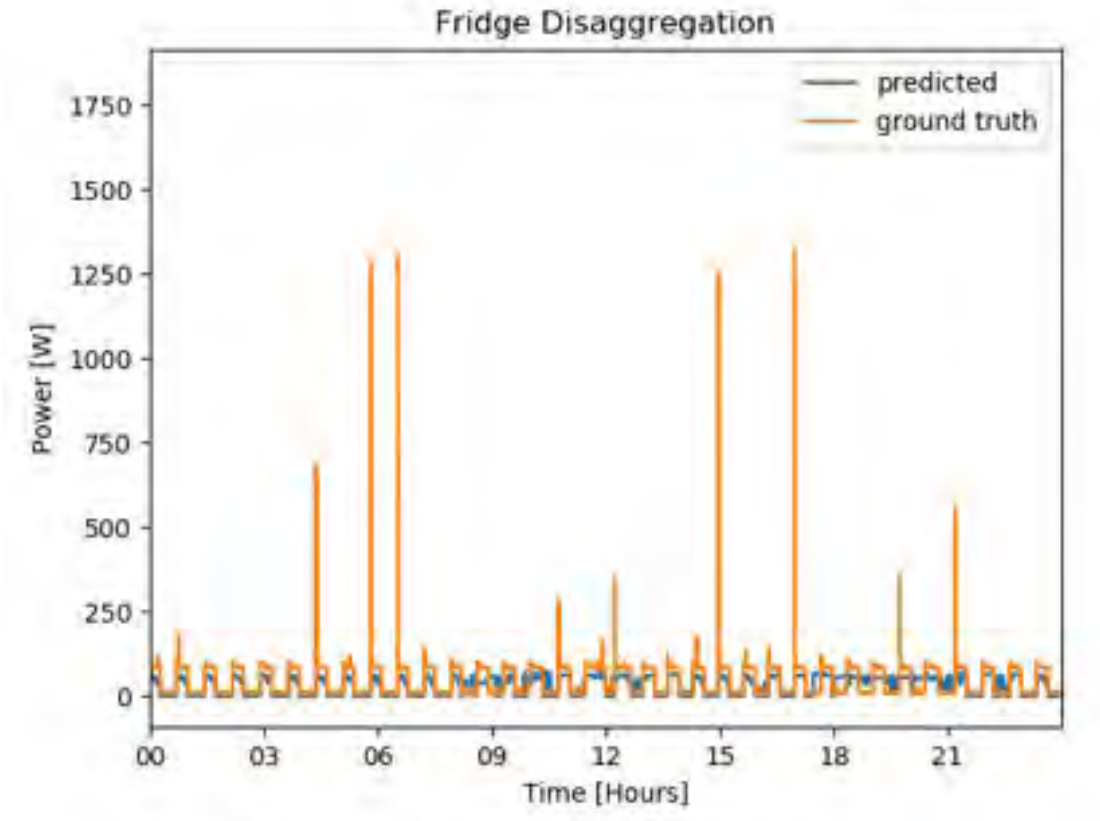

Figura 15: esempio di predizione del modello su un test set

In particolare, si sono utilizzati circa 3 anni di dati appartenenti alla casa 1 di UKDALE per addestrare la rete, più precisamente dati dal 2013-04-12 al 2015-07-01, mentre l'intera casa 2 è stata utilizzata per valutare la bontà del modello.
Nella figura precedente l'elettrodomestico utilizzato è stato il frigorifero, sul quale si è ottenuto un MAE pari a 28 sui dati del test set.
Per apprezzare meglio il confronto grafico tra la ground truth e la predizione effettuata dalla rete, si è effettuato uno zoom su una singola giornata della predizione; come si può osservare dal grafico in Figura 15, il segnale stimato dal modello segue fedelmente l'originale, ad eccezione di alcuni picchi irregolari, i quali molto probabilmente sono dovuti ad aperture improvvise del frigorifero e pertanto poco prevedibili.
In Figura 16 è possibile invece, osservare il medesimo addestramento su scala oraria, prendendo come riferimento il periodo tra la mezzanotte e le tre del mattino. In questo modo è possibile visualizzare in maniera ottimale come la rete riesce ad individuare correttamente le periodicità nel consumo del frigorifero.

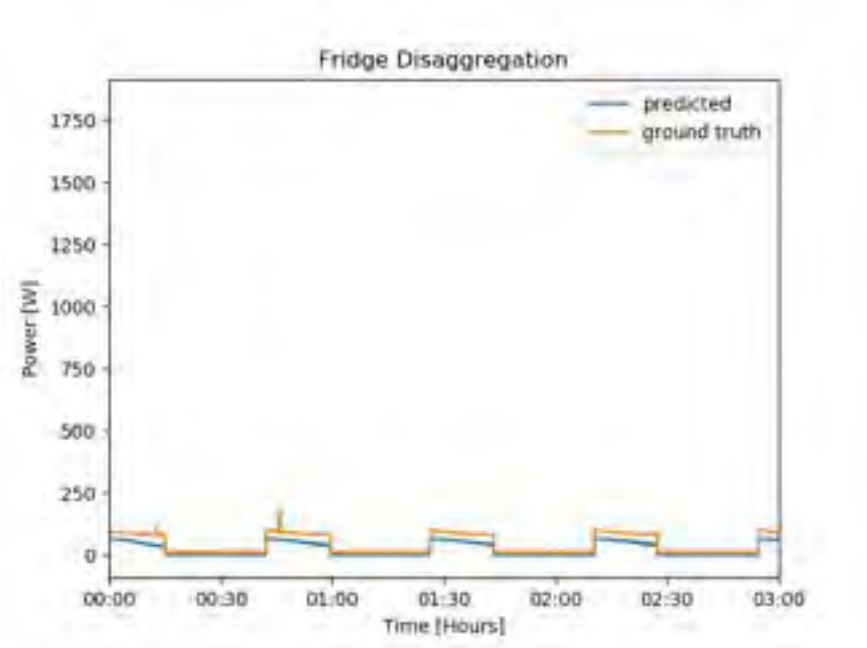

Figura 16: zoom della predizione su scala oraria

Come possiamo notare, il modello riesce a stimare con precisione l'istante preciso di attivazione del frigorifero (in questo caso l'attivazione può essere la salita di un fronte d'onda o la discesa dello stesso). L'unica mancanza di precisione assoluta la possiamo osservare nella scala del segnale stimato, in quanto, seppur di poco, non riesce ad effettuare un fit completo rispetto al segnale originale.
Come vedremo in seguito, questo problema è stato risolto agilmente, migliorando i risultati, tramite l'utilizzo del Denoising Auto Encoder, il quale non solo riesce ad individuare con precisione le attivazioni, ma effettua una stima accurata anche del fattore di scala.

Notiamo che il nostro modello ottiene un MAE di 28 sul frigorifero e 44 sulla lavatrice entrambi migliori di quelli ottenuti nel lavoro di riferimento [5] che sono riportati in Tabella 4.

LSTM	MAE
FRIGO	36
LAVATRICE	109

Tabella 4: MAE ottenuto sul dataset UKDALE in letteratura con LSTM

Il modello è stato addestrato esclusivamente sugli elettrodomestici sopra citati, in quanto rappresentano l'insieme di dispositivi in comune con i dati di Enea, permettendo, tra l'altro, l'utilizzo di questi modelli pre-addestrati su questi ultimi.

Ad ogni modo, alla luce dei risultati ottenuti possiamo affermare che la rete neurale ricorrente LSTM può essere applicata al problema di disaggregazione dell'energia sulle case di Enea, in quanto riesce a fornire una buona stima del segnale disaggregato e ci permette quindi di inferire una corretta progettazione ed implementazione del modello descritto in precedenza.
Tuttavia, seppur la rete LSTM risulta essere molto performante, almeno sui dati della letteratura, occorre far notare che possiede tempi di addestramento molto lunghi. Infatti, per addestrare il modello su tre anni di dati ed utilizzando una GPU (GeForce GTX 1050 Ti) occorrono circa quindici ore di training considerando 5 epoche, alle quali deve essere ovviamente aggiunto il tempo necessario alla disaggregazione sui dati di test; queste tempistiche sono causate principalmente dalla presenza degli stati bidirezionali con connessioni peephole, che da un lato migliorano le performance della rete, ma dall'altro aumentano drammaticamente i tempi di processamento.
Questo significa però che molto probabilmente la rete LSTM possiede un margine di miglioramento che è possibile scoprire effettuando un numero più ampio di test.

Inoltre, durante l'esecuzione di molteplici sessioni di addestramento, si è osservato che effettuando il test su case simili a quelle utilizzate per il training si ottiene un incremento notevole delle prestazioni del modello.
A questo proposito può risultare vantaggioso eseguire delle operazioni di clustering preliminari, in modo tale da suddividere le case in base alle loro similarità e applicare quindi sul test set il modello addestrato più appropriato, ottenendo un incremento significativo delle prestazioni.
Uno degli algoritmi più utilizzati per il clustering è il *K-means*; la scelta delle caratteristiche da valutare per esprimere una similarità, possono essere ad esempio il consumo medio mensile o annuo, il numero di utenze all'interno della casa, il numero o la tipologia di elettrodomestici.

2.4.2 Implementazione Denoising Auto Encoder

Il secondo modello che è stato preso in considerazione e che poi è stato implementato è il Denoising Auto-Encoder, che come descritto nei capitoli precedenti rappresenta un'istanza particolare di Auto-Encoder, ma con la peculiarità di aggiungere rumore ai dati di input, quindi "sporcarli", in modo da prevenire perdite di complessità del modello. Il rumore può essere aggiunto sia tramite operazioni di dropout, cioè non prendendo in considerazione l'effetto di alcuni neuroni, con una certa probabilità, oppure aggiungendo un rumore Gaussiano estratto da una distribuzione normale.
I due metodi considerati per l'aggiunta del rumore operano diversamente, ma ai fini pratici sono equivalenti; tuttavia, la nostra scelta è ricaduta sull'utilizzo di strati di dropout in quanto più semplici ed immediati sia da implementare che da interpretare.
Inoltre, il layer di dropout è molto utilizzato in letteratura anche per ridurre la probabilità di overfitting.
L'architettura mostrata in Figura 17 si riferisce proprio al particolare modello in cui il rumore viene immesso mediante operazioni di dropout e quindi tramite la perdita di alcuni input; i dati che invece riescono ad avanzare verso il centro della rete, vengono processati da vari strati totalmente connessi, con l'obiettivo di estrarre le features dominanti. Infine, l'output degli strati totalmente connessi vengono inoltrati ad uno strato convoluzionale con attivazione lineare, che fornisce il segnale ricostruito.

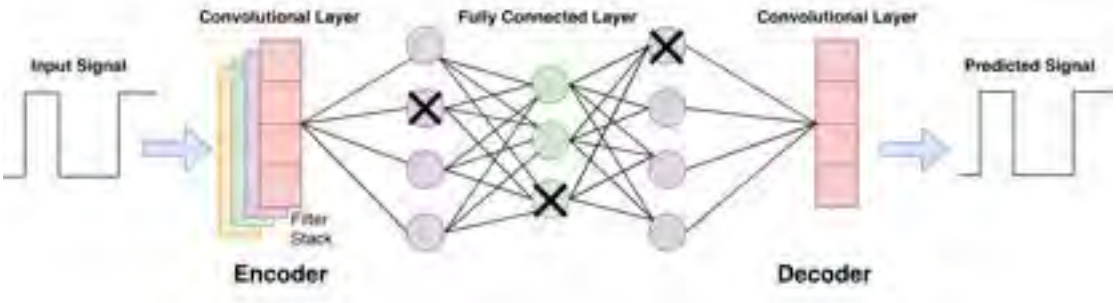

Figura 17: Rete realizzata con auto encoder

Esaminiamo ora in modo più dettagliato come è stato implementato il Denoising Auto-Encoder, specificando il funzionamento e l'utilità di ogni layer di cui è composto:

1. Strato di Input;
2. 1D conv (filter size=4, strides=1, number of filters=8, activation function=linear, padding=same);
3. Dropout(drop probability=0.1);
4. Fully Connected (N=size sequence dependent , activation function = ReLU);
5. Dropout(drop probability=0.1);
6. Fully Connected (N=128 , activation function = ReLU);
7. Dropout(drop probability=0.1);
8. Fully Connected (N=size sequence dependent , activation function = ReLU);
9. Dropout(drop probability=0.1);
10. Reshape;
11. 1D conv (filter size=4, strides=1, number of filters=1, activation function=linear, padding=same);

Come possiamo notare, gli strati utilizzati sono molteplici e ben differenti, eccetto alcuni casi, rispetto al modello ricorrente, proprio a causa della diversa natura. Il primo layer è uno strato convoluzionale, che riceve in ingresso un vettore di misurazioni puntuali e cerca di estrarre da queste delle features nascoste, applicando otto filtri differenti, ai quali viene poi applicata una funzione di attivazione lineare, proprio come avveniva per la LSTM. Successivamente viene applicato il primo layer di dropout con una probabilità di drop pari a 0.1; questo significa che l'output dello strato convoluzionale verrà ridotto del 10%.

A questo punto viene applicato il primo strato totalmente connesso, in cui comincia ad avvenire la compressione del segnale di ingresso, trasformandolo effettivamente in una forma più compatta. In questo caso il numero di neuroni dipende fortemente dalla dimensione della finestra scelta per eseguire l'addestramento; nello specifico viene impostato a $sequence * 8$, cioè per ogni elemento della finestra vengono stanziati otto neuroni, in modo tale da cercare di adattare la complessità del modello in base alla finestra scelta e quindi in base all'elettrodomestico che si sta considerando. La funzione di attivazione utilizzata, per tutti gli strati totalmente connessi, è la *Rectified Linear Unit*, disegnata in Figura 18, detta anche funzione rampa in quanto porta a zero tutti gli ingressi negativi e opera come una funzione lineare per tutti quelli positivi. Notiamo che questa funzione risulta non derivabile in un solo punto, cioè in $x = 0$, ma per convenzione (Keras in automatico), viene posto a zero come i valori negativi.

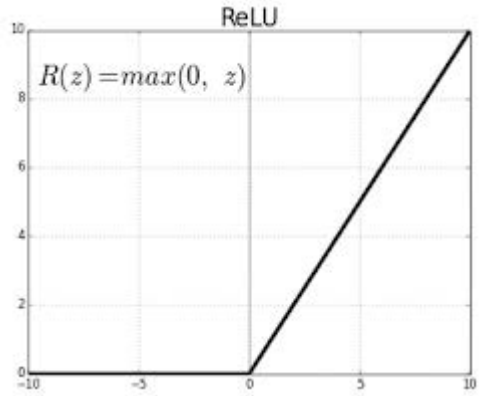

Figura 18: funzione Rectified Linear Unit

Successivamente, il segnale attraversa un ulteriore strato di dropout, dove viene perso un altro 10% di input, e viene inoltrato al layer totalmente connesso intermedio, costituito da 128 neuroni con funzione di attivazione ReLU. A questo punto il segnale è stato codificato in una forma più compatta e di fatto termina il compito dell'encoder. Da questo punto in poi il segnale viene processato dal decoder, il cui compito è quello di ricostruire il segnale disaggregato. Il decoder è speculare rispetto all'encoder ed esegue a ritroso le medesime operazioni; tuttavia risulta necessario motivare l'utilizzo di uno strato di Reshape.

Lo strato di Reshape ha il compito di cambiare la dimensionalità dell'input, lasciando comunque invariato il numero di elementi di cui è composto. Consideriamo, per esempio, il vettore $x = [1,2,3,4]$, tramite l'applicazione del Reshape possiamo trasformare questo vettore in una matrice $2X2$. Questa operazione viene applicata per invertire l'utilizzo di uno strato *flatten* (che schiaccia tutti gli elementi di una matrice all'interno di un vettore) e riportare il segnale nella stessa dimensione di partenza.

Come nel caso della LSTM, l'addestramento è stato eseguito utilizzando circa 3 anni di dati appartenenti alla casa 1 di UKDALE, mentre la casa 2 è stata utilizzata per valutare le prestazioni del modello. Anche l'elettrodomestico utilizzato è il medesimo; in questo modo si è replicato l'ambiente per ottenere un confronto chiaro tra i due modelli.
Analizzando i risultati ottenuti dall'addestramento del denoising autoencoder, possiamo affermare che questo modello fornisce delle prestazioni migliori in termini di errore (MAE); infatti, l'errore ottenuto sul frigorifero è di circa 18, mentre nel caso della LSTM abbiamo ottenuto un MAE pari a 28. Possiamo osservare questo forte miglioramento, anche graficamente in Figura 19.

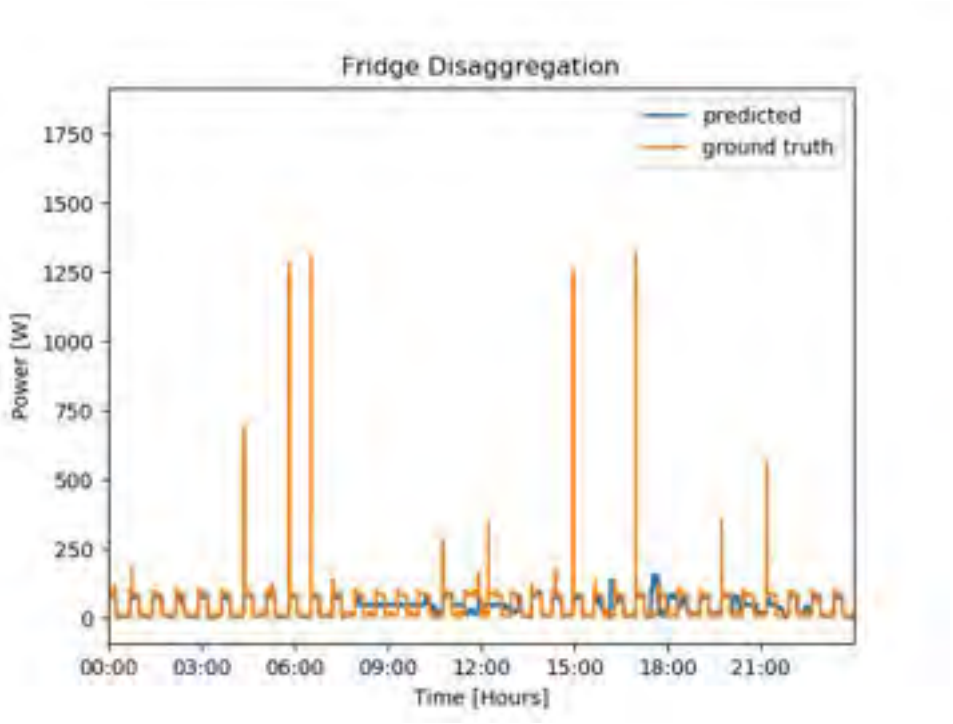

Figura 19: predizione del consumo del frigo ottenuta sul test set con la rete in Figura 17

Possiamo infatti notare come il denoising autoencoder riesca a fornire una stima molto più accurata rispetto alla rete LSTM, riuscendo a riprodurre in uscita un segnale che assomiglia molto all'originale anche per quanto riguarda i fattori di scala e non solo la forma d'onda.
Anche nel caso della lavatrice, riportato in figura 20, il denoising auto encoder riesce a fornire una buona stima del segnale originale, anche se con maggiori difficoltà, in quanto il frigorifero risulta comunque avere dei pattern regolari, più semplici da memorizzare. Di fatto, con la lavatrice si ottiene un MAE molto buono pari a 16.

Notiamo che i risultati ottenuti con questa rete sono migliori di quelli presentati nell'articolo di riferimento [5] che sono riportati in Tabella 5.

DAE	MAE
FRIGO	26
LAVATRICE	24

Tabella 5: MAE ottenuto sul dataset UKDALE in letteratura con il DAE

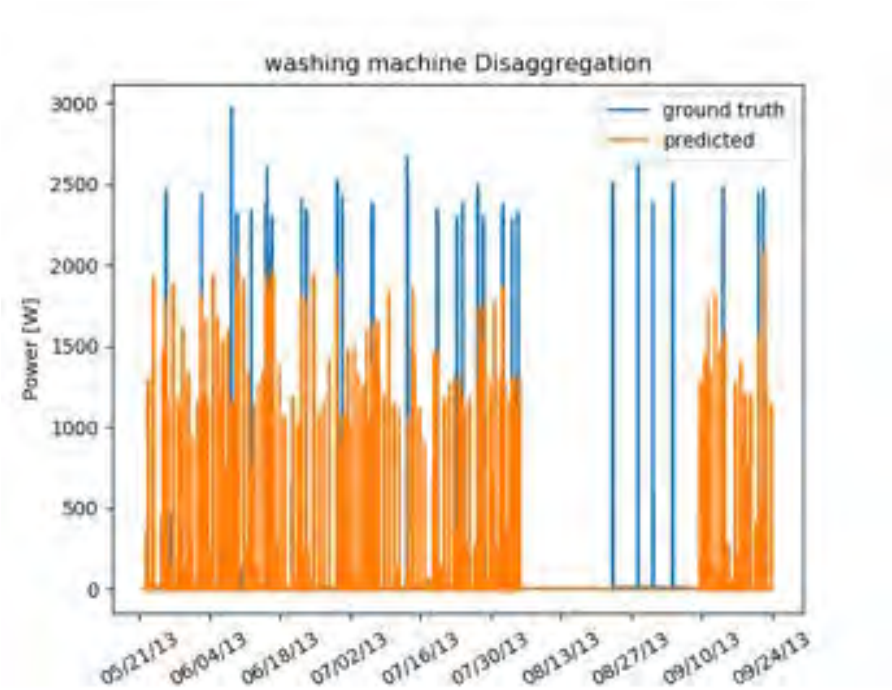

Figura 20: predizione del consumo della lavatrice ottenuta sul test set con la rete in Figura 17

Concludendo entrambi i modelli ottengono degli ottimi risultati sul dataset UKDALE, in entrambi i casi migliori di quelli presentati in letteratura. Tra i due, il modello migliore risulta essere quello basato sui DAE, in quanto ottiene il MAE più basso peraltro con tempi di addestramento molto più bassi.

2.5 Risultati sulle case ENEA

Una volta validata la bontà del modello scelto si è passato a considerare i dati delle case ENEA 7 e ENEA 8, in quanto erano le uniche case ad avere una mole di dati consistenti e coerenti.
In prima battuta si è tentato di replicare l'addestramento con le reti LSTM e denoising auto encoder descritte in precedenza ottenendo però dei risultati molto scarsi in quanto i dati del dataset UKDALE coprivano un arco temporale di 3 anni, mentre per le due case ENEA 7 e ENEA 8 si hanno solo circa 8 mesi di dati completi, insufficienti ad addestrare delle reti così complesse.
Inoltre, occorre tenere conto dell'intero processo di pulizia dei dati, eseguito preliminarmente all'addestramento e alla valutazione del modello, in quanto, oltre ad essere di scarsa quantità, i dati erano affetti da outliers improvvisi e buchi temporali, che come visto in precedenza, hanno obbligato l'utilizzo di diverse metodologie al fine di renderli più utilizzabili.
In questo modo però, se da un lato siamo riusciti a rimediare alla forte mancanza di dati, dall'altro abbiamo introdotti valori buoni da un punto di vista statistico, ma che molto probabilmente non rispecchiano perfettamente la realtà delle misurazioni. Questo comporta stime meno accurate da parte del modello, il quale potrebbe ottenere prestazioni sicuramente migliori operando con dati reali e completi.
In alcuni casi i dati mancanti su una giornata sono maggiori del 90%, in quanto in alcuni file relativi alla giornata sono presenti solo decine di misurazione quando normalmente una giornata di carico è composta da migliaia di questi ultimi. Come detto precedentemente, queste problematiche sono state affrontate da un punto di vista statistico, tuttavia limitante per eseguire un addestramento accurato di un modello di Machine Learning, in particolare nel campo delle reti neurali che richiedono una grande quantità di informazioni.

Di seguito possiamo osservare alcune delle predizioni che sono state restituite addestrando i modelli direttamente sui dati forniti, anziché sul dataset di letteratura. In particolare, sono stati utilizzati sette mesi di dati relativi al frigorifero di Enea 7 come training set e l'ultimo mese come test set.

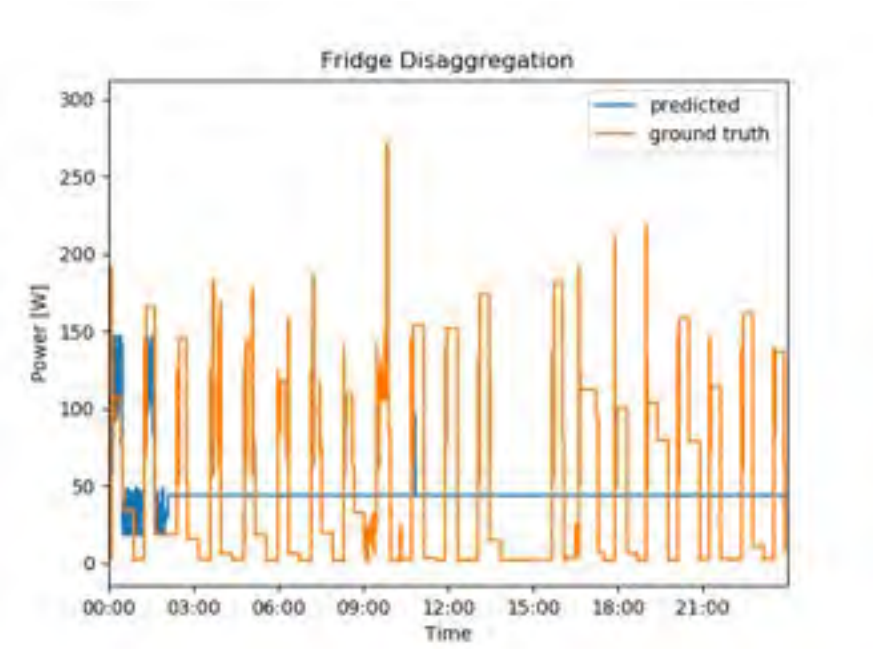

Figura 21: Predizione del consumo del frigo della casa Enea7 con modello LSTM su una giornata del mese di test

Come possiamo notare dalla figura 21, in cui è riportata la predizione su una giornata del mese di test, la rete, in alcuni istanti, cerca di stimare l'andamento del segnale disaggregato, ma per la maggior parte del tempo fornisce una stima che coincide con il valor medio della distribuzione. Questo andamento si ottiene quando la rete non riesce a minimizzare la funzione di errore se non facendo tendere la predizione alla media e questa è una situazione tipica che si verifica nel momento in cui la rete viene addestrata con pochi dati e poco puliti.

Come possiamo invece osservare dalla figura 22, la rete basata su modello DAE, che funzionava meglio sui dati di letteratura, non riesce nemmeno a stimare correttamente una parte dell'andamento del segnale, ma si limita a fornire in output il valore medio della distribuzione.

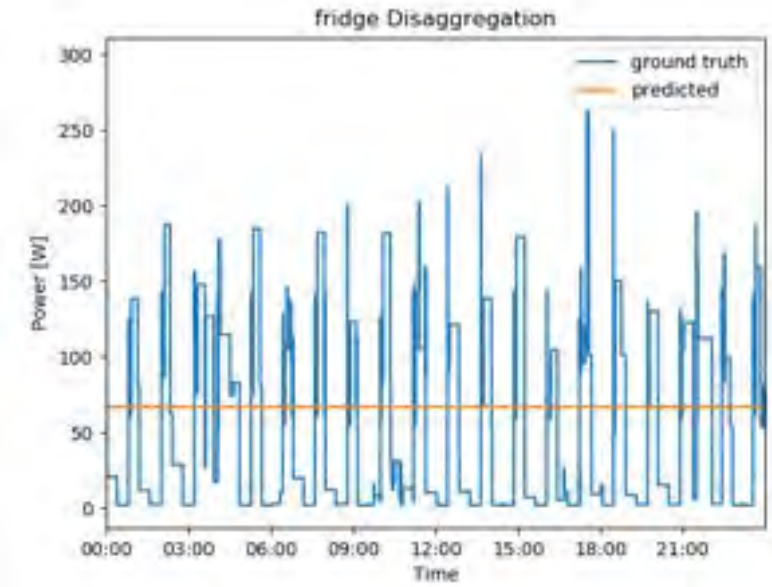

Figura 22:Predizione del consumo del frigo della casa Enea7 con modello DAE su una giornata del mese di test

In entrambi i casi, il modello implementato è troppo complesso rispetto alla piccola quantità di dati disponibile, rendendo la predizione poco accurata.

Tuttavia, il modello del denoising autoencoder addestrato sul dataset UKDALE si è rivelato utile alla previsione della disaggregazione dei consumi elettrici delle case Enea7 ed Enea8.
Come esempio, riportiamo nelle figure 23-26 la predizione ottenuta su frigorifero e lavatrice di Enea7 e Enea8 utilizzando il modello addestrato sulla casa 1 del dataset UKDALE.

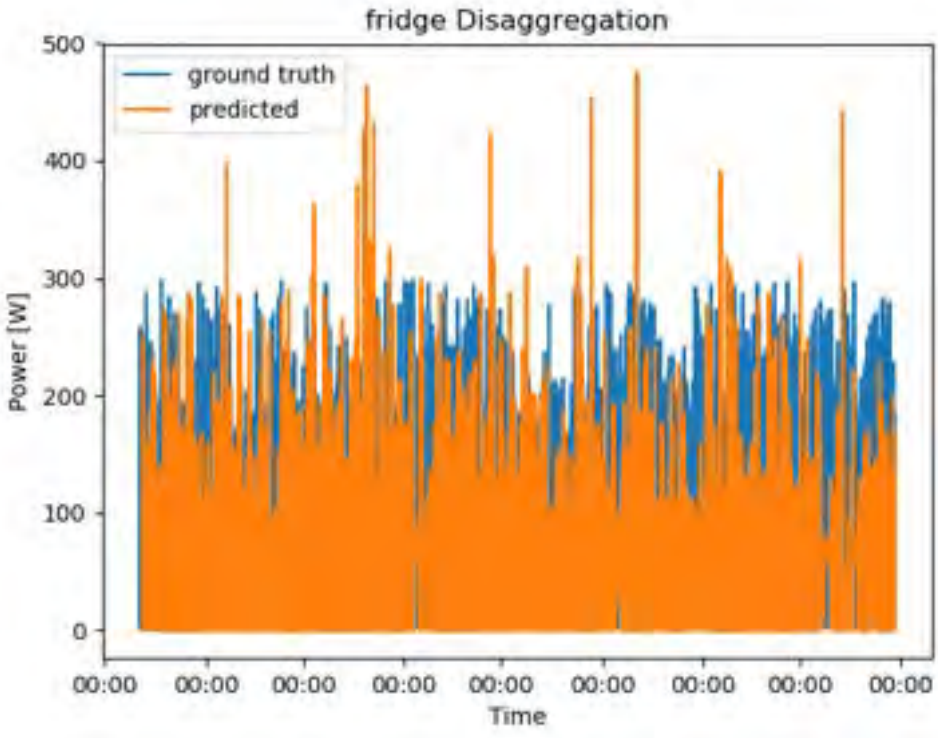

Figura 23: predizione ottenuta su 231 giorni di test del frigorifero di ENEA 7

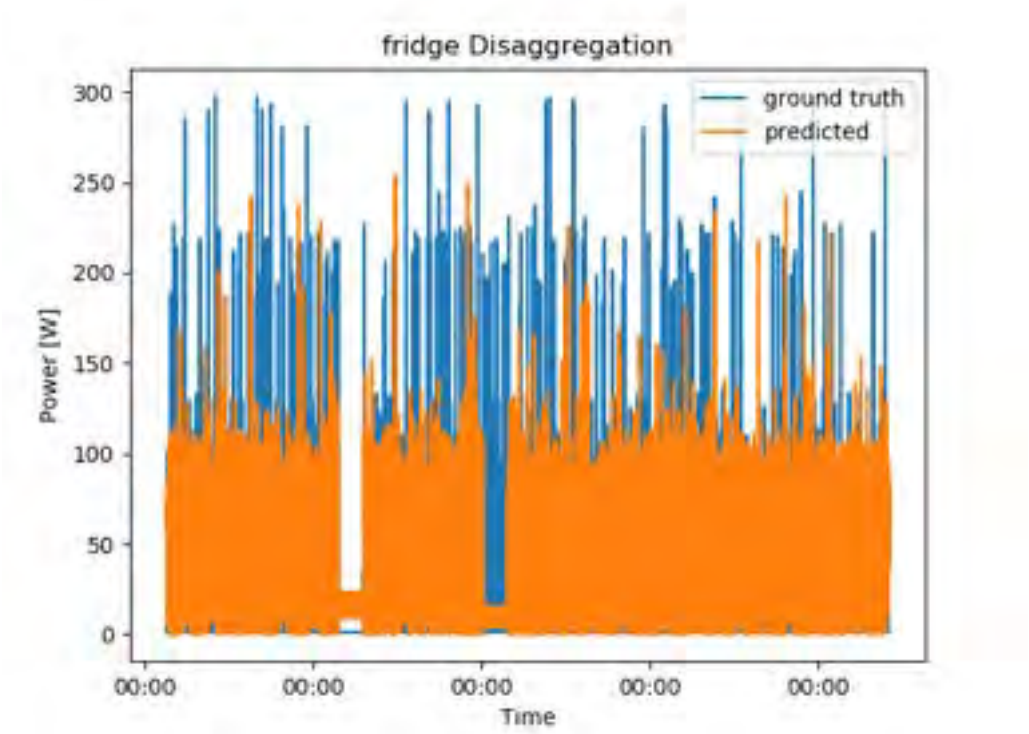

Figura 24: predizione ottenuta su 251 giorni di test del frigorifero di ENEA 8

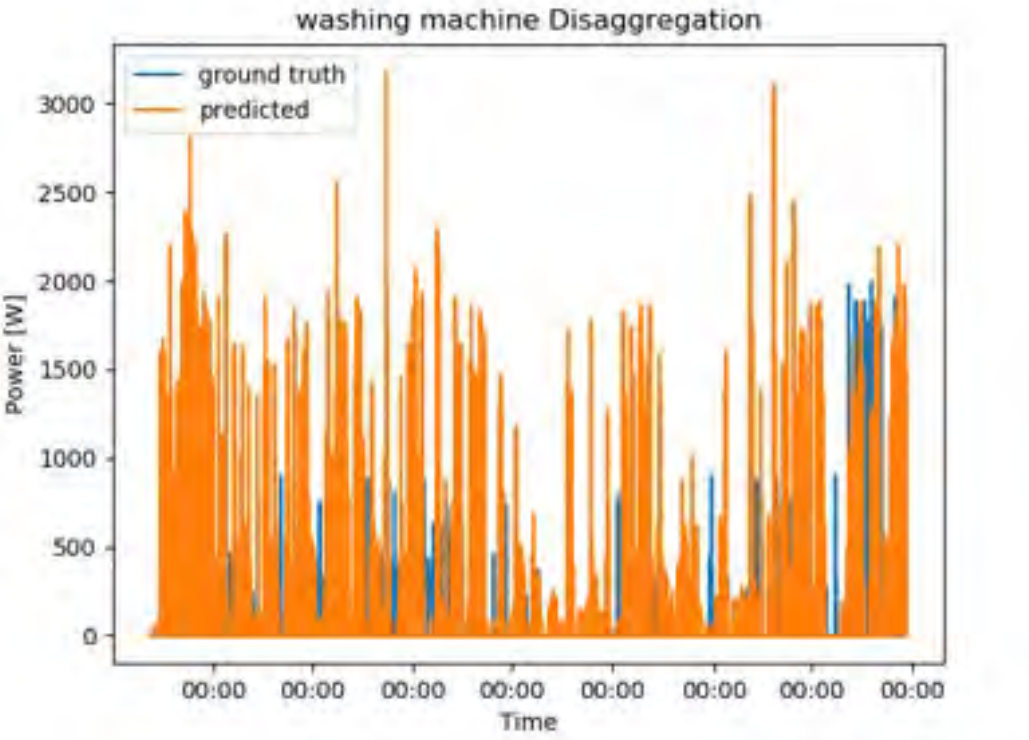

Figura 25: predizione ottenuta su 231 giorni di test della lavatrice di ENEA 7

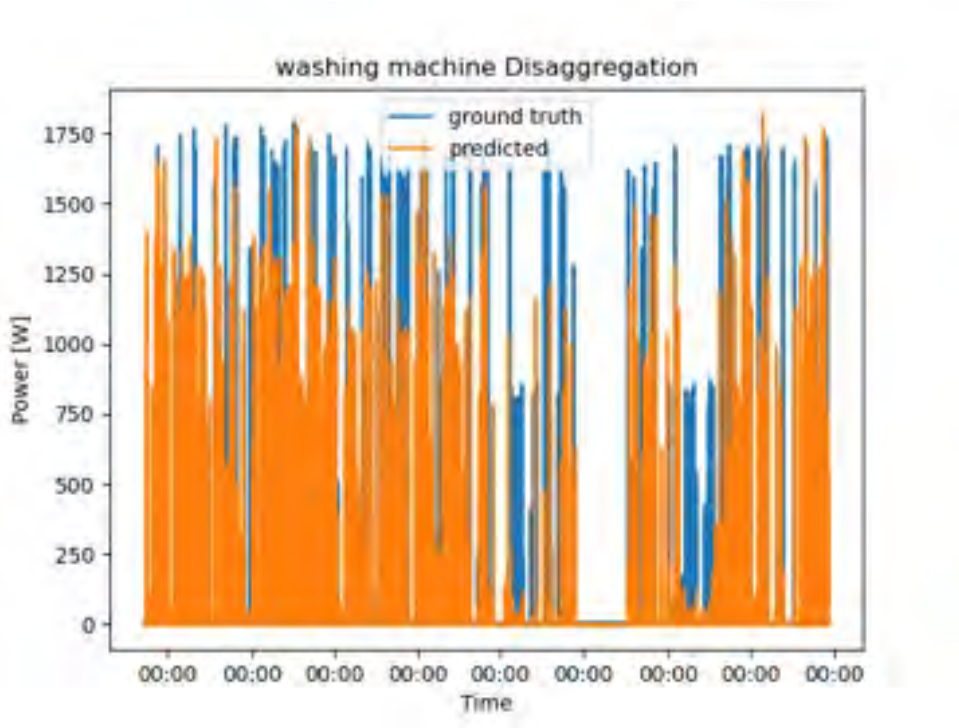

Figura 26: predizione ottenuta su 251 giorni di test della lavatrice di ENEA 8

In Tabella 6 è riportato il valore del MAE ottenuto sui due elettrodomestici nelle due case considerando come test set l'intero dataset a disposizione.

DAE	ENEA 7	ENEA 8
FRIGO	52,467	39,0329
LAVATRICE	24,804	14,9134

Tabella 6: MAE ottenuto sugli otto mesi di test delle case ENEA 7 e ENEA 8

Come possiamo notare dal confronto della tabella 6 con le tabelle 4 e 5, i risultati risultano essere comparabili ed appartenenti tutti al medesimo ordine di grandezza. In tutte le tabelle e in particolare nella tabella 4, possiamo però osservare come il MAE relativo alla lavatrice sia molto elevato, ad indicare che comunque risulta essere un elettrodomestico difficile da stimare, contrariamente a quanto accade invece con il frigorifero.

Cosa ancora più interessante è il risultato sul dato aggregato sulla singola giornata. Infatti, l'obiettivo del lavoro è quello di valutare la percentuale di consumo del singolo elettrodomestico per dare consigli al consumatore, e non si ha quindi interesse al dettaglio al secondo della previsione.
Riportiamo nelle Figura 27-30 l'andamento su tutto l'arco temporale a disposizione (231 giorni per ENEA 7 e 251 giorni per ENEA 8) della percentuale reale e quella prevista dal nostro modello sulle due case e sui due elettrodomestici.

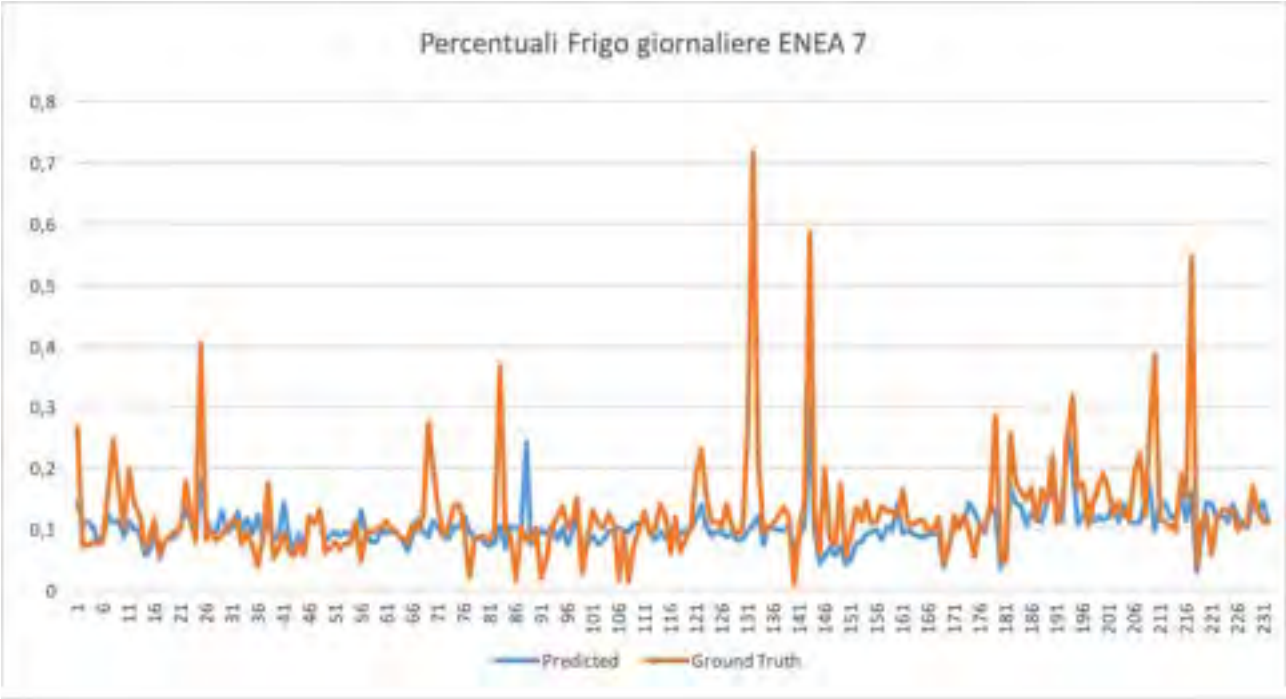

Figura 27: percentuali giornalieri previste e reali per il frigorifero di ENEA 7

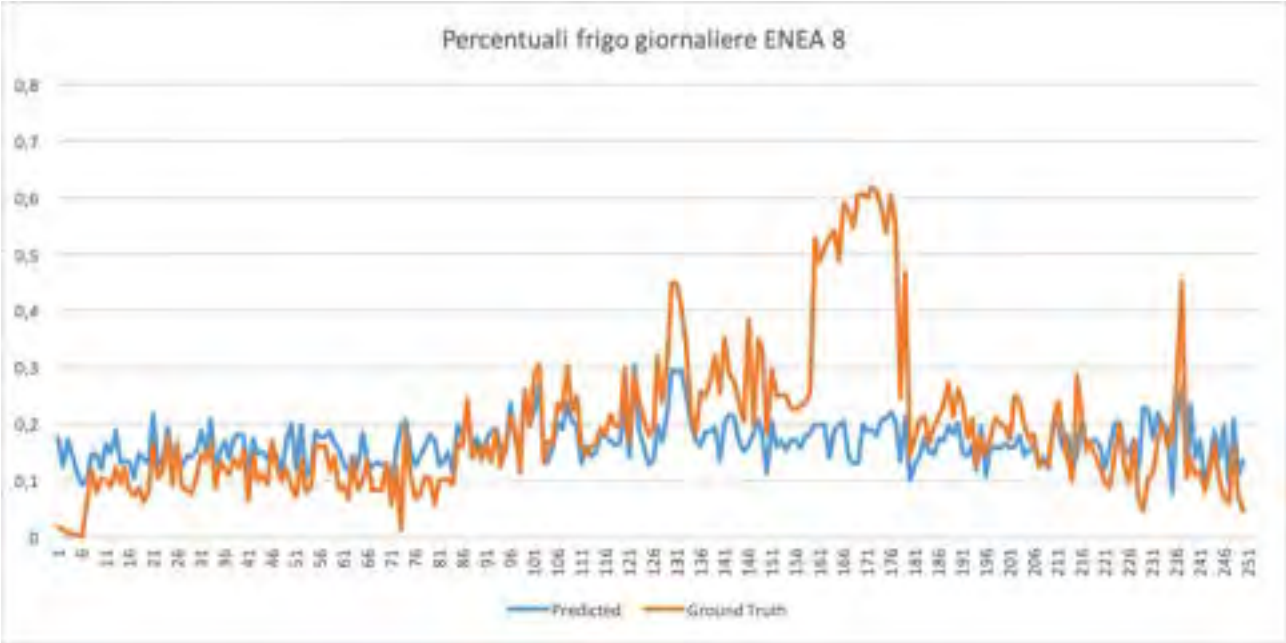

Figura 28: percentuali giornalieri previste e reali per il frigorifero di ENEA 8

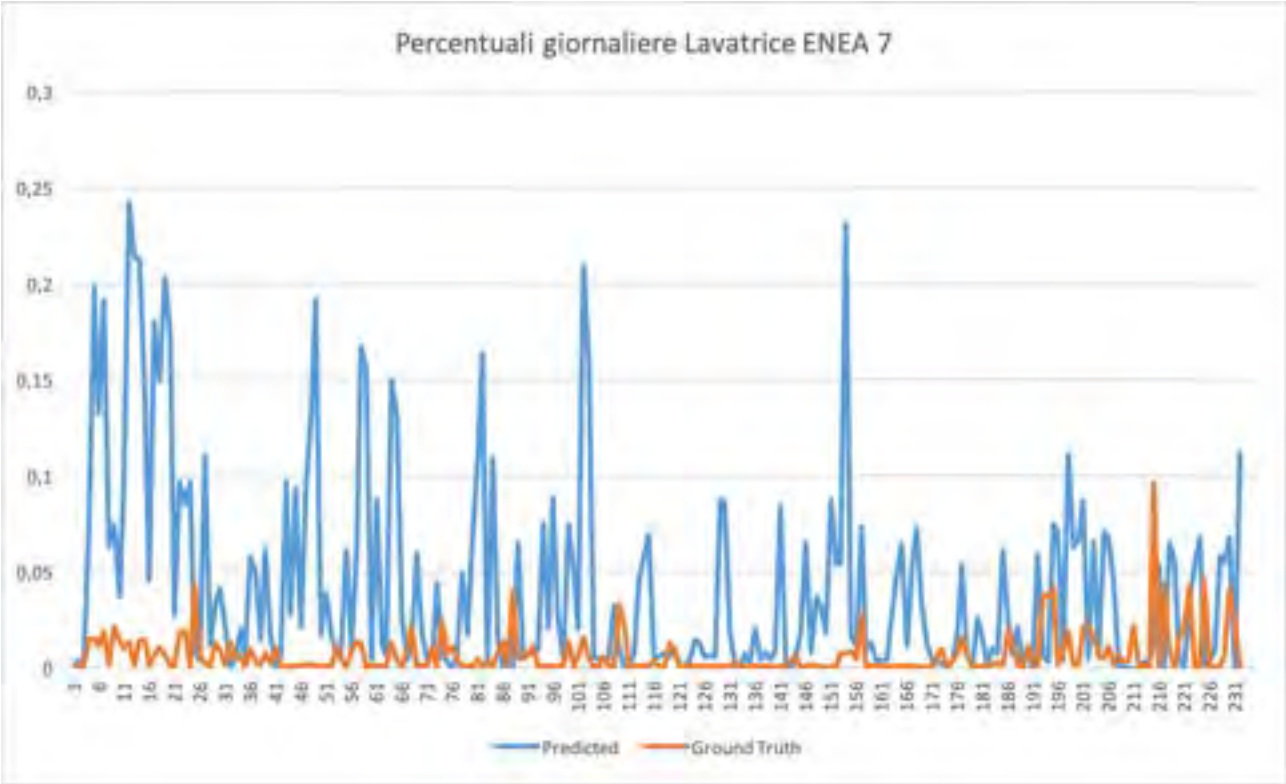

Figura 29: percentuali giornalieri previste e reali per la lavatrice di ENEA 7

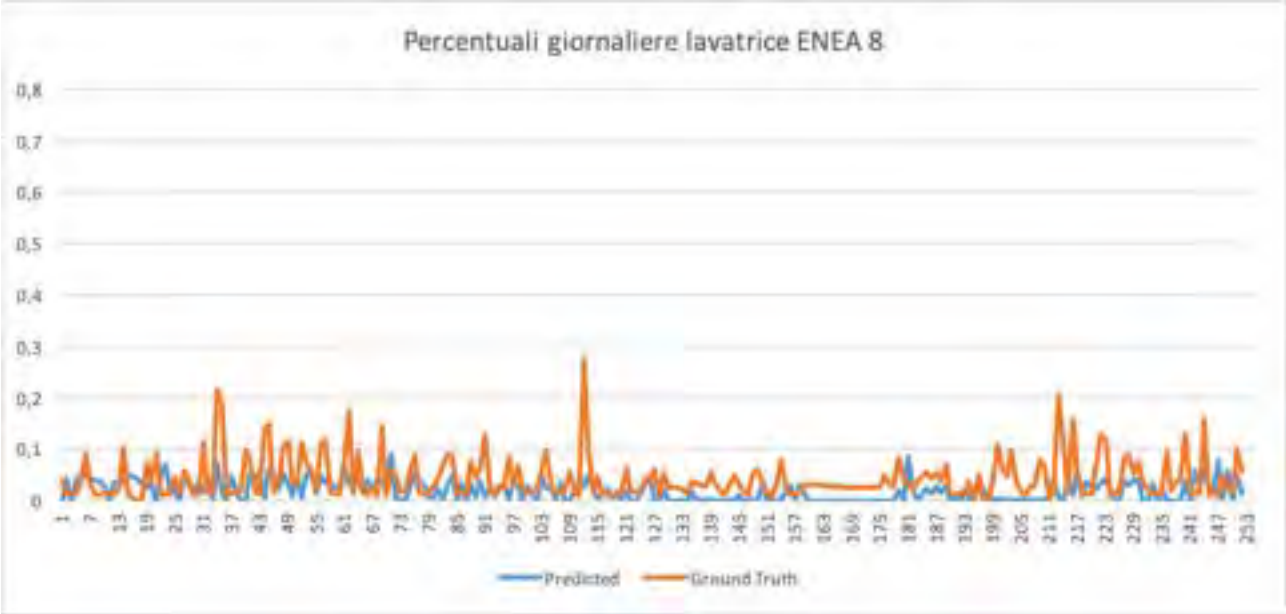

Figura 30: percentuali giornalieri previste e reali per la lavatrice di ENEA 8

3 Conclusioni

I risultati ottenuti dimostrano come il modello realizzato sia in grado di predire in maniera accurata la disaggregazione energetica di frigorifero e lavatrice su entrambe le case prese in considerazione. Il modello potrebbe essere più accurato se si raccogliesse una quantità maggiore di dati, in modo da poter addestrare il modello sui dati delle case stesse. Inoltre si potrebbero provare diversi modelli addestrati sulle diverse case presenti nel dataset UKDALE e con tecniche di clustering decidere quale modello utilizzare sulle nuove case prese in considerazione in modo da scegliere correttamente il modello relativo all'abitazione con il comportamento energetico più simile (notiamo a tal proposito come i risultati del nostro modello siano migliori sulla casa ENEA 8 che su ENEA 7).

Infatti, come sviluppo futuro, si è deciso di implementare un meccanismo di clustering che sia in grado di raggruppare le case fornite all'interno di gruppi ed assegnare ad ognuno di questi insiemi il modello più adatto e che fornisca le prestazioni più elevate. L'idea è quella di addestrare le reti neurali descritte in precedenza su case differenti appartenenti a cluster diversi; successivamente, ogni nuova casa che deve essere valutata viene prima processata dall'algoritmo di clustering, che la assegna, sulla base di caratteristiche comuni, al gruppo di case più simile. A questo punto, la casa può essere valutata mediante il modello assegnato a quel determinato cluster, ottimizzando le prestazioni in quanto si seleziona il modello migliore per lei.
In figura 31 è possibile osservare graficamente il processo di clustering, ad alto livello, offrendo quindi una base di quello che sarà parte della nuova implementazione.

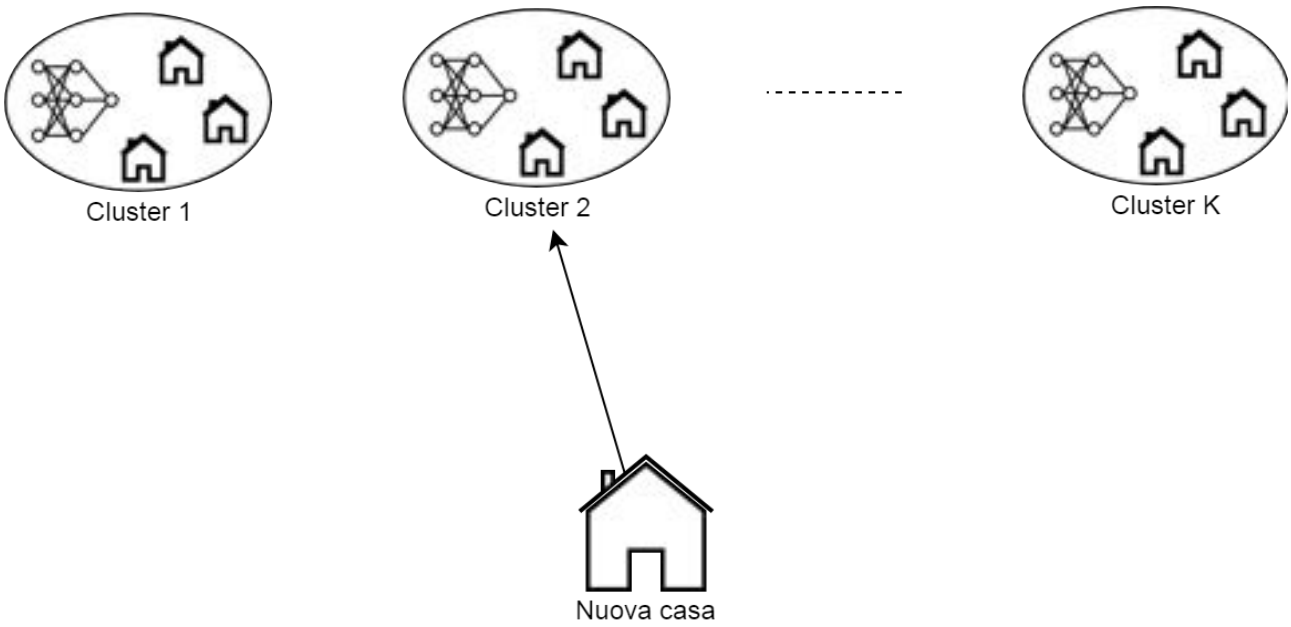

Figura 31: strategia di clustering per la disaggregazione del consumo elettrico

4 Riferimenti bibliografici

1. R. Bonfigli, A. Felicetti, E. Principi, M. Fagiani, S. Squartini, F. Piazza, “Denoising autoencoders for Non-Intrusive Load Monitoring: Improvements and comparative evaluation”, Energy and Buildings, 158, 2018, 1461-1474.
2. M. Figueiredo, A. de Almeida, B. Ribeiro, “Home electrical signal disaggregation for non-intrusive load monitoring (NILM) systems”, Neurocomputing, 96, 2012, 66-73.
3. W. He, Y. Chai, “An Empirical Study on Energy Disaggregation via Deep Learning”, *Advances in Intelligent Systems Research*, *133, 2016*.
4. J. Kelly, W. Knottenbelt, "The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes." *Scientific data* 2 (2015): 150007.
5. J. Kelly, W. Knottenbelt “Neural nilm: Deep neural networks applied to energy disaggregation” In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments* (pp. 55-64). ACM.
6. A. Sauhats, R. Varfolomejeva, O. Linkevics, R. Petrecenko, M. Kunickis, M. Balodis “Analysis and prediction of electricity consumption using smart meter data”. In: Power Engineering, Energy and Electrical Drives (POWERENG), 2015 IEEE 5th International Conference on. IEEE, 2015. p. 17-22.
7. C. Zhang, M. Zhong, Z. Wang, N. Goddard, C. Sutton “Sequence-to-point learning with neural networks for nonintrusive load monitoring, The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), Febbraio 2018, New Orleans, AAAI, 1-8.
8. Faustine, A., Mvungi, N. H., Kaijage, S., & Michael, K. (2017). A survey on non-intrusive load monitoring methodes and techniques for energy disaggregation problem. *arXiv preprint arXiv:1703.00785*.
9. I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, *Deep learning* (Vol. 1), 2016, Cambridge: MIT press.
10. H. Kim, M. Marwah, M. F. Arlitt, G. Lyon, and J. Han. Unsupervised Disaggregation of Low Frequency Power Measurements. Proceedings of the 11th SIAM International Conference on Data Mining, pages 747–758, 2011.
11. Zico Kolter, Tommi Jaakkola, and J Z Kolter. Approximate Inference in Additive Factorial HMMs with Application to Energy Disaggregation. In Proceedings of the International Conference on Artificial Intelligence and Statistics, pages 1472–1482, 2012.
12. Oliver Parson, S Ghosh, M Weal, and A Rogers. Non-intrusive Load Monitoring using Prior Models of General Appliance Types. In Proceeding of the twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12), Toronto, Canada, apr 2012.
13. Stephen Makonin, Fred Popowich, Ivan V. Bajic, Bob Gill, and Lyn Bartram. Exploiting HMM Sparsity to Perform Online Real-Time Nonintrusive Load Monitoring. IEEE Transactions on Smart Grid, 2015.
14. Bochao Zhao, Lina Stankovic, and Senior Member. On a Training-Less Solution for Non-Intrusive Appliance Load Monitoring Using Graph Signal Processing. IEEE Transactions on Smart Grid, 4, 2016.
15. Vladimir Stankovic, Jing Liao, and Lina Stankovic. A graph-based signal processing approach for lowrate energy disaggregation. In Computational Intelligence for Engineering Solutions (CIES), 2014 IEEE Symposium on, pages 81–87, 2014.

5 Abbreviazioni ed acronimi

NILM : Nonintrusive Load Monitoring

DAE: Denoising Auto Encoder

LSTM: Long Short Term Memory

nilmtk : Non-Intrusive Load Monitoring Toolkit

Curriculum del gruppo di lavoro:

VERONICA PICCIALLI

Nata nel 1975, Veronica Piccialli ha conseguito la laurea con lode in Ingegneria Informatica e il dottorato di ricerca in Ricerca Operativa, entrambi presso Sapienza Università di Roma, nel 2000 e nel 2004, rispettivamente. Nel 2006 è stata postdoc presso il Dipartimento Combinatorics and Optimization presso, University of Waterloo in Canada per 6 mesi con il Prof. Henry Wolkowicz. Dal 2004 al 2008 è stata postdoc presso l'Università di Roma "La Sapienza". Dal 2008 è professore assistente presso l'Università di Roma Tor Vergata. Nel 2013 ha ottenuto l'abilitazione nazionale come Professore Associato e nel 2017 l'abilitazione nazionale come Professore Ordinario. I suoi interessi di ricerca sono l'ottimizzazione non lineare, con particolare attenzione ai problemi di machine learning, ottimizzazione globale e teoria dei giochi. E' autrice e co-autrice di circa 30 pubblicazioni su riviste internazionali. Attualmente sta supervisionando 3 dottorandi, uno in collaborazione con l'Università della Lorena in Francia. È stata ricercatrice presso l'Università TU di Dresda e l'Alpen-Adria Universitat di Klagenfurt. E' stata invitata come relatrice alla conferenza "EWMINLP" a Marsiglia, 2010, e a Algorithms and Optimization in Action, IASI CNR, nel 2011. E' stata invitata a tenere diversi corsi di dottorato, come ad esempio all'EURO Summer Institute 2010 presso l'Università di Klagenfurt e alla COST/MINO PhD School on Advanced Optimization Methods nel 2016.

CHRISTIAN LA RICCIA

Nato nel 1993, Christian La Riccia sta perseguendo la laurea magistrale in Ingegneria Informatica, nei due anni previsti, con una media ponderata pari a 28,30. Attualmente sta lavorando al presente progetto, in collaborazione con Enea, per lo sviluppo della tesi nel campo del Deep Learning, all'interno del quale ha già avuto esperienze pregresse per progetti universitari, come lo sviluppo di reti convoluzionali per la classificazione di immagini. Dal 2016 lavora presso l'azienda Adeodata s.r.l come consulente e recentemente è stato spostato nel reparto di ricerca e sviluppo per l'implementazione di un software utilizzato nel campo farmaceutico. I suoi interessi si legano molto al mondo del Machine Learning in genere, al quale è fortemente appassionato.

CHIARA LITI

Nata nel 1990, Chiara Liti ha conseguito la laurea con lode in Ingegneria Gestionale presso l'Università di Roma Tor Vergata nel 2016. È iscritta al terzo anno del corso di dottorato in Computer Science, Control and GeoInformation presso la medesima università. Nel 2017 ha svolto un'internship di nove mesi con la Ladbrokes Coral Group lavorando ad un progetto di text mining e sentiment analysis . Attualmente si trova come studente visitatore presso l'Instituto de Matemáticas de la Universidad de Sevilla presso il quale sta svolgendo attività di ricerca nell'ambito del deep learning. I suoi di interessi di ricerca sono l'ottimizzazione, i data analytics e il machine learning e in particolare alle loro applicazioni e implicazioni nell'ambito delle neuroscienze.

ANDREA POMENTE

Nato nel 1990, Andrea Pomente ha conseguito la laurea magistrale in Ingegneria Gestionale presso l'Università di Roma Tor Vergata nel 2017. È iscritto al primo anno del corso di dottorato in Computer Science, Control and GeoInformation presso la medesima università. Dal 2017 lavora presso l'Università di Roma "Tor Vergata" come ricercatore nel campo dell'Osservazione della Terra e del Machine Learning. Dal 2018 ha iniziato una collaborazione con il Phi-Lab, il dipartimento dell'Agenzia spaziale europea focalizzata sull'intelligenza artificiale applicata all'osservazione della terra. Lo scopo della collaborazione è sviluppare algoritmi di Machine Learning per dati satellitari utilizzando le Deep Neural Network.