Progetto Machine Learning 2019-2020

Giovanni D'Agostino
Matricola:0255794
Email: giovannidag91@gmail.com

Marco Giorgi
Matricola:0266724
Email: marco.giorgi90@outlook.com

Il Non Intrusive Load Monitoring (NILM) è il processo utilizzato per monitorare il profilo energetico di un ambiente domestico e per disaggregare il consumo energetico totale in segnali di consumo dei singoli dispositivi. In termini di algoritmi esistono varie soluzioni che implementano questo metodo di disaggregazione, in questo paper la soluzione adottata è basata sulle reti neurali deep. Infatti, poiché NILM è un'attività di apprendimento automatico ben definita, i ricercatori hanno attivamente applicato vari modelli di deep learning a questa attività. Il problema principale nell'adottare soluzioni basata sulle reti neurali deep risiede nella difficoltà di addestramento a causa di un gradiente molto grande o di degrado della rete, per questo sono stati proposti vari framework di apprendimento tra i quali troviamo il Sequence-to-Sequence e la sua variante proposta da [1] (Chaoyun Zhang, Mingjun Zhong, Zongzuo Wang, Nigel Goddard, and Charles Sutton) ovvero il Sequence-to-Point, sulla quale si basa il seguente progetto.

1 Ambiente di Sviluppo

La rete deep del progetto che verrà mostrata nel seguente documento è stata implementata in Tensor Flow utilizzando le librerie di Keras e sfruttando la piattaforma di *Google Collaboratory*. Tra le GPU disponibili sulla piattaforma è stata utilizzata massivamente la *Tesla P100-PCIE* per avere tempi di sperimentazione più brevi e maggior memoria a disposizione. Il linguaggio di sviluppo utilizzato è Python e tra le librerie, oltre quella relative a *tensorflow*, sono state adottate *pandas*, *numpy* e *matplotlib.pyplot*.

2 Sequence-to-sequence

Le architetture Seq2Seq, introdotte per la prima volta da [2] (Kelly e Knottenbelt) definiscono una rete neurale F_s che mappa le sliding window $Y_{t:t+W-1}$ della potenza totale in ingresso alle finestre corrispondenti $X_{t:t+W-1}$ della potenza del dispositivo di uscita, cioè modellano $X_{t:t+W-1} = F_s(Y_{t:t+W-1}) + \varepsilon$, dove ε è il *random noise* Gaussiano di dimensione W. Quindi, per addestrare la rete su una coppia (X,Y) di sequenze complete, la funzione di *loss* è

$$L_s = \sum_{t=1}^{T-W+1} \log p(X_{t:t+W-1}|Y_{t:t+W-1}, \theta_s) \qquad (1)$$

dove θ_s sono i parametri della rete F_s. In pratica, un sottoinsieme di tutte le finestre possibili può essere utilizzato durante l'addestramento al fine di ridurre la complessità computazionale. Poiché ci sono più previsioni per x_t quando $2 \leq t \leq T-1$, una per ogni finestra scorrevole che contiene il tempo t, la media di questi valori previsti viene utilizzata come risultato della previsione.

Sebbene nelle prime fasi della progettazione è stato considerato questo tipo di architettura, l'idea è stata presto abbandonata a causa dell'elevato costo computazionale e quantità di memoria necessaria nella fase di training. La soluzione adottata in definitiva è stata invece l'architettura Sequence-to-Point, ideata successivamente da *[1]* e che ha permesso di ottenere risultati anche migliori ad un costo computazionale decisamente minore.

3 Sequence-to-point

Nel modello Sequence-to-Point (Seq2Point) invece di addestrare la rete per prevedere un'intera finestra di valori del dispositivo selezionato, ci si limita a prevedere soltanto il valore mediano della finestra. L'idea è quella di dare in input alla rete una *sliding window* di valori, corrispondenti ai consumi totali di un'abitazione, $Y_{t:t+W-1}$, e avere come output il valore x_τ posizionato al centro della finestra temporale corrispondente al dispositivo di cui si vuol fare la predizione, dove $\tau = t + \frac{W}{2}$.

Invece di mappare sequenza su sequenza, le architetture Seq2Point definiscono una rete neurale F_p che mappa le sliding window $Y_{t:t+W-1}$ dell'input al punto medio x_τ delle corrispondenti finestre $X_{t:t+W-1}$ dell'output. Il modello è dunque $x_\tau = F_p(Y_{t:t+W-1}) + \varepsilon$ e la funzione di *loss* per

addestrare la rete ha la seguente forma :

$$L_p = \sum_{t=1}^{T-W+1} \log p(x_\tau | Y_{t:t+W-1}, \theta_p) \quad (2)$$

dove θ_p sono i parametri di rete. Il vantaggio di questo modello rispetto ad altre varianti utilizzate in letteratura è che viene effettuata una singola previsione per ogni x_τ, piuttosto che una media di previsioni per ogni finestra.

Per considerare anche i punti iniziali e finali del dataset, che nella predizione dei punti mediani delle finestre, data una sequenza di input completa $Y = (y_1 ... y_T)$ riempiamo la sequenza con un numero pari a $W/2$ di zeri all'inizio e alla fine, con W che indica la dimensione della finestra scelta.

4 Dataset

I dataset *main_train.csv*, *fridge_train.csv* e *dishwasher_train.csv* contengono i dati utilizzati nella fase di training, e corrispondono rispettivamente al consumo energetico totale di un'abitazione, quello relativo al frigorifero e quello della lavastoviglie, tutti espressi in Watt. I dati si riferiscono ad un periodo compreso tra il 2019-01-01 00:00:00 e il 2019-03-14 23:59:59, il numero di tuple per ogni dataset è di 6082508 e non sono presenti all'interno dei dataset valori NaN o record replicati. Nella seguente tabella sono riportate le statistiche principali dei tre training set:

Set	Mean	Std	Min	Max
main	370.915	549.188	73.481	6048.7
fridge	37.2371	46.9888	2.2	1233.1
dishwasher	25.871	238.5801	0	2570.6

Il dataset messo a disposizione per il progetto risulta più piccolo rispetto a quelli ormai diventati standard per affrontare il problema della disaggregazione energetica (e.g. *UK-DALE*, *REDD*, ecc...) ed anche la frequenza di campionamento è diversa. Per questi motivi il modello di addestramento utilizzato sarà leggermente diverso da quello adottato da [1]

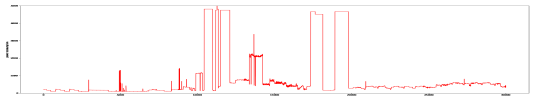

Fig. 1. Campione del consumo di potenza totale

Fig. 2. Campione del consumo di potenza frigorifero

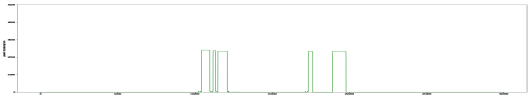

Fig. 3. Campione del consumo di potenza lavastoviglie

5 Scelte Progettuali

5.1 Preprocessing

Nella prima fase di progettazione i dataset sono stati caricati e suddivisi ognuno in due parti: la prima parte è stata utilizzata per il training del modello mentre la seconda per determinare gli iperparametri ottimali e valutare le prestazioni del modello su una parte del dataset non vista nella fase di addestramento. La divisione è stata effettuata considerando circa l'80% dei dati come *training set*, mentre il restante 20% come dati di *validation set*. Di fatto l'80% dei dati corrisponde ad un arco temporale compreso tra il 2019-01-01 ore 00:00:00 e il 2019-03-01 ore 00:00:00.

Nella seconda fase di progettazione, una volta determinate il modello con i relativi iperparametri e il numero di epoche opportuno, non è stato più necessario suddividere i dataset ed è stato effettuato l'addestramento del modello su tutti i dati a disposizione.

Come già accennato in precedenza, adottando uno schema di tipo *Seq2Point*, si è rivelato necessario effettuare un'operazione di padding dei dataset con un determinato numero di zeri sia all'inizio che alla fine della sequenza di dati. Successivamente è stata effettuata una trasformazione dei dati prima di essere inviati al modello per l'addestramento. Per quanto riguarda il consumo energetico relativo al frigorifero i dati sono stati standardizzati, semplicemento sottraendo la media e dividendo per la deviazione standard. Per quanto riguarda il consumo energetico della lavastoviglie i dati sono stati normalizzati, sottraendo il minimo e dividendo per la differenza tra il massimo e il minimo. Questa differenza tra i due approcci è anch'essa risultato di vari test ed il divario prestazionale è attribuibile alla diversa natura delle due sorgenti di consumo considerate, che verrà approfondita nelle seguenti sezioni.

5.2 Scelta della Finestra

Dal lavoro di [1], e in particolare dall'implementazione di Zhong[7], possiamo osservare che la dimensione della finestra è di 599, sia nella predizione dei consumi del frigorifero sia per la lavastoviglie. Ma è ragionevole pensare che questa finestra è stata specificatamente scelta perché risulta migliore per il dataset utilizzato (*UK-DALE*). Per questo per il progetto è stata utilizzata una finestra più grande, pari a 3999 per il modello di predizione per il frigorifero e 999 nel caso della lavastoviglie. Questi valori sono frutto di una lunga fase di sperimentazione manuale effettuata al variare di tutti gli iperparametri del modello, in cui è anche emerso che il valore ottimale della dimensione dei batch, che suddividono il dataset, sia per il frigorifero che per la lavastviglie è pari a 512.

5.3 Generatore

Uno dei problemi più grandi che è stato riscontrato nella progettazione è stato quello di gestire l'enorme quantità di dati da inviare al modello, avendo una disponibilità di memoria limitata. Per questo motivo per la fase di training si è scelto di generare *on-the-fly*, dividendo i dataset in batch e utilizzando i metodi della classe *keras.utils.Sequence* per generare da questi delle coppie *(finestra di dati, punto mediano)* da inviare al modello. In particolare per ogni batch di dati viene generato un numero di finestre pari alla dimensione del batch stesso e ognuna di esse contiene i valori energetici totali dell'abitazione all'interno di un determinato arco temporale. Ad ogni finestra viene affiancato un valore che corrisponde al punto mediano nella stessa finestra temporale considerata ma presa dai consumi energetici del dispositivo selezionato. questo insieme di coppie viene inviato al modello per l'addestramento.

Infine è possibile abilitare una funzione di *shuffle* in modo tale che i batch generati tra le epoche non si somigliano, limitando così la possibilità di overfitting e rendendo il modello più robusto.

Il generatore è stato implementato facendo riferimento a [3].

6 Architettura

6.1 Fridge

Il modello di rete adottato per il frigorifero (ispirato a [6]) presenta 4 layer nascosti di tipo convoluzionali 1D con filtri da 30, 30, 40, 50, rispettivamente con dimensioni del kernel 10, 8, 6, 5. Naturalmente utilizzando dei *Conv1D* è necessario passare in ingresso come argomento al primo layer le dimensioni dei dati in ingresso, nel nostro caso della forma (*size window*, 1). Subito dopo il quarto layer convoluzionale è stato aggiunto un layer di *Dropout* con rate di 0.2, questo per aggiungere un fattore di casualità e limitare la possibilità di andare in overfitting. Dopo la fase di filtraggio dei dati di input, si passa per uno layer *Dense* con un numero di neuroni pari a 1024, seguito anch'esso da un layer di *Drouout* identico al precedente.

Infine lo strato di output è costituito da un layer *Dense* con un unico neurone in modo da riflettere lo schema del *Seq2Point* e avere come output un unico valore. Tutti i layer *Conv1D* e *Dense* hanno impostati come funzione di regolarizzazione dell'output la funzione *ReLU*, mentre i restanti parametri sono stati lasciati come impostati di default dalle librerie.

La rete è stata addestrata tramite l'algoritmo *Adam* con learning rate pari a 0.0001 e i restanti parametri sono stati anche qui lasciati come quelli di default.

La funzione loss utilizzata è la *MAE*, ovvero l'errore medio assoluto. La sua maggiore efficacia in questo caso specifico rispetto alla *MSE*, la tipica funzione di loss utilizzata nei problemi di regressione, può essere ricondotto alla presenza di numerosi valori outlier nel dataset considerato.

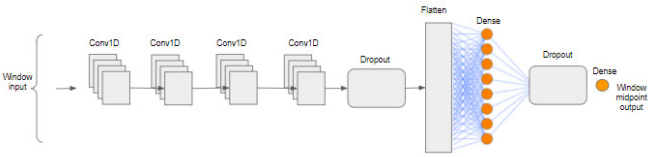

Fig. 4. Modello rete deep Fridge

6.2 Dishwasher

Il modello di rete deep proposto per trattare il caso della lavastoviglie può essere considerato una versione ridotta di quello utilizzato per il figorifero ed è composto da pochi strati: 2 layer *Conv1D*, rispettivamente con 30 e 40 filtri e dimensioni del kernel 10, 8, e 2 layer *Dense* da 512 e 1 neuroni, ma senza nessun layer di *Dropout*.

Alla luce dei risultati derivanti dai diversi test è stata scelta come funzione di loss la *Binary Crossentropy*, generalmente usata quando ci sono solo due classi di etichette (rappresentabili con 0 e 1), e che quindi risulta più adatta nell'individuare in maniera corretta quando il dispositivo è acceso o spento. Mentre per tutti i layer è stata utilizzata come funzione di attivazione la *ReLU* come nel caso della rete per il frigorifero, nel layer *Dense* finale è stata usata la funzione *Sigmoid* in quanto risulta essere la migliore soluzione adottando come funzione di loss la *Binary Crossentropy*.

La rete è stata addestrata tramite l'algoritmo *Adam* con learning rate pari a 0.0001, risultato migliore nella fase di sperimentazione, e lasciando i restanti valori di default.

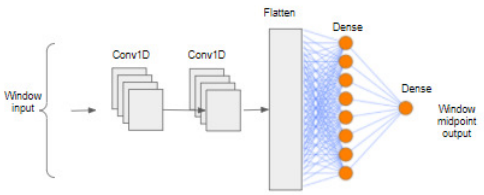

Fig. 5. Modello rete deep Dishwasher

7 Metriche

Poichè che la metrica F1 non è presente di default in keras per il monitoraggio del training è stato ritenuto opportuno definire in maniera custom una funzione che calcolasse la F1 sul validation set alla fine di ogni epoca. Per far questo è stata estesa classe *keras.metrics.Metric* seguendo lo schema indicato da [4] e [5].

La F1 finale è stata calcolata:

$$P_i(t) = \frac{\sum_{t=1}^{T} min(y_i(t), \hat{y}_i(t))}{\sum_{t=1}^{T} \hat{y}_i(t)} \tag{3}$$

$$R_i(t) = \frac{\sum_{t=1}^{T} min(y_i(t), \hat{y}_i(t))}{\sum_{t=1}^{T} y_i(t)} \quad (4)$$

$$F1_i = 2\frac{P_i \cdot R_i}{P_i + R_i} \quad (5)$$

Le reti sono state addestrate fine ad un numero di epoche ottimale calcolato inserendo un *early stopping* e monitorando il valore della F1 calcolato sul *validation set* per ogni epoca.

8 Risultati

I risultati sono stati calcolati sul *validation set* per un numero di epoche ottimale determinato dal early stopping considerando il miglior valore della F1 ottenuto e monitorando i valori della loss calcolati sia nel *training set* sia nel *validation set*.

Training Set	Epoche	Loss	F1
fridge	4	0.3325	0.7856
dishwasher	13	0.0049	0.9796

9 Conclusioni

Questo progetto rappresenta solo una delle tante implementazioni proposte per lo schema di Sequence-to-Point, e sarebbe interessante confrontare le prestazioni rispetto alle precedenti soluzioni attraverso uno dei dataset standard utilizzati per i problemi di disaggregazione energetica. Per questo specifico progetti non è stato possibile principalmente a causa delle limitazioni imposte dalla piattaforma di Google Colab nella sua versione gratuita.

I risultati ottenuti dunque sono stati effettuati sperimentalmente senza una cross validation, in quanto non è stato possibile definire una Grid Search per i suddetti limiti temporali e di risorse di memorizzazione. Per esempio abbiamo osservato che nell'addestramento del modello del fridge il miglioramento era direttamente proporzionale all'aumento della dimensione della finestra, naturalmente adattando anche il modello con l'opportuno numero di filtri, ma oltre un determinato valore la piattaforma messa a disposizione da Google portava ad errori di out-of-memory. Sarebbe stato interessante poter avere maggiore libertà nella ricerca degli iperparametri migliori per lo specifico caso di studio. Per questo motivo crediamo che, sebbene le soluzioni da noi sperimentate e qui proposte siano ottime, non siano ottime in senso assoluto.

References

[1] C. Zhang, M. Zhong, Z. Wang, N. Goddard, and C. Sutton. [*Sequence-to-point learning with neural networks for non-intrusive load monitoring*]. 2018

[2] J. Kelly and W. Knottenbel *"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.* 2015.

[3] A. Amidi and S. Amidi *"A detailed example of how to use data generators with Keras"* `https://stanford.edu/ shervine/blog/keras-how-to-generate-data-on-the-fly`

[4] Repository Github `https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/metrics.py`

[5] R. Bonfigli, A. Felicetti, E. Principi, M. Fagiani, S. Squartini, and F. Piazza. *"Denoising autoencoders for non-intrusive load monitoring: improvements and comparative evaluation," Energy and Buildings.* 2018

[6] Repository Github N. Batra et al `https://github.com/nilmtk/nilmtk-contrib/blob/master/nilmtk_contrib/disaggregate/seq2point.py`

[7] Repository Github M.Zhong `https://github.com/MingjunZhong/seq2point-nilm`