Università
degli Studi
di Ferrara

Time series analysis
gli ottimizzatori più famosi
a confronto

Anna Fabbri & Mattia Marella

23 gennaio 2025

Abstract

In questo studio, viene presentata un'analisi comparativa delle performance di quattro algoritmi di ottimizzazione — Stochastic Gradient Descent (SGD), Adagrad, RMSprop e Adam — nel contesto di una rete neurale ricorrente LSTM. Il dataset utilizzato, l'UCI Air Quality Dataset, offre un campione rappresentativo per analizzare la qualità dell'aria in un contesto urbano. L'analisi empirica evidenzia che RMSprop offre le migliori performance in termini di accuratezza (MAE, MSE, R^2) e stabilità nella convergenza, mentre Adam si distingue per rapidità di addestramento e robustezza generale. Adagrad, sebbene efficace nelle prime epoche, soffre di una rapida diminuzione del learning rate che ne limita l'efficacia a lungo termine. Infine, SGD, pur richiedendo un numero elevato di epoche per convergere, garantisce prestazioni finali competitive. I risultati forniscono indicazioni pratiche sull'uso ottimale degli algoritmi per applicazioni future nella modellazione di serie temporali.

1 Introduzione

Nel contesto del machine learning e del deep learning, il forecasting di serie temporali rappresenta una sfida cruciale con applicazioni in diversi settori, tra cui la finanza, l'energia e la gestione delle risorse. Le reti neurali ricorrenti (RNN) sono una delle architetture più utilizzate per modellare sequenze tem-

porali grazie alla loro capacità di mantenere informazioni su stati passati. Tuttavia, la scelta dell'ottimizzatore gioca un ruolo fondamentale nel determinare le performance e la velocità di convergenza del modello.

In questo progetto, abbiamo sviluppato un particolare modello basato su RNN: una rete LSTM (Long Short Term Memory). Essa è stata utilizzata per la previsione di serie temporali e per confrontare l'efficacia di quattro algoritmi di ottimizzazione: Stochastic Gradient Descent (SGD), Adagrad, RMSprop e Adam. L'obiettivo principale è analizzare le differenze in termini di accuratezza delle previsioni e comportamento durante il processo di addestramento, valutando i trade-off tra convergenza rapida e stabilità del gradiente. Questo studio fornisce un confronto empirico e approfondito che evidenzia i punti di forza e le debolezze di ciascun metodo.

1.1 Il Dataset

Il UCI Air Quality Dataset (presentato per la prima volta nel paper [1]) è un set di dati che misura la qualità dell'aria in una zona urbana di Milano. I dati sono stati raccolti tra il marzo 2004 e il febbraio 2005, offrendo un campione significativo per studiare l'andamento dei principali inquinanti atmosferici in un ambiente cittadino nel corso di un intero anno. È un dataset ben conosciuto ed infatti viene spesso utilizzato per analisi di serie temporali, regressione e classificazione per prevedere la concentrazione futura di inquinanti sulla base delle condizioni storiche e meteorologiche.

Il dataset contiene 9.358 istanze di risposte *medie orarie* provenienti da un array di 5 sensori chimici a ossido metallico integrati in un dispositivo multisensore per la qualità dell'aria. Il dispositivo è stato collocato sul campo in un'area significativamente inquinata.

Le features presenti nel dataset sono visibili nella tabella 1. I *missing values* non vengono contrassegnati con la tipica label *NaN*, bensì con un valore intero pari a -200.

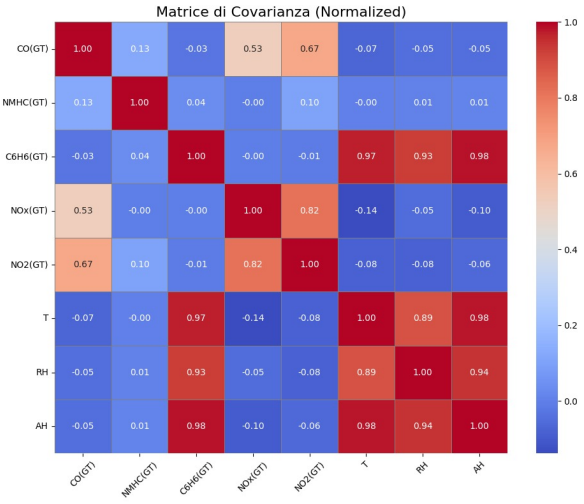

Figura 1: Covarianza delle features NON categoriche del dataset.

Dopo un'analisi della covarianza tra le feature volta ad escludere quelle tra loro correlate (che appesantirebbero il training), si è

Tabella 1: Descrizione delle variabili nel dataset UCI Air Quality

Attributo	**Descrizione**
Data	La data della misurazione.
Ora	L'ora corrispondente alla misurazione.
CO (GT)	Concentrazione di monossido di carbonio in mg/m^3.
PT08.S1 (CO)	Segnale del sensore per CO.
NMHC (GT)	Concentrazione di idrocarburi non metanici in µg/m^3.
C6H6 (GT)	Concentrazione di benzene in µg/m^3.
PT08.S2 (NMHC)	Segnale del sensore per NMHC.
NOx (GT)	Concentrazione di ossidi di azoto in µg/m^3.
PT08.S3 (NOx)	Segnale del sensore per NOx.
NO2 (GT)	Concentrazione di biossido di azoto in µg/m^3.
PT08.S4 (NO2)	Segnale del sensore per NO2.
PT08.S5 (O3)	Segnale del sensore per ozono.
T	Temperatura (°C).
RH	Umidità relativa (%).
AH	Umidità assoluta.

notato che alcune di esse sono effettivamente correlate tra loro. Ad esempio *AH* risulta molto correlata con *C6H6(GT)*, *T* ed *RH*, per cui è conveniente escluderla dal train, in quanto porta informazioni ridondanti (si veda l'immagine 1).

2 Related works

Non si può non menzionare il paper «A Review on Weather Forecasting Techniques» in quanto fonte fondamentale per comprendere l'evoluzione delle metodologie di previsione meteorologica.

L'importanza di questa revisione risiede anche nel confronto tra approcci deterministici e probabilistici, evidenziando come i modelli ensemble, che combinano più tecniche, offrano una robustezza superiore per gestire l'incertezza delle previsioni meteorologiche. Questa prospettiva si allinea con la crescente attenzione verso modelli ibridi che sfruttano sia le conoscenze fisiche dei sistemi atmosferici sia l'apprendimento automatico.

Anche il paper «Data-driven Real-time Short-term Prediction of Air Quality: Comparison of ES, ARIMA, and LSTM» risulta interessante: esso confronta tecniche di modellazione predittiva per la qualità dell'aria, tra cui ES, ARIMA e LSTM, evidenziando punti di forza e limiti. I risultati mostrano la superiorità delle LSTM su serie temporali complesse grazie alla capacità di apprendere

dipendenze a lungo termine, mentre ES si dimostra efficace nel breve termine. Questo lavoro offre spunti utili per la selezione dei modelli predittivi più adatti per l'analisi di dati ambientali e problemi di previsione di serie temporali.

L'articolo online *Weather forecasting with Recurrent Neural Networks in Python* ha invece fornito importanti spunti sull'architettura della rete utilizzata in questo lavoro. Esso esplora passo per passo la creazione di un modello di previsione del tempo utilizzando Python, analizzando le tecniche di preprocessing dei dati, la costruzione del modello LSTM e la valutazione delle performance del modello.

3 Il workflow

Nel presente progetto, l'obiettivo è confrontare le performance di diversi ottimizzatori, come Adam, SGD, AdaGrad e RMSProp, utilizzando sempre lo stesso modello di rete neurale per un task di previsione supervisionata.

Il workflow implementato nel notebook è suddiviso nei seguenti passaggi principali.

3.1 Scelta delle metriche

Le metriche utilizzate per il confronto sono: MSE e MAE, che misurano rispettivamente l'errore quadratico medio e l'errore assoluto medio, fornendo un'indicazione della distanza media tra le previsioni del modello e i valori reali, e R^2 (coefficiente di determinazione), che quantifica la proporzione di varianza nei dati osservati spiegata dal modello.

3.2 Preprocessing

3.2.1 Importazione delle librerie

L'ambiente di lavoro utilizza librerie standard di Python per l'apprendimento automatico e la gestione dei dati. Tra queste, le più importanti sono: *torch* per la creazione del modello; *numpy* e *pandas* per la manipolazione e l'analisi dei dati; *matplotlib* e *seaborn* per la visualizzazione grafica dei risultati; *ucimlrepo* per accedere direttamente al dataset UCI.

3.2.2 Feature engineering

I dati grezzi sono caricati in un DataFrame e trasformati per prepararli all'uso nel modello di rete neurale. Essi vengono sottoposti a quattro passaggi principali:

Feature selection Le variabili eccessivamente correlate tra loro vengono scartate (se ne sceglie una tra quelle correlate). Dopo l'analisi di covarianza sono state mantenute le seguenti: T, RH,

NO2(GT), *NMHC(GT)*. La feature *CO* è stata scelta come label;

Feature generation Vengono create alcune feature artificiali per aiutare la rete nell'apprendimento. In particolare, per ottenere un modello continuo degli orari di una giornata si è utilizzato una rappresentazione goniometrica che usa funzioni seno e coseno: *hour_sin*, *hour_cos*, *day_sin*, *day_cos*. Infine, è stata creata anche una flag booleana per verificare se un certo estremo temporale appartiene al fine settimana (*is_weekend*), in quanto i livelli di monossido di carbonio nell'aria (la label scelta) registrano un decremento rilevante nel corso del weekend.

A seguire, è stata ripetuta l'analisi di covarianza su tutte le features, controllando che nessuna avesse uno score superiore a 0.9 (si veda l'immagine 2).

Normalizzazione Le variabili indipendenti vengono scalate per standardizzare il loro range;

Divisione del dataset Il dataframe viene suddiviso in tre parti: training set, validation set e test set per garantire una valutazione robusta delle performance del modello ed implementare early stopping.

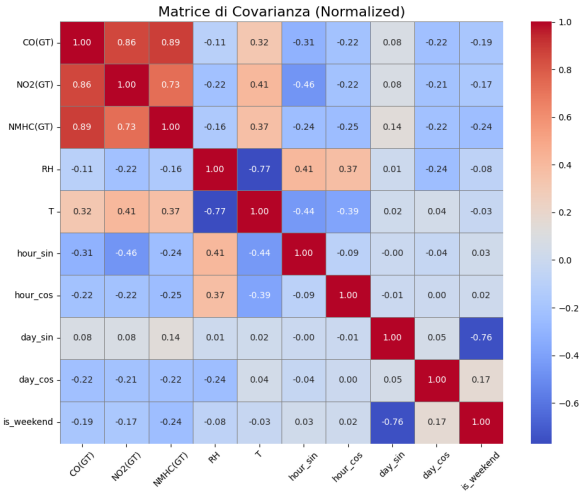

Figura 2: Covarianza delle features utilizzate.

3.3 Definizione del modello

Il modello utilizzato per la previsione si basa su una rete neurale ricorrente LSTM (Long Short-Term Memory), progettato per gestire serie temporali. La struttura del modello è la seguente:

LSTM Layer Il cuore del modello è costituito da un layer LSTM, che prende come input una threshold di 72 datapoint. Il numero di layer LSTM è pari ad 1 (la distribuzione dei dati è molto semplice e non è necessario un modello complesso), con un parametro di dropout tra i layer per evitare overfitting (valore predefinito: 0.3).

Fully Connected Layer Dopo che la sequenza è passata attraverso l'LSTM, l'output dell'ultimo timestep viene inviato a un layer Fully Connected che

riduce la dimensione dell'output alla dimensione finale desiderata. Questo layer è fondamentale per produrre la previsione finale.

Inizializzazione dei Pesi I pesi del modello sono inizializzati seguendo diverse tecniche: i pesi del layer LSTM vengono inizializzati utilizzando Xavier Uniform per i pesi di ingresso, Ortogonalità per i pesi ricorrenti (ottima per evitare vanishing o exploding gradient nei layer ricorrenti), e i bias vengono inizializzati a zero.

Il layer fully connected utilizza anch'esso l'inizializzazione Xavier Uniform per i pesi, e zero initialization per i bias.

3.4 Implementazione degli ottimizzatori

La peculiarità di questo progetto è stata l'integrazione degli ottimizzatori definiti in classe ("a mano") nelle funzioni per il training della rete che la libreria Torch offre *off the shelf*.

In seguito, essi sono anche stati confrontati con i corrispettivi ottimizzatori offerti da Torch.

3.4.1 Cos'è un ottimizzatore?

Gli ottimizzatori sono algoritmi utilizzati nell'addestramento di modelli di machine learning per aggiornare i parametri del modello, minimizzando la funzione di perdita. Ogni ottimizzatore segue una strategia differente per la correzione e l'adattamento dei pesi nel tempo, migliorando l'efficienza e la stabilità dell'apprendimento. Di seguito una panoramica dei principali ottimizzatori implementati.

3.4.2 Gli ottimizzatori analizzati

SGD L'SGD è l'ottimizzatore più semplice e diretto, ed è la base di molte altre tecniche di ottimizzazione. Funziona aggiornando i parametri del modello in base al gradiente della funzione di perdita, e necessita di un solo iperparametro: il learning rate.

AdaGrad AdaGrad è un algoritmo adattivo che modifica il tasso di apprendimento per ogni parametro, aumentando la dimensione del passo per parametri con gradienti piccoli e riducendo la dimensione del passo per parametri con gradienti grandi.

L'ottimizzatore mantiene una somma accumulata dei quadrati dei gradienti, e divide il gradiente per la radice quadrata di questa somma accumulata.

RMSProp RMSProp è un ottimizzatore che migliora l'SGD applicando un adattamento del tasso di apprendimento per

ogni parametro. Questo viene fatto tenendo traccia della moving average dei quadrati dei gradienti.

Esso aggiorna i parametri dividendo il gradiente per la radice quadrata della media dei quadrati dei gradienti, riducendo così il tasso di apprendimento quando il gradiente è grande.

ADAM Adam è uno degli algoritmi di ottimizzazione più popolari grazie alla sua efficienza e capacità di adattarsi alle variazioni del gradiente. Funziona combinando i vantaggi di *Momentum* e *RMSProp*, utilizzando due stime momentanee: una per il momento primo (media del gradiente) e una per il momento secondo (varianza del gradiente).

3.5 Alcune osservazioni tecniche

Durante l'allenamento, sono affiorate alcune interessanti considerazioni:

- SGD e AdaGrad necessitano di un learning rate iniziale più alto degli altri ottimizzatori. Nel primo ciò è necessario per una convergenza veloce, nel secondo perché tale iperparametro è inversamente proporzionale alla radice quadrata della somma accumulata dei quadrati dei gradienti, quindi tende a decrescere rapidamente.
- Un batch di 32 datapoint è un buon compromesso tra stabilità e velocità di addestramento.

4 Risultati

Si veda la tabella 2 per i risultati presentati di seguito. L'immagine sottostante mostra la loss ottenuta con gli ottimizzatori in esame.

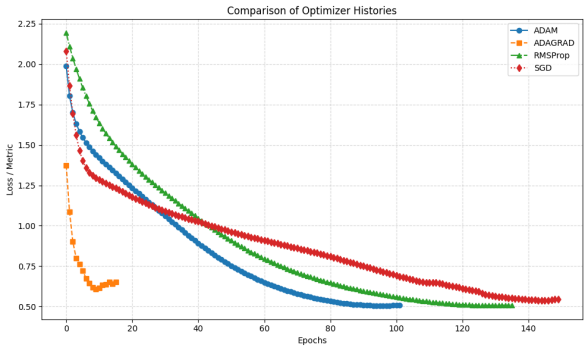

Figura 3: Grafico delle validation loss per ogni ottimizzatore (con early stopping).

4.1 ADAM

L'ottimizzatore ADAM mostra prestazioni sufficienti nelle metriche considerate: durante l'addestramento, la validation loss diminuisce più rapidamente rispetto ad altri metodi (si veda figura 3), ma questo comportamento può portarlo a rimanere intrappolato in un minimo locale, interrompendo prematuramente il miglioramento. In termini di tempo di esecuzione e numero di epoche ne-

Tabella 2: Risultati dell'allenamento con early stopping (pazienza pari a 6 epoche) utilizzando diversi ottimizzatori implementati a mano.

Optimizer	Time (s)	MSE	MAE	R^2	Test Loss	Epochs
ADAM	79,01	0,5720	0,5713	0,5009	1,7270	102
AdaGrad	10,10	0,5393	0,5618	0,5294	1,6312	16
RMSprop	80,04	0,5298	0,5457	0,5377	1,6015	136
SGD	86,11	0,5169	0,5484	0,5490	1,5699	150

cessarie, le sue prestazioni risultano inferiori rispetto agli altri ottimizzatori.

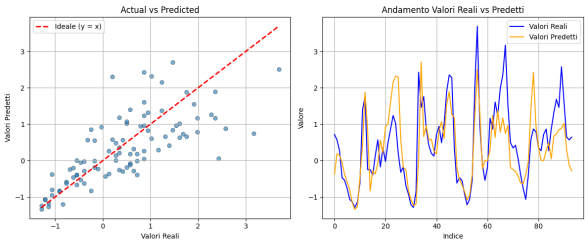

Figura 4: Adam: confronto tra dati reali e predetti.

4.2 AdaGrad

AdaGrad soffre di una rapida diminuzione del learning rate, che causa una riduzione iniziale veloce della loss durante le prime epoche (si veda figura 3), seguita però da un arresto del miglioramento. Questo limite compromette la capacità di raggiungere risultati ottimali.

4.3 RMSProp

Analizzando i risultati ottenuti utilizzando gli ottimizzatori definiti durante il corso (tabella 2), si osserva che RMSprop offre le migliori performance in termini di MAE, MSE e R^2. Sebbene il tempo di allenamento e il numero di epoche siano più elevati rispetto agli altri metodi, questo comportamento consente a RMSprop di convergere in modo più graduale (si veda la figura 3), evitando di rimanere intrappolato in minimi locali e raggiungendo minimi globali più stabili.

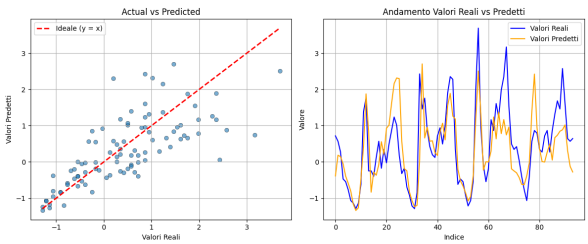

Figura 5: AdaGrad: confronto tra dati reali e predetti.

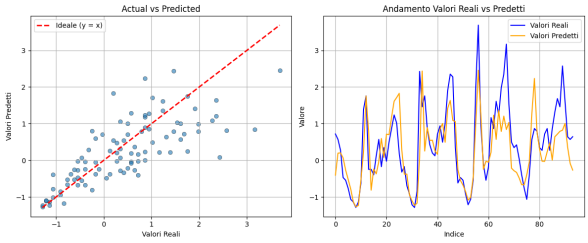

Figura 6: RMSProp: confronto tra dati reali e predetti.

4.4 SGD

SGD si distingue come il metodo più lento: dopo una discesa rapida della loss nelle fasi iniziali, il miglioramento prosegue a un ritmo graduale e quasi costante, richiedendo un numero elevato di epoche per convergere (figura 3). Tuttavia, questo comportamento gli permette di ottenere prestazioni finali competitive.

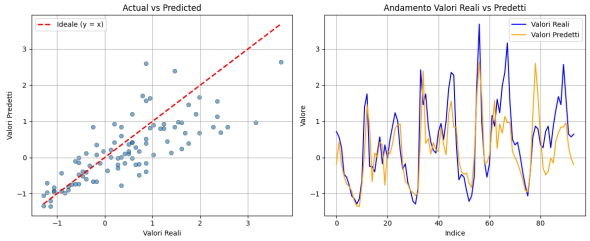

Figura 7: SGD: confronto tra dati reali e predetti.

4.5 Confronto con ottimizzatori di Pytorch

Ripetendo l'addestramento del modello utilizzando gli ottimizzatori predefiniti della libreria PyTorch, mantenendo invariato il learning rate, si osservano generalmente risultati migliori rispetto agli ottimizzatori implementati manualmente (si veda la tabella 3), con l'eccezione di SGD, dove la nostra versione si dimostra superiore.

Adam raggiunge prestazioni migliorate, riducendo significativamente sia il tempo sia il numero di epoche necessarie per l'addestramento.

AdaGrad mostra un comportamento più stabile: la diminuzione iniziale della loss non rallenta bruscamente (si veda 8), permettendo al processo di apprendimento di proseguire più a lungo.

RMSProp inizia con una discesa più graduale, garantisce una loss ben decrescente e un comportamento complessivamente soddisfacente.

Al contrario, le prestazioni di SGD peggiorano rispetto alla nostra implementazione, evidenziando un miglior adattamento dei parametri nella versione personalizzata.

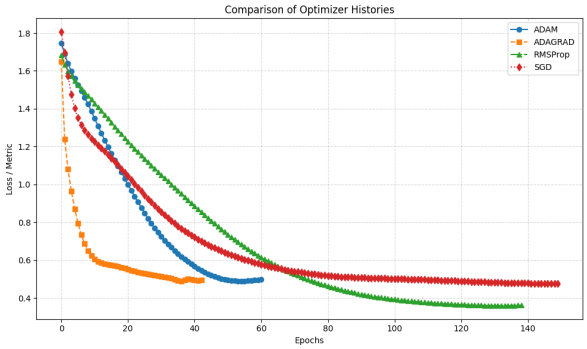

Figura 8: Grafico della validation loss con gli ottimizzatori definiti in Pytorch (con early stopping).

4.6 Confronto delle funzioni obiettivo in diverse epoche

A parità di epoche, AdaGrad converge più rapidamente rispetto agli altri ottimizzatori (figura 9), risultando la scelta migliore se

Tabella 3: Risultati dell'allenamento con early stopping (pazienza pari a 6 epoche) utilizzando diversi ottimizzatori offerti da Torch.

Optimizer	Time (s)	MSE	MAE	R^2	Test Loss	Epochs
ADAM	38,84	0,4857	*0.5167*	0,5762	1,4629	61
AdaGrad	35,70	*0.4836*	0,5353	0,5780	1,4659	43
RMSprop	91,30	0,4969	0,5365	0,5664	1,5034	139
SGD	86,78	0,6198	0,6201	0,4591	1,8854	150

l'obiettivo è la velocità (meno di 40 epoche). Tuttavia, prolungando l'addestramento, tutti gli algoritmi, ad eccezione di SGD, mostrano una tendenza all'overfitting e finiscono per bloccarsi (figura 10).

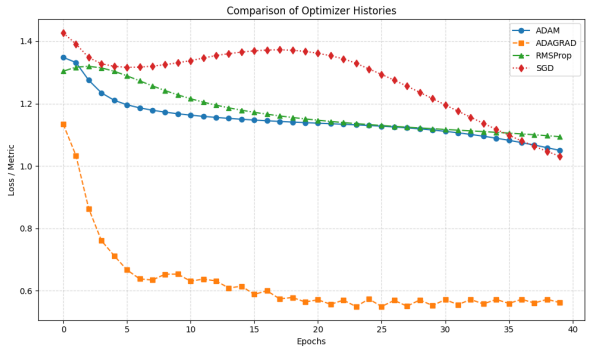

Figura 9: Loss in 40 epoche.

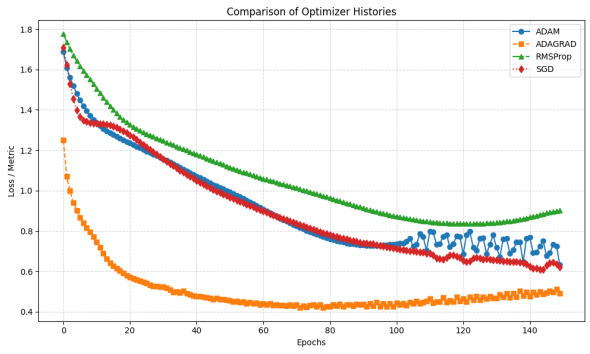

Figura 10: Loss in 150 epoche.

Riferimenti bibliografici

[1] S. De Vito et al. «On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario». In: *Sensors and Actuators B: Chemical* 129.2 (2008), pp. 750–757.

[2] Garima Jain e Bhawna Mallick. «A Review on Weather Forecasting Techniques». In: *International Journal of Advanced Research in Computer and Communication Engineering* 5.12 (2016).

[3] Rohan Kosandal. *Weather forecasting with Recurrent Neural Networks in Python*. 2019. URL: `https://medium.com/analytics-vidhya/weather-forecasting-with-recurrent-neural-networks-1eaa057d70c3`.

[4] Iryna Talamanova et al. «Data-driven Real-time Short-term Prediction of Air Quality: Comparison of ES, ARIMA, and LSTM». In: *International Conference on Intelligent Systems Design and Applications*. 2022, pp. 322–331.