Anomaly Transformer: un'architettura basata sull'attenzione per il rilevamento di anomalie nelle serie temporali

Introduzione

Il rilevamento di anomalie nelle serie temporali è un compito cruciale in diversi ambiti applicativi, come il monitoraggio di sistemi, l'esplorazione spaziale e il trattamento delle acque. L'obiettivo è identificare osservazioni o pattern che deviano significativamente dal comportamento normale del sistema, segnalando potenziali problemi o guasti. Tradizionalmente, questo problema è stato affrontato con tecniche statistiche classiche o modelli di deep learning basati su reti neurali ricorrenti (RNN). Tuttavia, queste metodologie presentano limitazioni nella capacità di catturare relazioni a lungo termine e di distinguere in modo efficace anomalie rare da normali fluttuazioni nei dati.

L'**Anomaly Transformer**, proposto da *Jiehui Xu et al. (2022)*, rappresenta un'innovativa architettura di rete neurale basata sul meccanismo dell'attenzione, progettata specificamente per il rilevamento di anomalie nelle serie temporali in modo non supervisionato. L'idea chiave alla base di questo modello è che le anomalie, a causa della loro rarità, hanno difficoltà a stabilire associazioni significative con l'intera serie temporale, concentrando le loro relazioni principalmente sui punti temporali adiacenti. Questa intuizione, definita come "bias di concentrazione adiacente" (*adjacent-concentration bias*) , fornisce un criterio discriminante intrinseco tra punti normali e anomali, che gli autori sfruttano attraverso il concetto di "Associazione Discrepanza" (*Association Discrepancy*).

Il meccanismo dell'attenzione

Prima di addentrarci nei dettagli dell'Anomaly Transformer, è fondamentale comprendere il meccanismo dell'attenzione, che costituisce il nucleo di questa architettura e, più in generale, dei modelli Transformer. L'attenzione, introdotta nel celebre articolo "**Attention Is All You Need**" di *Vaswani et al. (2017)*, ha rivoluzionato il campo dell'elaborazione del linguaggio naturale e si è dimostrata efficace anche in altri domini, come la visione artificiale e l'analisi delle serie temporali.

In sostanza, l'attenzione consente a un modello di pesare dinamicamente l'importanza di diverse parti dell'input durante l'elaborazione di ogni elemento dell'output. Questo meccanismo permette di catturare relazioni a lungo termine e di modellare dipendenze complesse tra elementi distanti nella sequenza, superando le limitazioni delle reti neurali ricorrenti tradizionali (RNN).

Matematicamente, l'attenzione può essere descritta come una funzione che mappa una query e un insieme di coppie chiave-valore in un output, dove query, chiavi, valori e output sono tutti vettori. L'output viene calcolato come una somma pesata dei valori, dove il peso

assegnato a ciascun valore è determinato da una funzione di compatibilità tra la query e la chiave corrispondente. Una delle implementazioni più comuni dell'attenzione è la "*Scaled Dot-Product Attention*", descritta dall'equazione:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

dove:

- Q è la matrice delle query
- K è la matrice delle chiavi
- V è la matrice dei valori
- d_k è la dimensione delle chiavi

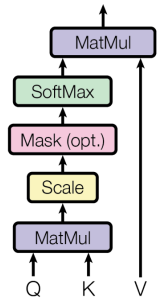
Questa formula calcola l'attenzione su un insieme di query simultaneamente. In primo luogo, vengono calcolati i prodotti scalari tra la query e tutte le chiavi, che vengono poi saldati per $\sqrt{d_k}$ e normalizzati tramite la funzione softmax per ottenere i pesi da applicare ai valori. La somma pesata dei valori produce infine l'output dell'attenzione. Il fattore di scala $\sqrt{d_k}$ aiuta a stabilizzare i gradienti durante l'addestramento.

Un'estensione dell'attenzione è la **self-attention multi-head**, la quale permette al modello di considerare simultaneamente diverse rappresentazioni dello stesso input. In pratica, invece di eseguire una singola funzione di attenzione, vengono eseguite diverse funzioni di attenzione in parallelo, ciascuna su una proiezione lineare diversa dell'input. I risultati di queste attenzioni parallele vengono poi combinati per produrre l'output finale.

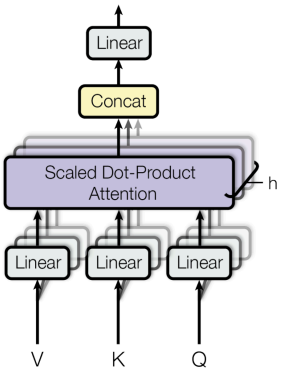
$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_n)W^O$$

$$\text{Dove } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Scaled Dot-Product Attention



Multi-Head Attention



Architettura classical del Transformer

Il Transformer adotta una struttura encoder-decoder, comune nei modelli di trasduzione di sequenze. L'encoder elabora la sequenza di input e la mappa in una rappresentazione continua, mentre il decoder genera la sequenza di output un elemento alla volta, basandosi sulla rappresentazione dell'encoder e sugli output precedentemente generati.

Sia l'encoder che il decoder sono composti da stack di layer identici. Ogni layer dell'encoder contiene due sub-layer: un meccanismo di self-attention multi-head e un semplice feed-forward network completamente connesso. Ogni layer del decoder contiene tre sub-layer: i due sub-layer dell'encoder più un ulteriore sub-layer di attenzione encoder-decoder multi-head.

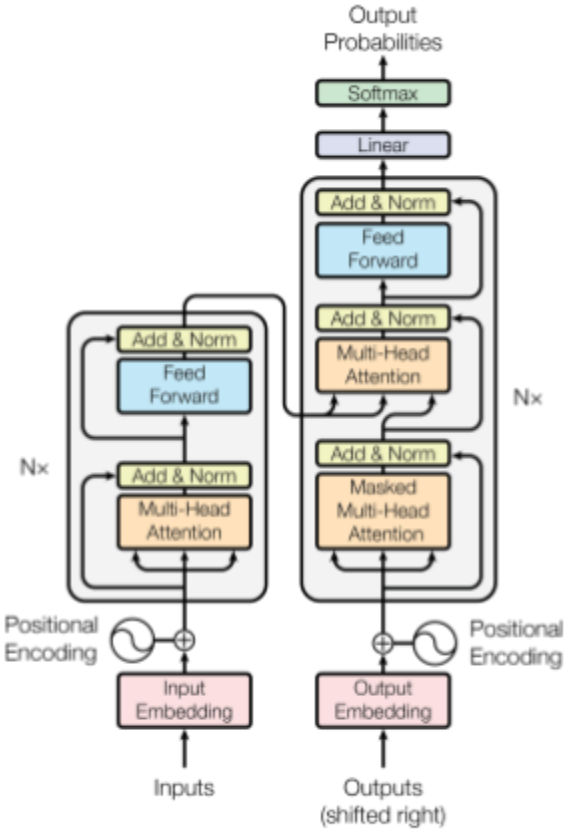


Figura 1: Transformer - architettura

Il Transformer utilizza la self-attention in tre modi diversi:

1. **Self-attention nell'encoder**: consente a ciascuna posizione nell'encoder di prestare attenzione a tutte le posizioni nel layer precedente dell'encoder.
2. **Self-attention nel decoder**: consente a ciascuna posizione nel decoder di prestare attenzione a tutte le posizioni nel decoder fino a quella posizione inclusa. Questo è necessario per mantenere la proprietà autoregressiva del decoder.

3. **Attenzione encoder-decoder**: consente a ciascuna posizione nel decoder di prestare attenzione a tutte le posizioni nella sequenza di input.

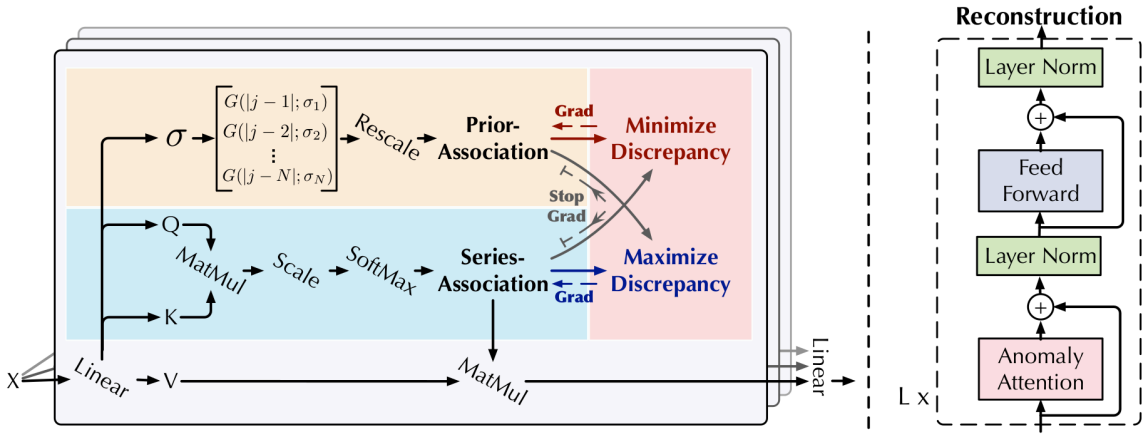
Codifica Poszionale

Poiché il Transformer non ha ricorrenza né convoluzione, non ha una nozione intrinseca dell'ordine delle parole in una sequenza. Per ovviare a questo, il modello utilizza una "codifica posizionale" che viene aggiunta agli embeddings di input in fondo agli stack dell'encoder e del decoder. Questa codifica fornisce al modello informazioni sulla posizione relativa o assoluta dei token nella sequenza.

$$PE(pos,\ 2i) \ = \ sin(pos/10000^{\ 2i/d_{model}})$$

$$PE(pos,\ 2i\ +\ 1) \ = \ cos(pos/10000^{\ 2i/d_{model}})$$

L'architettura dell'Anomaly Transformer



L'Anomaly Transformer[1], illustrato in figura, segue una struttura simile ai modelli Transformer ma priva di decoder e che introduce un nuovo meccanismo di attenzione chiamato **Anomaly Attention** per calcolare la discrepanza di associazione (Association Discrepancy). Questo meccanismo, cuore dell'Anomaly Transformer, è progettato per modellare sia l'associazione a priori (**prior-association**) che l'associazione di serie (**series-association**) di ciascun punto temporale.

Il modello fa uso di uno stack di livelli (L, tipicamente 3) di Anomaly Transformer ciascuno dei quali fa uso di più multi-head (h, tipicamente 8) Anomaly Attention per

[1] Per i dettagli implementativi del modello Anomaly Transformer fare riferimento al package model del progetto Python allegato al presente documento.

processare l'input ($h = 8$).

Formalmente, data in input la serie $X \in R^{Nxd}$ il modello si definisce come:

$$Z^l = LayerNorm(AnomalyAttention(X^{l-1}) + X^{l-1})$$

$$X^l = LayerNorm(FeedForward(Z^l) + Z^l)$$

Dove $X^l \in R^{Nxd_{model}}$ rappresenta l'output del livello l-esimo con d_{model} canali, $Z^l \in R^{Nxd_{model}}$ rappresenta il livello l-esimo nascosto e $X^0 \in R^{Nxd_{model}} = Embedding(X)$ è l'input dato al primo livello del modello.

L'associazione a priori (*prior-association*) rappresenta il *bias di concentrazione adiacente*, catturato da un kernel gaussiano con un parametro di scala σ apprendibile. Rappresenta una misura dell'associazione dei punti temporali con i loro vicini.

L'associazione di serie (*series-association*), invece, viene appresa direttamente dai dati grezzi attraverso il meccanismo di self-attention e rappresenta una misura dell'associazione dei punti temporali con il resto della serie.

Inizializzazione: $Q, K, V, \sigma = X^{l-1}W^l_Q, X^{l-1}W^l_K, X^{l-1}W^l_V, X^{l-1}W^l_\sigma$

Prior-Association: $P^l = Rescale\left(\left[\frac{1}{\sqrt{2\pi}\sigma_i} exp\ -\left(\frac{|j-i|^2}{2\sigma_i^2}\right)\right]_{i,j\in\{1,..,N\}}\right)$

Series-Association: $S^l = softmax\left(\frac{QK^T}{\sqrt{d_{model}}}\right)$

Reconstruction: $\widehat{Z}^l = S^l V$

Dove:

- $Q, K, V \in R^{Nxd_{model}}$, $\sigma \in \Re^{Nx1}$ sono i comuni parametri della self-attention con l'aggiunta del parametro appreso di scala
- $W^l_Q, W^l_K, W^l_V \in \Re^{d_{model}xd_{model}}$
- $W^l_\sigma \in \Re^{d_{model}x1}$
- $P^l, S^l \in \Re^{NxN}$
- $Rescale$ è la divisione delle righe della matrice data per la somma delle stesse.

Nella versione *multi-head* proposta dagli autori si ha $Q, K, V \in \Re^{Nx\frac{d_{model}}{h}}$, $\sigma \in \Re^{Nxh}$ per h

head di Anomaly Attention. I blocchi uniscono l'output dei livelli nascosti $\widehat{Z}_m^l$ per ottenere il risultato finale (ricostruzione) $\widehat{Z}^l$.

Il risultato finale fornito dal modello X^l è la serie temporale *"ricostruita"*, ovvero la serie temporale processata dal modello. Essa rappresenta informalmente una serie derivata da quella fornita in ingresso utile per rilevare le anomalie sotto opportune elaborazioni.

La discrepanza tra *prior-association* e *series-association* costituisce la base per il rilevamento delle anomalie. Formalmente, l'Associazione Discrepanza (*Association Discrepancy*) è definita come la divergenza Kullback-Leibler simmetrizzata tra l'associazione a priori e l'associazione di serie per tutti i livelli usati.

$$AssDis(P,S;X) = \left[\frac{1}{L}\sum_{l=1}^{L}\left(KL\left(P^l_{i,:}||S^l_{i,:}\right) + KL\left(S^l_{i,:}||P^l_{i,:}\right)\right)\right]_{i=1,\dots,N}$$

Dove $AssDis(P,S;X) \in \Re^{Nx1}$ è la l'Associazione Discrepanza per ogni punto della serie temporale e $KL(P,Q) = \sum_{x\in X}(P(x)log(\frac{P(x)}{Q(x)}))$. Puntualmente, le anomalie presentano un basso valore di Associazione Discrepanza, infatti la divergenza *Kullback-Leibler* è una misura della similarità tra due distribuzioni di probabilità e nel caso di anomalie si ottiene una differenza tra le distribuzioni *prior-association* e *series-association*. Infatti, le anomalie non riescono a stabilire associazioni significative con l'intera serie temporale (series-association) ma solo con i punti a loro vicini (prior-association).

Apprendimento Minimax dell'Associazione

Per migliorare la capacità del modello di distinguere tra punti normali e anomali, gli autori propongono una strategia di apprendimento minimax. Questa strategia mira ad amplificare la discrepanza di associazione tra i due tipi di punti (anomali e non).

- **Fase di minimizzazione:** In questa fase, la *prior-association* viene guidata ad approssimare la *series-association* appresa dai dati grezzi. Questo processo consente alla *prior-association* di adattarsi ai diversi pattern temporali presenti nei dati.

- **Fase di massimizzazione:** In questa fase, la *series-association* viene ottimizzata per aumentare la discrepanza di associazione. Questo spinge la *series-association* a concentrarsi maggiormente su relazioni a lungo termine, rendendo più difficile la ricostruzione delle anomalie e, di conseguenza, più facile la loro identificazione.

La funzione di perdita complessiva del modello incorpora sia l'errore di ricostruzione che

la discrepanza di associazione, bilanciate da un iper-parametro λ:

$$L_{total}(\widehat{X}, P, S, \lambda; X) = ||X - \widehat{X}||^2 - \lambda||AssDis(P, S; X)||_1$$

In particolare nelle fasi di minimizzazione e massimizzazione si usano due varianti della funzione di perdita:

- **Minimizzazione**: $L_{total}(\widehat{X}, P, S_{detach}, -\lambda; X)$
- **Massimizzazione**: $L_{total}(\widehat{X}, P_{detach}, S, \lambda; X)$

Dove *detach* significa fermare la backpropagation del gradiente, ovvero nelle fasi di minimizzazione e massimizzazione si fissano rispettivamente i gradienti relativi ai persi per la a *series-association* e la *prior-association*. Questo consente che la fase di massimizzazione o minimizzazione impatti una sola delle due *-association.

Criterio di anomalia basato sull'associazione

Il criterio finale per il rilevamento di anomalie combina la discrepanza di associazione normalizzata con il criterio di ricostruzione:

$$AnomalyScore(X) = Softmax(-AssDis(P, S; X)) \odot \left[\left|\left| X_{i,:} - \widehat{X}_{i,:} \right|\right|_2^2 \right]_{i=1,\dots,N}$$

Dove ⊙ è il prodotto elemento per elemento. Questo criterio sfrutta sia le informazioni sulla rappresentazione temporale che sulla discrepanza di associazione, consentendo al modello di rilevare anomalie in modo più accurato ed efficace.

La classificazione finale dei punti anomali si effettua stabilendo una soglia δ sul valore di *anomaly-score* oltre la quale il punto temporale è identificato come anomalo. Ci sono varie strategie per definire questa soglia, gli autori propongono di usare **K-Means** con il metodo della **Gap Statistics** (Tibshirani et al., 2001) per trovare il cluster non il maggiore anomaly score e definire δ in modo da catturare gli r punti nel cluster.

Analisi di sensitività degli iperparametri sul modello Anomaly Transformer

Sebbene gli autori del modello Anomaly Transformer abbiano studiato la sensitività degli iperparametri del modello, si effettua comunque uno studio in tal senso mirato alla scoperta del comportamento del modello quando non è possibile usare un alta dimensionalità d_{model} uguale o superiore 512. Nello specifico, si è deciso di procedere con la **grid search** in modo da testare tutti i parametri scelti facendo uso del solo dataset **MSL** (*Mars Science Laboratory*) a causa della lunghezza e della potenza di calcolo richiesta dalla procedura.

Specifichiamo inoltre che, dati i lunghi tempi di training del modello, si è deciso di ridurre d_{model} a 64 e il numero di *epoch* di training a sole 2. Tale decisione è giustificata dai seguenti dati sperimentali, infatti cambiano solo gli iperparametri specificati e lasciano gli altri come suggerito dagli autori, ovvero:

- **Learning rate**: 10^{-4}
- **Lambda** λ: 1, 3
- **Anomaly ratio**: 1
- **Batch Size**: 128
- **Numero di livelli** l: 3
- **Numero di attention heads** h: 8
- **Tipo di Kernel**: Gaussiano
- **Funzione di perdita**: Norma L2 (MSE)

Non si ottiene una differenza sostanziale in termini di prestazioni predittive del modello.

	Accuracy	**Precision**	**Recall**	**F-Score**
$d_{model} = 512$ $epoch = 3$	0.9845047	0.9181818	0.9363893	**0.9271962**
$d_{model} = 64$ $epoch = 2$	0.9865807	0.9205660	0.9550605	**0.9374961**

Questo risultato evidenzia che nel caso di dati ad alta dimensionalità, come nel caso del dataset MSL con $d = 55$, anche a fronte di una sostanziale riduzione della dimensionalità del modello si ottengono ottime performance predittive.

Dunque, le configurazioni[2] di funzioni e iperparametri usate sono le seguenti:

- **Learning rate**: 10^{-4}
- **Lambda** λ: 0.5, 3
- **Anomaly ratio**: 1
- **Epochs**: 2
- **Batch Size**: 128
- **Dimensione** d_{model}: 64
- **Numero di livelli** l: 3
- **Numero di attention heads** h: 8
- **Tipo di Kernel**: Gaussiano, No (*prior-association* come parametro appreso) e sigmoide.
- **Funzione di perdita**[3]: Norma L1, Norma L2 (MSE), Cross Entropy e Divergenza KL

Per un totale di 24 possibili combinazioni totali.

Dove, data la distanza $dist_{i,j} = |i - j|_{i,j\in[1,..N]}$ e il parametro appreso σ_i le funzioni usate come kernel sono

$$Gaussian = \left[\frac{1}{\sqrt{2\pi\sigma_i}} exp\left(-\frac{dist_{i,j}}{2\sigma_i^2}\right)\right]_{i,j\in\{1,..,N\}}$$

$$NO = Softmax(\sigma, dim = -1)$$

$$Sigmoid = tanh(dist_{i,j} * \sigma_i)$$

La migliore combinazione trovata in termini di funzioni e iperparametri in base alla metrica f-score è la seguente.

- **Learning rate**: 10^{-4}
- **Lambda** λ: 3
- **Anomaly ratio**: 1
- **Epochs**: 2
- **Batch Size**: 128

[2] Tali configurazioni sono limitate dall'hardware a disposizione, infatti si dispone di una Nvidia RTX 4060 con soli 8GB di memoria video a fronte di una RTX Titan da 24GB usata dagli autori.

[3] Per i dettagli implementativi delle funzioni di perdita usate rimandiamo alla visione della documentazione del package *torch.nn* disponibile qui.

- **Dimensione** d_{model}: 64
- **Numero di livelli** l: 3
- **Numero di attention heads** h: 8
- **Tipo di Kernel**: No (prior-association come parametro appreso).
- **Funzione di perdita**: Divergenza KL

Le metriche di classificazione ad essa associata sono:

Accuracy	**Precision**	**Recall**	**F-Score**	**Train Time (s)**
0.9890773	0.9209094	0.9805563	**0.9497973**	160.2966

Basandosi sulla metrica *F-Score* possiamo osservare un miglioramento dello 1,23% rispetto a quanto ottenuto usando gli iperparametri suggeriti dagli autori, tuttavia questa piccola differenza potrebbe essere dovuta ad una marginale randomicità presente nelle implementazioni (pytorch) dei metodi descritti. Infatti, eseguendo altre due volte il processo di training e test la media della metrica *F-Score* ammonta a 0,936436. Questo evidenzia come non vi sia un concreto e significativo miglioramento delle prestazioni rispetto all'uso degli iperparametri consigliati dagli autori.

Inoltre, sempre in termini di *F-Score* i valori prodotti dalle singole configurazioni testate nella grid search variano da $\simeq 0,91$ a $\simeq 0,949$, dunque non solo non si evidenzia alcun sostanziale miglioramento ma si evidenzia soprattutto la robustezza del modello a varie configurazioni di funzioni e parametri. Ipotizziamo che questa particolare proprietà del modello sia dovuta alla struttura minimax che in fase di training riesce comunque a guidare il modello verso un'ottima identificazione delle anomalie.

Algoritmi di ottimizzazione per il modello Anomaly Transformer

L'algoritmo di ottimizzazione scelto dagli autori è **ADAM** (*Adaptive Moment Estimation*), ovvero un algoritmo che si basa sulla stima adattiva dei momenti del primo e del secondo ordine dei gradienti. In particolare ADAM risulta essere ben adatto a problemi di grandi dimensioni e computazionalmente efficiente in termini di tempo e memoria.

Analizziamo ora in termini di metriche per la predizione e tempo di training le prestazioni offerte anche da altri algoritmi di ottimizzazione, in particolare concentriamo la nostra attenzione sui seguenti[4].

- **SGD** (*Stochastic Gradient Descent*): Un ottimizzatore di base che aggiorna i parametri in base al gradiente negativo della funzione di perdita. Può essere combinato con tecniche come momentum e weight decay.
- **RMSprop** (*Root Mean Square Propagation*): Mantiene una media mobile esponenziale del quadrato dei gradienti passati e divide il gradiente per la radice quadrata di questa media. Questo aiuta ad affrontare il problema dei gradienti che scompaiono o esplodono.
- **Adadelta**: Simile ad *Adagrad* (*Adaptive Gradient Algorithm*), adatta la velocità di apprendimento per ogni parametro in base alla somma dei suoi gradienti quadrati passati. I parametri con gradienti grandi avranno velocità di apprendimento ridotte, mentre quelli con gradienti piccoli avranno velocità di apprendimento più elevate. *Adadelta* però, non accumula tutti i gradienti passati. Invece, mantiene una media mobile esponenziale dei gradienti quadrati.
- **AdamW**: Una variante di *Adam* con weight decay disaccoppiato (Adam usa pesi di decadimento β_1e β_2 per i momenti del primo e del secondo ordine). Questo può portare a una migliore generalizzazione.

Le prove effettuate sono state fatte variando solo l'algoritmo di ottimizzazione usato dal modello e lasciando gli altri parametri come quelli consigliati dagli autori, sempre a meno di d_{model} e del numero di epoche, ridotti per motivazioni temporali.

- **Learning rate**: 10^{-4}
- **Lambda** λ: 3
- **Anomaly ratio**: 1
- **Epochs**: 2
- **Batch Size**: 128

[4] Per comprendere a fondo le caratteristiche di ciascuno degli algoritmi testati rimandiamo alla visione della documentazione del package *torch.optim* disponibile qui.

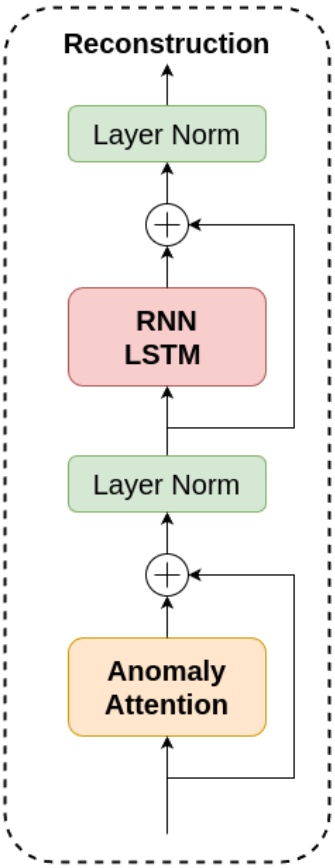
- **Dimensione** d_{model}: 64
- **Numero di livelli** l: 3
- **Numero di attention heads** h: 8
- **Tipo di Kernel**: Gaussiano
- **Funzione di perdita**: Norma L2 (MSE)

I risultati ottenuti sono riportati nella seguente tabella.

	Accuracy	**Precision**	**Recall**	**F-Score**	**Train Time (S)**
Adam	0.9863501	0.9194589	0.9540304	0.9364257	**199.2156**
AdamW	0.9862144	0.9189424	0.9532578	0.9357856	**199.3804**
SGD	0.9896608	0.9215214	0.9858357	0.9525943	**199.6484**
Adadelta	0.9875848	0.9199461	0.9662632	0.9425360	**200.2806**
RMSprop	0.9857802	0.9179940	0.9499099	0.9336793	**200.7332**

Come è lecito aspettarsi l'algoritmo Adam è risultato il più performante dal punto di vista temporale, confermando ancora una volta la bontà di tale algoritmo nell'addestramento di modelli. Inoltre, come visto in precedenza, le piccole fluttuazioni delle metriche inerenti alla performance predittiva sono del tutto riconducibili alla casualità dell'implementazione del modello sottostante.

Uso di RNN in combinazione con Anomaly Transformer



Viste le buone prestazioni fornite dal modello, anche a seguito di una sua semplificazione, si è pensato se si potesse ottenere un miglioramento delle prestazioni mediante l'applicazione di **reti neurali ricorrenti** (RNN) di tipo LSTM (*Long-Short Term Memory*).

Le **Long Short-Term Memory** (LSTM) sono un tipo di unità ricorrente progettata per superare il problema della memoria a breve termine. Le LSTM introducono un meccanismo sofisticato di "cancelli" che regolano il flusso di informazioni attraverso la rete, consentendo loro di ricordare informazioni rilevanti per periodi più lunghi e dimenticare quelle irrilevanti.

Nello specifico, si è pensato di usare uno o più livelli di una rete RNN di tipo LSTM al posto del blocco Feed Forward per cercare aumentare la capacità del modello di riconoscere le anomalie andando a cercare relazioni di lungo/breve termine nella ricostruzione della serie ottenuta con il meccanismo dell'attenzione. In particolare, l'architettura modificata secondo questa idea è riportata nel diagramma a sinistra di questo paragrafo.

Le prove effettuate comprendono l'uso di diversi numeri di livelli della RNN partendo da un minimo di 1 e terminando con un massimo di 16. Tali prove sono state effettuate usando gli iperparametri consigliati dagli autori e le semplificazioni precedentemente introdotte, come si riporta di seguito.

- **Learning rate**: 10^{-4}
- **Lambda** λ: 3
- **Anomaly ratio**: 1
- **Epochs**: 2
- **Batch Size**: 128
- **Dimensione** d_{model}: 64
- **Numero di livelli** l: 3
- **Numero di attention heads** h: 8
- **Tipo di Kernel**: Gaussiano
- **Funzione di perdita**: Norma L2 (MSE)

I risultati ottenuti sono riportati nella seguente tabella

RNN Layers	Accuracy	Precision	Recall	F-Score	Train Time (S)
0	0.9874491	0.9198478	0.9649756	**0.9418714**	199.9097
1	0.9892809	0.9203423	0.9833891	**0.9508217**	202.9323
2	0.9875441	0.9175610	0.9688385	**0.9425028**	208.6800
4	0.9880054	0.9212782	0.9689673	**0.9445211**	216.3183
8	0.9849254	0.9196620	0.9389647	**0.9292131**	234.4055
16	0.9871642	0.9189189	0.9631728	**0.9405256**	271.9105

Dove 0 come numero di livelli RNN indica il modello originale.

In termini di *F-Score* i risultati migliori si ottengono nel caso in cui si usa un solo livello per la rete RNN LSTM, tuttavia ancora una volta ciò è dovuto alla casualità intrinseca nei dettagli implementativi. Infatti, eseguendo il training e il test del modello due volte e facendo la media della metrica *F-Score* si ottiene un valore di *0.9353132*, completamente in linea con i risultati precedentemente ottenuti senza l'uso di RNN.

Il modello Anomaly Transformer si conferma dunque non solo robusto rispetto a cambiamenti negli iperparametri ma anche a cambiamenti infrastrutturali. Questo risultato evidenzia ancora una volta la bontà del modello e dell'idea alla base. Dunque, possiamo concludere questa fase di sperimentazione affermando che la strategia minimax alla base, volta a minimizzare l'Association Discrepancy per le anomalie, permette di ottenere sempre ottimi risultati.

Anomaly Attention vs Self Attention per Anomaly Detection

Come ultima prova sperimentale del lavoro condotto sul modello Anomaly Transformer ci siamo chiesti quale sia il reale vantaggio di questo modello rispetto all'uso di un modello con sola attenzione classica (transformer) per la ricerca di anomalie. Tale questione, è lecita vista l'impatto avuto dal meccanismo dell'attenzione nella sostituzione delle RNN in diversi campi applicativi.

Per confrontare il meccanismo dell'Anomaly Attention con il normale meccanismo dell'attenzione (Self Attention) descritto nelle fase iniziali di questo documento, definiamo quindi un modello che faccia uso della **Multi-Head Attention** per ottenere la ricostruzione della serie. Essendo il modello Anomaly Transformer praticamente privo di decoder, usiamo lo stack di encoder del transformer interpretando l'output come ricostruzione della serie. Il modello viene allenato usando l'errore di ricostruzione.

Formalmente, tale modello è definito come segue.

$$X^l = LayerNorm(FeedForward(A^{l-1}) + A^{l-1})$$

$$A^l = LayerNorm(MHAttention(X^{l-1}) + X^{l-1})$$

Dove

- $X^l \in R^{Nxd_{model}}$ rappresenta l'output del livello l-esimo con d_{model} canali
- $A^l \in R^{Nxd_{model}}$ rappresenta l'output del meccanismo dell'attenzione multi-head.

Inoltre, ricordando le definizioni precedentemente introdotte

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_n)W^O$$

$$\text{Dove } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

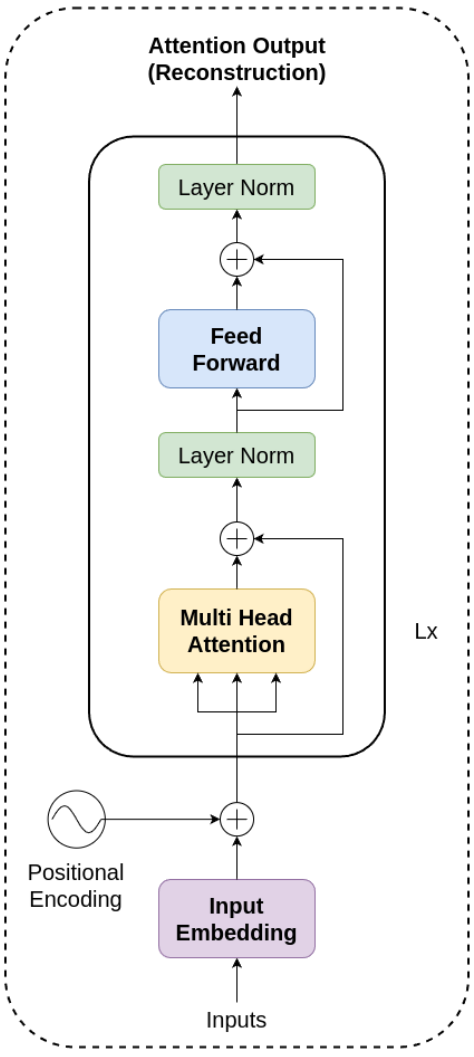
Possiamo definire

$$MHAttention(X^{l-1}) = MultiHead(X^{l-1}W^l_Q, X^{l-1}W^l_K, X^{l-1}W^l_V)$$

Dove:

- $Q, K, V \in R^{Nxd_{model}}$ sono i parametri della self-attention

- $W^{l}_{Q}, W^{l}_{K}, W^{l}_{V} \in \Re^{d_{model} x d_{model}}$ sono le matrici dei pesi per ottenere query Q, key K e value V.



In particolare il modello[5] sarà composto da E_{layers} livelli di encoder che produrranno in uscita la ricostruzione della serie. L'architettura di tale modello è riportata a sinistra di questo paragrafo.

La funzione di perdita usata per stimare l'errore è la norma al quadrato, ovvero:

$$L(X, \widehat{X}) = \left|\left|X - \widehat{X}\right|\right|^2$$

Dove $\widehat{X}$ è la ricostruzione della serie ottenuta in output dal modello finale.

In modo analogo a quanto visto per il transformer, la rete *Feed-Forward* è completamente connessa e consiste di due trasformazioni lineari con attivazione *ReLU* nel mezzo. Tale rete ha dimensione d_{ff} pari a $4d_{model}$ e formalmente corrisponde alla seguente formulazione.

$$FFN(X) = max(0, xW_1 + b_1)W_2 + b_2$$

Per quanto concerne l'embedding e il positional encoding vengono usati quelli dell'Anomaly Transformer in modo da ridurre al minimo le differenze tra i due e capire concretamente i reali vantaggi dell'Anomaly Attention rispetto alla Self-Attention. Anche il numero di livelli di stack encoder e di head del meccanismo dell'attenzione sono mantenuti uguali a quelli usati nell'Anomaly Transformer: $E_{layer} = 3$e $h = 8$.

La classificazione dei punti anomali è fatta in modo del tutto analoga a quanto avviene per il modello Anomaly Transformer, ovvero invece di usare Anomaly Score si usa l'errore di ricostruzione.

[5] Per comprendere i dettagli implementativi del modello descritto rimandiamo alla visione del package *self_attention* presente nel progetto Python di riferimento.

$$AnomalyScore(X) = \left[\left|\left| X_{i,:} - \widehat{X}_{i,:} \right|\right|_2^2 \right]_{i=1,\dots,N}$$

Si userà quindi una soglia δ sul valore dell'errore di ricostruzione per determinare se il punto analizzato è anomalo o meno.

Per comparare i due modelli abbiamo effettuato diversi cicli di training e test dei modelli usando iperparametri identici. In particolare per il modello Anomaly Transformer gli iperparametri sono stati lasciati ai valori predefiniti consigliati dagli autori a meno di quelli cambiati per la comparazione.

I risultati ottenuti usando i due modelli fanno esclusivo riferimento al dataset **MSL** (*Mars Science Laboratory*) per ragioni temporali.

Modello	**Accuracy**	**Precision**	**Recall**	**F-Score**	**Train Time**
Self-attention $d_{model} = 128$ $epoch = 3$	0.8515604	0.4048789	0.8698172	**0.5525562**	38.5978
Self-attention $d_{model} = 512$ $epoch = 5$	0.7999457	0.3189789	0.7916559	**0.4547337**	352.9563
Anomaly Attention $d_{model} = 128$ $epoch = 3$	0.9883039	0.9201558	0.9734741	**0.9460643**	312.0443
Anomaly Attention $d_{model} = 512$ $epoch = 5$	0.9840706	0.9182741	0.9317538	**0.9249648**	645.7070

Basandosi sulla metrica *F-Score* possiamo concludere che l'uso dell'**Anomaly Attention** garantisce un miglioramento delle prestazioni pari a $\simeq 43\%$ rispetto alla self-attention classifica. Inoltre, il modello che fa uso della self-attention classica produce valori bassi di *precision* indicando la presenza di numerosi **falsi positivi**, ovvero punti normali classificati come anomali.

Dettagli implementativi e codice sorgente

Per tutti i dettagli relativi alle implementazioni mostrate in questo documento fare riferimento al codice fornito in allegato o pubblicamente disponibile qui.

Conclusioni

L'Anomaly Transformer rappresenta un significativo passo avanti nel campo dei modelli non supervisionati per il rilevamento di anomalie nelle serie temporali. Sfruttando il meccanismo dell'attenzione e il concetto di discrepanza di associazione, questo modello è in grado di catturare relazioni complesse e di distinguere in modo efficace anomalie rare da normali fluttuazioni nei dati. I risultati sperimentali su diversi benchmark dimostrano la superiorità di questa architettura rispetto a metodi precedenti, aprendo nuove prospettive per l'applicazione del deep learning nel rilevamento di anomalie in contesti reali.