Alma Mater Studiorum · Università di Bologna

**Scuola di Ingegneria e Architettura**
**Corso di Laurea Magistrale in Ingegneria Informatica**
**Dipartimento di Informatica - Scienza e Ingegneria DISI**
**Tesi di laurea in Intelligent Systems**

# Anomaly Prediction with Temporal Convolutional Networks for HPC Systems

**Relatore:**
***Prof.ssa***
***MICHELA MILANO***

**Presentata da:**
***PIZZIGATI LORENZO***

**Correlatore:**
***Dott.***
***ANDREA BORGHESI***

**Sessione III**
**Anno Accademico 2018-2019**

*A Luca,*
*Mio Fratello*

# Indice

# Elenco delle figure

# 1 Introduzione

La tecnologia moderna ci offre le conoscenze e le infrastrutture necessarie per poter risolvere problemi molto complessi in tempi relativamente brevi. Tuttavia, prima dell'era contemporanea, non era possibile superare certi limiti relativi all'analisi sperimentale in molti ambiti della scienza (e non solo) come la fisica, la chimica, la biologia, ecc. In particolare, per poter effettuare certe analisi e simulazioni che spesso diamo per scontate è necessario un numero enorme di calcoli matematici, molto superiori a quelle che potrebbe effettuare la popolazione globale unita in un unica entità.

**La nascita dei primi calcolatori elettronici**, avvenuta dopo la seconda guerra mondiale, ha dato una spinta propulsoria non indifferente nella velocità con cui venivano effettuati questi calcoli, il limite indotto dall'anatomia del cervello umano nell'effettuare queste operazioni venne superato.

Così, più nuove scoperte venivano alla luce grazie alla disponibilità dei nuovi mezzi tecnologici, più l'avidità di conoscenza e di innovazione dell'uomo veniva alimentata e, pochi decenni più avanti, l'avanzamento tecnologico nel campo dei calcolatori elettronici (che comunque muoveva passi a ritmi elevati, seguendo la legge di Moore) iniziò ad essere superato dalle nuove necessità, in termini di potenza di calcolo, date dalle frontiere che si stavano aprendo alla scienza e all'industria.

**L'idea che diede un nuovo impulso alla capacità di calcolo di un singolo sistema di elaborazione** nascque durante gli anni 80' e fa riferimento ad un nuovo paradigma strutturale dei calcolatori elettronici. **Si passò così dall'esecuzione di algoritmi matematici su di una singola CPU ad una parallelizzazione su molteplici CPU**, aventi una memoria condivisa da cui ricevere i dati su cui eseguire calcoli complessi. Molti programmatori dovettero affrontare la sfida dell'adattamento ad un nuovo stile di programmazione, che passava da sequenziale a parallela. **Venne così standardizzato agli inizi degli anni 90' il primo sistema HPC (High Performance Computing)**.

## 1.1 Sistemi HPC

La strada per giungere alla creazione delle infrastrutture moderne di cui oggi disponiamo fu tuttavia tortuosa.

**Inizialmente, due differenti paradigmi** per la strutturazione di un sistema HPC presero campo. **Il primo**, che richiedeva più investimenti in termini economici ma garantiva prestazioni più elevate, faceva riferimento alla costruzione di un unico sistema altamente ottimizzato. **Il secondo**, più economico ma meno efficiente, consisteva nel collegare tra loro più calcolatori pre-esistenti (COW - Clusters Of Workstations), per esempio tramite una rete LAN (Local Area Network).

Negli anni che seguirono la tecnologia dei calcolatori elettronici fece notevoli passi in avanti:

- Aumentava costantemente la velocità con cui veniva eseguita una singola operazione (frequenza di clock).
- Aumentava costantemente la capienza delle singole memorie.
- Vennero introdotti nuovi protocolli di comunicazione che garantivano un notevole aumento della velocità di scambio dei dati tra diverse unità di calcolo.

Tutto ciò abbassò i costi di acquisto o di costruzione di questi sistemi e ne rese possibile l'accesso alle industrie su larga scala.

Infine, **l'ultima innovazione** per quanto riguarda la composizione di tali infrastrutture avvenne con la nascita dei sistemi ibridi, composti non solo da CPU (le quali soffrono di alcune limitazioni date dalla natura generalizzata dei compiti che possono eseguire) ma anche da GPU ed FPGA (specializzate nell'eseguire velocemente e parallelamente semplici operazioni).

Figura 1: Fast Eastern branch of Russian Academy of Sciences - HPC System

### 1.1.1 I sistemi HPC - Oggi

Arriviamo così ai giorni nostri. Attualmente, **i migliori sistemi HPC offrono prestazioni dell'ordine dei PetaFLOPS** ($10^{15}$ operazioni in virgola mobile al

secondo) ma hanno un costo che ammonta a svariate decine di milioni di euro, il che rende ipoteticamente inaccessibile lo sfruttamento di tali prestazioni alla maggior parte delle aziende e degli enti di ricerca.

Incidentalmente, è nata negli ultimi anni la tecnologia del **Cloud Computing, un nuovo paradigma che consiste nell'erogazione di servizi on-demand tramite l'utilizzo della rete internet**. In particolare, è ora possibile "affittare" la potenza di calcolo fornita da un sistema HPC anche senza possederne uno "in casa".

Questo nuovo paradigma offre la **soluzione a due vecchi problemi**:

1. Il primo consiste nel fatto che solamente medie e grandi imprese possono permettersi l'acquisto o la costruzione di un sistema del genere.

2. Il secondo consiste nel fatto che nella maggior parte dei casi un ente non ha la necessità di utilizzare costantemente e/o totalmente il sistema, il che lo porta all'inutilizzo più o meno costante nel suo insieme o in alcune delle sue parti. Generalmente questo conduce alla conclusione che i costi necessari all'acquisto o alla costruzione di un sistema HPC ne offuschi i guadagni generati. Poter quindi affittare a terzi la propria potenza di calcolo inutilizzata aumenta la convenienza nell'effettuare un'operazione commerciale di questa portata.

Arrivati a questo punto sorge però un nuovo problema: se si affitta a terzi la potenza di calcolo del proprio sistema HPC, **è necessario garantire quella che viene definita "qualità del servizio"(QoS - Quality of Service)** che, tra i vari parametri che la descrivono, comprende la **disponibilità del sistema (availability)**. Questo significa che un cluster HPC non deve mai (o quasi) essere inaccessibile a chi ha pagato per usufruirne.

É chiaro che un sistema di queste dimensioni (1000-10000 $m^3$), potenza e complessità, può potenzialmente e facilmente mostrare svariati tipi di problemi che possono portarlo in uno stato anomalo e quindi renderlo inaccessibile a coloro per cui ne è stato garantito l'utilizzo.

**L'oggetto di questo progetto di tesi** nasce da questa problematica, e si propone di trovare una soluzione costruendo un modello, basato sull'utilizzo di tecniche facenti riferimento all'Intelligenza Artificiale (ed in particolare al Machine Learning), che sia in grado di prevedere in anticipo il sorgere di nuove anomalie nel sistema, di modo che un amministratore possa intervenire tempestivamente, evitando così che questo debba essere reso inaccessibile all'utenza.

**La struttura della tesi** è la seguente:

- **Nella parte seguente del primo capitolo** presenterò una breve panoramica storica e funzionale del sistema HPC specifico su cui verrà implementato il progetto: il super-computer MARCONI di proprietà di CINECA.

- **Nel secondo capitolo** parlerò prima dello stato dell'arte per quanto riguarda la predizione di anomalie ed in un secondo momento delle Temporal Convolutional Neural Network (TCN), che è la particolare tipologia di rete neurale che è stato deciso di adottare per realizzare un modello in grado di prevenire in modo automatizzato le future anomalie del sistema.

- **Nel terzo capitolo** viene data una panoramica dell'ambiente di monitoraggio dei dati che serviranno a perfezionare il modello della rete neurale e, in un secondo momento, verrà introdotto il software lato client necessario ad attuarne l'estrazione.

- **Il quarto capitolo** è centrale ed in questa sede entrerò nel vivo del progetto: chiarificherò prima la logica secondo cui sono stati scelti i particolari nodi di MARCONI su cui testare il funzionamento della rete neurale, parlerò poi delle scelte effettuate durante il processo di estrazione e rielaborazione dei dati. Solamente arrivati fino a questo punto potrò presentare il modello vero e proprio della rete neurale, introducendo prima la versione basilare del modello prodotto, e successivamente analizzando in modo fine i miglioramenti apportati ed i risultati ottenuti grazie ad ognuno di questi. Alla fine del capitolo è presentato un prototipo di rete neurale che utilizza un approccio differente rispetto a quello adottato nel progetto in esame.

- **Nel quinto ed ultimo capitolo** trarrò delle conclusioni finali ed introdurrò quelli che potranno essere gli sviluppi futuri a questo progetto.

## 1.2 MARCONI

**Il sistema HPC** su cui è indirizzato in progetto di predizione delle anomalie è **MARCONI**, di proprietà di **CINECA**, un consorzio no-profit con sedi a Bologna, Milano e Roma, formato da 67 università italiane e 13 istituzioni, e precisamente del **dipartimento SCAI (SuperComputing and Innovation)**, specializzato nei sistemi HPC.

**SCAI è il più grande centro di calcolo in Italia ed uno dei più grandi in Europa**. L'obiettivo che si propone è quello di accelerare la ricerca scientifica fornendo la potenza di calcolo dei propri sistemi HPC, mirando a sviluppare e promuovere i propri servizi a favore della comunità scientifica italiana ed europea. SCAI offre il proprio supporto ai ricercatori che lavorano in svariati ambiti della scienza, come la fisica, la fisica delle particelle, la scienza dei materiali e la chimica.

Per far fronte a queste sfide il dipartimento SCAI ha a disposizione vari sistemi HPC (Galileo, D.A.V.I.D.E, PICO e MARCONI). MARCONI è il sistema di punta del dipartimento anche se ha visto la luce solamente negli ultimi anni. Infatti, egli vede la propria nascita nel 2016, anche se da quel momento ha subito diversi sviluppi della sua infrastruttura che lo hanno portato a diventatare **uno dei più potenti sistemi HPC al mondo (attualmente al 19esimo posto in termini di prestazioni)**, grazie alla sua **capacità di calcolo di 20 PFlop/s** (20 milioni di miliardi di operazioni in virgola mobile eseguite in un secondo) ed una **capacità di memorizzazione di 17 PB** ($17 \cdot 10^{15}$ byte).

Diamo una rapida occhiata alla sua **storia**:

1. Inizia la produzione di un sistema preliminare nel giugno del 2016, basato sulla famiglia di prodotti Intel® Xeon® processor E5-2600 v4 (Broadwell) e con una potenza computazionale di 1 PFlop/s.

2. Alla fine del 2016 è stata aggiunga una nuova sezione, equipaggiata con la nuova generazione della famiglia di prodotti Intel Xeon Phi (Knights Landing), basata su un'architettura multi-core che abilita una configurazione complessiva di 250 mila core. Questa sezione introduce nuova potenza di calcolo al sistema, precisamente di 11 PFlop/s.

3. Nell'agosto del 2017 viene aggiunta una terza partizione basata su Intel Xeon 8160 (SkyLake). Vengono così aggiunti al sistema 2300 nodi.

4. Il sistema allo stato attuale vede l'aggiunta di 912 nodi SkyLake, i quali hanno anche rimpiazzato la versione preliminare del 2016. La potenza di calcolo raggiunta, come già citato, è di 20 PFlop/s.

### 1.2.1 Architettura

**L'architettura di MARCONI, ad alto livello**, può essere vista come una rete di normali computer connessi tra loro, chiamati **nodi della rete**. Ogni nodo possiede tre tipi di memoria ed un numero variabile di core, i quali possiedono una **FPU (Floating Point Unit)** che è responsabile di eseguire i calcoli. Le memorie seguono una gerarchia che, procedendo dall'alto verso il basso, vede diminuire costantemente la capacità di memorizzazione a favore di un aumento della velocità di accesso ai dati.

In particolare, nel punto più in alto di questa gerarchia vediamo la **memoria condivisa**, responsabile nella ricezione dei dati da parte dell'utente e della memorizzare permanentemente dei risultati forniti dalla FPU. Seguono i **dischi privati**, di dimensioni che tipicamente vanno dai 64 ai 256 GB, responsabili della memorizzazione dei risultati parziali dei calcoli. In fondo alla gerarchia ci sono le

**memorie cache**, di rapidissimo accesso ma le cui capacità di memorizzazione è dell'ordine dei KB. Esse sono responsabili della memorizzizazione dei dati utilizzati più frequentemente.

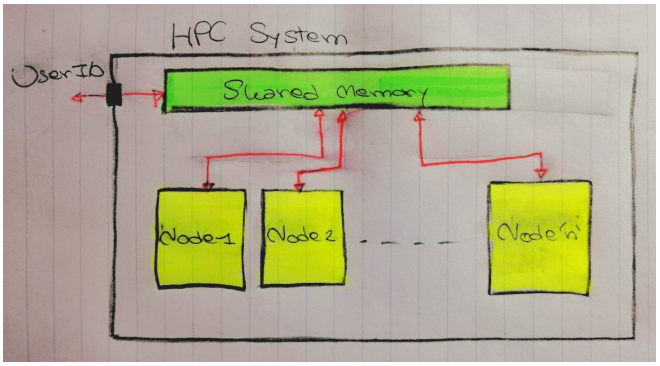



Figura 2: Architettura semplificata - MARCONI

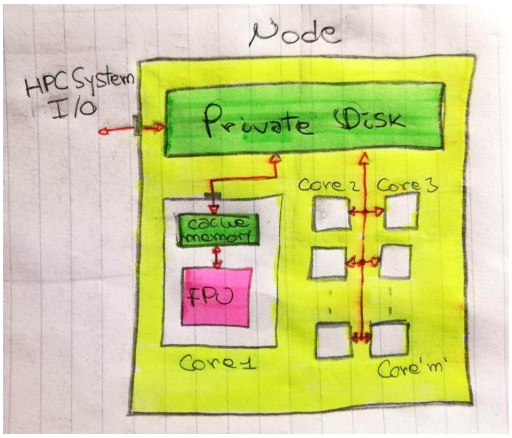



Figura 3: Architettura semplificata del singolo nodo - MARCONI

# 2 Predizione di anomalie: Stato dell'arte e TCN

Il problema della predizione di anomalie è stato ed è ampliamente discusso all'interno della comunità scientifica, date le sue innumerevoli applicazioni in ambito industriale e non solo. Tuttavia, **non siamo ancora arrivati al punto di considerare una particolare soluzione uno standard**, in quanto le nuove scoperte nel campo dell'intelligenza artificiale suggeriscono periodicamente nuove soluzioni migliori delle precedenti in termini di efficacia, generalizzabilità e/o efficienza.

Prima di parlare delle caratteristiche facenti riferimento alla soluzione adottata, cioè l'utilizzo di una **Temporal Convolutional Neural Network (TCN)**, farò una panoramica sulle soluzioni che vengono considerate lo stato dell'arte nella risoluzione di questo problema.

## 2.1 Stato dell'arte

L'utilizzo di ***reti neurali profonde*** **(Deep Neural Network)**, ideate da Yoshua Bengio nel 2006, ha rapidamente aperto il campo a nuove sperimentazione di reti neurali in grado di analizzare delle sequenze di dati legate tra loro. **I nuovi modelli** messi in campo hanno spesso rimpiazzato le vecchie soluzioni afferenti al Machine Learning a causa della loro **abilità nel gestire ed analizzare delle serie temporali multivariate**, cioè caratterizzate da più di una variabile (tavolta centinaia o migliaia) legata ad un singolo intervallo temporale.

**Questa particolare tipologia di rete neurale prende il nome di *Ricorrente***, in quanto la stessa operazione viene ripetuta per ogni elemento della sequenza, avente l'output dipendente dagli elementi precedentemente processati.

Analizziamo ora la struttura e la funzione della versione basica delle **RNN (Recurrent Neural Network)**.

### 2.1.1 Vanilla RNN

Nelle reti neurali tradizionali (Multilayer Perceptron - MLP) viene assunto che tutti gli input della rete siano indipendenti tra loro, una grossa limitazione se consideriamo che spesso abbiamo bisogno di introdurre delle **dipendenze temporali tra i dati in input**.

Questa funzionalità viene introdotta mediante l'utilizzo di ***pesi temporali*** **come parametri aggiuntivi da far apprendere alla rete neurale**. Questi pesi collegano tra loro elementi della sequenza temporalmente connessi. In particolare, l'input al tempo $t$ è collegato da dei pesi $v_t^i$ all'input registrato al tempo $t-1$, uno per ogni variabile. Tutto ciò rende le RNN applicabili a compiti quali il riconoscimento della grafia ed il riconoscimento vocale.

**Una caratteristica delle Vanilla RNN** è che possono imparare dipendenze temporali limitatamente ad un periodo $T$ a cui corrisponde un certo numero di elementi precedentemente processati dalla rete, il che ne vincola la capacità di riconoscimento di dipendenze temporali a tali elementi. Precisamente, ogni input $x_t^i$ può essere influenzato solamente dagli input $x_u^i$ tali che $t - u < T$. Inoltre, **l'aggiunta di parametri da apprendere e di neuroni fa sorgere altre problematiche quali la velocità di apprendimento della rete neurale**.

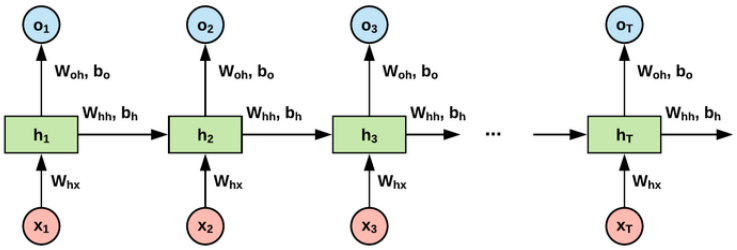



Figura 4: Vanilla RNN

### 2.1.2 Long short-term memory - LSTM

Il limite relativo all'apprendimento di dipendenze nella sequenza di dati limitatamente ad un periodo $T$ viene superato con l'ideazione delle **Long Short-Term Memory (LSTM), una particolare rete ricorrente che ha l'abilità di decidere se ricordare o meno un particolare elemento della sequenza**. Questa nuova funzionalità è resa possibile grazie ad una modifica della rete che consiste con un **ampliamento strutturale del singolo neurone**, che ora viene identificato come *cella di memoria*, e grazie all'aggiunta di **nuove connessioni tra un neurone e quello successivo nella sequenza**, che identificano *lo stato della cella*.

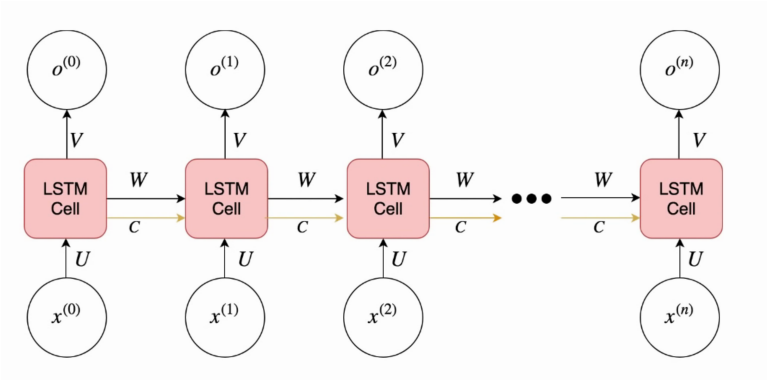



Figura 5: Architettura LSTM

La nostra nuova cella ha così una nuova connessione in input e in output, ci rimane ora da capire come queste nuove connessioni vengono aggiornate.

Per comprendere ciò, è necessario indagare la **struttura interna della cella**. Senza entrare troppo nei dettagli, possiamo identificare la nuova struttura grazie all'aggiunta di tre elementi, chiamati *gate*:

- **Input Gate** → Decide se aggiornare o meno la cella di memoria.
- **Forget Gate** → Decide se dimenticare o meno il valore memorizzato nella cella, resettandolo a 0.
- **Output Gate** → Decide se rendere visibile o meno il valore memorizzato nella cella alle celle successive.

Questi nuovi elementi introducono nella rete tre nuovi vettori di parametri per ogni cella, il che aumenta considerevolmente il numero di valori che la rete deve apprendere durante la fase di allenamento, e di conseguenza il tempo necessario a completarlo con efficacia.

La seguente immagine mostra l'architettura interna di una singola cella della rete, ed in particolare l'**aumento di complessità** rispetto alla struttura del singolo neurone di una Vanilla RNN.

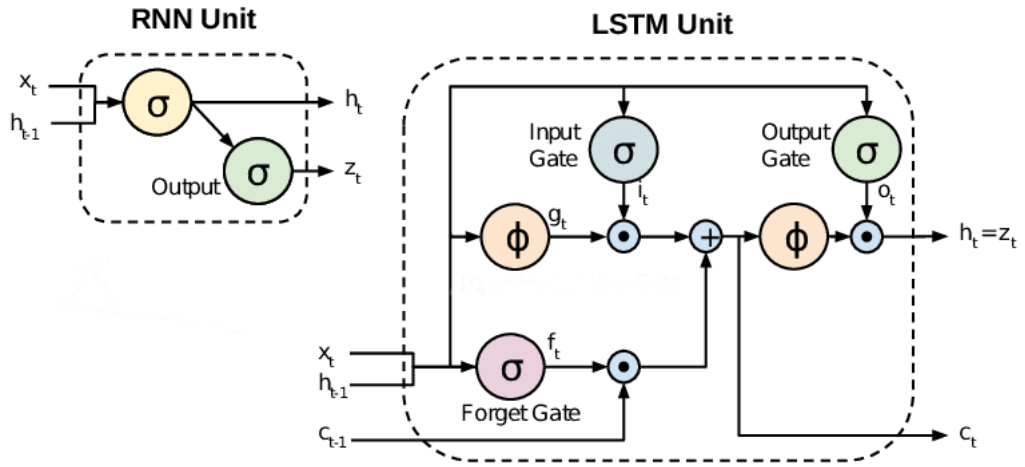



Figura 6: [LSTM - Cella di memoria] VS [Vanilla RNN - Neurone]

### 2.1.3 Gated Recurrent Unit - GRU

**Una modifica alle LSTM** volta a limitare il problema della pesantezza della rete, soprattutto per quanto riguarda la velocità di apprendimento, trova la sua

sintesi nelle GRU **(Gated Recurrent Unit)**. In particolare, questo tipo di rete corrisponde ad una LSTM con una **cella di memoria semplificata**. Dalle LSTM viene rimosso l'output gate, quindi il valore memorizzato nella cella viene esposto senza controllo al resto della rete.

Teoricamente parlando, **l'efficacia di una LSTM è per natura strettamente superiore a quelle di una GRU**. Tuttavia, nel momento in cui dalla teoria ci spostiamo alla pratica, risulta che **le GRU lavorano meglio su dei dataset di minori dimensioni rispetto alle LSTM** in quanto richiedono meno tempo per apprendere i valori dei parametri della rete. Se invece analizziamo la capacità di apprendere dipendenze temporali tra elementi distanti nella sequenza di ingresso, le LSTM producono quasi sempre risultati visibilmente migliori rispetto alle GRU.

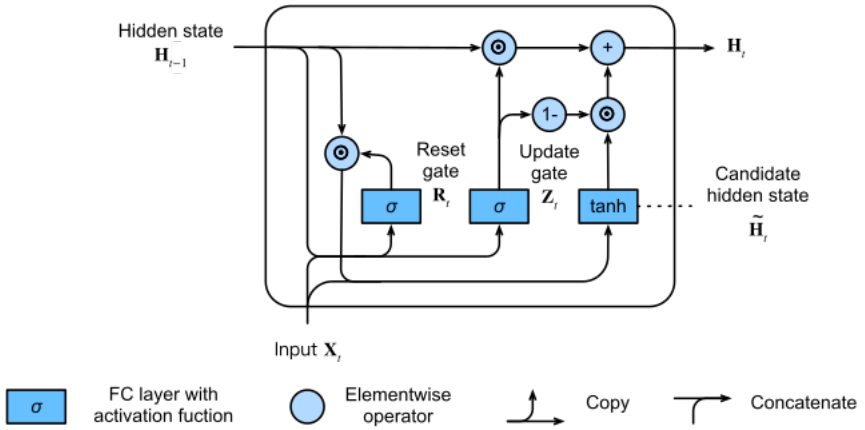



Figura 7: GRU cell

## 2.2 Temporal Convolutional Network - TCN

Prima di introdurre le TCN, diamo una rapida occhiata alla tipologia di rete neurale che sta alla base di queste ultime, le **Convolutional Neural Network (CNN)**.

Rispetto ad una tradizionale rete neurale, le CNN adottano una differente struttura di connessioni tra uno strato della rete ed il successivo. In particolare, invece di avere tutti i neuroni di uno strato connessi ad ogni singolo neurone dello strato successivo, abbiamo una **matrice di pesi (connessioni), chiamata *kernel*, condivisa tra tutti i neuroni di un livello della rete**, il che lascia intuire come il numero di parametri di cui una rete neurale deve apprendere i valori ottimali diminuiscano enormemente, a favore della velocità di apprendimento della rete. Questi kernel possono essere visti come una finestra che scorre tra i neuroni di

uno strato della rete e, nel momento in cui si trovano in una particolare posizione, **influenzano l'output del neurone che stanno processando in modo che questo sia vincolato al valore attualmente in input ai neuroni localmente vicini** tramite un'operazione matematica chiamata *convoluzione*, il cui simbolo è '$*$':

$$I(i) * K = \sum_{n=-k}^{k} I(i+n) \cdot K(n)$$

con $k = \frac{<lunghezza\_lato\_kernel> - 1}{2}$, in questo caso il kernel è monodimensionale.

In genere, in ogni singolo strato della rete è presente più di un kernel. Questo significa che ad ogni neurone della rete sono associati più neuroni nello strato successivo, il che aumenta considerevolmente la dimensione della rete.

Le reti neurali convoluzionali sono spesso utilizzate nel campo del riconoscimento d'immagine e, sebbene siano più complesse rispetto a quanto descritto fino ad ora, non è necessario introdurre altri elementi descrittivi al fine di poter comprendere la loro evoluzione per quanto riguarda l'analisi di sequenze temporali.

### 2.2.1 Architettura di una TCN

La base di partenza nella costruzione di una TCN è, come detto, quella di una CNN. Fatta eccezione per alcuni elementi come lo strato di *pooling*, i quali non sono stati citati nella sezione precedente per non creare confusione.

Per adattare la struttura di una CNN all'analisi di serie temporali sono stati introdotti vari elementi che risolvono l'insorgere di nuove problematiche date dal nuovo paradigma funzionale per cui le si vuole adattare. In generale, **non esiste uno standard che definisce la struttura di una TCN**, per questo differenti soluzioni implicano differenti strutture.

Tuttavia, viene definita in letteratura una **base di partenza per la costruzione di TCN** che verranno poi riadattate per un determinato scopo, ed è proprio questa struttura che descriverò in questa sezione.

### 2.2.2 Convoluzioni Causali Dilatate

Come per le Vanilla RNN, anche con le TCN è necessario **definire a priori il periodo temporale che si vuole considerare quando si vanno a cercare le dipendenze tra un elemento della sequenza e gli elementi precedenti** ma, come vedremo, la struttura delle convoluzioni dilatate permette la definizione di un intervallo temporale esponenzialmente superiore a quello delle RNN a parità di risorse e di parametri da apprendere.

A differenza della convoluzione classica, quella dilatata agisce **collegando tra loro i neuroni di uno stesso strato non strettamente adiacenti gli uni agli altri**, ma quelli ad una particolare distanza tra loro definita dal *fattore di dilatazione*. La convoluzione dilatata viene definita come segue:

$$I(i) *_d K = \sum_{n=-k}^{k} I(i + n \cdot d) \cdot K(n)$$

con $d$ pari al fattore di dilatazione, che generalmente viene scelto pari ad una potenza di 2. Nel caso in cui viene scelto $d = 1$ allora la convoluzione dilatata si riduce alla convoluzione classica.

**Nelle TCN vengono impilati vari strati di neuroni, caratterizzati da valori di *d* che crescono di strato in strato seguendo l'ordine delle potenze della dimensione del kernel**. In questo modo si aumenta il *campo di ricezione* di un singolo neurone ad un numero di valori della sequenza temporale pari al fattore di dilatazione dell'ultimo strato moltiplicato per la dimensione del kernel. Per esempio, se impostiamo la dimensione del kernel a 2 ed impiliamo 4 strati di neuroni seguendo questo schema, con fattori di dilatazione pari a [1, 2, 4, 8], otterremo una rete convoluzionale in cui ogni singolo elemento della sequenza possiede conoscenza relativa ai 16 elementi della sequenza a lui precedenti.

In generale, **una convoluzione può essere *causale* o *non causale***. Una convoluzione si dice ***non causale*** **se il kernel non è centrato nell'ultimo elemento di ogni sua dimensione**. In parole povere, questo significa che in una convoluzione non causale l'output della convoluzione tra l'elemento attualmente processato ed il kernel coinvolge anche elementi successivi a quello corrente. In genere le convoluzioni non causali vengono utilizzate nell'elaborazione dei pixel di un'immagine. Diversamente, quando viene considerata una sequenza temporale non si dovrebbe introdurre conoscenza specifica sugli elementi successivi nella sequenza, per questo **nelle TCN vengono utilizzate delle convoluzioni causali**.

Nella seguente illustrazione verrà mostrata la struttura di strati convoluzionali causali dilatati impilati gli uni sugli altri, in modo da far chiarezza e rendere più comprensibile la descrizione fatta fino ad ora.

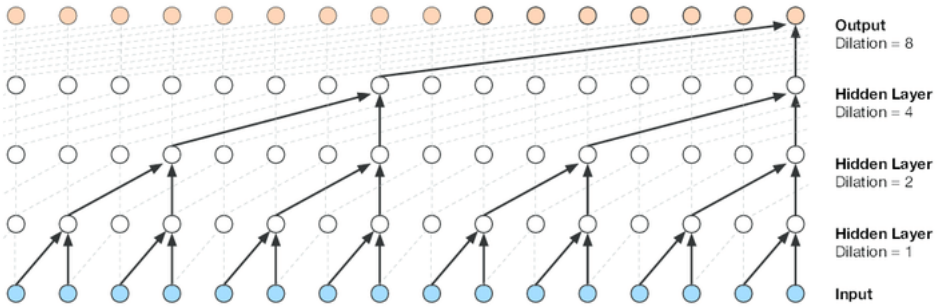



Figura 8: Stack of Causal Dilated Convolutions with $d = [1, 2, 4, 8]$

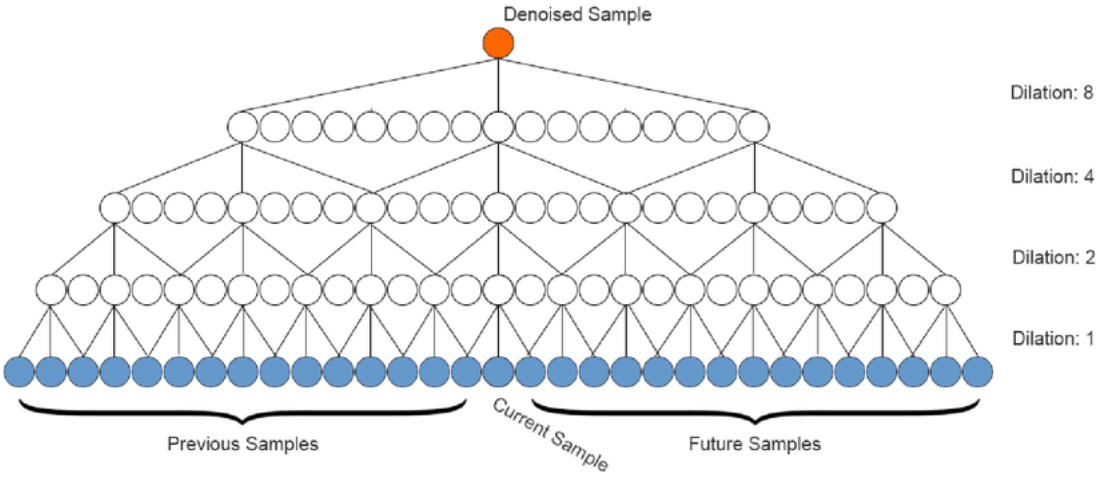


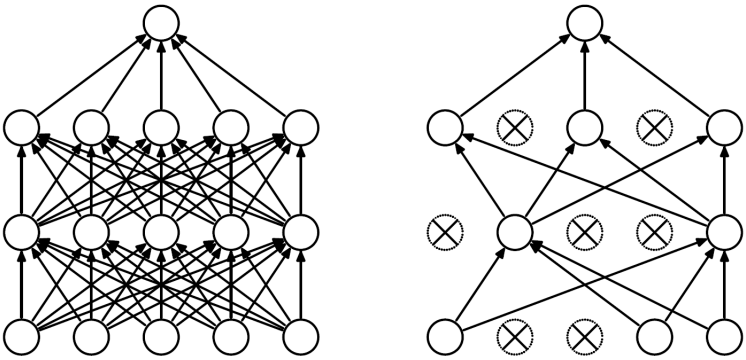

Figura 9: Stack of Non-Causal Dilated Convolutions with $d = [1, 2, 4, 8]$

### 2.2.3 Dropout & Weights Normalization

Una rete neurale, in generale, deve essere in grado di **imparare con precisione un certo comportamento** evitando il cosidetto **problema dell'*underfitting*** ma, allo stesso tempo, deve essere in grado di **generalizzare una certa funzione**, evitando quindi di replicare con troppa adesione il comportamento dei dati in ingresso durante la fase di allenamento, facendo sorgere **il problema opposto, cioè quello dell'*overfitting***.

Per evitare il secondo di questi due problemi, sono state ideate varie **tecniche di *regolarizzazione*** (L1 regularization, L2 regularization, ecc) **che servono ad evitare che la rete aderisca troppo profondamente ai dati visti durante la fase di apprendimento**.

In particolare, nelle TCN viene adottata una tecnica che ha ottenuto un grande successo in molti tipi di reti neurali. Questa viene chiamata ***Dropout*, e consiste nell'eliminazione di un sotto-insieme casuale di neuroni nello strato successivo a quello corrente**, e quindi delle sue connessione con i neuroni del layer corrente. In genere viene definito un fattore che definisce quanti neuroni, rispetto al totale, devono essere eliminati durante l'elaborazione di un insieme di dati in input per un dato layer. Questi neuroni eliminati, insieme alle loro connessioni, verranno reintrodotti durante l'elaborazione dell'insieme successivo di dati in input, che vedranno rimossi un altro sotto-insieme di neuroni e delle loro connessioni.



(a) Standard Neural Net (b) After applying dropout.

Figura 10: Dropout

Oltre al Dropout, viene applicata un'altra trasformazione che modifica gli elementi costitutivi della rete, sto parlando della **normalizzazione dei pesi (Weights Normalization)**. Questo tipo di normalizzazione che, come suggerito dal nome, viene applicata ai pesi delle connessioni tra i vari elementi della rete piuttosto che ai valori restituiti in output dai neuroni stessi, **serve ad accelerare la convergenza del gradiente durante il processo di ottimizzazione della rete**, e quindi la velocità di apprendimento della stessa.

### 2.2.4 Connessioni Residue

L'ultimo elemento che caratterizza la versione di base di una TCN sono le *Connessioni Residue*. Queste connessioni servono a **limitare il problema, presente soprattutto nelle reti profonde, dell'esplosione e dello svanimento del gradiente** (*vanishing/exploding gradient*). Considerando che la rete in esame possiede svariati strati impilati di convoluzioni dilatate, sembra che questo elemento aggiuntivo faccia proprio al caso nostro.
Inoltre, le connessioni residue **velocizzano e facilitano l'apprendimento di**

**funzioni elementari** come l'identità, aumentando ulteriormente la velocità di apprendimento della rete in svariate circostanze.

Fondamentalmente, **una connessione residua consiste nel confrontare l'output di una trasformazione** (nel nostro caso le convoluzioni dilatate) **con l'input della trasformazione stessa**.

**Una problematica** che può mostrarsi nel fare ciò è quando una trasformazione produce un risultato che ha una dimensione differente rispetto al dato prima di essere processato. Questo problema può essere risolto introducendo delle *convoluzioni 1x1*, le quali possono essere opportunamente impostate per aggiustare la dimensione del dato in ingresso a quella del dato trasformato.

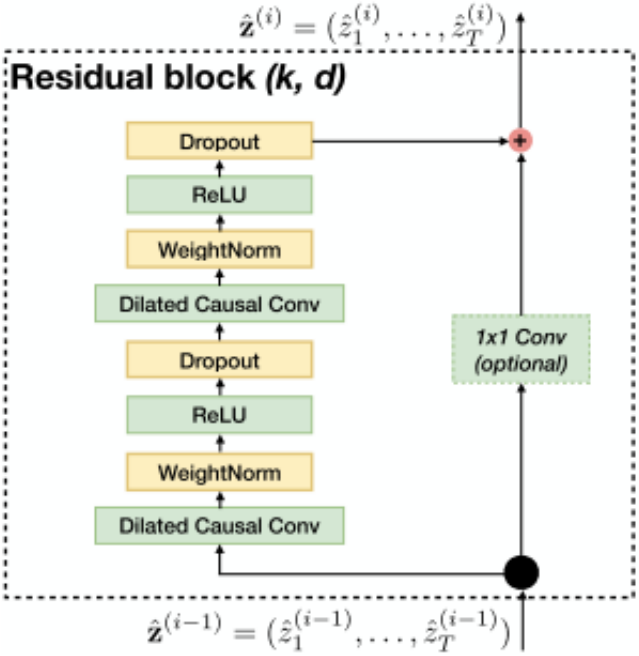



Figura 11: Residual Block

## 2.3 TCN contro tutti

Ora che abbiamo illustrato il funzionamento e l'architettura delle TCN e delle varie reti ricorrenti, possiamo trarre delle conclusioni e spiegare il motivo per cui è stata scelto di adottare questa soluzione piuttosto che quelle più comunemente utilizzate per problemi concernenti l'analisi di sequenze e serie temporali.

Tra le varie reti ricorrenti analizzate quelle che offrono più accuratezza nei risultati sono le LSTM, e per dataset di piccole dimensioni le GRU. Ebbene, **rispetto a questa particolare tipologia di rete le TCN offrono prestazioni**

**altalenanti in termini di accuratezza** (per certi compiti hanno un'efficacia visibilmente superiore, altre volte invece vengono superate dalle RNN), ma allo stesso tempo **richiedono molta meno memoria** in quanto le TCN non necessitano di salvare dei risultati intermedi, come invece accade per le LSTM e le GRU. Inoltre, le TCN offrono maggiore controllo per quanto riguarda l'impostazione del campo di ricezione di un particolare elemento della sequenza, infatti in questo caso basta aggiustare la dimensione dei kernel od il fattore di dilatazione delle convoluzioni.

Rispetto alle Vanilla RNN, le TCN possiedono la capacità di poter **imparare dipendenze temporali più lunghe, a parità di risorse**. Inoltre, le TCN hanno un'**impostazione strutturale che limita i problemi dell'esplosione o dello svanimento del gradiente** (ReLU, Residual Connections, ecc) nonostante siano particolarmente profonde e quindi siano delle ottime candidate per subire questo tipo di degradazione delle prestazioni durante la fase di ottimizzazione. Diversamente, **le RNN soffrono terribilmente l'esistenza di questa problematica**, e ciò è causato dalla struttura sequenziale che lega elementi differenti della stessa successione temporale. Per le Vanilla RNN, è la necessità di risolvere questo problema ad aver spinto maggiormente all'ideazione delle LSTM.

Tuttavia, **le RNN si comportano meglio delle TCN in fase di valutazione e test della rete**. Infatti, durante questa fase la rete è sgravata dal compito dell'apprendimento, e quindi non sono presenti alcune problematiche che piuttosto caratterizzano la fase precedente. In particolare, durante le fasi finali di implementazione di una rete neurale, le TCN necessitano di processare l'intera sequenza di elementi nella serie temporale che rientrano nel campo di ricezione dell'elemento corrente. Diversamente, per le RNN gli stati intermedi necessari a valutare le dipendenze tra il dato corrente e quelli precedenti sono sufficienti allo scopo, e quindi gli elementi precedenti della sequenza temporale possono essere scartati.

Complessivamente, sembra che le TCN abbiamo delle caratteristiche ottimali nei confronti delle reti ricorrenti e quindi giustificano, almeno preliminarmente, la scelta effettuata nell'adozione di questo modello per la risoluzione del problema della predizione di anomalie. Inoltre, è necessario citare il fatto che, essendo le TCN una novità in letteratura, i possibili miglioramenti futuri potrebbero fare in modo che queste superino definitivamente le RNN. Potrebbe risultare quindi utile acquisire consapevolezza sugli usi, i limiti e le potenze di quest'architettura. Inoltre, è necessario contribuire in maniera attiva allo sviluppo di questa tipologia di rete neurale, in modo che un giorno si possa convergere verso uno standard nella risoluzione di problemi facenti riferimento all'analisi di serie temporali.

# 3 Fonte dei dati

Prima di entrare nel vivo del progetto è bene considerare che, nel momento in cui stiamo per andare a realizzare un modello (nel nostro caso una TCN) utilizzando la tecnologia del Machine Learning, sono di **primaria importanza i dati che gli verranno forniti per le fasi di apprendimento e di test**. Infatti, in generale è meglio prevedere di dedicare più tempo all'estrazione e all'elaborazione dei dati piuttosto che all'implementazione del modello.

Nel nostro caso, **i dati di cui faremo uso per allenare il modello di predizione delle anomalie** consistono nelle registrazioni raccolte dai molti sensori posizionati internamente ai nodi del sistema HPC MARCONI (temperatura della CPU, velocità delle ventole, ecc), alcuni invece sono inseriti manualmente dagli amministratori del sistema (stato del nodo, ecc), infine altri dati vengono raccolti dall'ambiente software di riferimento (tempo di avvio del sistema operativo, ecc).

In questo capitolo analizzeremo prima l'ambiente di monitoraggio dei dati raccolti in tempo reale, dopodichè andremo ad analizzare il **framework utilizzato per l'ottenimento effettivo dei dati (Examon)**.

## 3.1 Ambiente di monitoraggio

Come è facilmente intuibile, ottenere i dati dal sistema è un arduo compito in assenza di un ambiente che faciliti questo lavoro. È necessario valutare e gestire problematiche quali l'aggregamento dei dati raccolti, la trasmissione degli stessi dal componente che li raccoglie ad un ambiente indicizzato (un database) da cui attingere i dati pseudo-ordinati, ecc.

Ebbene, nel caso dei sistemi HPC di CINECA, abbiamo a disposizione un ambiente che facilita il lavoro sgravandoci da tutti quei compiti di basso livello che stanno al di fuori della logica specifica del dato utile ai nostri scopi.

Vista ad alto livello, questo ambiente consiste di **tre strati facilmente distinguibili**:

1. Livello di raccolta dei dati.
2. Livello di comunicazione.
3. Livello di visualizzazione, immagazzinamento e trasferimento alle applicazioni soprastanti (come il modello di ML oggetto del progetto).

Il primo dei tre, il **livello di raccolta dei dati**, è composto dalle API dei sensori e dei moduli preposti alla raccolta dei dati provenienti dall'ambiente software ospitato dai nodi stessi.

Il secondo, **il livello di comunicazione** invece funge da intermediario tra i livelli confinanti ed è basato sul protocollo *MQTT*. Infatti, l'**MQTT Broker utilizza un meccanismo di *publish/subscribe*** all'interno del quale le fonti di dati fungono da publishers, che inviano dati grezzi al broker, mentre i tool più ad alto livello fungono da subscribers, i quali richiedono i dati relativi ad un particolare insieme di argomenti (*topic*). In questo schema, il ruolo dell'MQTT broker è quello di **creare e gestire gli argomenti relativi ai dati inviati dai publishers**. Nel momento in cui un publisher invia dei dati relativi ad un certo argomento (specificato come parametro aggiuntivo al dato), l'argomento viene creato ed è reso disponibile al broker. Ogni subscriber interessato a quell'argomento riceverà i dati richiesti nel momento in cui saranno resi disponibili all'MQTT broker.

**Il terzo livello** comprende tutti quei tool che presentano i dati all'utente. *Grafana*, per esempio, è un tool dedicato alla visualizzazione grafica in tempo reale dei dati raccolti dai sensori. Invece, *Cassandra* è un database da cui si potrà attingere per estrarre dati storici. *Apache Spark* è un framework adibito al calcolo distribuito, mentre *Apache Streaming* è il protocollo utilizzato per il trasferimento di dati dai database ai contesti applicativi più ad alto livello.

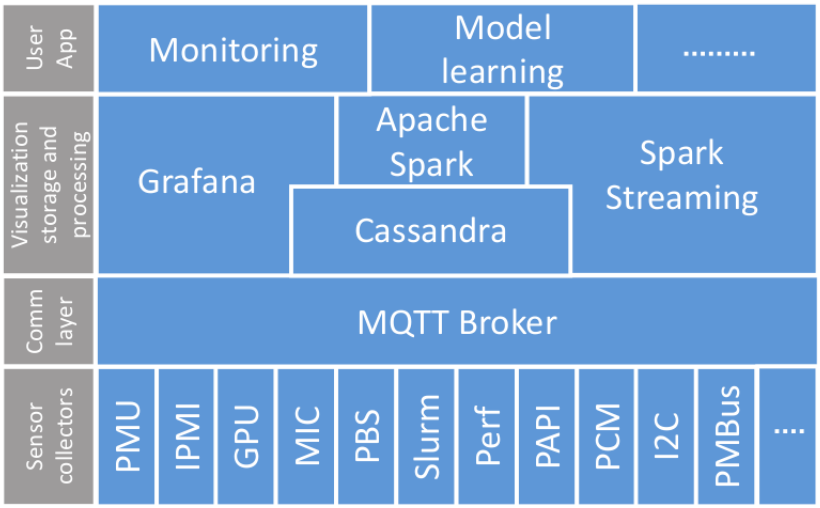



Figura 12: Struttura ad alto livello dell'ambiente di monitoraggio

## 3.2 Examon

**Examon è un framework per la raccolta, l'immagazzinamento e l'analisi di dati per sistemi HPC**, dei quali ha accesso grazie a Cassandra (ed

in particolare KairosDB). L'accesso ad Examon è reso possibile grazie al client (examon-client, disponibile grazie al lavoro di Francesco Beneventi).

**I principali componenti del framework** sono dei demoni (processi computazionali che continuano la loro esecuzione potenzialmente all'infinito), i quali gestiscono i dati raccolti dai sensori software e hardware su un singolo nodo (quelli citati nella sezione precedente) e li espongono all'utente raggruppati in dei *plugin*, a seconda delle loro origini.

In questo progetto sono stati utilizzati **tre plugin**:

- `Ganglia` → Fornisce statistiche relative all'utilizzo della CPU (velocità, frequenza, tempo di inattività, ecc) e dello stato del nodo (memoria liberia nel disco e nella RAM, spazio disponibile nei buffer, tempo di avvio del sistema operativo, ecc).

- `Confluent` → Fornisce informazioni relative al consumo energetico, alla temperatura di ciascun componente, dei dischi e della rete di comunicazione.

- `Nagios` → Raccoglie dati relativi allo stato generico del nodo.

Quello di cui andremo a parlare da qui alla fine del paragrafo è il modo in cui **i dati forniti dai plugin, che chiameremo *metriche***, sono strutturati internamente ad Examon e di come possono essere raccolti dall'utente per utilizzarli in un'applicazione ad alto livello.

**NOTA**: In questa tesi c'è un'abuso della notazione *metrica*. In particolare, quando ci spostiamo nel contesto del Machine Learning, con *metrica* si intende una funzione volta a valutare la misura del successo (o fallimento) di un modello in fase di test.

I dati forniti da Examon si presentano tramite una tripla del tipo `<timestamp, name, value>`:

- `timestamp` indica l'istante temporale che determina la fine dell'intervallo in cui quella metrica è stata registrata.

- `name` è il nome della metrica in analisi.

- `value` è il valore registrato per quella particolare metrica, ed è calcolato facendo una media aritmetica tra i valori registrati dal sensore nell'intervallo temporale che fa riferimento alla tripla.

| | timestamp | name | value |
|---|---|---|---|
| 0 | 2019-10-18 10:00:57+02:00 | boottime | 1.56174e+09 |
| 1 | 2019-10-18 10:01:59+02:00 | boottime | 1.56174e+09 |
| 2 | 2019-10-18 10:02:59+02:00 | boottime | 1.56174e+09 |
| 3 | 2019-10-18 10:13:40+02:00 | load_five | 219.48 |
| 4 | 2019-10-18 10:14:00+02:00 | load_five | 220.96 |
| 5 | 2019-10-18 10:14:20+02:00 | load_five | 220.8 |
| 6 | 2019-10-18 10:26:01+02:00 | swap_free | 1.6384e+07 |
| 7 | 2019-10-18 10:26:41+02:00 | swap_free | 1.6384e+07 |
| 8 | 2019-10-18 10:27:21+02:00 | swap_free | 1.6384e+07 |

Figura 13: Tabella - Estrazione casuale senza filtri

È necessario ora introdurre una struttura che faciliti l'utente nell'ottenimento di dati con particolari caratteristiche, in base alle sue necessità, tramite l'utilizzo di interrogazioni simili allo stile utilizzato per interrogare i database tramite SQL.

Per fare ciò, **viene definita per ogni metrica un insieme di parametri descrittivi aggiuntivi chiamati *tag keys***, i quali possono far riferimento ad una o più metriche.

```
In [32]: sq.DESCRIBE().execute()
```

Out[32]:

| | name | tag keys |
|---|---|---|
| 0 | 1U_Stg_HDD4_Pres | [chnl, cluster, health, node, org, plugin] |
| 1 | 1U_Stg_HDD1_Pres | [chnl, cluster, health, node, org, plugin] |
| 2 | 1U_Stg_HDD3_Pres | [chnl, cluster, health, node, org, plugin, type] |
| 3 | CPU_2_PECI | [chnl, cluster, health, node, org, part, plugi... |
| 4 | BMC_Temp | [chnl, cluster, health, node, org, part, plugi... |
| 5 | BB__3_0V_VBAT | [chnl, cluster, health, node, org, part, plugi... |
| 6 | DIMM_6 | [chnl, cluster, health, node, org, part, plugi... |
| 7 | DIMM_8 | [chnl, cluster, health, node, org, part, plugi... |
| 8 | DIMM_9 | [chnl, cluster, health, node, org, part, plugi... |

Figura 14: Tabella - Estrazione dei tag associati alle metriche

**NOTA**: `sq` è un istanza della classe `ExamonQL`, la quale rappresenta una connessione ad Examon.

Ogni tag può assumere un insieme definito di valori che è ottenibile interrogando opportunamente Examon. Nell'esempio seguente viene mostrato il risultato restituito dall'interrogazione per ottenere i possibili valori associabili al tag `plugin`.

```
In [8]: df = sq.DESCRIBE(tag_key = 'plugin') \
            .execute()

        display(df)
```

| | tag values |
|---|---|
| 0 | confluent_pub |
| 1 | ipmi_pub |
| 2 | pmu_pub |
| 3 | ganglia_pub |
| 4 | nagios_pub |

Figura 15: Tabella - Valori che può assumere il tag `plugin`

In linea di massima questi sono gli elementi necessari che, opportunamente gestiti e combinati tra loro, sono necessari per effettuare un'analisi sulla struttura dei dati su Examon. Una volta identificata la tipologia dei dati che si vuole ottenere, è possibile effettuare richieste vere e proprie per ottenere dati nella forma `<timestamp, name, value>`.

# 4 Progetto

Finalmente abbiamo gli strumenti necessari per entrare nel vivo del progetto.

I dati storici forniti da Examon comprendono tutti i nodi di tutti i cluster HPC detenuti dal dipartimento SCAI del consorzio CINECA. Come è facilmente intuibile, l'ammontare di dati potenzialmente estraibili sono superiori a quelli necessari per allenare efficientemente la rete neurale, è quindi auspicabile attuare una selezione finalizzata a scegliere dei nodi di MARCONI considerati come "buoni", in modo da ridurre i tempi di estrazione dei dati e di allenamento della TCN. Una volta effettuata questa scelta, è necessario estrarre i dati relativi ai nodi selezionati, i quali si presentano nella forma grezza descritta nel capitolo precedente (`<timestamp, metric_name, value`), e quindi elaborali in modo che i dati generati da diverse fonti (sensori, moduli software, ecc) siano coerenti tra loro ed organizzati in modo da essere processabili da una rete neurale. Successivamente, vengono effettuate delle ulteriori modifiche in modo da adattare i dati al nostro problema e al particolare tipo di rete neurale che si è deciso di implementare. Una volta trasformati i dati grezzi nella forma desiderata è possibile procedere con l'implementazione della TCN, cui segue la fase di allenamento necessaria al perfezionamento del modello ed infine la fase di test e di analisi dei risultati.

Per quanto riguarda l'ideazione del modello, **si è deciso di ricorrere ad una tecnica di Machine Learning chiamata *Supervised Learning***. Questa consiste nel dare al modello la capacità di giudicare quanto una previsione sia corretta, basandosi su un'informazione aggiuntiva ai dati in ingresso che ci dice il risultato sperato di una previsione. In questo modo la rete neurale può, in base ad un confronto tra la previsione ottenuta ed il risultato sperato, modificare il suo comportamento per cercare di ottenere risultati migliori per le previsioni effettuate successivamente a quella corrente.

Questa tecnica si contrappone a quella dell'*Unsupervised Learning*, in cui questa informazione aggiuntiva non è prevista, e si adatta meglio a dei modelli di Machine Learning in cui l'obiettivo non è effettuare previsioni, bensì quello di ottenere informazioni "nascoste" nei dati forniti. Per esempio, un compito per cui viene spesso utilizzata questa tecnica è quello di trovare dei gruppi omogenei su un insieme molto vasto di dati.

## 4.1 Nagios - Stato del nodo

Lo scopo della nostra rete neurale è quello di prevedere in anticipo se un nodo del cluster MARCONI andrà in uno stato di anomalia, quindi l'informazione necessaria per permettere alla TCN di perfezionarsi deve fare riferimento allo stato del nodo.

**Nagios** è il plugin che fornisce questa indicazione e **contiene gli stati di ogni nodo dei cluster HPC**.

**Questa informazione viene aggiornata ogni 15 minuti** e contiene l'attributo `value`, che si presenta come una descrizione testuale dello stato del nodo, e `state`, che può assumere valori interi da 0 a 3 ed indica il codice dello stato. Non esiste una corrispondenza biunivoca tra i valori che possono assumere i parametri `value` e `state`. In particolare, `value` può presentare descrizione diverse associabili allo stesso valore del parametro `state`.

Le configurazioni del valore di questi parametri a cui noi siamo interessati sono quelle in cui `state` assume il valore 2, mentre `value` contiene al suo interno la stringa `DRAIN`.

| timestamp | state | value |
|---|---|---|
| 2019-12-18 10:00:00.100000+01:00 | 0 | ok () |
| 2019-12-18 10:00:00.100000+01:00 | 0 | C() W() O({mmgetstate-active:rdma-on:no-releva... |
| 2019-12-23 16:01:10.823000+01:00 | 2 | C(/marconi_scratch[df=1],/marconi_work[df=1]) ... |
| 2019-12-23 16:01:10.823000+01:00 | 2 | ALLOCATED+DRAIN matches a critical state |
| 2019-12-23 16:15:00.025000+01:00 | 2 | C(/marconi_scratch[df=1],/marconi_work[df=1]) ... |
| 2019-12-23 16:15:00.025000+01:00 | 2 | ALLOCATED+DRAIN matches a critical state |

Figura 16: Estrazione dati dal plugin Nagios - Stati critici e stati ordinari

**Ciò che vogliamo ottenere dalla nostra rete neurale non è una semplice previsione delle anomalie del nodo, ma una previsione sull'insorgere di nuove anomalie, il che esclude la circostanza in cui viene prevista un'anomalia per un nodo che è già in stato anomalo**. Infatti, effettuare della analisi sui risultati della rete senza considerare quest'eventualità può indurre in delle considerazioni errate sulla bontà del modello prodotto. Inoltre, la creazione di un rete neurale in grado di prevedere efficacemente futuri stati anomali quando lo stato corrente è già anomalo, ma che allo stesso tempo non sia in grado di prevedere uno stato anomalo se questo è preceduto da uno stato non anomalo, potrebbe risultare inutile nella pratica quotidiana.

## 4.2 Selezione dei nodi

Per scegliere i nodi da cui estrarre i dati da dare in pasto alla rete neurale è necessario fare alcune considerazioni.

Innanzitutto, bisogna tenere conto del fatto che **alcuni nodi della rete potrebbero essere in qualche modo compromessi in un dato intervallo temporale. Questo nodi sono caratterizzati da un numero di anomalie eccessivamente elevato**, questo significa che utilizzare i dati che li caratterizzano per perfezionare la rete neurale potrebbe condizionare la TCN ad apprendere in modo migliore l'insorgere di anomalie per questi nodi, piuttosto che per i nodi "sani" cui è invece indirizzato il modello che si vuole creare.

| | node | num_criticalities |
|---|---|---|
| **0** | r065c01s01 | 183 |
| **1** | r065c01s02 | 283 |
| **2** | r065c01s03 | 248 |
| **3** | r089c18s01 | 20263 |
| **4** | r090c10s01 | 20302 |
| **5** | r097c18s02 | 26971 |

Figura 17: Numero di anomalie presentate dai nodi "sani" e dai nodi "compromessi" - Le prime 3 righe fanno riferimento a nodi "sani", le ultime 3 a nodi "compromessi"

In secondo luogo, bisogna considerare che **nei nodi "sani" avremo moltissimi dati associati ad uno stato ordinario della rete e invece pochissimi associati ai momenti in cui insorgono nuove anomalie (*dataset sbilanciato*). Alla luce di ciò è auspicabile scegliere i nodi caratterizzati dal maggior numero di intervalli in cui la rete passa da uno stato ordinario ad uno stato anomalo**.
Infine, è necessario ricordare che **abbiamo a che fare con una gran quantità di nodi potenzialmente analizzabili, per cui sarebbe opportuno trovare un modo per ridurre la scelta di questi ad un insieme ristretto**. In particolare, i nodi di cui è composto il cluster MARCONI in cui sono ospitati tutti e tre plugin necessari all'ottenimento dei dati (Nagios, Confluent e Ganglia) sono 6785.

---

**NOTA**: Risulta dalle sperimentazioni che estrarre i dati forniti da Nagios per tutti i nodi di MARCONI ed aventi un certo valore per i parametri `state` e `value`, e successivamente contare quante anomalie sono state registrate per questi nodi, non è un compito arduo anche per un normalissimo PC e si completa in pochi

minuti (3min 34s eseguendo l'algoritmo dal mio PC). Mentre selezionare i nodi con il maggior numero di passaggi da stato ordinario a stato anomalo è risultato computazionalmente intrattabile (sempre eseguendo l'algoritmo dal mio PC).

---

Fatte queste considerazioni si può procedere con la scelta del nodo. **La strategia** che ho implementato è mostrata nel diagramma di flusso di Figura 18.

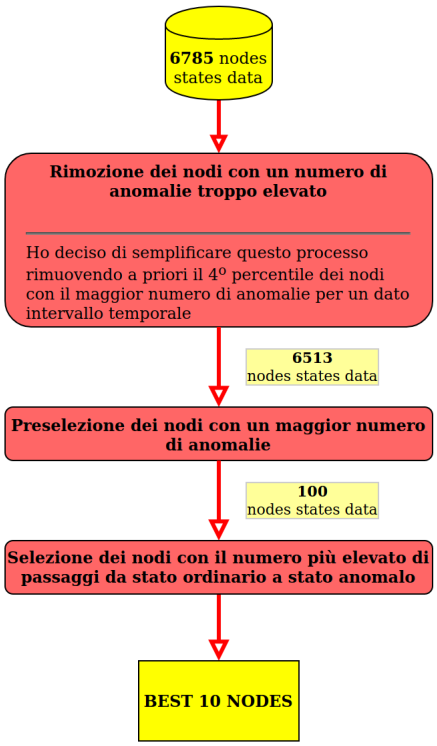



Figura 18: Strategia di selezione dei nodi - Diagramma di flusso

Nella Figura 19 è invece riportato l'esempio del risultato di una selezione, indicante per ogni nodo scelto il numero di passaggi da stato ordinario a stato anomalo riscontrati durante l'intervallo di riferimento. La selezione riportata come esempio differisce rispetto a quella che ho realmente effettuato nell'intervallo temporale di riferimento dell'estrazione degli stati dei nodi (1 mese nell'esempio; 3 mesi nella realtà).

| node | #rising_edges |
|---|---|
| r090c05s04 | 21 |
| r071c15s04 | 20 |
| r071c14s04 | 19 |
| r071c09s02 | 19 |
| r071c14s02 | 17 |
| r090c08s04 | 16 |
| r071c16s04 | 16 |
| r092c01s02 | 15 |
| r092c02s01 | 15 |
| r092c08s04 | 15 |

Figura 19: Nodi selezionati - Numero di passaggi di stato nell'intervallo di estrazione

## 4.3 Estrazione dei dati

Una volta effettuata la scelta dei nodi da analizzare ed estratti i dati sullo stato degli stessi, come illustrato del paragrafo precedente, si può procedere con l'estrazione dei dati che verranno presi in considerazione per effettuare delle previsione sugli stati futuri dei nodi.

**Le interrogazioni su Examon** vengono effettuate seguendo uno stile simile a quello adottato dai più comuni linguaggi di programmazione per le interrogazioni ai database. Nel nostro caso abbiamo inserito come parametri per filtrare le estrazione la data e l'ora di inizio e fine dell'intervallo temporale su cui si vogliono ottenere i dati (rispettivamente: `18 Ottobre 2019 ore 10:00` e `11 Gennaio 2020 ore 06:36`), oltre che ai nodi precedentemente selezionati e l'indicazione del plugin che fornisce i dati che si vuole ottenere.

```
node = 'r090c05s04'
plugin_name = 'ganglia_pub'
t_start = '18-10-2019 10:00:00'
t_stop = '11-01-2020 06:36:00'

data = sq.SELECT('*') \
         .FROM('*') \
         .WHERE(plugin=plugin_name, node=node) \
         .TSTART(t_start) \
         .TSTOP(t_stop) \
         .execute()
```

Figura 20: Query di esempio - Filtri: Plugin; Nodo; Data/ora di inizio estrazione; Data/ora di fine estrazione;

Queste informazioni vengono fornite dai plugin Confluent e Ganglia e comprendono, rispettivamente, 33 e 35 metriche distinte.

| | timestamp | name | value |
|---|---|---|---|
| 0 | 2019-10-18 10:00:57+02:00 | boottime | 1.56174e+09 |
| 1 | 2019-10-18 10:04:59+02:00 | bytes_in | 1249.77 |
| 2 | 2019-10-18 10:04:59+02:00 | bytes_out | 1123.37 |
| 3 | 2019-10-18 10:01:08+02:00 | cpu_aidle | 71.5 |
| 4 | 2019-10-18 10:01:08+02:00 | cpu_idle | 27.9 |
| 5 | 2019-10-18 10:01:08+02:00 | cpu_nice | 0 |
| 6 | 2019-10-21 11:23:28+02:00 | cpu_num | 68 |
| 7 | 2019-10-18 10:00:57+02:00 | cpu_speed | 1400 |
| 8 | 2019-10-18 10:01:08+02:00 | cpu_steal | 0 |
| 9 | 2019-10-18 10:01:08+02:00 | cpu_system | 0.6 |

(a) Ganglia

| | timestamp | name | value |
|---|---|---|---|
| 0 | 2019-10-18 10:00:00.019000+02:00 | HDD_0_Status | Present |
| 1 | 2019-10-18 10:00:00.019000+02:00 | HSBP_PSOC_Temp | 37 |
| 2 | 2019-10-18 10:00:00.019000+02:00 | HSBP_Temp | 28 |
| 3 | 2019-10-18 10:00:00.019000+02:00 | MTT_CPU1 | Unavailable |
| 4 | 2019-10-18 10:00:00.019000+02:00 | NM_Capabilities | Unavailable |
| 5 | 2019-10-18 10:00:00.019000+02:00 | P1_Status | Present |
| 6 | 2019-10-18 10:00:00.019000+02:00 | P1_Therm_Ctrl_% | 0 |
| 7 | 2019-10-18 10:00:00.019000+02:00 | P1_Therm_Margin | -27 |
| 8 | 2019-10-18 10:00:00.019000+02:00 | PS1_Curr_Out_% | 20 |
| 9 | 2019-10-18 10:00:00.019000+02:00 | PS1_Input_Power | 572 |

(b) Confluent

Figura 21: Sotto-insieme di esempio delle metriche estratte: (a) Ganglia (b) Confluent

Nella loro forma grezza i dati si presentano esattamente nella forma illustrata in Figura 21, nei prossimi due paragrafi analizzeremo il processo di trasformazione citato ad inizio capitolo necessario a dare coerenza ed organizzazione alle metriche raccolte.

**Il processo di rielaborazione dei dati** è stato diviso in **due parti**. **Nella prima** di queste viene effettuata una rielaborazione generica indipendente dal tipo di rete neurale cui i dati verranno serviti. Invece, **nella seconda** parte del processo verrà effettuata una rielaborazione ulteriore che specializza il dataset per la TCN che si vuole realizzare.

## 4.4 Post-elaborazione dei dati all'estrazione

Generalmente parlando, **una rete neurale accetta in ingresso delle *osservazioni* processate sequenzialmente**.

**NOTA**: Con *osservazione* si intende l'insieme dei dati raccolti per un nodo della rete in un dato intervallo temporale.

**Ognuna di queste è composta da un insieme di attributi (*features*) di cui la rete neurale deve scoprire le relazioni che influiscono maggiormente sul risultato finale**. Nel nostro caso in esame, gli attributi corrispondono

alle metriche raccolte dai plugin, mentre il risultato finale è la predizione sullo stato futuro del nodo. Alla luce di ciò, risulta chiara la necessità di rielaborare i dati estratti in modo che il risultato finale sia conforme alla forma dello strato di ingresso della rete neurale (*input layer*).

**NOTA**: Non vi è alcuna differenza tra il processo effettuato per rielaborare i dati di Ganglia e quelli di Confluent.

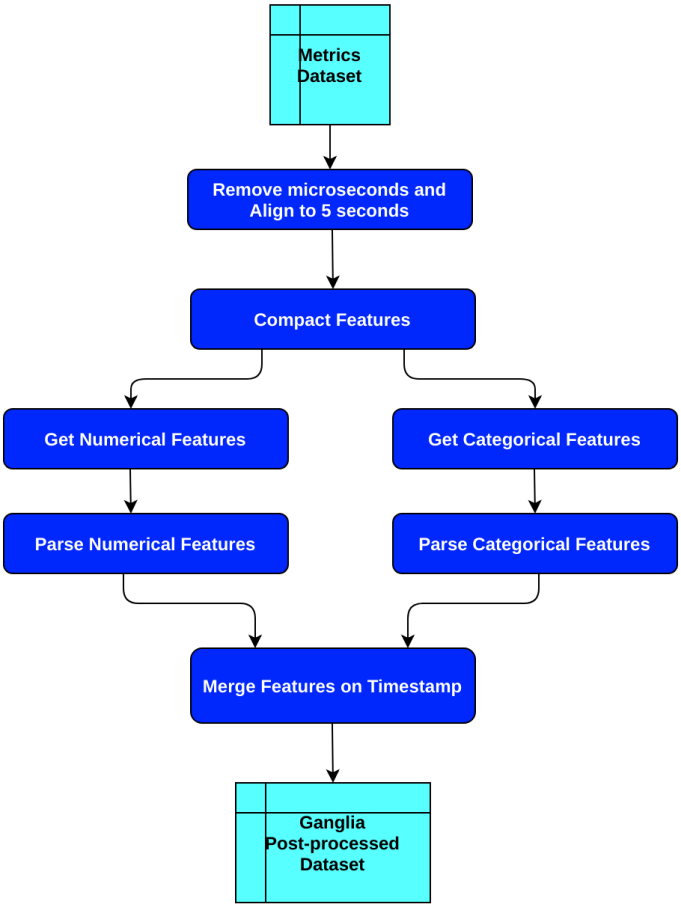



Figura 22: Processo di rielaborazione dei dati estratti - Diagramma di flusso

Per illustrare i risultati intermedi del processo di rielaborazione dei dati, faremo riferimento per semplicità ad un sotto-insieme delle metriche estratte da Ganglia. Come si evince dal diagramma, la prima operazione effettuata è stata quella di **eliminare i microsecondi dal timestamp e di quantizzare i dati su una base di 5 secondi**.

| timestamp | name | value |
|---|---|---|
| 2019-10-18 10:00:55+02:00 | boottime | 1.56174e+09 |
| 2019-10-18 10:01:55+02:00 | boottime | 1.56174e+09 |

Una volta allineati i timestamp, si può procedere nel cambiare l'organizzazione del dataset dalla forma `<timestamp, metric_name, value>` alla forma `<timestamp, metric_1_val, metric_2_val, ... , metric_N_val>`.

| timestamp | boottime | machine_type | os_name | swap_total |
|---|---|---|---|---|
| 2019-10-18 10:00:55+02:00 | 1.561743e+09 | x86_64 | Linux | 16383996.0 |
| 2019-10-18 10:01:55+02:00 | 1.561743e+09 | x86_64 | Linux | 16383996.0 |

A questo punto è necessario **separare gli attributi di tipo numerico e quelli di tipo categorico** e, una volta effettuata questa operazione, si possono rielaborare distintamente i due tipi di dato.
In particolare, **i dati di tipo numerico** vengono riaggregati su una base di 5 minuti. Per fare ciò, è necessario raggruppare, per ogni attributo, i valori compresi nell'intervallo di 5 minuti attualmente in esame e, per perdere meno contenuto informativo possibile, viene calcolata la **media** e la **varianza** di questi valori. Inoltre, considerando che questi valori si riferiscono a dei sotto-intervalli di durata differente, la media e la varianza vengono calcolate dando **pesi differenti** ai valori, i quali crescono proporzionalmente alla durata del sotto-intervallo a cui si riferiscono.

| timestamp | avg:boottime | var:boottime | avg:swap_total | var:swap_total |
|---|---|---|---|---|
| 2019-10-18 10:05:00+02:00 | 1.561743e+09 | 0.0 | 16383996.0 | 0.0 |
| 2019-10-18 10:10:00+02:00 | 1.561743e+09 | 0.0 | 16383996.0 | 0.0 |

**I dati di tipo categorico** vengono aggregati sempre su una base di 5 minuti. Una volta fatto ciò, per rendere coerenti i dati di tipo numerico e quelli di tipo categorico, è necessario **convertire le categorie ad un tipo numerico**. Questo compito richiede semplicemente di modificare ogni categoria per un particolare attributo in un numero intero, creando una corrispondenza biunivoca tra la categoria di partenza ed il valore numerico ad essa assegnato.

Tuttavia, effettuando questa operazione abbiamo implicitamente assegnato agli attributi di tipo categorico un ordine logico/matematico, il quale induce una forma di condizionamento che potrebbe compromettere l'efficacia dell'apprendimento della rete neurale.

Per risolvere questa problematica, viene generalmente utilizzata una **tecnica chiamata *One-Hot Encoding***, la quale consiste nell'eliminare l'attributo categorico in esame, e creare un nuovo attributo per ognuno dei valori che può assumere l'attributo rimosso. I nuovi attributi devono essere di tipo binario e, per ogni osservazione, viene assegnato il valore `1` solamente al nuovo attributo corrispondente al valore che assumeva il dato categorico rimosso (tutti gli altri nuovi attributi assumono il valore `0`, per quella particolare osservazione).

| Sample | Value | Numerical | Value_Red | Value_Blue | Value_Green |
|---|---|---|---|---|---|
| 1 | Red | 1 | 1 | 0 | 0 |
| 2 | Green | 3 | 0 | 0 | 1 |
| 3 | Blue | 2 | 0 | 1 | 0 |

Figura 23: One-Hot Encoding - Esempio attributo `'colore'`

Per esempio, se avessimo un attributo chiamato `'colore'` che può assumere tre tipi di categorie (`'red'`, `'green'`, `'blue'`) mappate nei valori 1, 2 e 3, il nuovo dataset conterrà tre nuovi attributi, chiamati `'red'`, `'green'` e `'blue'`. Se il valore dell'attributo `'colore'` fosse stato `'blue'`, allora i valori dei tre nuovi attributi saranno `[0,0,1]`.

Di seguito è riportato un esempio dei risultati ottenuti applicando queste trasformazioni ai dati di tipo categorico.

| timestamp | machine_type | os_name |
|---|---|---|
| 2019-10-18 10:00:55+02:00 | x86_64 | Linux |
| 2019-10-18 10:01:55+02:00 | x86_64 | Linux |

(a) Prima

| timestamp | machine_type_0 | os_name_0 |
|---|---|---|
| 2019-10-18 10:00:00+02:00 | 1.0 | 1.0 |
| 2019-10-18 10:05:00+02:00 | 1.0 | 1.0 |

(b) Dopo

**Il passaggio finale** per i dati estratti facenti riferimento al plugin Ganglia consiste nell'unire nuovamente i dati di tipo numerico e quelli di tipo categorico in un unico dataset.

| timestamp | avg:boottime | var:boottime | avg:swap_total | var:swap_total | machine_type_0 | os_name_0 |
|---|---|---|---|---|---|---|
| 2019-10-18 10:05:00+02:00 | 1.561743e+09 | 0.0 | 16383996.0 | 0.0 | 1.0 | 1.0 |
| 2019-10-18 10:10:00+02:00 | 1.561743e+09 | 0.0 | 16383996.0 | 0.0 | 1.0 | 1.0 |

**NOTA**: I dati di Ganglia e Confluent erano stati estratti separatamente per cui, alla fine dei passaggi appena illustrati, è necessario unire i due distinti dataset.

## 4.5 Pre-elaborazione dei dati alla TCN

La successiva rielaborazione dei dati è necessaria a rendere il dataset conforme a ciò che si aspetta di ricevere in ingresso la specifica TCN realizzata.

Prima di tutto, facciamo una **breve digressione sul funzionamento della TCN** in modo da poter introdurre più facilmente le modifiche al dataset effettuate in questa fase.

**Per prevedere i futuri passaggi di stato da ordinario ad anomalo per un dato nodo in un particolare intervallo temporale, la TCN considera i dati raccolti nei 127 intervalli temporali atomici precedenti a quello corrente** (in una fase secondaria del progetto vengono considerati solamente i 31 intervalli precedenti). Va considerato che alcuni dati potrebbero non essere presenti e quelli raccolti potrebbero non essere ben formati. Inoltre, ad ogni iterazione la rete neurale deve conoscere i dati relativi allo stato del nodo nell'intervallo temporale successivo e i dati relativi ai 127 intervalli temporali precedenti per effettuare una previsione. Alla luce di ciò, risulta che **la mancanza dei dati relativi anche solo ad un singolo intervallo temporale può corrompere le previsioni effettuate per 128 iterazioni** (quella precedente, per cui manca l'indicazione sullo stato futuro del nodo, e le 127 successive, per cui mancano i dati associati ad almeno un intervallo temporale precedente).

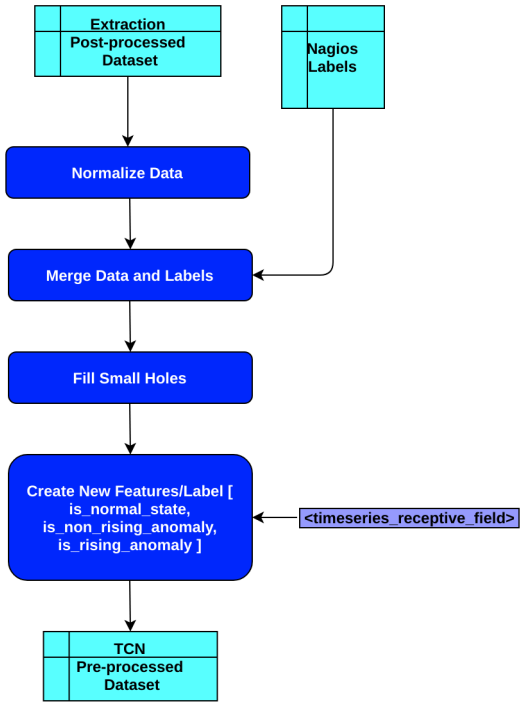



Figura 24: Processo di pre-elaborazione dei dati per la TCN - Diagramma di flusso

Come effettuato nel paragrafo precedente, per analizzare i risultati intermedi dell'ulteriore rielaborazione del dataset, faremo riferimento ad un sottoinsieme degli attributi disponibili per ogni singolo intervallo temporale.

La prima rielaborazione effettuata consiste nella **normalizzazione dei valori sui singoli attributi**, utile ad accelerare la convergenza del gradiente durante la fase di allenamento della TCN. In particolare, si vuole standardizzare la scala di riferimento di tutti gli attributi nell'intervallo compreso tra il valore 0 ed il valore 1:

$$Norm\_x_{i,j} = \frac{x_{i,j} - min_i(x_{i,j})}{max_i(x_{i,j}) - min_i(x_{i,j})} \quad \forall i \in \{0, ..., N-1\}, j \in \{0, ..., M-1\}$$

con $N = numero\ osservazioni$ ed $M = numero\ attributi$.

| timestamp | avg:bytes_in | var:bytes_in | avg:bytes_out | var:bytes_out | HDD_0_Status_0 | MTT_CPU1_0 |
|---|---|---|---|---|---|---|
| 2019-11-02 18:30:00+01:00 | 0.025164 | 6.741918e-05 | 0.000336 | 3.177244e-09 | 0.0 | 0.0 |
| 2019-11-02 18:50:00+01:00 | 0.004530 | 0.000000e+00 | 0.000012 | 0.000000e+00 | 0.0 | 0.0 |

Una volta normalizzati tutti gli attributi, si può procedere con l'**unificazione dei dati di ogni osservazione con lo stato del nodo (*label*)**, effettuata utilizzando il timestamp come valore chiave per attuare questo passaggio.

| timestamp | avg:bytes_in | var:bytes_in | avg:bytes_out | var:bytes_out | HDD_0_Status_0 | MTT_CPU1_0 | label |
|---|---|---|---|---|---|---|---|
| 2019-11-02 18:30:00+01:00 | 0.025164 | 0.000067 | 0.000336 | 3.177244e-09 | 0.0 | 0.0 | 0 |
| 2019-11-02 18:50:00+01:00 | 0.004530 | 0.000000 | 0.000012 | 0.000000e+00 | 0.0 | 0.0 | 0 |

Come citato all'inizio del paragrafo, l'intervallo temporale su cui è stata effettuata l'estrazione (circa 3 mesi) presenta delle zone in cui mancano dei dati. Ogni sotto-intervallo di dati mancanti invalida 128 osservazioni, per cui ho deciso di **creare sinteticamente i dati** per quei sotto-intervalli di minima durata. In particolare, ho deciso di coprire i buchi di durata inferiore ai 45 minuti. Quest'ultimo valore è stato scelto in modo da trovare un compromesso tra il condizionamento introdotto nella rete creando dati fittizi, e la rivalidazione delle osservazioni che erano state rese inutilizzabili a causa di piccoli buchi nel dataset.

La **strategia adottata per creare i valori sintetici degli attributi mancanti** consiste nell'effettuare un'**interpolazione lineare** tra i valori assunti dagli

attributi dell'osservazione immediatamente precedente al buco, e quelli dell'osservazione immediatamente successiva allo stesso.
Propongo un semplice esempio su un singolo attributo: se ci mancano i dati per un intervallo di 20 minuti, ed i valori immediatamente precedente e successivo al buco sono `10` e `20` (per quell'attributo), allora i 4 dati mancanti assumeranno i valori `[12,14,16,18]`.

La seguente illustrazione ci mostra il risultato prodotto dalla creazione sintetica dei dati necessari a coprire un buco di 15 minuti. Si può notare come i valori assunti dagli attributi creati decrescano linearmente.

| timestamp | avg:bytes_in | var:bytes_in | avg:bytes_out | var:bytes_out | HDD_0_Status_0 | MTT_CPU1_0 | label |
|---|---|---|---|---|---|---|---|
| 2019-11-02 18:30:00+01:00 | 0.025164 | 0.000067 | 0.000336 | 3.177244e-09 | 0.0 | 0.0 | 0 |
| 2019-11-02 18:50:00+01:00 | 0.004530 | 0.000000 | 0.000012 | 0.000000e+00 | 0.0 | 0.0 | 0 |

(a) Data with hole

| timestamp | avg:bytes_in | var:bytes_in | avg:bytes_out | var:bytes_out | HDD_0_Status_0 | MTT_CPU1_0 | label |
|---|---|---|---|---|---|---|---|
| 2019-11-02 18:30:00+01:00 | 0.025164 | 0.000067 | 0.000336 | 3.177244e-09 | 0.0 | 0.0 | 0.0 |
| 2019-11-02 18:35:00+01:00 | 0.020005 | 0.000051 | 0.000255 | 2.382933e-09 | 0.0 | 0.0 | 0.0 |
| 2019-11-02 18:40:00+01:00 | 0.014847 | 0.000034 | 0.000174 | 1.588622e-09 | 0.0 | 0.0 | 0.0 |
| 2019-11-02 18:45:00+01:00 | 0.009689 | 0.000017 | 0.000093 | 7.943110e-10 | 0.0 | 0.0 | 0.0 |
| 2019-11-02 18:50:00+01:00 | 0.004530 | 0.000000 | 0.000012 | 0.000000e+00 | 0.0 | 0.0 | 0.0 |

(b) Hole removed

Arrivati fin qui, ci resta solamente il compito di **adattare la label indicante lo stato del nodo all'impostazione necessaria alla rete per predirre il passaggio di stato da ordinario ad anomalo**, piuttosto che la semplice previsione di un'anomalia. Per fare ciò, ho deciso di eliminare l'attributo `'label'` di partenza (lo stato del nodo), a favore di tre nuovi attributi che possono assumere valori binari (0 od 1) :

- `'is_normal_state'` → indica se in un determinato intervallo temporale il nodo di riferimento si trovava in stato ordinario.
- `'is_non_rising_anomaly'` → indica se in un determinato intervallo temporale il nodo di riferimento si trovava in stato anomalo ma, allo stesso tempo, nell'intervallo temporale immediatamente precedente il nodo era già in stato anomalo.

- `'is_rising_anomaly'` → indica se in un determinato intervallo temporale il nodo subisce un cambiamento di stato da ordinario ad anomalo, cioè se lo stato del nodo nell'intervallo temporale immediatamente precedente era ordinario, ed ora invece risulta anomalo.

**L'attributo che interessa maggiormente alla rete neurale**, necessario per perfezionarsi nell'effettuare previsioni sull'insorgere di nuove anomalie, è l'ultimo dei tre citati.

Di seguito viene mostrato il risultato finale della fase di rielaborazione dei dati.

| timestamp | avg:bytes_in | var:bytes_in | avg:bytes_out | var:bytes_out | HDD_0_Status_0 | MTT_CPU1_0 | is_normal_state | is_non_rising_anomaly | is_rising_anomaly |
|---|---|---|---|---|---|---|---|---|---|
| 2019-11-02 18:30:00+01:00 | 0.025164 | 0.000067 | 0.000336 | 3.177244e-09 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 2019-11-02 18:35:00+01:00 | 0.020005 | 0.000051 | 0.000255 | 2.382933e-09 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 2019-11-02 18:40:00+01:00 | 0.014847 | 0.000034 | 0.000174 | 1.588622e-09 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 2019-11-02 18:45:00+01:00 | 0.009689 | 0.000017 | 0.000093 | 7.943110e-10 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 2019-11-02 18:50:00+01:00 | 0.004530 | 0.000000 | 0.000012 | 0.000000e+00 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |

Figura 25: Dataset rielaborato e processabile dalla TCN

## 4.6 Problemi relativi al Dataset

Prima di addentarci nell'implementazione della TCN, è necessario citare alcuni **problemi intrinseci al dataset completo** che abbiamo ora a disposizione. In particolare, sono due gli ostacoli che potrebbero compromettere l'accuratezza delle predizioni effettuate dal modello, ed entrambi sono relativi non tanto ai dati necessari ad effettuare una predizione, quanto alle indicazioni che abbiamo a disposizione sul risultato della predizione che, come ampliamente discusso, fa riferimento al passaggio di stato da ordinario ad anomalo (per un particolare nodo HPC in un determinato intervallo temporale).

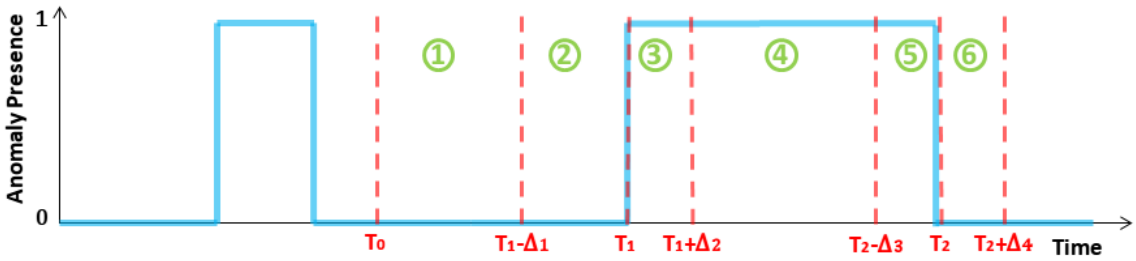
**In primo luogo**, ciò che risulta una volta completata l'estrazione e l'elaborazione dei dati è che **il dataset è molto sbilanciato**. Un dataset risulta sbilanciato quando una classe potenzialmente prevedibile dalla rete è sotto-rappresentata rispetto alle altre classi. Nel nostro caso la nostra rete può prevedere due classi (`'is_rising_anomaly'` $\in$ `[0,1]`) e risulta che il numero di osservazioni relative a `'is_rising_anomaly'=0` è estremamente più elevato rispetto a quelle della classe `'is_rising_anomaly'=1`.

| | total | #non_rising_anomalies | #rising_anomalies |
|---|---|---|---|
| **observations** | 104402 | 104360 | 42 |

Figura 26: Dataset sbilanciato - Numero di osservazioni per ogni classe

Questa situazione generalmente porta alla sintesi di un modello molto bravo nel predirre i dati per le classi più rappresentate, a scapito delle classi meno rappresentate, per cui si prevedono risultati peggiori. Nel caso binario (il nostro), questo problema si riduce al rischio di ottenere un modello capace di effettuare predizioni per una sola classe, che nel nostro caso corrisponde a `'is_rising_anomaly'=0`, ed ovviamente non è ciò che vogliamo ottenere dalla nostra rete.

**In secondo luogo**, è emerso un problema che sta a monte del progetto di tesi e riguarda l'inserimento manuale da parte degli amministratori di sistema dei dati su Nagios relativi allo stato dei nodi.




In particolare, quando un nodo presenta una reale anomalia nessun operatore è in grado di agire istantaneamente. Quindi, nell'intervallo di tempo tra il momento in cui il nodo passa realmente in stato anomalo ($T_1 - \Delta_1$) e quello in cui l'operatore registra lo stato `DRAIN` ($T_1$), passano un certo numero di intervalli di tempo ($\Delta_1$) in cui il nodo risulta in stato normale, anche se così non è. Nel nostro caso, è necessario accettare questo fenomeno senza agire di conseguenza.

In generale, possiamo identificare **sei fasi che descrivono il modo in cui viene gestita un'anomalia dagli amministratori di sistema**, e quindi dello stato del nodo:

1. Il nodo funziona correttamente e quindi ci sono dei processi in esecuzione.
   - Intervallo temporale: $T_0 \rightarrow T_1 - \Delta_1$
   - Stato attuale: `[1,0,0]`
   - Stato reale : `[1,0,0]`

**2.** Il nodo inizia a funzionare in modo non corretto ma nessuno se ne è ancora accorto, gli utenti possono accedere ai nodi e ci sono dei processi in esecuzione.

- Intervallo temporale: $T_1 - \Delta_1 \rightarrow T_1$
- Stato attuale: `[1,0,0]`
- Stato reale : `[0,0,1] + [0,1,0]`

**3.** Il nodo funziona in modo non corretto, se ne sono accorti ed è stato reso non raggiungibile dagli utenti. Ci potrebbero essere dei processi ancora in esecuzione ma in fase di terminazione.

- Intervallo temporale: $T_1 \rightarrow T_1 + \Delta_2$
- Stato attuale: `[0,0,1] + [0,1,0]`
- Stato reale : `[0,1,0]`

**4.** Il nodo funziona in modo non corretto, se ne sono accorti ed è stato reso non raggiungibile dagli utenti. I processi precedentemente in esecuzione sono ora terminati.

- Intervallo temporale: $T_1 + \Delta_2 \rightarrow T_2 - \Delta_3$
- Stato attuale: `[0,1,0]`
- Stato reale : `[0,1,0]`

**5.** L'anomalia è stata risolta ed il nodo funziona correttamente, ma ancora non è stato reso raggiungibile dagli utenti. Non ci sono processi in esecuzione.

- Intervallo temporale: $T_2 - \Delta_3 \rightarrow T_2$
- Stato attuale: `[0,1,0]`
- Stato reale : `[1,0,0]`

**6.** Il nodo funziona correttamente e gli utenti possono accedervi. Potenzialmente ci sono dei processi in esecuzione, ma potrebbe passare altro tempo affinchè il nodo torni ad essere effettivamente attivo.

- Intervallo temporale: $T_2 \rightarrow T_2 + \Delta_4$
- Stato attuale: `[1,0,0]`
- Stato reale : `[1,0,0]`

**NOTA**: "Stato" si riferisce alla tripla `['is_normal_state','is_non_rising_anomaly','is_rising_anomaly']`

Sinteticamente, nel momento in cui l'operatore registra lo stato anomalo ($T_1$) passa un certo intervallo di tempo ($\Delta_2$) in cui i nodi vengono "chiusi" ma in cui i processi attualmente presenti restano in esecuzione. Quando i processi in esecuzione terminano ($T_1 + \Delta_2$) la macchina risulta ancora in stato `DRAIN`, anche se in realtà è in `IDLE`. Quando l'operatore risolve il problema ($T_2 - \Delta_3$) lo stato rimane etichettato come anomalo per un altro intervallo temporale ($\Delta_3$), anche se così non è. Infine lo stato viene modificato nuovamente in normale ($T_2$) e il nodo torna in funzione.

## 4.7 Implementazione della TCN - Punto di Partenza

**Come modello di partenza**, ho deciso di implementare una TCN basilare per come viene descritta nel paper di Shaojie et. al [X], con l'**unica modifica** riguardante il dropout che, nel mio caso, viene impostato in modo del tutto generale con una ratio di 0.2 (In ogni fase dell'allenamento vengono eliminate il 20 % delle connessioni totali, che verranno poi reinserite prima della fase successiva), diversamente da quanto specificato nel paper scientifico appena citato, in cui in ogni fase di allenamento è prevista la cancellazione delle connessioni relative ad un'intera metrica (scelta casualmente).

Prima di analizzare questa prima versione della rete neurale implementata, è bene fare una brevissima **digressione sulle librerie** di cui ho fatto uso per la sintesi di quest'ultima. In particolare, grazie alla sua facilità d'uso ho deciso di utilizzare `Keras` **come interfaccia di alto livello**, una libreria open-source scritta in Python, progettata per il Machine Learning e le reti neurali. Mentre **come interfaccia di basso livello (backend) ho deciso di utilizzare** `TensorFlow`, una libreria open-source realizzata da Google (sempre per il ML e le reti neurali), comprendente moduli sperimentati e ottimizzati per l'esecuzione di basso livello. Infatti, quest'ultima può essere utilizzata per la sintesi di algoritmi di ML su un vasto tipo di architetture hardware (ASIC, FPGA, GPU e CPU).

Torniamo sulla nostra TCN. Ebbene, come punto di partenza ho deciso di **impostare a 128 in campo di ricezione della rete convoluzionale**, relativamente alle osservazioni precedenti a quella che deve essere processata. Questo significa che, nel momento in cui la TCN elabora un'osservazione, ha la possibilità di tenere in considerazione i valori registrati nei 127 intervalli temporali precedenti. Ossia che, per prevedere l'insorgere di un'anomalia, può attingere alle informazioni registrate da 10 ore e 40 minuti prima dell'intervallo corrente

($128 \cdot 5min = 640min = 10h\ 40min$; ogni osservazione aggrega le informazioni registrate durante i 5 minuti precedenti).

In più, cercando di mantenere il prototipo iniziale in più semplice possibile, si è deciso di **limitare la capacità previsionale della rete all'insorgere di un'anomalia relativamente all'intervallo temporale successivo rispetto a quello in esame**. Vale a dire che l'**obiettivo che si vuole raggiungere** è quello di poter prevedere in anticipo di 5 minuti l'insorgere di una futura anomalia.

**Il processo logico che ho deciso di seguire per ottenere la sintesi completa del modello**, una volta ottenuto il dataset, è il seguente (ref. Figura 29):

**1a.** In primo luogo, è necessario **dividere il dataset di partenza** in modo da ottenere dei dati da utilizzare per la fase di allenamento (**Training/Validation data and labels**) e di test (**Test data and labels**) della rete neurale. La divisione tra i dati di allenamento e di test è d'obbligo: le osservazioni destinate alla fase di test sono indispensabili per valutare le performance del modello su dei dati mai visti prima. La divisione tra dati di allenamento e validazione non è sempre necessaria: può risultare utile nel caso in cui si vogliano effettuare delle valutazioni sulle performance del modello mentre questo è in fase di allenamento. Personalmente, ho deciso di destinare il 20% dei dati per la fase di test, ed il restante per la fase di allenamento. Di questi ultimi, ne ho dedicato il 20% per il processo di validazione (16% sul totale), destinando quindi solamente il 64% del dataset di partenza per l'allenamento effettivo la rete.

Inoltre, ho deciso di utilizzare un **generatore di serie temporali** per esporre i dati al modello. Quest'approccio porta a **due vantaggi**: **(1)** in questo modo si possono mantenere in memoria solamente i dati attualmente processati dalla rete, piuttosto che l'intero dataset, che nel nostro caso comprende $10^5 \cdot 113$ valori da 64 bit (`float64`) e **(2)** il generatore solleva dal compito della formattazione dei dati da esporre alla rete.

In particolare, per questo secondo punto, è necessario aprire una **breve parentesi sulla formattazione dei dati che si aspetta la rete in ingresso**, dettata dal primo strato convoluzionale. Infatti, durante la fase di allenamento e per ogni osservazione, la convoluzione causale si aspetta in ingresso non solamente l'osservazione stessa, ma anche le osservazioni contenute nel campo di ricezione temporale della TCN. Inoltre, l'informazione che specifica se una particolare osservazione coincide con l'insorgere di un'anomalia deve essere assegnata all'osservazione precedente, in quanto la rete deve essere in grado di effettuare delle previsioni avanti nel futuro, piuttosto che descrivere correttamente lo stato presente.

**1b.** Parallelamente, si può procedere con la **costruzione del modello di partenza** che, al netto di improvvisi colpi di fortuna o di una grande esperienza maturata negli anni in quest'ambito, non garantirà quasi mai i risultati sperati.

**2.** Una volta costruito il modello si può procedere con la **fase di allenamento**, utilizzando il sotto-insieme del dataset di partenza (Training/Validation data and labels) destinato a questo scopo. Durante questa fase occorre **effettuare delle scelte**:

- Il **numero di osservazioni da elaborare in ogni pacchetto per il calcolo del gradiente** (discretizzato in gruppi di osservazioni), il quale va ad aggiornare opportunamente i pesi delle connessioni durante la fase di *backpropagation*. Questo parametro viene indicato come `batch_size` e può variare a seconda che si stiano processando i dati di allenamento, di validazione o di test. Nel mio caso, ho deciso di impostare questo valore a 64 per i dati di allenamento, e 32 per i dati di validazione e test.
- La **Loss Function** che si vuole utilizzare per calcolare le differenze tra il risultato sperato e quello effettivo di una previsione. Nel nostro caso si è scelto di utilizzare la funzione `"binary_crossentropy"`, che risulta calzante nel caso in cui si voglia prevedere la classe di appartenenza di un'osservazione, nel momento in cui la scelta si riduce a due classi:

  $$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

  con $N$ pari al numero di elementi nella batch (`batch_size`), $y$ come risultato atteso (0 od 1) ed $\hat{y}$ corrispondente alla predizione effettuata dalla rete, cioè la probabilità che l'osservazione elaborata appartenga alla classe `1`.

  Dalla formula si può intuire come il primo termine della somma ($y_i \cdot log(\hat{y}_i)$) si annulli quando il risultato atteso coincide alla classe `0`, penalizzando solamente la differenza tra 0 ed $\hat{y}_i$ (si ricorda che $log(1) = 0$ e $log(0) = -\infty$) (Analogamente se il risultato atteso coincide con la classe `1`).
- L'**algoritmo di ottimizzazione** necessario per trovare la configurazione ottimale dei pesi delle connessioni della rete neurale che minimizzi la Loss Function: ho deciso di utilizzare ***Adam* (Adaptive moment estimation)** in quanto, rispetto al più comunemente usato *SGD* (Stochastic Gradient Descent) questo fa uso di un diverso *learning rate* per ogni connessione della rete, ed è stato dimostrato da risultati empirici che questo approccio velocizza l'apprendimento del modello.

- Il **numero di epoche** per cui si vuole allenare la rete: durante ogni epoca il modello processa l'intera porzione del dataset destinato a questo fase, a meno di scelte progettuali differenti (k-fold Cross-Validation, ecc).

**3.** Una volta completata la fase di allenamento si può procedere con la **valutazione dei risultati** inferibili dai valori ottenuti dalla *Loss Function* durante la fase stessa. In particolare, ho deciso di confrontarne, per le diverse epoche, le differenze tra *training* e *validation loss*, in modo da capire il momento in cui il modello inizia a degenerare mostrando overfitting. In questo modo, si può aggiustare il numero delle epoche per cui si vuole allenare la rete, cercando di ottenere un risultato ottimale per la successiva iterazione di allenamento, che funga da compromesso tra underfitting ed overfitting.


Figura 27: Training/Validation Loss - Overfitting

Una tecnica che svolge questo compito in modo automatizzato prende il nome di *Early Stopping* e, talvolta, l'ho utilizzata in sostituzione al controllo manuale appena descritto.

**4.** Si può ora procedere con la **fase di test** in cui, in prima istanza, vengono effettuate dalla rete delle previsioni per dei dati mai visti prima (Test data).

**5.** Durante queste fase vengono **valutate le prestazioni della rete** a seguito delle operazioni descritte al punto precedente. Occorre quindi selezionare una **metrica** per poter valutare i risultati delle previsioni ( *"accuracy", "precision", "recall", "f_score",* ecc), in linea con le specifiche del problema da risolvere a monte del progetto. Per calcolare queste metriche è necessario conoscere i concetti di

- **True Positive (TP)**: il numero di osservazioni per cui è stata prevista una futura anomalia in modo corretto.
- **False Positive (FP)**: il numero di predizioni per cui è stata prevista una futura anomalia in modo non corretto.
- **True Negative (TN)**: il numero di predizioni per cui non è stata prevista una futura anomalia in modo corretto.
- **False Negative (FN)**: il numero di predizioni per cui non è stata prevista una futura anomalia in modo non corretto.

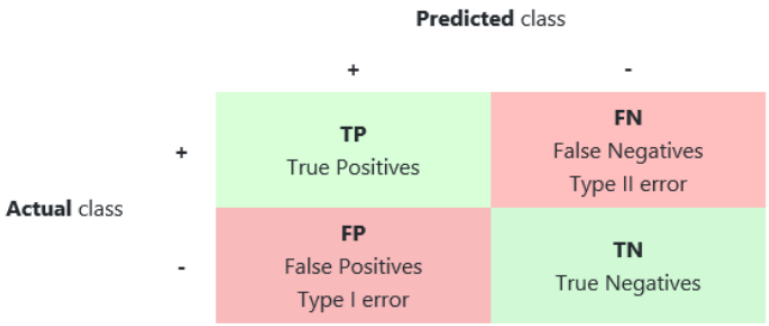



Figura 28: Matrice di Confusione (*Confusion Matrix*)

Nel nostro caso, **la metrica adottata corrisponde a *f1_score***, la quale tiene conto in egual misura di *precision* ($\frac{TP}{TP+FP}$) e *recall* ($\frac{TP}{TP+FN}$). In parole povere ed in modo specifico al nostro problema, si vuole tenere conto della *precisione* per fare in modo che l'amministratore di sistema possa intervenire preventivamente sul numero maggiore di anomalie del sistema, mentre si vuole tenere in considerazione il *richiamo* per fare in modo che l'amministratore di sistema venga notificato di una futura anomalia per il minor numero di volte possibile, se questa non è stata prevista in modo corretto.

Una volta ottenuto il risultato della metrica adottata, si può valutare se questo è in linea con le specifiche del progetto e, in caso positivo, si può considerare la rete neurale come ultimata, altrimenti è necessario introdurre delle modifiche alla rete in modo da migliorarne le performance.

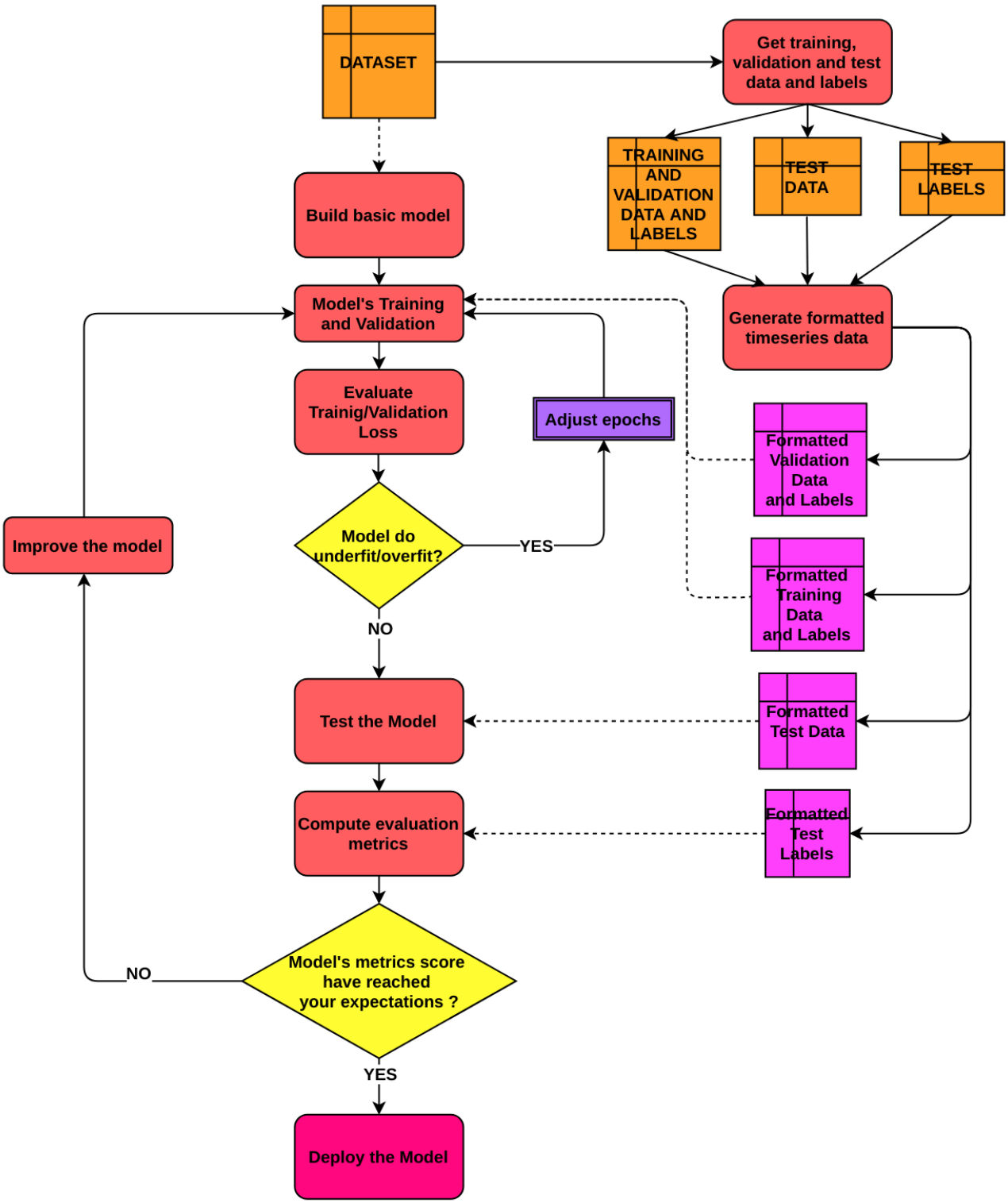



Figura 29: TCN model deployment flow chart

Come ampliamente discusso, l'architettura della versione di base della TCN emula quasi perfettamente quella descritta nel capitolo dedicato allo stato dell'arte. Tuttavia, occorre **aggiungere uno strato di output** che, in questo caso, è composto da un solo neurone. In particolare, quest'ultimo fornisce la **probabilità secondo cui l'osservazione attualmente elaborata conduca al futuro insorgere di un'anomalia**, per questo motivo la **funzione di attivazione** di tale neurone è la *funzione sigmoidea*, la quale ha il compito di mappare i valori in ingresso (precedentemente sommati in modo pesato dalle connessioni in ingresso al neurone) in un valore compreso tra 0 ed 1 esclusi.

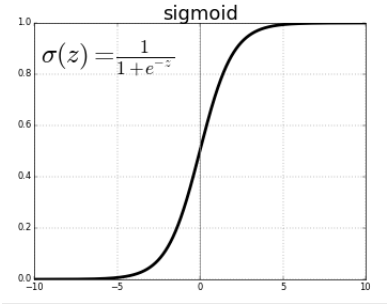



Figura 30: *Sigmoid Function*

Il codice scritto in Python, necessario per la sintesi della TCN grezza, è il seguente:

```
timeseries_receptive_field = 128
n_features = x_train.shape[1]
kernel_size = 2

dilation_rates = tcnu.get_dilation_rates(timeseries_receptive_field, kernel_size)
```

```
--- Dilation Rates ---
[1, 2, 4, 8, 16, 32, 64]
```

```
X_in = Input(shape=(timeseries_receptive_field, n_features))
X = X_in

for dilation_rate in dilation_rates :
    filters = n_features if dilation_rate == dilation_rates[-1] else 1

    dilated_convolution = Conv1D(filters, kernel_size, \
                                 padding='causal', dilation_rate=dilation_rate, activation='relu')

    X = tfa.layers.WeightNormalization(dilated_convolution)(X)
    X = Dropout(0.2)(X)

X_out = Add()([X_in, X])
X_out = Flatten()(X_out)
X_out = Dense(1, activation='sigmoid')(X_out)

tcn = Model(inputs=X_in, outputs=X_out)
tcn.compile(optimizer='adam', loss='binary_crossentropy')
```

Di seguito viene illustrata una schematizzazione basilare della rete risultante:

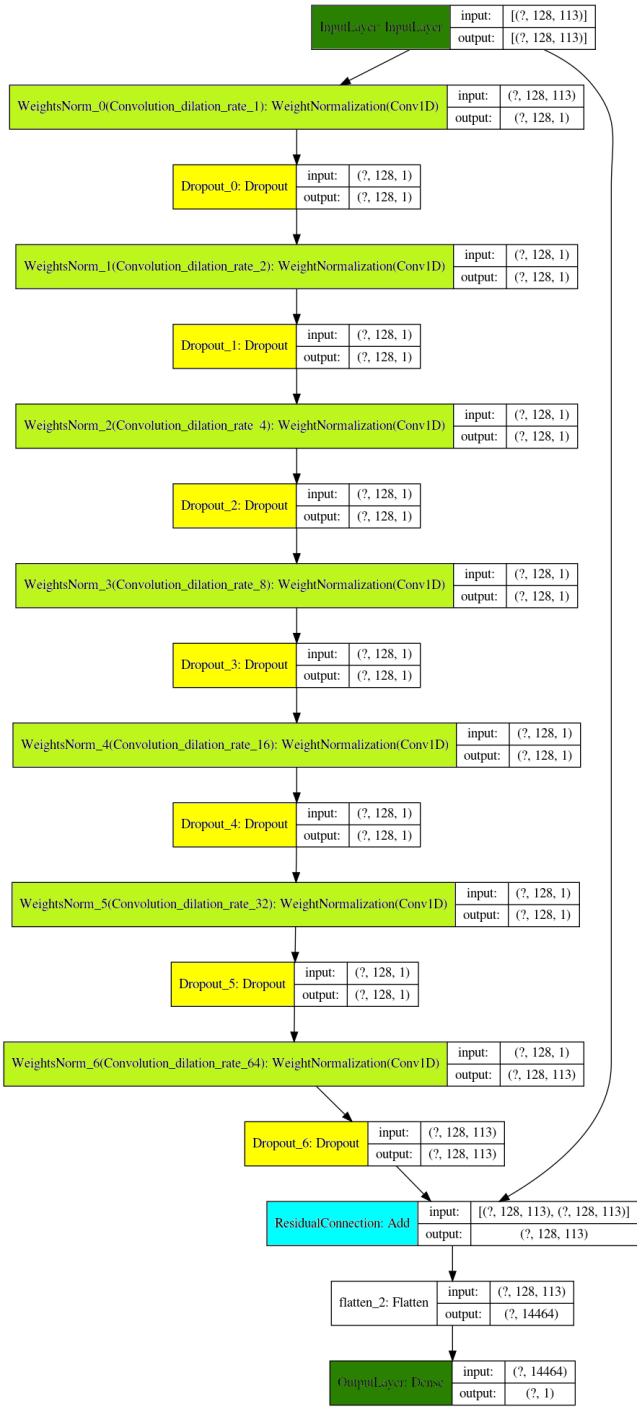



Figura 31: Architettura di base della TCN sintetizzata

Accidentalmente, i **risultati ottenuti dall'allenamento** di questa prima implementazione sono pessimi.

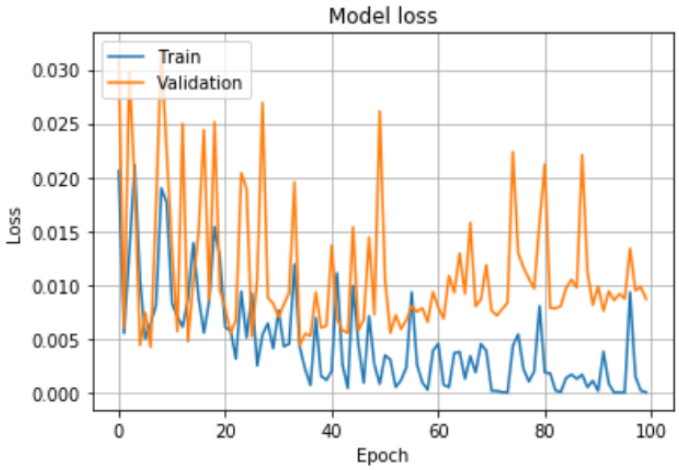



Figura 32: Training/Validation Loss - 100 *Epochs* - First TCN

**In prima istanza**, possiamo analizzare i trend suggeriti dal grafico che mappa l'evoluzione del Training/Validation Loss durante lo scorrere delle epoche: la discontinuità mostrata nei trend, in particolare per quanto riguarda la presenza di numerosi picchi, dimostrano come il modello soffra del problema dell'esplosione del gradiente, il quale è spesso accoppiato al problema dello svanimento del gradiente, anche se non è direttamente inferibile dal grafico in esame. Questa analisi non è sufficiente a trarre delle conclusioni oggettive, pertanto andiamo ad analizzare i **risultati ottenuti dopo la fase di test**.

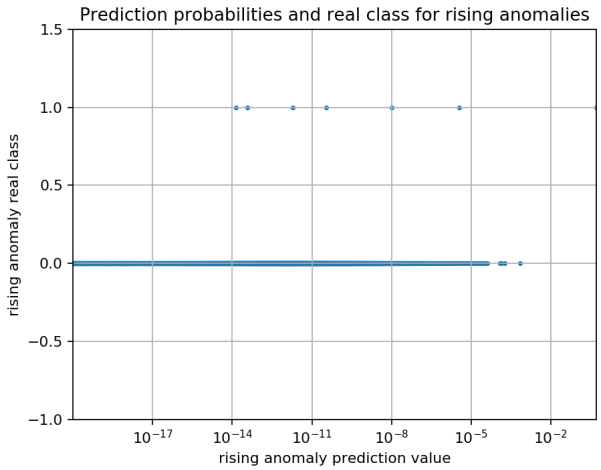



Figura 33: Prediction results - First TCN

Come si può notare dal grafico sopra, che mappa sull'asse `x` i valori ottenuti per le predizioni effettuate in fase di test e sull'asse `y` la classe di appartenenza effettiva, **pare che la rete non sia in grado di prevedere il futuro insorgere di anomalie** e, in più, pare non esista alcuna distribuzione non casuale per quanto riguarda le previsioni effettuate.

Non mi aspettavo buoni risultati dopo questa prima implementazione, quindi possiamo iniziare ad aggiungere qualche elemento al modello per cercare di ottenere risultati migliori.

## 4.8 Miglioramenti Incrementali

Complessivamente, le **modifiche introdotte gradualmente nella rete** sono le seguenti:

1. **Riduzione del dataset** alle sole osservazioni con una distanza temporale minore di un certo valore (1000) rispetto all'insorgere di una nuova anomalie. Riduzione del campo di ricezione temporale da 128 a 32 osservazioni.
2. **Inserimento di una connessione residua** per ogni strato convoluzionale.
3. **Incremento del numero di filtri** per ogni strato convoluzionale (da 1 al numero di attributi di un'osservazione).
4. **Inserimento del Dropout** a monte dell'output layer.

**Per valutare le differenze** tra le performance ottenute dai vari modelli confronteremo **tre parametri**: In primo luogo valuteremo il **tempo di allenamento del modello, su 100 epoche**, risultante dal dataset in esame. Successivamente valuteremo **la media, su 5 esecuzioni indipendenti**, dei valori ottenuti per la metrica *f1_score*. In ultimo faremo delle **considerazioni riguardanti le soglie impostate** per assegnare ad una determinata osservazione una classe di appartenenza, a partire dalla probabilità ottenuta in fase di test per quella determinata classe.

In particolare, nel momento in cui ad un'osservazione viene assegnata una probabilità di appartenenza alla classe `1`, si deve decidere se quest'ultima sia abbastanza alta per essere effettivamente considerata premonitrice dell'insorgere di una futura anomalia. Per fare ciò, viene impostato un **limite inferiore di probabilità** sotto il quale si considera che l'osservazione non conduca ad una futura anomalia.

Per decidere questa soglia ho adottato una tecnica che consiste nell'**identificare, per ogni esecuzione indipendente, l'impostazione di questo limite inferiore che porta al risultato migliore in termini di *f1_score***.

Per esempio, durante un'esecuzione potrei valutare i seguenti limiti inferiori [0.1, 0.2, ... , 0.9], a questo punto verrebbero assegnate diverse classi di appartenenza (classe `0` o classe `1`) alle osservazioni effettuate, a seconda che la probabilità che queste vi appartengano siano inferiori o meno alla soglia corrente. In questo modo, a partire dalle diverse classi assegnate per i differenti valori delle soglie, si possono calcolare i valori della *f1_score*. Se ipotizziamo che questi siano [0.1, 0.12, 0.08, 0.33, 0.39, 0.4, 0.44, 0.08, 0.2, 0.5], non serve altro che scegliere il valore più alto, in questo caso 0.5, e quindi recuperare la soglia impostata per ottenere tale divisione in classi che ha portato al risultato ottimale che, in questo caso, risulta essere 0.9.

### 4.8.1 Prima Modifica: Riduzione del Dataset e del Campo di Ricezione Temporale

In questa fase si è scelto di attuare delle modifiche che sono indirizzate, in prima istanza, a **diminuire il tempo di allenamento della rete neurale**, che risulta aggirarsi intorno ai **3min 44s**. Inoltre, questi cambiamenti portano ulteriori vantaggi.

In particolare, si è deciso di filtrare il dataset di partenza, preservando solamente le osservazioni al più distanti 1000 timestep rispetto all'insorgere di un'anomalia (1000 timestep = 5000 min = 3days 11h 20min). In questo modo il dataset si riduce di 5 volte (da 100k a 20k osservazioni, circa) e, eliminado solo osservazioni che non conducono a future anomalie, **si è limitato il problema dello sbilanciamento del dataset**.

In secondo luogo ho deciso di **diminuire il campo di ricezione dello stack di convoluzioni dilatate da 128 a 32**, questo significa che ora le osservazioni che verranno analizzate dalla TCN potranno dare uno sguardo allo storico non oltre le osservazioni registrate nelle 2h 40min precedenti. In questo modo, lo stack di convoluzioni dilatate si riduce da 7 a 5 convoluzioni distinte.

In Figura 34 si può dare uno sguardo all'**architettura della rete** risultante.

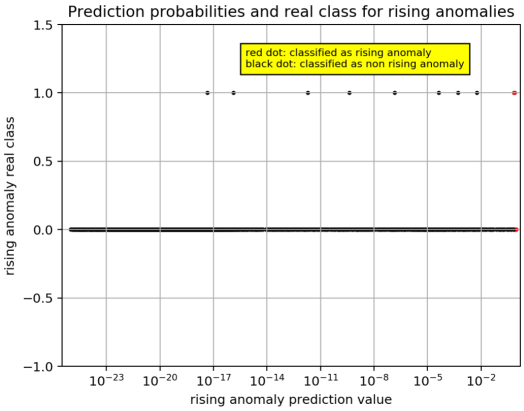
I **risultati ottenuti dopo 5 esecuzioni indipendenti della fase di allenamento e di test** fanno pensare che, con un dataset meno sbilanciato, la rete neurale assume un comportamento che si discosta dalla casualità mostrata per l'implementazione basilare descritta nel paragrafo precedente.

In particolare, **i valori calcolati per *f_score*** hanno una media di **0.16**, con un **massimo** registrato a **0.27** ed un **minimo** a **0.1**. Sicuramente questi valori non sono un punto di arrivo, il quale potrebbe prevedere un punteggio per l'*f_score* non inferiore a 0.5, ma sicuramente è un **notevole miglioramento rispetto alla versione grezza della rete e del dataset**, con i quali si erano ottenuti dei risultati inferiori di almeno un ordine di grandezza.

Allo stesso tempo, per quanto riguarda i **limiti inferiori (in termini di probabilità) che determinano gli assegnamenti delle osservazioni ad una classe**, si presenta una **media** (sulle 5 esecuzioni indipendenti) di **0.42**, con un **massimo** registrato a **0.70** ed un **minimo** registrato a **0.25**. Tutto sommato, sembra che queste esecuzioni indipendenti portino a dei risultati con una **scarsa varianza delle soglie**, se consideriamo che siamo all'interno dello stesso ordine di grandezza (i decimi).

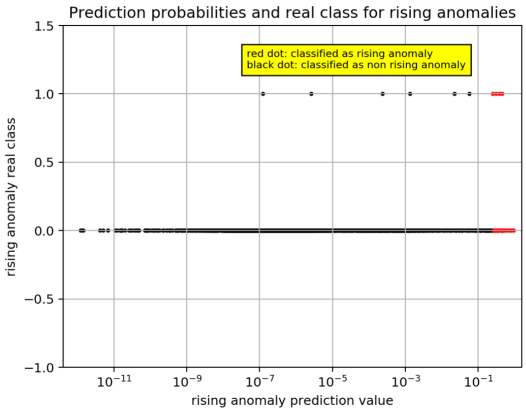
**Ciò che ricerchiamo per quanto riguarda le soglie** ottenute che portano ai risultati ottimali è proprio questo, cioè che siano il più simile possibile tra loro (i.e. con bassa varianza). **La motivazione di questa necessità** risiede nel fatto che, una volta sintetizzato il modello, non è possibile cambiare dinamicamente le soglie, quindi bisogna averne scelta una che porti in generale a dei buoni risultati.

## Execution 1



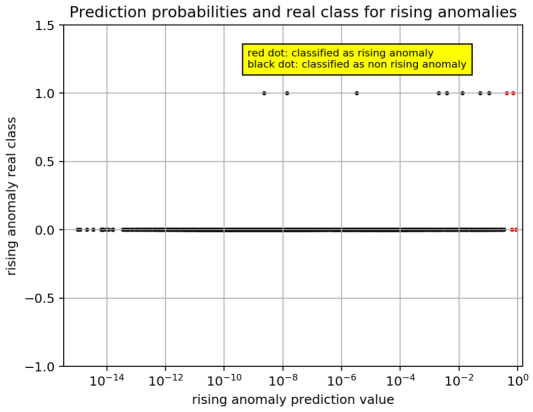

```
True Negative (TN): 3902
False Positive (FP): 13
False Negative (FN): 8
True Positive (TP): 2
f1_score :  0.16
LB Threshold:  7.03e-01
UB Threshold:  1
```

## Execution 2



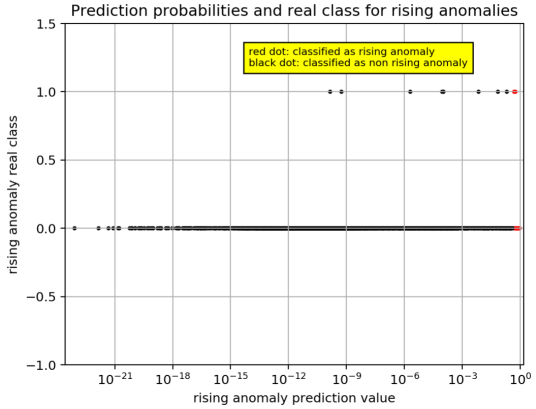

```
True Negative (TN): 3852
False Positive (FP): :  63
False Negative (FN) 6
True Positive (TP): 4
f1_score :  0.1039
LB Threshold 2.53e-01
UB Threshold:  1
```

## Execution 3



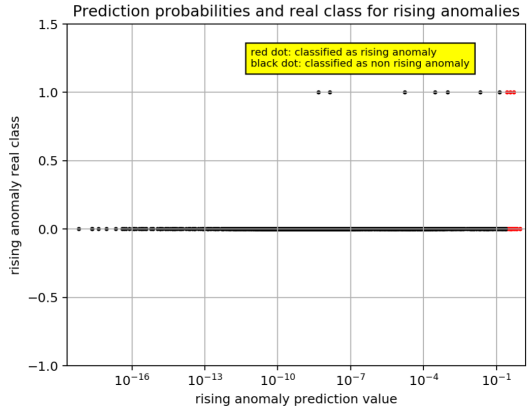

```
True Negative (TN): 3912
False Positive (FP): :  3
False Negative (FN) 8
True Positive (TP): 2
f1_score :  0.2667
LB Threshold 4.24e-01
UB Threshold:  1
```

## Execution 4




```
True Negative (TN): 3890
False Positive (FP): :  25
False Negative (FN) 8
True Positive (TP): 2
f1_score :  0.1081
LB Threshold 4.80e-01
UB Threshold:  1
```

## Execution 5




```
True Negative (TN): 3892
False Positive (FP): 23
False Negative (FN): 7
True Positive (TP): 3
f1_score :  0.1667
LB Threshold:  2.71e-01
UB Threshold:  1
```

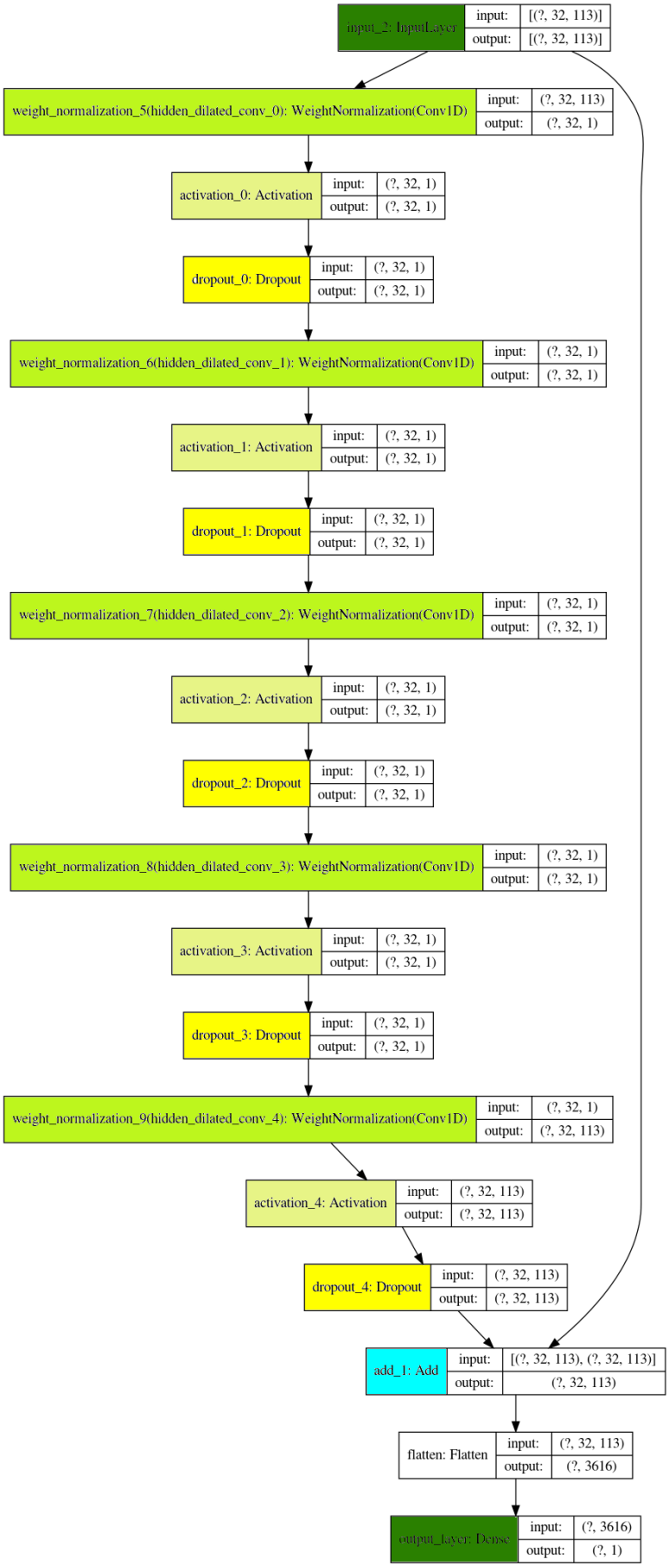



Figura 34: Architettura TCN - Riduzione del dataset e del Campo di Ricezione

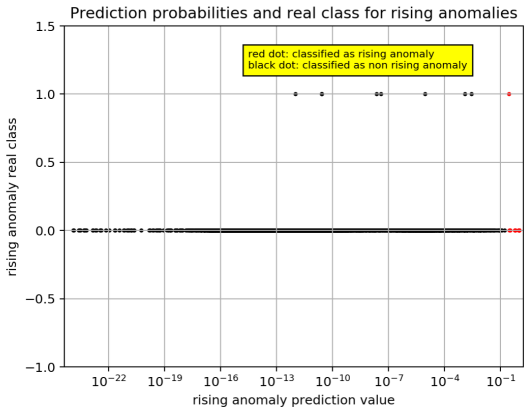
### 4.8.2 Seconda Modifica: Inserimento Connessioni Residue

La seconda modifica effettuata è stata guidata dalla **necessità di ridurre il problema dello svanimento del gradiente**. Precisamente, sono state introdotte delle **ulteriori connessioni residue**, le quali permettono alla rete di apprendere le differenze tra i valori in uscita da uno strato convoluzionale ed il successivo, piuttosto che il valore dell'uscita stessa. Questo passaggio introduce **nuove connessioni per ogni strato convoluzionale**, quindi nuovi pesi da apprendere, pari al numero delle uscite dello stesso (i.e. `<timeseries_receptive_field> x <n_features>`; nel nostro caso: `32 x 113`). In questo modo la rete viene appesantita, anche se questo non risulta un grosso problema in quanto, grazie alle riduzioni del dataset e del campo di ricezione degli strati convoluzionali effettuati precedentemente, la rete è diventata molto veloce in fase di apprendimento. Precisamente, il **tempo necessario a completare questa fase** risulta essere circa di **7min 42s**, più o meno raddoppiato rispetto alla modifica precedente.

In Figura 35 si può dare uno sguardo all'**architettura della rete** risultante.

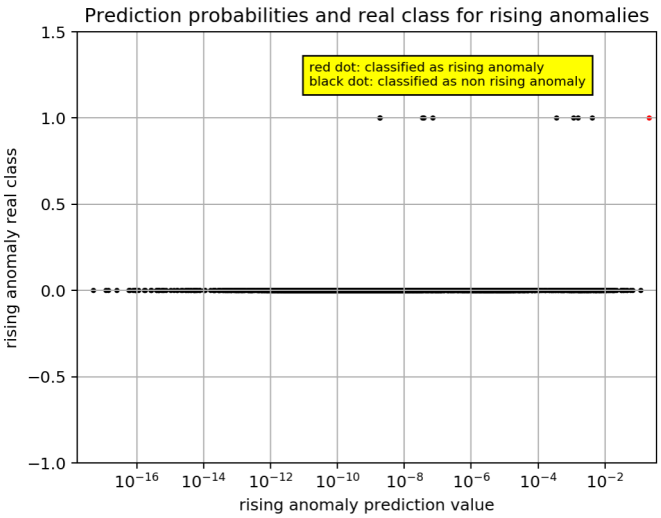
I **risultati ottenuti dopo 5 esecuzioni indipendenti della fase di allenamento e di test** dimostrano come il modello abbia beneficiato di un **miglioramento delle performance rispetto alla versione precedente** dello stesso. In particolare, la **media dei valori ottenuti per l'*f_score*** ha subito un incremento di 0.032, attestandosi a **0.192**. Inoltre, per quanto riguarda la **varianza dei valori ottenuti per il limite inferiore di probabilità della classe** `1`, necessario ad identificare un'osservazione come premonitrice dell'insorgere di una futura anomalia, osserviamo un **lieve miglioramento**. Infatti, il **minimo** ottenuto per questo valore risulta essere **0.1**, mentre il **massimo** si attesta a **0.352**, con una **media sulle 5 esecuzioni** pari a **0.231**.

## Execution 1



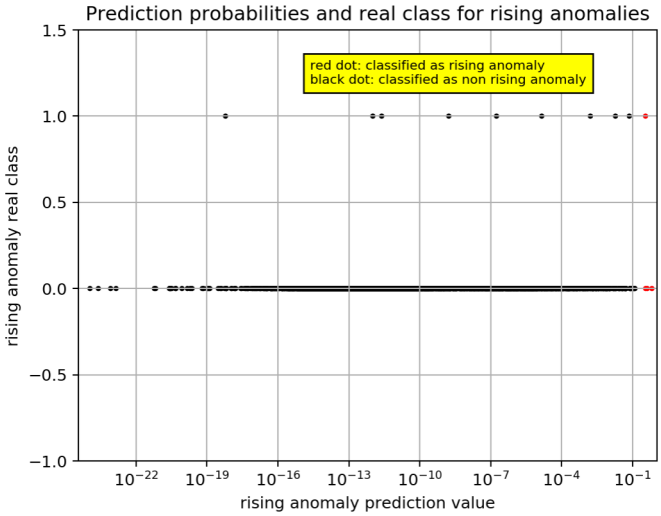

```
True Negative (TN): 3904
False Positive (FP): 11
False Negative (FN): 9
True Positive (TP): 1
f1_score :  0.09
LB Threshold:  2.8e-01
UB Threshold:  1
```

## Execution 2



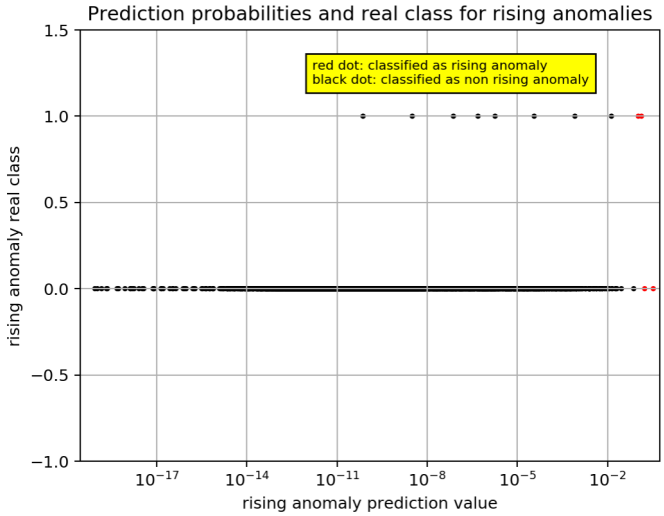

```
True Negative (TN): 3915
False Positive (FP): :  0
False Negative (FN) 9
True Positive (TP): 1
f1_score :  0.18
LB Threshold 2.08e-01
UB Threshold:  1
```

## Execution 3



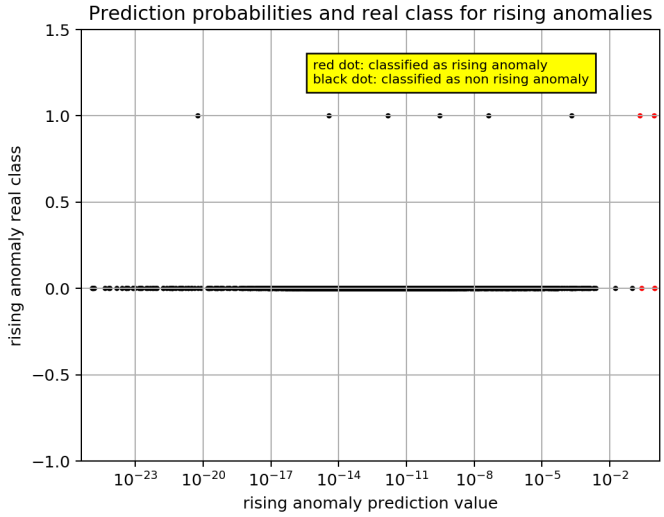

```
True Negative (TN): 3912
False Positive (FP): :  3
False Negative (FN) 9
True Positive (TP): 1
f1_score :  0.14
LB Threshold 3.52e-01
UB Threshold:  1
```

## Execution 4




```
True Negative (TN): 3913
False Positive (FP): :  2
False Negative (FN) 8
True Positive (TP): 2
f1_score :  0.286
LB Threshold 1e-1
UB Threshold:  1
```

## Execution 5




```
True Negative (TN): 3912
False Positive (FP): 3
False Negative (FN): 8
True Positive (TP): 2
f1_score :  0.267
LB Threshold:  2.17e-01
UB Threshold:  1
```

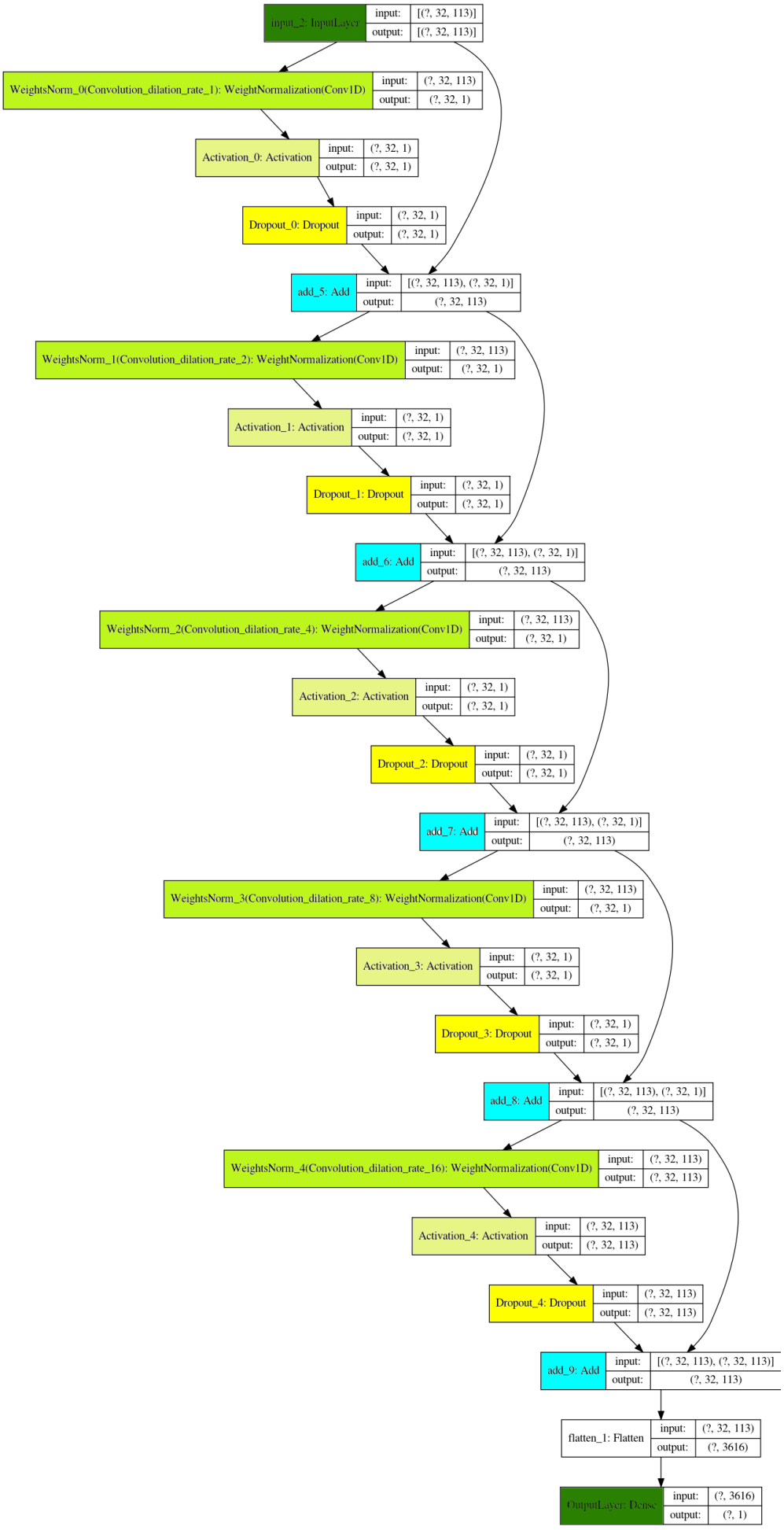


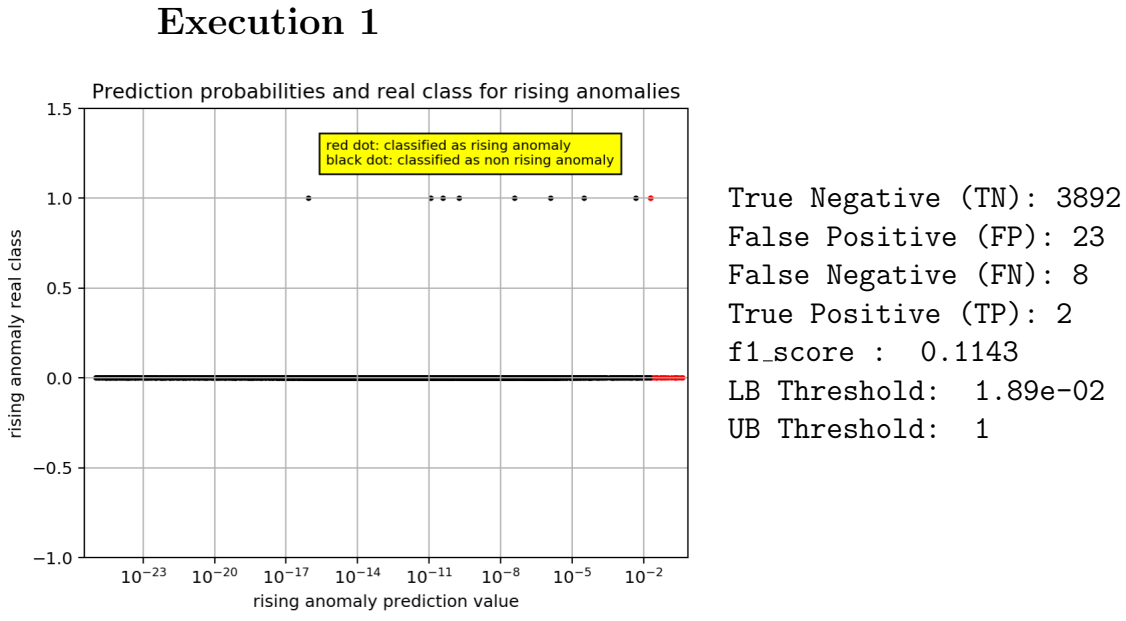

Figura 35: Architettura TCN - Inserimento delle Connessioni Residue

### 4.8.3 Terza Modifica: Incremento Filtri

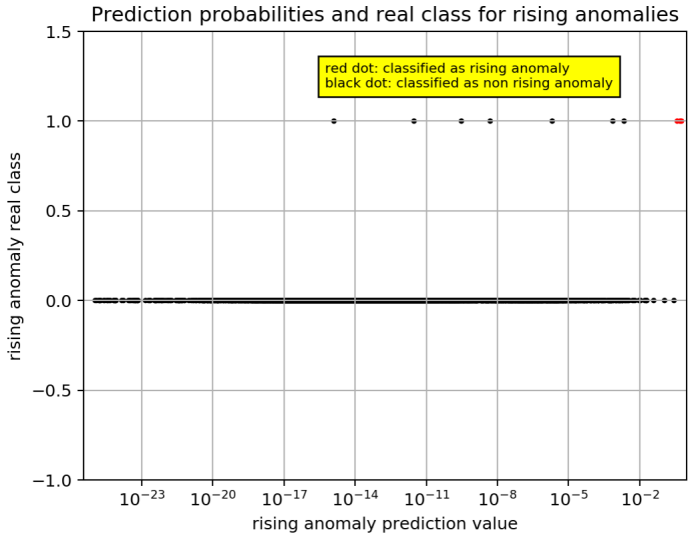
La terza modifica effettuata consiste nell'**aumentare il numero di filtri generati da ogni strato convoluzionale**, precisamente di un fattore pari al numero di attributi per ogni ossevazione -1. In questo modo la rete viene appesantita ulteriormente. In particolare, per ogni strato convoluzionale vengono aggiunte un numero di connessioni pari al campo di ricezione dello strato di convoluzioni causali dilatate, moltiplicate per il numero di nuovi filtri. In generale, **le prestazioni** (in termini di tempo di allenamento della rete neurale) rimangono comunque accettabili e si aggirano intorno agli **11min 28s**.

In Figura 36 si può dare uno sguardo all'**architettura della rete** risultante.

Per quanto riguarda le **performance predittive** della TCN, possiamo osservare un **ulteriore miglioramento rispetto alla versione precedente**. In particolare, il **valore medio registrato per le *f_score* durante le 5 esecuzioni indipendenti** risulta incrementato di 0.058, attestandosi a **0.25**. Tuttavia, per quanto riguarda le **differenze tra il valore minimo e massimo dei limiti inferiori di probabilità affinchè un'osservazione venga classificata come classe** `1`, osserviamo un notevole peggioramento rispetto alla modifica precedente. Infatti, il **minimo** ottenuto per questo valore risulta essere **0.019**, mentre il **massimo** si attesta a **0.937**, con una **media sulle 5 esecuzioni** pari a **0.417**.

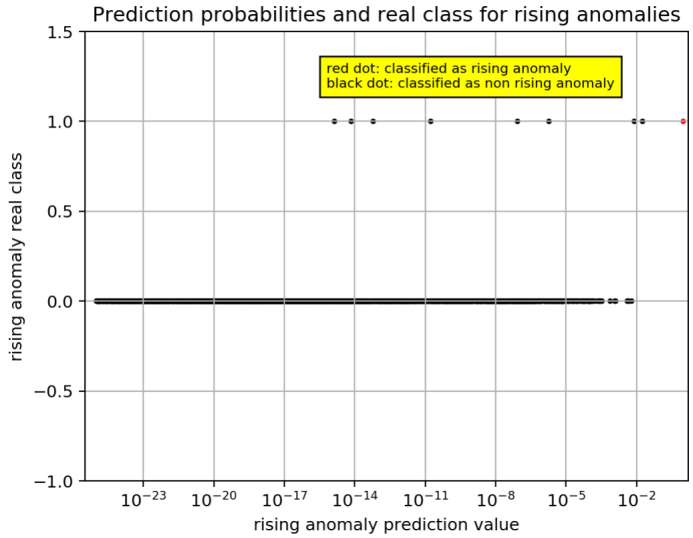


## Execution 2



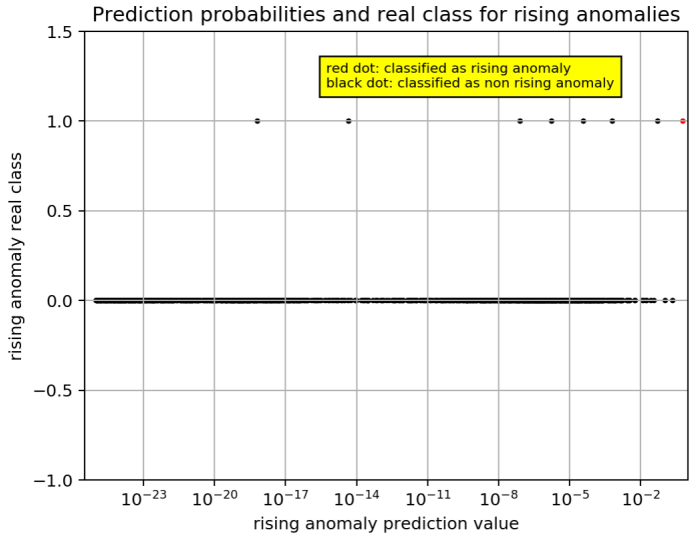

```
True Negative (TN): 3915
False Positive (FP): :  0
False Negative (FN) 7
True Positive (TP): 3
f1_score :  0.4615
LB Threshold 3.79e-01
UB Threshold:  1
```

## Execution 3



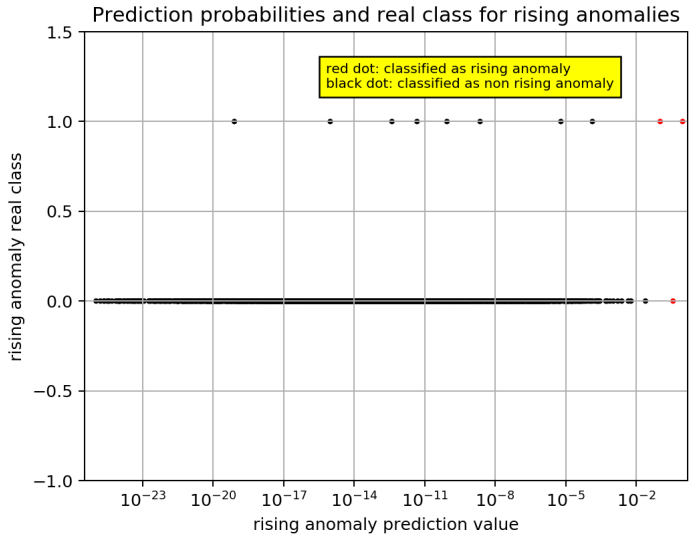

```
True Negative (TN): 3915
False Positive (FP): :  0
False Negative (FN) 9
True Positive (TP): 1
f1_score :  0.1818
LB Threshold 9.73e-01
UB Threshold:  1
```

## Execution 4




True Negative (TN): 3915
False Positive (FP): : 0
False Negative (FN) 9
True Positive (TP): 1
f1_score : 0.1818
LB Threshold 6.13e-03
UB Threshold: 1

## Execution 5




True Negative (TN): 3914
False Positive (FP): 1
False Negative (FN): 8
True Positive (TP): 2
f1_score : 0.3077
LB Threshold: 1e-1
UB Threshold: 1

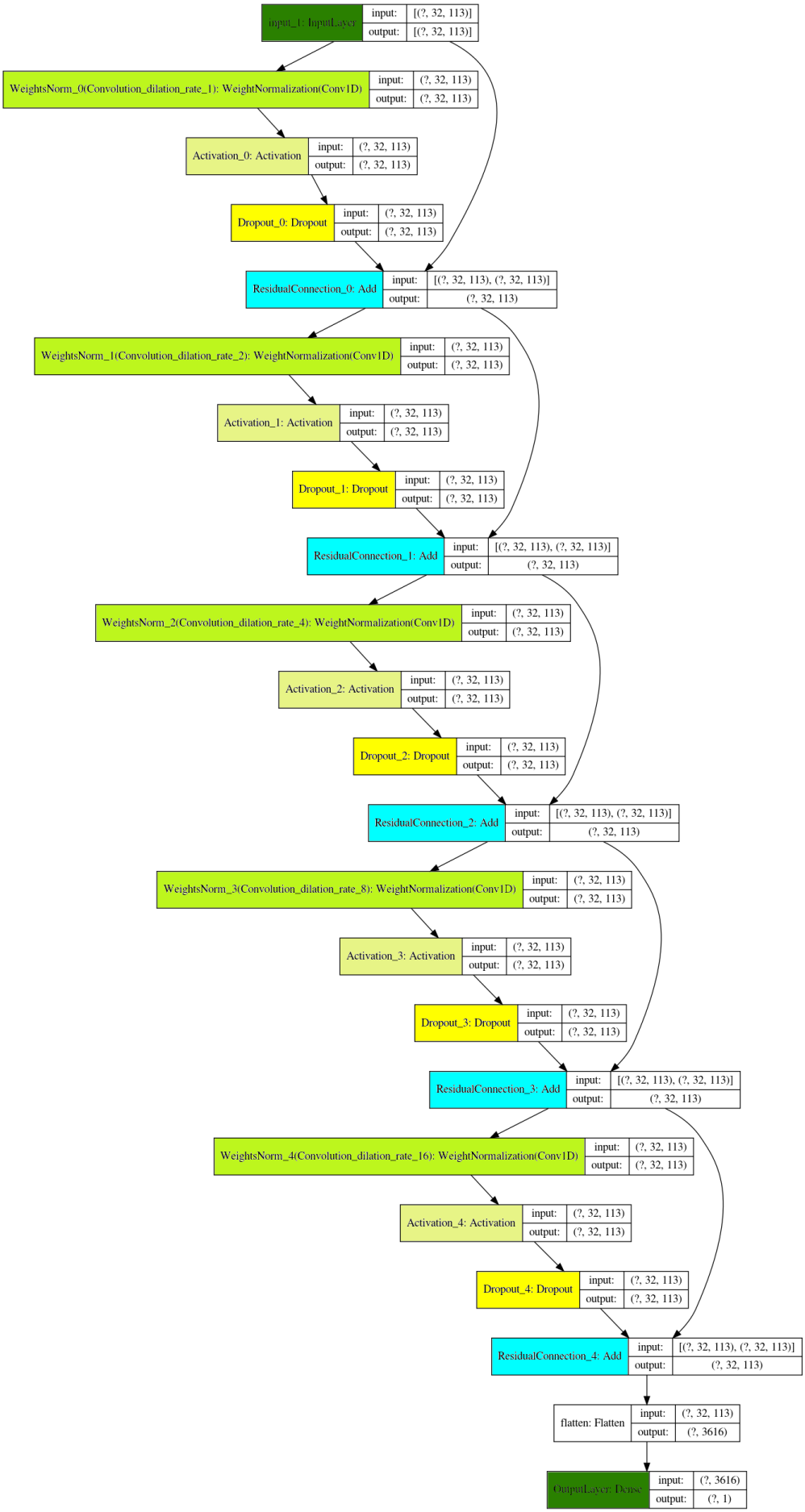


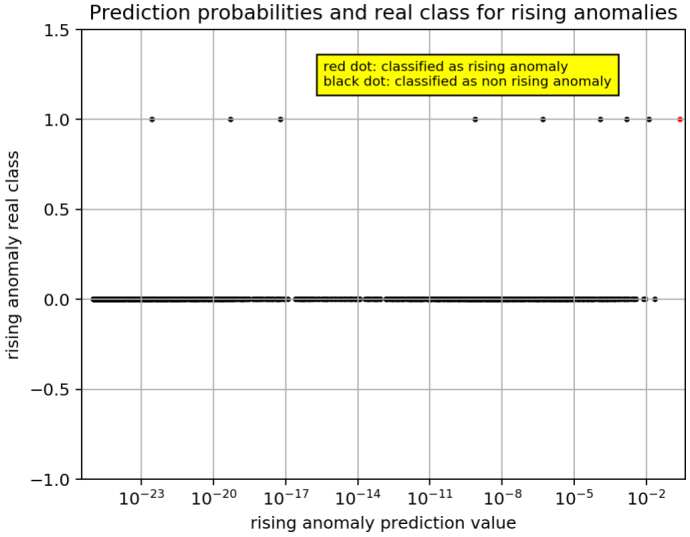

Figura 36: Architettura TCN - Strati Convoluzionali: Aumentro Filtri

### 4.8.4 Quarta Modifica: Inserimento Dropout Finale

La quarta ed ultima modifica della TCN consiste nell'**aggiunta del dropout tra l'ultima connessione residua e lo strato di uscita della rete**. In questo modo, si è tentato di **ridurre ulteriormente l'overfitting del modello** anche se, come vedremo, quest'aggiunta non ha portato alcun miglioramento per quanto riguarda le prestazioni della rete. Avendo introdotto solamente una modifica marginale, il tempo di allenamento della rete sul dataset in esame risulta essere molto simile a quello ottenuto applicando le precedenti modifiche.

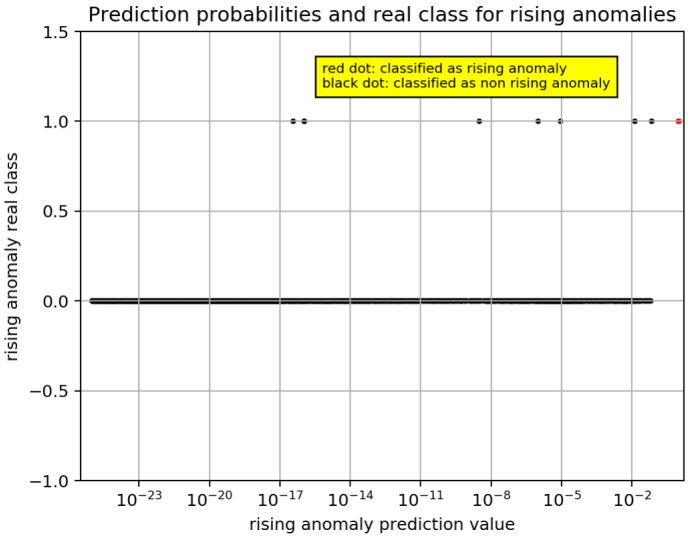
Come preannunciato, il **valore medio registrato per le *f_score* durante le 5 esecuzioni indipendenti** risulta decrementato di 0.04, attestandosi a **0.21**. Inoltre, per le **differenze tra il valore minimo e massimo dei limiti inferiori di probabilità della classe** `1`, osserviamo un'**ulteriore peggioramento rispetto alla modifica precedente** che, ricordo, aveva dimostrato ottenere scarsi risultati secondo questo aspetto. Infatti, il **minimo** ottenuto per questo valore risulta essere **0.0016**, mentre il **massimo** si attesta a **0.937**, con una **media sulle 5 esecuzioni** pari a **0.44**.

## Execution 1



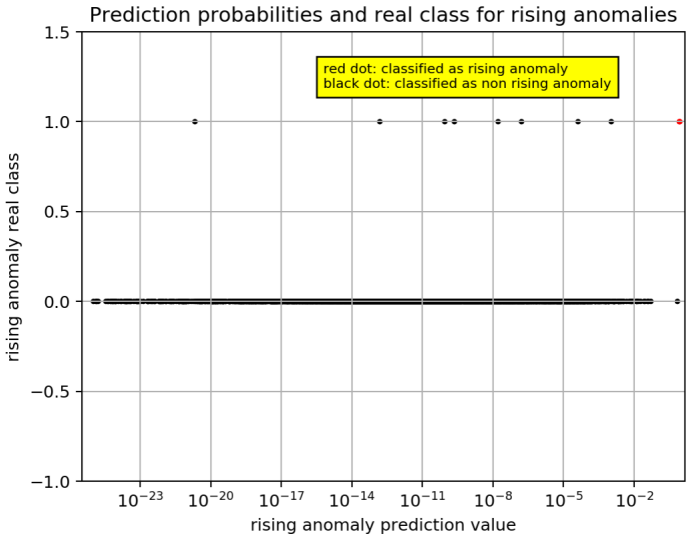

```
True Negative (TN): 3915
False Positive (FP): 0
False Negative (FN): 9
True Positive (TP): 1
f1_score :  0.1818
LB Threshold:  2.53e-2
UB Threshold:  1
```

## Execution 2



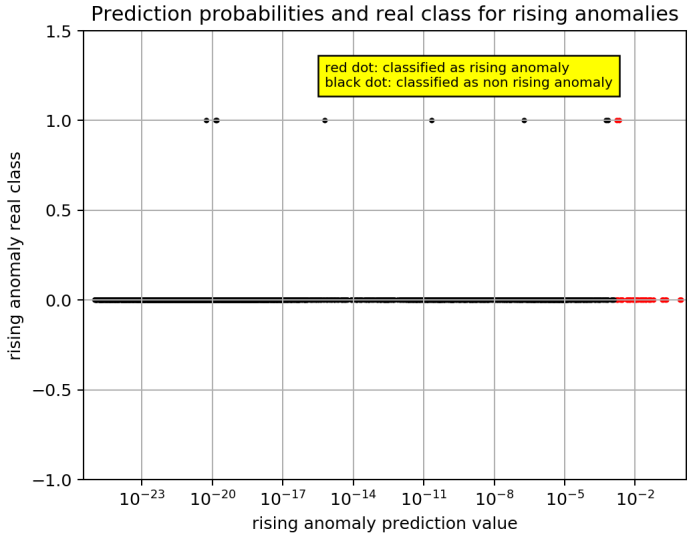

```
True Negative (TN): 3915
False Positive (FP): :  0
False Negative (FN) 8
True Positive (TP): 2
f1_score :  0.333
LB Threshold 9.37e-1
UB Threshold:  1
```

## Execution 3



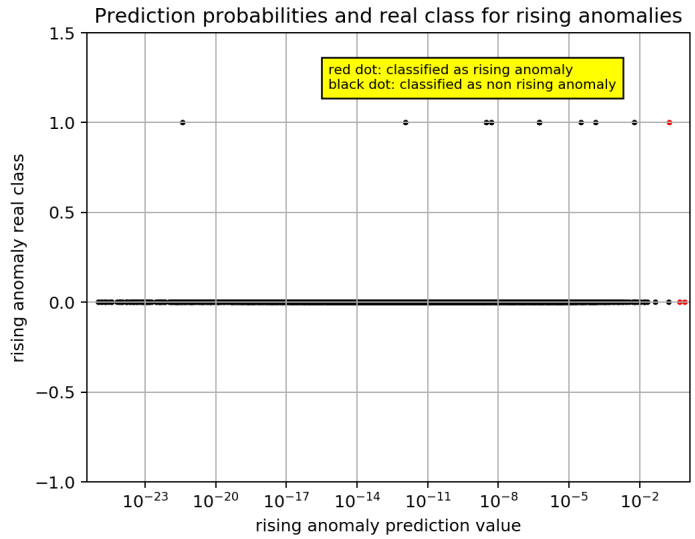

```
True Negative (TN): 3915
False Positive (FP): :  0
False Negative (FN) 8
True Positive (TP): 2
f1_score :  0.333
LB Threshold 8.29e-1
UB Threshold:  1
```

## Execution 4




True Negative (TN): 3871
False Positive (FP): : 44
False Negative (FN) 8
True Positive (TP): 2
f1_score : 0.0714
LB Threshold 1.6e-3
UB Threshold: 1

## Execution 5




True Negative (TN): 3913
False Positive (FP): 2
False Negative (FN): 9
True Positive (TP): 1
f1_score : 0.1538
LB Threshold: 1.81e-1
UB Threshold: 1

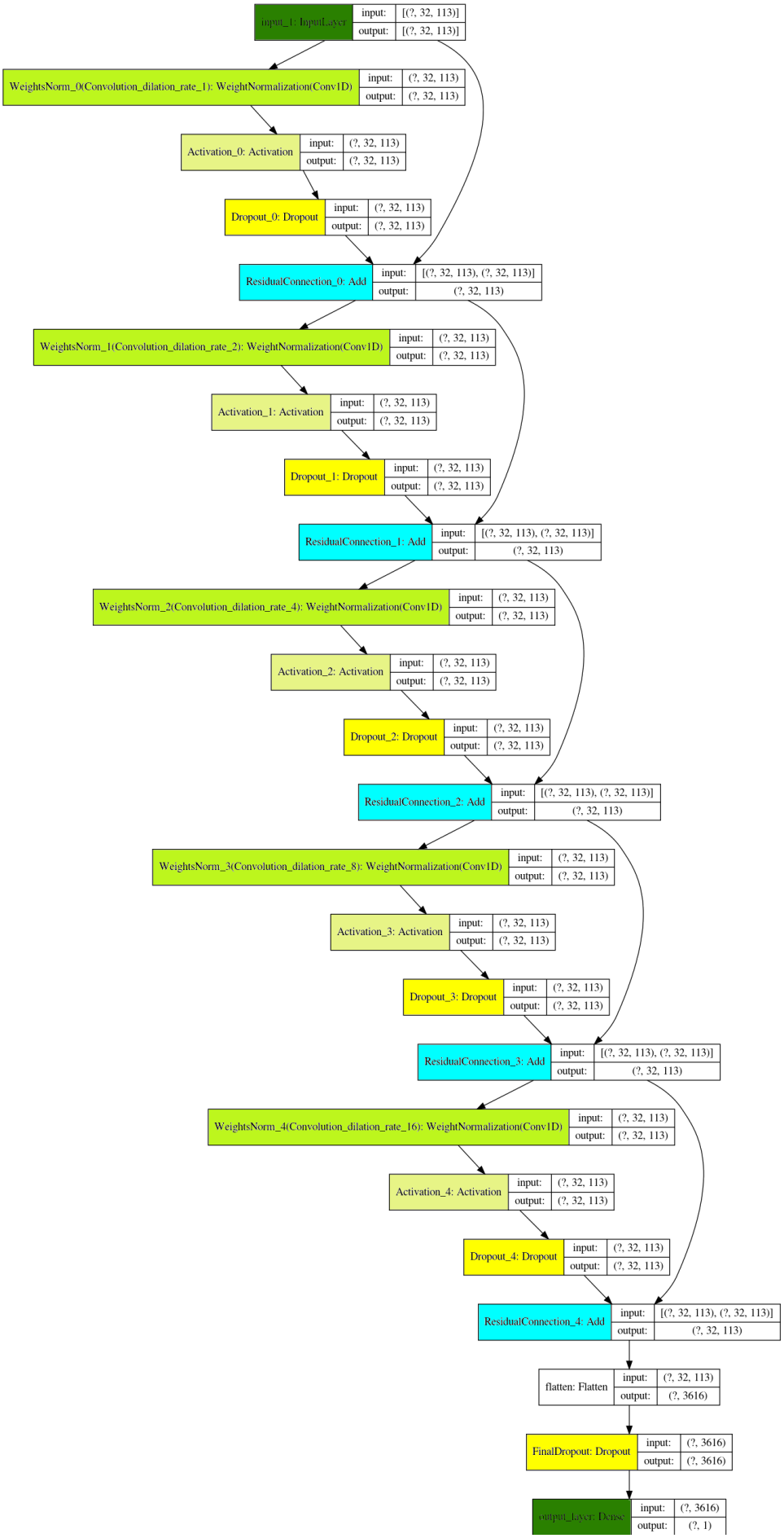



Figura 37: Architettura TCN - Inserimento Dropout finale

## 4.9 Un nuovo approccio

Dalla necessità di ridurre e, nel migliore dei casi, risolvere i problemi relativi al dataset (sbilanciamento e ritardo nell'assegnamento delle anomalie), ho pensato di **creare un nuovo modello** che, seppur architetturalmente simile alla TCN sintetizzata, può potenzialmente **aprire una diramazione rispetto alla via seguita fino ad ora**. In particolare, ciò che si vuole ottenere dal nuovo modello non è la previsione diretta riguardante l'insorgere di una nuova anomalia ma, piuttosto, vogliamo far apprendere al modello un modo per **prevedere la distanza (in termini temporali) dall'insorgere di una nuova anomalia**.

A titolo esemplificativo, prendiamo in esame l'indicazione sullo stato della rete, rispetto alle 10 osservazioni precedenti ed inclusa un'osservazione classificata come `is_rising_anomaly`. Allo stato attuale, il risultato che ci aspettiamo di ottenere dalla rete neurale, per queste 10 osservazioni, in cui l'ultima corrisponde all'anomalia, è il seguente: `[0,0,0,0,0,0,0,0,1,0]` (ricordo che questa indicazione viene traslata di una posizione all'indietro). Invece, seguendo il nuovo approccio, il risultato dovrà essere il seguente: `[9,8,7,6,5,4,3,2,1,0]`.

L'ottenimento di un modello in grado di effettuare questo tipo di previsioni offre **numerosi vantaggi**. **In primo luogo**, la capacità predittiva della rete viene generalizzata a qualsiasi distanza temporale rispetto al momento corrente. **In secondo luogo**, sulle previsioni della rete si possono eseguire delle ulteriori analisi statistiche a valle delle singole previsioni effettuate. Per esempio, si possono identificare degli intervalli omogenei in cui le previsioni conducono a risultati simili e, accidentalmente, si potrebbe identificare il ritardo nell'assegnamento da parte degli amministratori di sistema dello stato anomalo del nodo.

Rispetto all'architettura della TCN precedentemente sintetizzata non ho introdotto alcuna modifica, se non per la **funzione di attivazione dello strato di uscita** che, nel caso precedente era la sigmoide, mentre ora è la **ReLU**. In più, è necessario effettuare una modifica anche alla **Loss Function** che, se nel caso precedente risultava calzante la binary cross-entropy, ora invece risulta inadeguata, ed andrebbe opportunamente sostituita con una tra MAE (*Mean Absolute Error*), MSE (*Mean Squared Error*), RMSE (*Root Mean Squared Error*), ecc. Personalmente, ho scelto di utilizzare **MAE**.

Per quanto riguarda i **risultati ottenuti**, possiamo con tranquillità affermare che sembra non esserci quasi alcuna differenza tra una rete che effettua previsioni casuali ed il nostro modello. In pratica, questo significa che bisognerà svilupparlo ulteriormente per fare in modo che porti a risultati tangibili.

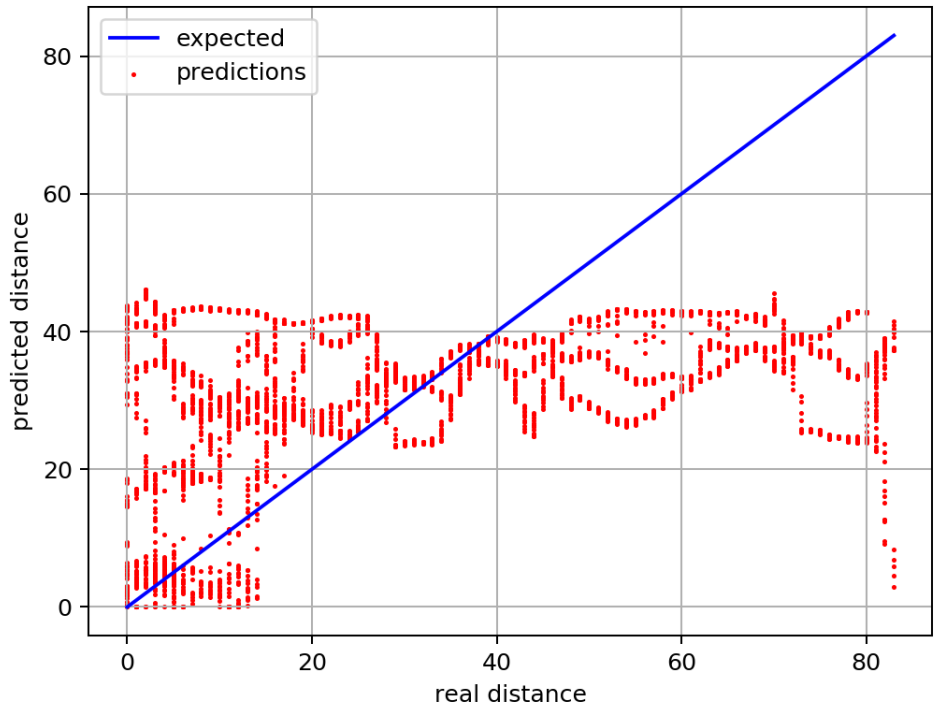



Figura 38: Previsione di distanze dalla prossima anonmalia; `x`: Distanza prevista - `y`: Distanza reale

**NOTA**: Le distanze sono discretizzate in ore.

Dal grafico risultante si può notare ciò che avevo precedentemente anticipato, ad occhio umano non è inferibile alcuna omogeneità nel comportamento delle previsioni effettuate dalla rete neurale, se non una lieve tendenza, per le osservazioni più prossime all'insorgere di una nuova anomalia, ad effettuare previsioni meno discontanti dal risultato sperato.

# 5 Conclusioni e Sviluppi futuri

**Fin dal principio si è assunto che il modello ideato avrebbe difficilmente ottenuto ottimi risultati**. Questa considerazione è dovuta al fatto che il **dataset di partenza era soggetto ad alcune problematiche** difficili (se non impossibili) da risolvere allo stato attuale. In particolare, è risultato subito palese la presenza di uno sbilanciamento tra le osservazioni associate ad uno stato ordinario del nodo e quelle associate ad uno stato anomalo. In secondo luogo, è emerso che l'identificazione di un'anomalia da parte di un'amministratore di sistema avviene in ritardo rispetto all'insorgere dell'anomalia stessa.

**La dimostrazione empirica sulla correttezza delle assunzioni riguardanti lo sbilanciamento del dataset** sono state confermate nel momento in cui si è deciso di applicare un filtro sulla distanza temporale di un'osservazione dall'insorgere di una futura anomalia, che ha condotto all'ottenimento dei primi risultati utili. In particolare, si è riuscito ad ottenere una media su 5 esecuzioni indipendenti *f_score* di 0.16, a fronte dei risultati precedenti in cui questa assumeva valori di uno o due ordini di grandezza inferiore.

**Si sono ottenuti ulteriori miglioramenti aggiungendo le connessioni residue tra i vari strati della TCN**, in primo luogo, e successivamente **aumentando il numero di filtri convoluzionali tra i vari strati di convoluzioni causali dilatate della rete**, facendo raggiungere all'*f_score* una media di 0.25 . Tuttavia, non è stata trovata una configurazione ottimale della rete neurale che, allenata in modo indipendente per 5 differenti esecuzioni, sia riuscita ad isolare le osservazioni che conducono ad una futura anomalia applicando la stessa soglia di probabilità.

Complessivamente, si può dire che **il tempo di esecuzione della fase di allenamento della TCN** sia buona, richiedendo circa 11 minuti per essere ultimata, nel momento in cui il dataset è composto da $2 \cdot 10^4$ osservazioni, comprendenti 113 attributi l'una.

Nonostante non siano stati raggiunti risultati ottimali nella costruzione di un modello capace di prevedere la maggior parte delle future anomalie del sistema, le performance ottenute dalla migliore versione della TCN apre la strada a **nuove sperimentazioni**. In particolare, si potrebbe osare con l'**aumento del numero di epoche per la fase di allenamento**, così come con **diverse dimensioni per i lotti di osservazioni** (attualmente impostate a 64 per la fase di allenamento e a 32 per le fasi di validazione e test) **che determinano il calcolo della Loss Function** e, di conseguenza, degli aggiornamenti dei pesi delle connessioni della rete neurale.

Inoltre, si può adottare un approccio più radicale agendo direttamente sull'architettura del modello. In particolare, non sarebbe una brutta idea tentare di applicare una **riduzione della dimensionalità dei dati in ingresso alla rete** (PCA, Autencoder, ecc), in questo modo verrebbero eliminate a priori tutte le ridondanze nei dati in ingresso alla stessa, facendo in modo che l'apprendimento si concentri solamente sulle informazioni necessarie.
Parimenti, si può adottare un approccio più drastico dirigendosi verso un nuovo paradigma architetturale. Infatti, in letteratura sono presenti diversi tipi di reti neurali studiate per l'analisi di dati strutturati in sequenze o connesse tra loro da un legame temporale. Vale la pena quindi **costruire una LSTM** (*Long Shot-Term Memory*) **od una GRU** (*Gated Recurrent Unit*) **e confrontare i risultati ottenuti con la TCN oggetto del progetto di tesi**.

Altrimenti, si potrebbe agire a monte della rete limitando i problemi relativi allo sbilanciamento del dataset. Una tecnica spesso utilizzata a tale fine consiste nel **riprodurre molteplici volte le osservazioni appartenenti alla classe sottorappresentata**, nel nostro caso quella indicante l'insorgere di una futura anomalia, ricordandosi però di applicare lo stesso trattamento anche alle osservazioni rientranti nel campo di ricezione della parte convoluzionale della TCN.

Se invece si vuole osare ed adottare un cambiamento di approccio più radicale, un tentativo che vale la pena effettuare consiste nell'**ideare un modello che apprenda in modo preciso e puntuale il comportamento ordinario dello stato di un nodo, in modo da prevedere un'anomalia nel momento in cui ci si accorge che il comportamento del nodo si allontana dall'ordinario**. Quello appena descritto è il metodo più comunemente adottato per quanto riguarda la risoluzione del problema della previsione delle anomalie.
Infine, si potrebbe osservare il problema da una differente prospettiva, consistente nella **previsione della distanza dall'insorgere di una futura anomalia**, piuttosto che l'anomalia stessa. Realizzando un prototipo di questo tipo nella fase finale del progetto, si è aperta un'ulteriore via sperimentale che potrebbe portare alla costruzione di un modello capace di generalizzare l'anticipo con cui vengono previste nuove anomalie.

# Riferimenti bibliografici


[1] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[2] Francesco Beneventi, Andrea Bartolini, Carlo Cavazzoni, and Luca Benini. Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1038–1043. IEEE, 2017.

[3] Andrea Borghesi, Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini. A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems. *Engineering Applications of Artificial Intelligence*, 85:634–644, 2019.

[4] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[5] Andriy Burkov. *The hundred-page machine learning book*. Andriy Burkov Quebec City, Can., 2019.

[6] Enzo Busseti, Ian Osband, and Scott Wong. Deep learning for time series modeling. *Technical report, Stanford University*, pages 1–5, 2012.

[7] John Cristian Borges Gamboa. Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887*, 2017.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data Augmentation for Time Series Classification using Convolutional Neural Networks. In *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, Riva Del Garda, Italy, September 2016.

[10] Kevin Leto. *Anomaly Detection in HPC Systems*. Tesi di laurea, Università degli Studi di Bologna, 2019.

[11] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[12] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89. Presses universitaires de Louvain, 2015.

[13] Sabyasachi Sahoo. Residual blocks — building blocks of resnet, 2018. `https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec`.

[14] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016.

[15] Sagar Sharma. Epoch vs batch size vs iterations, 2017. `https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9`.

[16] Chi-Feng Wang. The vanishing gradient problem, 2019. `https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484`.

[17] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[18] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[19] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, 2017.

[20] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*, pages 298–310. Springer, 2014.