UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA BIOMEDICA

OTTIMIZZAZIONE DI UN'ARCHITETTURA NEURALE PER CLASSIFICAZIONI MULTI-LABEL

Relatore:

Prof. Loris Nanni

Laureando:

Mattia Piazza

ANNO ACCADEMICO 2021–2022

Data di laurea 21/07/2022

Indice

Introduzione

L'addestramento Multi-label è una delle classi di apprendimento che ammette, per ogni istanza o *pattern*, l'assegnazione di più etichette contemporaneamente, utilizzata con successo negli ambiti più svariati [1] come per *tag recommendation* [2], bioinformatica [3-6], raccolta di informazioni [7, 8], riconoscimento vocale [9, 10] e classificazione di recensioni e commenti negativi da parte degli utenti [11, 12] per citarne alcune.

In questa tesi è proposta la topologia chiamata LSTM_GRU per classificazioni *multi-label* che essenzialmente combina in sequenza un livello *Long Short-Term Memory* (LSTM), un livello di *dropout* fissato con probabilità 0.4%, un livello *Gated Recurrent Unit* (GRU) [17], un ulteriore livello di *dropout* sempre fissato con probabilità 0.4%, un livello *fully connected* e prima dell'uscita una funzione sigmoidea. Entrambi, LSTM e GRU, modellano stati temporali nascosti con meccanismi di *gating* e soffrono di attivazioni intermedie che sono funzione di caratteristiche a basso livello.

Le reti LSTM e GRU sono state allenate con differenti varianti dell'ottimizzazione Adam [18] le quali, non essendo stabili, garantiscono la diversità negli *ensemble* e migliorano le prestazioni del metodo Adam. Combinando gli ensamble proposti con IMCC [7].

Con l'aumento di complessità di un classificatore, è lecito aspettarsi un miglioramento della performance dello stesso. Tuttavia, il guadagno marginale in termini di performance potrebbe, in determinate situazioni, non giustificare l'incremento di parametri del sistema. Inoltre, in taluni casi, come ad esempio in presenza di dataset sparsi e con un numero ridotto di esempi, un sistema più complesso potrebbe avere più difficoltà a generalizzare rispetto ad un sistema più semplice.
In questa tesi è stato confrontato lo stato dell'arte per la classificazione multi-label con una serie di modelli di machine learning più semplici come i classificatori Tree Bagger.

Nel primo capitolo si introducono le reti neurali artificiali e si illustrano le principali tipologie di reti neurali con le loro caratteristiche.

Nel secondo capitolo si illustrano i livelli utilizzati nella creazione delle topologie che vengono descritte, invece, nel terzo capitolo.

Nel quarto capitolo sono inseriti, assieme ai vari dataset ed *ensemble* utilizzati, i risultati.

Finalmente, nel quinto capitolo, arriviamo alle conclusioni.

1. Reti Neuronali Artificiali

1.1 Intelligenza Artificiale

L'intelligenza artificiale, dall'inglese *artificial intelligence* (abbreviato AI), è una disciplina dell'informatica che permette di dotare elaboratori elettronici di determinate caratteristiche tipicamente umane, quali, ad esempio, percezioni visive, spazio-temporali e decisionali, attraverso lo studio di fondamenti teorici, metodologie e tecniche che consentono la programmazione e la progettazione di *hardwere* e *softwere*; sono capaci, dunque, di riprodurre parzialmente l'attività intellettuale umana.
In particolare, è considerato un comportamento intelligente quello di un programma in grado di apprendere dall'esperienza al fine di migliorare le sue prestazioni nella risoluzione di un dato task.

1.2 Reti Neurali Artificiali

Ispirate ai neuroni biologici e alla struttura di una rete neurale biologica, le reti neurali artificiali sono modelli computazionali composti da neuroni artificiali, definiti come modello matematico, connessi tra loro [34]. Come le sinapsi nel nostro cervello, ogni connessione tra neuroni trasmette un segnale che può essere amplificato, oppure attenuato, attraverso un peso che viene costantemente aggiornato durante il processo di apprendimento.
Un neurone di una rete neurale artificiale può essere rappresentato nel modo seguente:

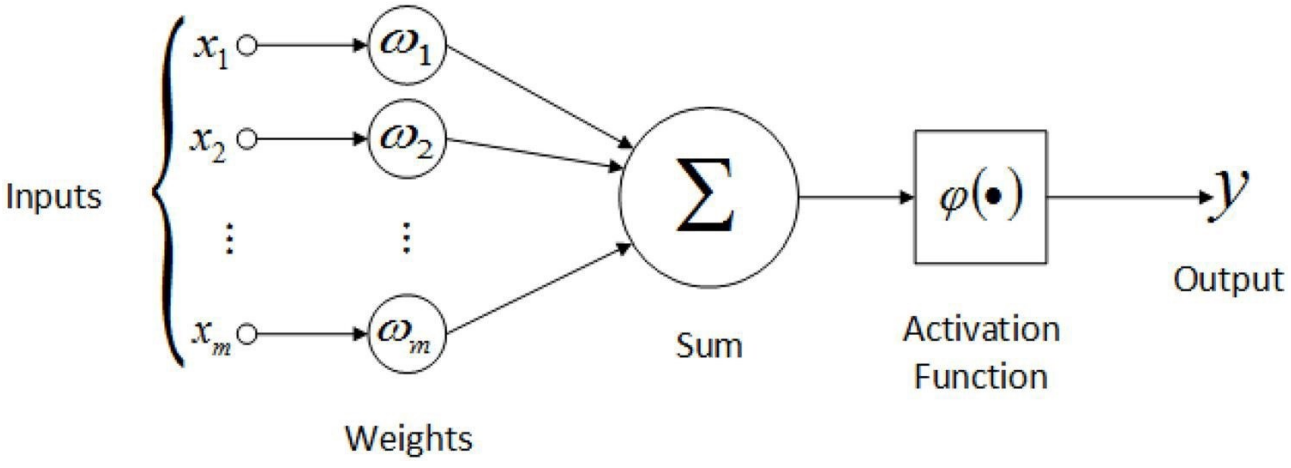

Figura 1.2.1 Neurone Artificiale

Ad ogni ingresso x_i è associato un peso w_i. La somma di tutti gli ingressi pesati, $\sum x_i w_i$, poi passata attraverso una funzione di attivazione non lineare, φ, che restituirà l'uscita del neurone y. Il termine *bias* è stato ommesso per semplicità. A questo punto l'uscita y servirà come ingresso per il neurone del livello successivo. Questi neuroni sono organizzati all'interno di reti con più livelli. Solitamente il primo livello che si incontra è quello di ingresso, che riceve i dati in entrata alla rete, fino ad arrivare al livello di uscita che restituisce il risultato finale. I responsabili maggiori dell'apprendimento della rete sono i livelli nascosti che separano i due livelli d'ingresso e uscita, il loro numero può variare da zero a più, creando una mappatura non lineare tra ingresso e uscita.
Il numero di livelli e il numero di neuroni, come altri parametri quale la velocità di apprendimento (*learning rate*), devono venire settati manualmente in quanto non possono essere appresi dalla rete neurale, questi parametri vengono chiamati iperparametri e sono gli elementi caratteristici della rete.

1.3 Deep Learning

L'apprendimento profondo (in inglese *Deep Learning*) è un approccio moderno nell'implementazione di intelligenze artificiali e consiste in un'organizzazione gerarchica di molti livelli; ogni livello calcola i valori per il successivo elaborando via via un'informazione sempre più completa.
Il recente successo del Deep Learning è dovuto a fattori quali la disponibilità di dataset di grandi dimensioni e lo sviluppo di sistemi di calcolo parallelo basati su GPU (GPU *computing*) che hanno contribuito all'utilizzo sempre più frequente di tal metodologia; l'elevata diffusione ha certamente permesso lo sviluppo di tecniche di Deep-Learning le quali hanno raggiunto, e in alcuni casi superato, lo stato dell'arte in moltissimi settori rendendole le tecniche più utilizzate nell'ambito delle AI.

1.4 Recurrent Neural Network

Le reti neurali ricorrenti (in inglese *Recurrent Neural Network* o RNN) ricoprono una parte del lavoro svolto; questa tipologia di rete prevede delle connessioni di *feedback* (in genere verso neuroni dello stesso livello, ma anche verso livelli precedenti). Essendo richiesta la considerazione del comportamento della rete in più stati temporali, il flusso di informazioni nella fase di addestramento è più complesso rispetto ad altre tipologie di rete. Le RNN sono particolarmente indicate quando si lavora con pattern che rappresentano sequenze [35], in quanto al tempo t rendono disponibile l'informazione

processata al tempo *t*-1, *t*-2 e così via, questo comportamento è chiamato *effetto memoria* (di breve termine).
La cella è la parte centrale della rete ricorrente che possiede uno stato (o memoria) $h_{(t)}$ per ogni istanza temporale. È costituita da un numero prefissato di neuroni (può essere vista a tutti gli effetti come un livello della rete). Lo stato $h_{(t)}$ dipende dall'input $x_{(t)}$ e dallo e dallo stato precedente $h_{(t)}$:

$$h_{(t)} = f(h_{(t-1)}, x_{(t)}) \tag{1.4.1}$$

A seconda della struttura e di come viene trattato lo stato di memoria $h_{(t)}$ esistono diverse tipologie di celle: un esempio sono le *Long-Short Term Memory* (LSTM) e le *Gated Recurrent Unit* (GRU). Quest'ultime funzionano come le RNN, ma hanno la capacità di apprendimento a lungo termine grazie a meccanismi chiamati *gates*, grazie ai quali sono in grado di decidere quali informazioni aggiungere o rimuovere [37]

1.5 Temporal Convolutional Neural Network

Le *Temporal Convolutional Neural Networks* (TCNs) [26] sono una classe di reti neurali che estraggono informazioni da sequenze di input o da serie temporali grazie ad una gerarchia di convoluzioni. Generalmente, reti neurali ricorrenti (RNN) risolvono problemi legati alle serie temporali introducendo un modo per conservare e memorizzare l'informazione. Tuttavia, le reti neurali convoluzionali (CNN) possono eguagliare o addirittura superare le prestazioni delle RNN, come dimostrato in Bai et al. [38].

Le TCN utilizzano dei livelli convolutivi 1D impilati l'uno sull'altro per creare una rete profonda, utili per poter eseguir la convoluzione sulla dimensione temporale.

Ognuno di questi livelli possiede un fattore di dilatazione (*dilatation factor*) il quale, man mano che la rete diventa profonda, aumenta esponenzialmente consentendo ai primi livelli di cercare informazioni temporalmente vicine tra loro e ai livelli più profondi di individuare dipendenze a lungo termine grazie alla *feature* estratta dai livelli precedenti. Questo permette alle TCN di avere un *receptive field* molto più ampio, superando uno dei limiti presenti nelle RNN.

La dimensione del *receptive field* è calcolata come segue:

$$R = (f-1)(2^K - 1) + 1 \tag{1.5.1}$$

dove *f* è la dimensione del filtro usato nelle convoluzioni e *K* è il numero di livelli convoluzionali.

Il fattore di dilatazione delle convoluzioni è $2^{(k-1)}$.

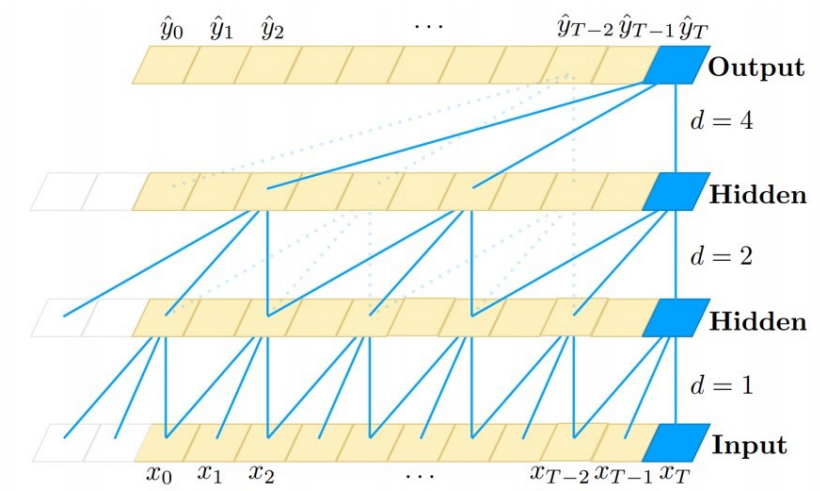

Figura 1.5.1 Esempio di struttura interna di una TCN, d è il *dialtation factor*

1.6 Tree Bagger

Un albero di classificazione (in inglese *classification tree*) è un modello predittivo composto da nodi foglia, rappresentano le classificazioni, e ramificazioni che collegano i vari nodi foglia, si possono tradurre come l'insieme delle proprietà che portano a quelle classificazioni.

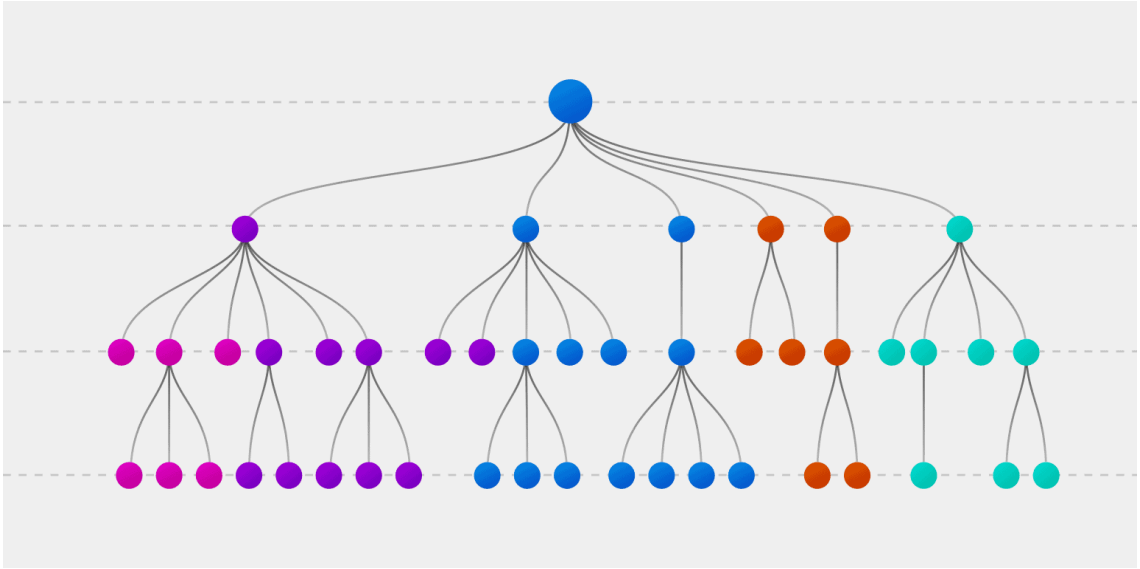

Figura 1.6.1 Struttura di un albero di classificazione

Ogni nodo interno risulta essere una macro-classe costituita dall'unione delle classi associate ai suoi nodi figli. La strategia nell'allenamento degli alberi di classificazione consiste nell'accettare il numero di *features*, selezionate casualmente ad ogni decisione, come dati in ingresso opzionali [40].

Alberi di classificazione individuali tendono ad 'overfittare'; Il classificatore Tree Bagger costituisce un'*ensemble* di alberi di classificazione. *Bagging* sta per ***b**ootstrap-**agg**regat**ing**,* tecnica con la quale, generalmente, modelli dello stesso tipo vengono addestrati su dataset differenti ottenuti tramite campionamento casuale con rimpiazzo (*bootstrap*). Ogni albero, all'interno dell'*ensamble,* è allenato con la sua replica *bootstrap* dei dati in ingresso in modo indipendente dagli altri alberi. Questo permette di migliorare la generalizzazione e ridurre gli effetti dell'*overfitting*.

Per ogni classe $c \in C$ e ogni albero $t = 1, \ldots, T$ calcola la probabilità $P_t(c|x)$, ovvero la probabilità a posteriori della classe c data l'osservazione x usando l'albero t. C è il l'insieme di tutte le classi distinte dei dati di addestramento. La media pesata a posteriori della classe sugli alberi selezionati è pari a:

$$\widehat{P}_{bag}(c|x) = \frac{1}{\sum_{t=1}^{T} \alpha_t I(t \in S)} \sum_{t=1}^{T} \alpha_t \widehat{P}_t(c|x) I(t \in S) \qquad (1.6.1)$$

dove S è l'insieme degli indici degli alberi selezionati che compongono la selezione e $I(t \in S)$ è pari a 1 se t appartiene al set S, 0 altrimenti.
Infine, la classe predetta è quella che ottiene la media pesata più grande:

$$\widehat{y}_{bag} = arg_{c \in C} max\left\{\widehat{P}_{bag}(c|x)\right\} \quad (1.6.2)$$

1.7 Multi-label

Utilizzato in svariati ambiti applicativi che spaziano dal *tag recommendation* alla bioinformatica, dall'*information retrival* al *rule mining*, *web mining* e molti altri, l'apprendimento Multi-label viene sicuramente utilizzato in svariati ambiti applicativi proprio per la possibilità di far fronte ad oggetti del mondo reale con molteplici significati semantici.
L'apprendimento Multi-label, infatti, è una delle classi di apprendimento che ammette per ogni istanza o pattern l'assegnazione di più etichette contemporaneamente.

Le modifiche della topologia di reti neurali proposte in questo elaborato per la classificazione Multi-label sono principalmente due, nei quali la parte centrale è data dalla combinazione di Gated Recurrent Units (GRU) e Temporal Convolutional Neural Networks (TCN).
I metodi di ottimizzazione differenti, utilizzati per sviluppare ogni ensemble, permettono un aumento delle performance e un rimescolamento delle *features* per ogni rete rispetto all'uso del metodo Adam originale.
Inoltre, combinando tra loro i set proposti di GRU e TCN con lo stato dell'arte per la classificazione Multi-label, Incorporating Multiple Clusterin Centers (IMCC), si è ottenuto un ulteriore miglioramento delle prestazioni.

1.8 Indicatori di Performance

Le classificazioni Multi-label sono valutate utilizzando diversi indicatori di performance in modo da riuscire a valutare i risultati ottenuti confrontandoli con quelli di studi precedenti, come, ad esempio, con quelli della *Incorporating Multiple Cluster Center* (IMCC), stato dell'arte per la classificazione multi-label. Sia X un insieme di dati che include m campioni $x_i \in \Re^d$ ciascuno avente un'etichetta $y_i \in \{0, 1\}^l$, dove l è il numero di etichette. Siano H, F l'insieme di etichette predette, dove $h_i \in \{0, 1\}^l$ è il vettore

dell'etichetta predetta rispetto al campione x_i e $f_i \in \Re^l$ è il valore di confidenza di ciascuna previsione. I seguenti indicatori di performance [7] possono essere definiti per H, F:

- Hamming Loss, è il rapporto fra le etichette classificate erroneamente e il numero totale di etichette,

$$HLoss(H) = \frac{1}{ml}\sum_{i=1}^{m}\sum_{j=1}^{l} I(y_i(j) \neq h_i(j)) \qquad (1.8.1)$$

  dove *I*() è la funzione indicatrice. Hamming loss è una funzione *loss function*, dunque il suo valore ottimo è 0 e il suo limite superiore è 1. Deve, perciò, essere minimizzata: se il valore dell'hamming loss è 0 allora non sono presenti errori nel vettore delle etichette predette.

- One Error, è il rapporto tra le istanze le cui etichette, con il più alto livello di confidenza, sono erroneamente classificate; deve essere minimizzato:

$$OneError(F) = \frac{1}{m}\sum_{i=1}^{m} I(h_i(arg\max_j f_i) \neq y_i(arg\max_j f_i)) \qquad (1.8.2)$$

  dove *I*() è la funzione indicatrice.

- Ranking Loss, è il rapporto medio delle coppie di etichette ordinate in modo contrario per ciascuna istanza. Può essere ottenuto dal valore di confidenza, considerando il numero di confidenze tra coppie di etichette correttamente classificate (un'etichetta classificata correttamente ha un ranking migliore rispetto ad una classificata erroneamente). Ranking loss è una *loss function* e dunque deve essere minimizzata.

- Coverage, è il numero medio di step necessari per spostarsi in basso nella lista delle etichette classificate di una istanza in modo da coprire tutte le relative etichette. Dovrebbe essere minimizzato

- Average Precision, è il rapporto medio delle etichette classificate in modo migliore rispetto ad una particolare etichetta. Deve essere massimizzato.
  Per la classificazione del dataset ATC, un altro tipo di indicatori è solitamente usato [19]

- Aiming, è il rapporto tra le etichette correttamente predette e tutte le etichette previste:

$$Aiming(H) = \frac{1}{m}\sum_{i=1}^{m}\frac{||h_i \cap y_i||}{||h_i||} \tag{1.8.3}$$

- Recall, valuta quante etichette "positive" sono state effettivamente classificate, indica quanto il sistema è selettivo:

$$Recall(H) = \frac{1}{m}\sum_{i=1}^{m}\frac{||h_i \cap y_i||}{||y_i||} \tag{1.8.4}$$

- Accuracy, è la percentuale media di etichette correttamente predette rispetto alle etichette totali:

$$Accuracy(H) = \frac{1}{m}\sum_{i=1}^{m}\frac{||h_i \cap y_i||}{||h_i \cup y_i||} \tag{1.8.5}$$

- Absolute True, è la percentuale delle etichette correttamente predette rispetto alle previsioni totali:

$$AbsTrue(H) = \frac{1}{m}\sum_{i=1}^{m} I(h_i = y_i) \tag{1.8.6}$$

- Absolute False, è la percentuale delle previsioni errate rispetto alle previsioni totali:

$$AbsFalse(H) = \frac{1}{m}\sum_{i=1}^{m}\frac{||h_i \cup y_i|| - ||h_i \cap y_i||}{l} \tag{1.8.7}$$

2. Livelli e approccio di ottimizzazione

In questo capitolo analizzeremo i differenti livelli usati nelle varie topologie e l'approccio di ottimizzazione Adam.

2.1 Long Short-Term Memory (LSTM)

Il livello LSTM permette di apprendere dipendenze a lungo termine tra fasi temporali nelle serie storiche e sequenze di dati. Il livello ottimizza il gradiente su lunghe sequenze durante l'allenamento.
Possiamo vedere come, nella figura 2.1.1 che illustra il flusso di una serie storica X con C features, h_t e x_t definiscano, rispettivamente, l'uscita (chiamata anche unità nascoste) e lo stato della cella al tempo t.

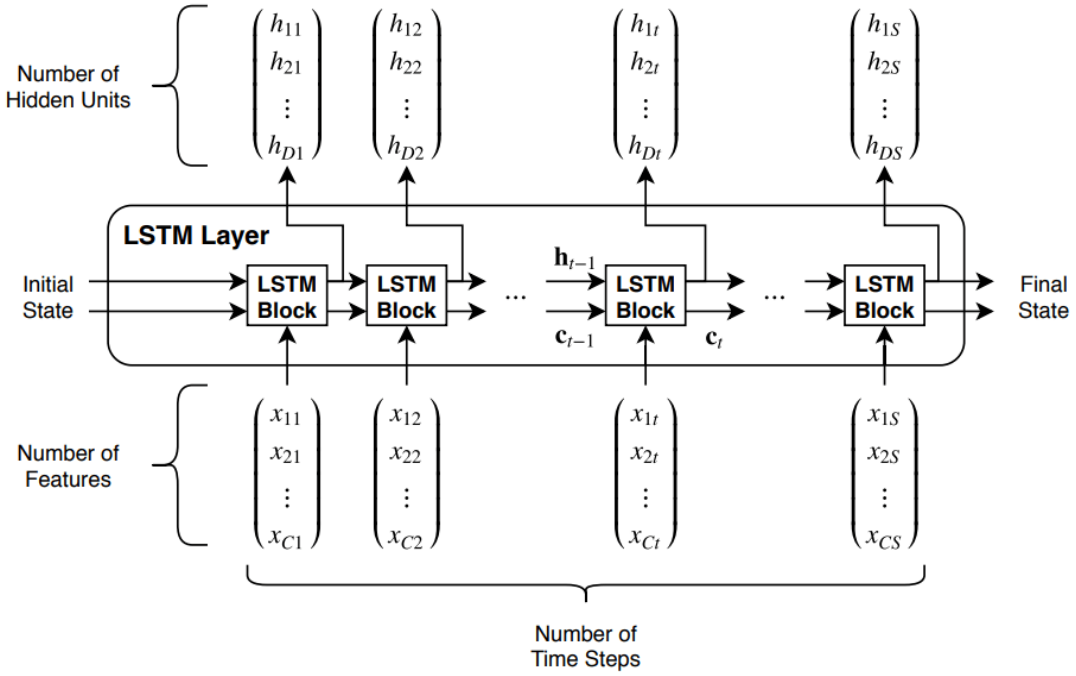

Figura 2.1.1 Architettura di un livello LSTM [24]

In generale, al tempo t, il blocco LSTM utilizza lo stato corrente della cella (c_{t-1}, h_{t-1}) e i valori successivi della sequenza x_{ct} per calcolare l'output e aggiornare lo stato della cella attraverso l'aggiunta o la rimozione di informazioni. Il livello utilizza dei cancelli per compiere questi aggiornamenti.

Le componenti principali di un livello LSTM sono l'*input gate* che determina il livello dello stato della cella da aggiornare, il *forget gate* che decide cosa eliminare, sempre dallo stato della cella, il *candidate gate* aggiunge informazioni allo stato della cella ed infine l'*output gate* determina il livello dello stato della cella aggiunto allo stato nascosto.

Sia x_t la sequenza di input e inizializziamo h_0. Successivamente, definiamo l'input gate i_t, il forget gate f_t, il candidate gate g_t e l'output gate o_t come:

$$i_t = \sigma_g(W_i x_t + R_i h_{t-1} + b_i) \quad (2.1.1)$$

$$f_t = \sigma_g(W_f x_t + R_f h_{t-1} + b_f) \quad (2.1.2)$$

$$g_t = \sigma_c(W_g x_t + R_g h_{t-1} + b_g) \quad (2.1.3)$$

$$o_t = \sigma_g(W_o x_t + R_o h_{t-1} + b_o) \quad (2.1.4)$$

Dove W_i, R_i, b_i, W_f, R_f, b_f, W_g, R_g, b_g, W_o, R_o, b_o sono matrici e vettori, σ_g denota la funzione di attivazione dei vari *gate*, σ_c è la funzione di attivazione per l'aggiornamento dello stato.
La funzione sigmoidea è la funzione di attivazione predefinita di un livello LSTM definita come:

$$\sigma(x) = (1 + e^{-x})^{-1} \quad (2.1.5)$$

Definiamo poi lo stato della cella come:

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (2.1.6)$$

dove Θ è il prodotto Hadamard (o *component-wise*)
Il vettore di uscita è

$$h_t = o_t \odot \sigma_c(c_t) \quad (2.1.7)$$

2.2 Gated Recurrent Unit

Introdotte nel 2014 da Kyunghyun Cho et al. [17] le GRU sono una tipologia di cella presente nelle reti neurali; sono usate principalmente per risolvere il problema del *vanishing gradient* (o *exploding gradient*) ovvero la scomparsa del gradiente. Questo problema è tipico di reti neurali profonde che utilizzano metodi di addestramento basati sulla retro-propagazione dell'errore (*backpropagation*).
Utilizzando funzioni di attivazione non lineari standard (come sigmoide, tangente iperbolica o la funzione logistica), caratterizzate dall'avere il gradiente nell'intervallo [0, 1]; questo significa che applicando la regola di derivazione a catena i parametri del modello decrescono esponenzialmente nei livelli lontani dall'output.

Utilizzando la cella GRU si cerca di mantenere i vantaggi della cella LSTM (*Long-Short Term Memory*), infatti può essere considerata come una semplificazione di quest'ultima, riducendo parametri e complessità. La cella GRU ha la possibilità, rispetto

alla LSTM, di decidere quale parte della vecchia informazione è rilevante per comprendere la nuova [25] grazie al *forget gate*.

Per problemi come lo *speech signal modeling*, *polyphonic music modeling* e *natural language processing*, la cella GRU ha prestazioni simili alla LSTM [9]. Generalmente hanno prestazioni migliori su dataset di piccole dimensioni [38] e sembrerebbe che lavorino meglio su problemi per la riduzione del rumore (*de-noise task*) [39].

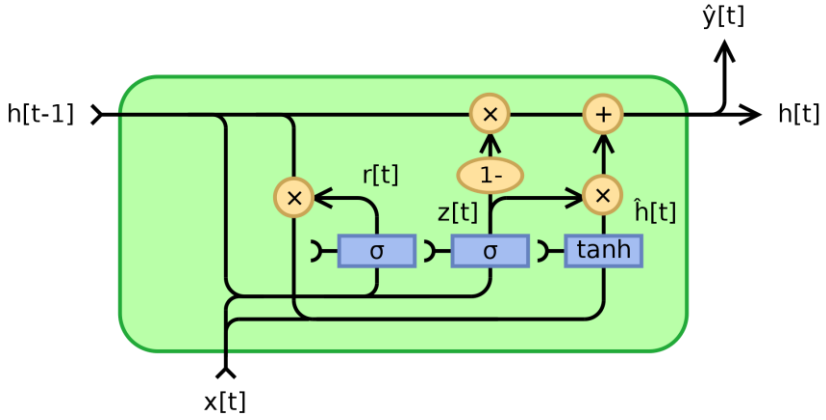

Figura 2.2.1 Struttura di una cella GRU

Le componenti fondamentali di una cella GRU sono il *reset gate* e l'*update gate*: il primo stabilisce quanta informazione passata dimenticare, mentre il secondo quale informazione può dimenticare e quale passare all'*output*.

Sia x_t la sequenza di *input* e h_t quella di *output*; inizializziamo lo stato di memoria della cella all'istante $t = 0$ come $h_0 = 0$. Definiamo l'*update gate vector* z_t e il *reset gate vector* r_t come

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \quad (2.2.1)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad (2.2.2)$$

dove W_z, U_z, b_z, W_r, U_r, b_r sono matrici e vettori e σ è la funzione sigmoidea.
Definiamo poi:

$$\hat{h}_t = \phi(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \quad (2.2.3)$$

come
l'*activation vector* candidato, dove ϕ è la funzione di attivazione tangente iperbolica e Θ è il prodotto Hadamard (o *component-wise product*).
Sapendo che r_t determina la quantità di informazione passata che è rilevante per l'*activation vector* candidato, si ha che

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \quad (2.2.4)$$

è l'*output vector*, che va a rappresentare lo stato di memoria della cella all'istante t.

2.3 Dropout

Un livello di dropout azzera casualmente gli elementi di input con una certa probabilità. I nodi individuali o vengono esclusi dalla rete (*dropped out*) con una probabilità (1 - *P*) oppure mantenuti all'interno con probabilità *P*, il risultato è una rete ridotta. Questo approccio di regolarizzazione obbliga la rete neurale ad apprendere *features* più robuste permettendo di ridurre la dipendenza di addestramento tra neuroni.
Utilizzando un livello di questo tipo il tempo necessario per completare un *epoch* è minore, ma il numero di iterazioni all'incirca raddoppia.

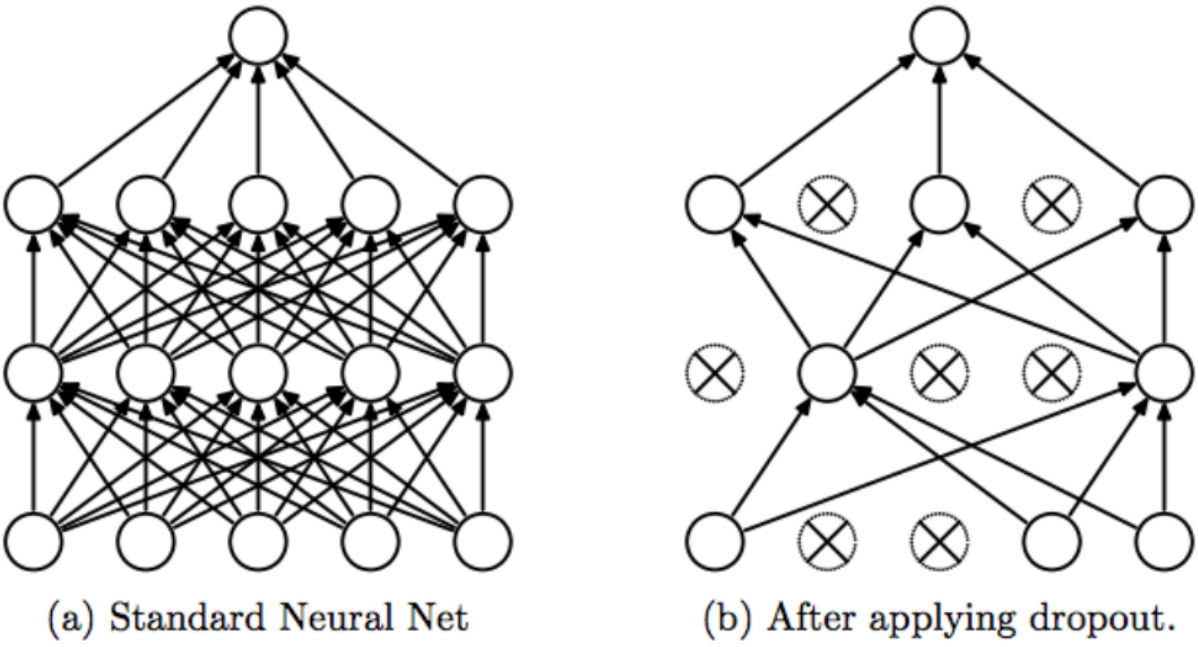

Figura 2.3.1 Effetto di un livello di dropout [27]

2.4 Batch-Normalization

Questa tecnica viene tipicamente utilizzata nelle reti molto profonde per standardizzare gli *input* di ogni *mini-batch*, riducendo drasticamente il numero di periodi di addestramento necessari.
Calcolando media e varianza di ciascuna variabile di input, la Batch-Normalization può essere implementata durante l'apprendimento.
Calcolo della media:

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_{ij} \tag{2.4.1}$$

e della varianza:

$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_{ij} - \mu_i)^2 \tag{2.4.2}$$

dove *m* è il numero di elementi di *input* e x_{ij} è l'input *j*-esimo relativo alla *i*-esima dimensione.

Infine, la normalizzazione diventa:

$$\hat{x_{ij}} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{2.4.3}$$

dove $\epsilon = 10^{-5}$.

Il livello di *Batch-Normalization* viene identificato con l'id BN.

2.5 Pooling

Inserito dopo il nucleo principale GRU/TCN, il livello di *Pooling* ha lo scopo di ridurre la dimensionalità dei dati processati mantenendo solo l'informazione più rilevante, diminuendo così la probabilità di *overfittig*.

2.6 Fully Connected Layer

Quando tutti gli *input* di un livello sono collegati a ogni unità di attivazione del livello successivo si dice livello Fully-connected (o completamente connesso). Tipicamente, nei più popolari modelli di apprendimento, gli ultimi livelli sono completamente

connessi i quali compilano i dati estratti dai livelli precedenti per fornire il risultato finale della rete. È tra i livelli più dispendiosi in termini di tempo. In termini di tempo è il secondo livello più dispendioso, dopo il livello convoluzionale.
È composto da *l* neuroni (*l* è il numero di etichette di output di un dato problema) completamente connessi con il livello precedente.

2.7 Sigmoid Layer

Come funzione di attivazione per il livello finale è stata utilizzata la funzione sigmoidea in modo da riportare il valore di attivazione nell'intervallo [0…1], poi interpretato come valore di probabilità di ogni etichetta. Ciascun neurone del livello *fully-connected* fornisce una valutazione (da 0 a 1) per una singola etichetta.

2.8 Ottimizzazione Adam

Adam (*Adaptive momentum estimation*) è un metodo di ottimizzazione introdotto in [18]; combinando concetti di momento e di gradiente adattivo, calcola il learning rate adattivo per ciascun parametro. La regola di aggiornamento si basa sul valore del gradiente al tempo *t* e sulle medie mobili esponenziali (*Exponential Moving Average*, EMA) m_t (primo momento) e u_t (secondo momento) come:

$$m_t = \rho_1 m_{t-1} + (1 - \rho_1) g_t \tag{2.8.1}$$

$$u_t = \rho_2 u_{t-1} + (1 - \rho_2) g_t^2 \tag{2.8.2}$$

dove g_t è il gradiente al tempo *t*, g_t^2 è il quadrato del gradiente nei termini del quadrato delle sue componenti, ρ_1 e ρ_2 (solitamente settati a 0.9 e 0.999, rispettivamente) rappresentano il tasso di decadimento esponenziale per il primo e per il secondo momento; i due momenti sono inizializzati a 0: $m_t = u_t = 0$.

Nelle prime iterazioni i valori delle due medie mobili, a causa della loro inizializzazione a zero, potrebbero essere molto piccoli, gli autori del metodo Adam hanno proposto una nuova versione che presenta una correzione ai due momenti:

$$\hat{m}_t = \frac{m_t}{(1 - \rho_1^t)} \tag{2.8.3}$$

$$\hat{u}_t = \frac{u_t}{(1 - \rho_2^t)} \tag{2.8.4}$$

L’ultimo aggiornamento per ogni parametro θ_t della rete è:

$$\theta_t = \theta_{t-1} - \lambda \frac{\hat{m}_t}{\sqrt{\hat{u}_t} + \epsilon} \tag{2.8.5}$$

Dove λ è il learning rate, ϵ è un numero positivo molto piccolo, tale da preservare una possibile divisione per 0 (solitamente $\epsilon = 10^{-8}$).

3. Architettura di LSTM_GRU

L‘obiettivo di questa tesi è quello di ottimizzare la topologia di una rete neurale per migliorare la prestazione della stessa.
Combinazioni di livelli presentati precedentemente formano diverse strutture allenate con i dataset “Image” e “mAn”. Dati i tempi computazionali dell’architettura LSTM_GRU, sono stati avviati test esaustivi.

L’addestramento della rete è avvenuto attraverso differenti varianti dell’ottimizzazione Adam. I principali iperparametri sono stati settati come segue:
è stato utilizzato un alto *learning rate*: 0.01, inoltre sono stati specificati anche il *gradient decay factor*: 0.5 e lo *squared gradient decay factor*: 0.999. Il mini-batch size è stato fissato a 30, mentre il numero di *epoches* a 150.
La rete del modello in esame è composta da un primo livello LSTM con N livelli nascosti (N = 125) seguito da un dropout layer che oscura i nodi con una probabilità di 0.4. Successivamente troviamo un livello GRU con M livelli nascosti (M = 100) seguito da un ulteriore livello di dropout con probabilità di 0.4. La rete si conclude con un fully connected layer e un livello con la sigmoide.

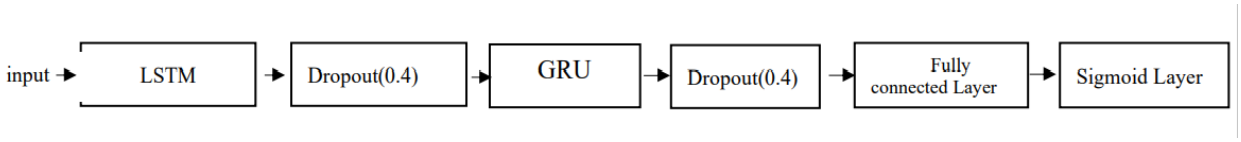

Figura 3.1 Schema del modello LSTM_GRU

La *loss function* è la *binary cross entropy loss* tra gli *output* (valori predetti) e i *target* o attuali etichette, e calcola la perdita di un set di *m* osservazioni calcolando la seguente media:

$$\text{CELoss} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{l}\mathbf{y}_i(j)\cdot\log\left(\mathbf{h}_i(j)\right) + (1-\mathbf{y}_i(j))\cdot\log\left(1-\mathbf{h}_i(j)\right),$$

dove $y_i \in \{0, 1\}^l$ e $h_i \in \{0, 1\}^l$ sono le attuali e predette etichette

4. Risultati sperimentali

4.1 Datasets

La valutazione dei diversi approcci richiede dataset con categorizzazione multi-label binaria. Sono stati selezionati i seguenti dodici dataset per l'ampia differenza che presentano tra loro (come ad esempio classificazione di immagini, musica, biologia, farmaci) e perché spesso utilizzati per confrontare topologie di classificazioni Multi-label:

- Cal500 [28]: è un dataset composto da 502 canzoni popolari Western rappresentate da 68
  features numeriche. Ogni istanza è stata annotata manualmente da annotatori umani utilizzando 174 etichette distinti. A loro volta le etichette sono suddivise in 6 categorie semantiche: strumentazione, caratteristiche vocali, generi, emozioni, qualità acustica e termini d'uso.
- Scene [29]: è un dataset di 2407 immagini a colori di scene differenti (divise in immagini di training e di test) suddivise in 6 categorie: spiaggia (369), tramonto (364), campi (327), fogliame d'autunno (360), montagna (223) e città (210). Nel dataset sono presenti sessantadue immagini che appartengono a due categorie e una che appartiene a tre. Tenendo conto delle categorie in comune il numero totale di etichette è 15. Ogni immagine in questo dataset è passata attraverso quattro fasi della procedura di *preprocessing*.
  Nella prima fase l'immagine viene convertita nello spazio uniforme CIE Luv. Nella seconda fase si divide l'immagine in 49 blocchi con una griglia 7x7. Con la terza fase si calcolano la media e la varianza di ogni blocco. La media è l'equivalente di un'immagine a bassa risoluzione, mentre la varianza corrisponde a caratteristiche di trama computazionalmente poco costose. Infine, nell'ultima fase l'immagine viene trasformata in un vettore con 294 caratteristiche di dimensioni (49x3x2).
- Image [30]: è un dataset di 2000 scene naturali raggruppate in cinque categorie base: deserto (340), montagna (268), mare (341), tramonto (216) e alberi (378). Questo dataset è ideato con l'intento di produrre un numero maggiore di immagini che appartengono a due (442) e tre (15) categorie. Le immagini di questo dataset utilizzano la stessa procedura di preprocessing di [29].

- Yeast [2]: è un dataset biologico per la classificazione di 2417 microarray di dati e profili filogenetici. Ogni istanza viene rappresentata da 103 feature numeriche e 14 etichette distinte. Ogni gene può avere più di un'etichetta.
- Arts [7]: è un dataset costituito da 5000 immagini artistiche, ciascuna descritta da 462 features numeriche in cui ogni immagine può appartenere ad alcune delle 26 classi presenti.
- Liu [29]: è un dataset di farmaci raccolti per prevedere in silico i loro effetti collaterali. Comprende 832 farmaci rappresentati da 2892 classi.
- ATC [31]: è una raccolta di 3883 prodotti farmaceutici codificati Anatomical Therapeutic Chemical (ATC). Ogni istanza `e rappresentata da 42 feature e 14 classi
- ATC_f: è una variante della raccolta citata sopra che include le stesse istanze, ma rappresentate da un descrittore 806-dimensionale.
- mAn [6]: è un dataset di proteine rappresentate da 20 feature e 20 etichette.
- Bibitex, Enron, Health: Questi dataset possiedono dati altamente sparsi usati in [7]
-

Un resoconto di tutti i dataset, contenente il numero di istanze, features, etichette e il numero medio di etichette per istanza (LCard), è riportato nella Tabella 1.5.1. I dataset 1-5 sono presenti nel toolkit IMCC di MatLab [7] disponibile all'indirizzo: https://github.com/keauneuh/Incorporating-Multiple-Cluster-Centers-forMulti-Label-Learning/tree/master/IMCCdata.

Name	#patterns	#features	#labels	LCard
CAL500	502	68	174	26.044
Image	2000	294	5	1.236
Scene	2407	294	5	1.074
Yeast	2417	103	14	4.24
Arts	5000	462	26	1.636
ATC	3883	42	14	1.265
ATC_f	3883	700	14	1.265
Liu	832	2892	1385	71.160
mAn	3916	20	20	1.650
bibtex	7395	1836	159	2.402
enron	1702	1001	53	3.378
health	5000	612	32	1.662

Tabella 4.1.1 Resoconto dei dodici dataset utilizzati

4.2 Presentazione degli Ensemble

Gli *ensemble* combinano gli output di più modelli per migliorare le prestazioni del sistema e ridurre l'*overfitting*. Aumentando la varietà dei classificatori è possibile migliorare le predizioni e la genericità dell'*ensemble*.

Sono valutati diversi ottimizzatori per la creazione di *ensemble*: ottimizzatore Adam [18], diffGrad [33] e undici nuove varianti di questo metodo (DGrad, Cos#1, Exp, Sto, CLRW, EpochCos#1, EpochDecay, Sqrt, Hyp, CyclicExp e LRClipping).

L'obiettivo di questi esperimenti è quello di testare le prestazioni delle nuove topologie; I risultati vengono confrontati con l'attuale stato dell'arte, ovvero l'*Incorporating Multiple Clustering Center* (IMCC).

IMCC [7] prevede due fasi: 1) aumento dei dati del *train set* (*data augmentation*) tramite la creazione di esempi virtuali e 2) allenamento *multi-label*. È proprio l'aumento dei dati che permette all'IMCC di avere il suo principale aumento di prestazioni.

Tutti i classificatori sono fusi tra loro a livello di confidenza con l'*average rule*. Di seguito viene riportata una breve descrizione degli *ensemble* utilizzati:

- StoGRU: è l'*ensemble* costituito da 40 GRU_A (GRU con *N* unità nascoste, nell'esperimento $N = 50$, seguito da un livello di *pooling*, un livello *fully connected* e la funzione sigmoidea), combinate e fuse tra loro tramite *average rule*;
- StoGRU_B: come StoGRU, ma costituita da GRU_B (identica a GRU_A, ma con un livello convoluzionale seguito da un livello di *batch-normalization* inseriti subito prima della rete stessa);
- StoTCN: è l'*ensemble* costituito da 40 TCN, combinate e fuse tra loro tramite *average rule*;
- StoTCN_B: come StoTCN ma costituita da TCN_B (identiche a TCN_A, ma con un livello convoluzionale messo subito prima della rete);
- StoGRU_TCN: è l'*ensemble* di 40 GRU_TCN (combinazione in sequenza di GRU_A, senza il livello di *pooling*, e TCN_A), fuse tra loro con metodo stocastico;
- LSTM_GRU_ è l'*ensemble* composto da 40 LSTM_GRU (presentata nel capitolo 3);
- ENNbase: è la fusione tramite l'*average rule* di StoGRU e StoTCN.;
- ENN: è la fusione tramite *average rule* degli ensamble StoGRU e StoTCN, StoGRU_B, StoTCN_B e StoGRU_TCN;
- ENNnew: è la fusione tramite *average rule* di ENN e LSTM_GRU;
- LeaveOO_Sparse: utilizza n-1 dataset per selezionare il metodo più adatto per classificare il dataset restante. In questo caso considera solo dataset sparsi;
- LeaveOO_NonSparse: come LeaveOO_Sparse, ma considera solo dataset non-sparsi;
- LeaveOO: come i precedenti, ma questa volta considera qualsiasi dataset;

Sono state aggiunte anche fusioni tra ENNnew/Enn e IMCC:

- ENN+IMCC: utilizza la regola dell'addizione tra ENN e IMCC; i risultati di ENN sono stati normalizzati prima della fusione visti i differenti range di valori rispetto a IMCC;
- ENN+3×IMCC: come la precedente, ma i risultati di IMCC sono pesati secondo un fattore 3;
- ENNnew+IMCC: utilizza la regola dell'addizione tra ENNnew e IMCC; i risultati di ENNnew sono stati normalizzati prima della fusione;

- ENNnew+3×IMCC: come la precedente, ma i risultati di IMCC sono pesati di un valore 3.

4.3 Risultati ottenuti

Nella tabella 4.3.1 sono riportati le varie *average precision* ottenute dagli *ensemble* descritti in precedenza (righe) sul determinato *dataset* (colonne).

Average Precision	Cal500	Image	Scene	Yeast	Arts	ATC	ATC_f	Liu	mAn	Average
IMCC	0.50	0.836	0.904	0.773	0.619	0.866	0.922	0.523	0.978	0.769
StoGRU	0.498	0.851	0.911	0.74	0.561	0.872	0.872	0.485	0.979	0.752
StoGRU_B	0.49	0.861	0.908	0.741	0.555	0.877	0.848	0.485	0.978	0.749
StoTCN	0.498	0.847	0.913	0.764	0.506	0.882	0.9	0.498	0.977	0.754
StoTCN_B	0.497	0.855	0.917	0.765	0.541	0.883	0.903	0.505	0.976	0.760
StoGRU_TCN	0.491	0.852	0.916	0.752	0.592	0.89	0.913	0.51	0.977	0.766
LSTM_GRU	0.493	0.837	0.888	0.768	0.637	0.873	0.742	0.551	0.976	0.752
ENNbase	0.502	0.855	0.922	0.756	0.552	0.888	0.916	0.497	0.979	0.763
ENN	0.499	0.859	0.924	0.762	0.582	0.893	0.916	0.505	0.979	0.769
ENNnew	0.498	0.866	0.922	0.778	0.628	0.892	0.917	0.521	0.978	0.778
LeaveOO_Sparce	---	---	---	---	0.628	---	---	0.53	---	0.579
LeaveOO_NonSparse	0.501	0.854	0.918	0.781	---	0.882	0.925	---	0.979	0.834
LeaveOO	0.5	0.847	0.918	0.781	0.635	0.882	0.923	0.53	0.979	0.777
ENN+IMCC	0.502	0.856	0.922	0.783	0.631	0.883	0.927	0.521	0.979	0.778
ENN+3×IMCC	0.503	0.845	0.917	0.777	0.627	0.876	0.926	0.524	0.979	0.775
ENNnew+IMCC	0.502	0.855	0.922	0.782	0.634	0.883	0.927	0.528	0.979	0.779
ENNne+3×IMCC	0.503	0.847	0.913	0.778	0.629	0.875	0.925	0.527	0.979	0.775
TreeBagger	0.523	0.858	0.898	0.774	0.600	0.891	0.904	0.522	0.985	0.773

Tabella 4.3.1 Confronto delle average precision ottenute dai vari ensemble

Osserviamo che i test sono stati svolti su 9 dataset. Inoltre, per dataset sparsi (Liu, Arts, bibtex, enron e health) è stata performata una riduzione delle dimensioni tramite PCA mantenendo il 99% della varianza.
LSTM_GRU non converge se viene applicata la normalizzazione sul dataset ATC_f ; ma non si ottengono comunque buoni risultati se non viene applicata nessuna normalizzazione. Se viene applicata la PCA alo stesso dataset la sua prestazione rimane comunque minore di quella ottenuta con altri metodi.

Viene utilizzata di solito quando il numero di parametri è maggiore del numero di osservazioni. *L'overfitting* è una situazione particolare dove, a causa delle grandi dimensioni delle strutture, la rete ha bisogno di regolare molti parametri con troppi gradi di libertà. A causa di ciò la rete può risultare estremamente accurata durante l'allenamento, ma non durante il test.

Dalla tabella precedente possiamo trarre le seguenti conclusioni:

- LSTM_GRU lavora molto bene in dataset sparsi, mentre è risulta simile alle prestazioni degli altri modelli basati su GRU/TCN per dataset non sparsi.
- LeaveOO_Sparse aumenta le performance di ENN+3×IMCC, mentre LeaveOO_NonSparse e LeaveOO sembrano non ottenere prestazioni interessanti
- ENNnew+IMCC è *l'ensemle* migliore, è migliore di IMCC su ogni dataset ed è più robusto di ENN+IMCC per dataset sparsi (Liu, Arts, bibtex, enron e health)

In generale, la nuova topologia LSTM_GRU tende a superare solo in alcuni casi le prestazioni degli ensemble precedenti. Unendola con ENN e poi fondendola tramite la regola dell'addizione con IMCC supera lo stato dell'arte

5. Conclusioni

L’apprendimento Multi-label è una delle classi che ammette l’assegnazione di più etichette contemporaneamente per ogni istanza o *pattern*.

In questo elaborato sono state proposte delle combinazioni di ensemble di LSTM_GRU con altri modelli, addestrati con nuove varianti del metodo di ottimizzazione Adam. La rete quando combinata con IMCC risulta in una classificazione multi-label migliore.

Per validare l’approccio usato sono stati utilizzati nove dataset con domini applicativi molto differenti tra loro.

L’unione con altre topologie di deep learning e nuove ottimizzazioni potrebbero essere un punto di partenza interessante per esperimenti futuri

Bibliografia

[1] E. G. Galindo and S. Ventura, "Multi label learning: a review of the state of the art and ongoing research," Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 4, no. 6, pp. 411-444, 2014, doi: doi.org/10.1002/widm.1139.

[2] A. Elisseeff and J. Weston, A kernel method for multi-labelled classification (NIPS). MIT Press Direct, 2001.

[3] L. Nanni, A. Lumini, and S. Brahnam, "Neural networks for Anatomical Therapeutic Chemical (ATC)," ArXiv, vol. abs/2101.11713, 2021.

[4] X. Cheng, W.-Z. Lin, X. Xiao, and K.-C. Chou, "pLoc_bal-mAnimal: predict subcellular localization of animal proteins by balancing training dataset and PseAAC," Bioinformatics, vol. 35, no. 3, pp. 398-406, 2018, doi: 10.1093/bioinformatics/bty628.

[5] L. Chen et al., "Predicting gene phenotype by multi-label multi-class model based on essential functional features," Molecular genetics and genomics : MGG, vol. 296, no. 4, pp. 905-918, 2021, doi: 10.1007/s00438-021-01789-8.

[6] Y. Shao and K. Chou, "pLoc_Deep-mAnimal: A Novel Deep CNN-BLSTM Network to Predict Subcellular Localization of Animal Proteins," Natural Science, vol. 12, 5, pp. 281-291, 2020, doi: 10.4236/ns.2020.125024.

[7] S. Shu et al., "Incorporating Multiple Cluster Centers for Multi-Label Learning," ArXiv, vol. abs/2004.08113, 2020.

[8] M. Ibrahim, M. U. G. Khan, F. Mehmood, M. Asim, and W. Mahmood, "GHS-NET a generic hybridized shallow neural network for multi-label biomedical text classification," Journal of biomedical informatics, vol. 116, p. 103699, 2021, doi: 10.1016/j.jbi.2021.103699.

[9] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Light Gated Recurrent Units for Speech Recognition," IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 2, no. 2, pp. 92-102, 2018, doi: 10.1109/TETCI.2017.2762739.

[10] Y. Kim and J. Kim, "Human-Like Emotion Recognition: Multi-Label Learning from Noisy Labeled Audio-Visual Expressive Speech," presented at the 2018

IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018.

[11] M. B. Messaoud, I. Jenhani, N. B. Jemaa, and M. W. Mkaouer, "A Multi-label Active Learning Approach for Mobile App User Review Classification," in KSEM, 2019.

[12] J. P. Singh and K. Nongmeikapam, "Negative Comments Multi-Label Classification," 2020 International Conference on Computational Performance Evaluation (ComPE), pp. 379-385, 2020, doi: 10.1109/ComPE49325.2020.9200131.

[13] W. Zhang, F. Liu, L. Luo, and J. Zhang, "Predicting drug side effects by multilabel learning and ensemble learning," BMC Bioinformatics, vol. 16, no. 365, 2015.

[14] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Random k-Labelsets for Multilabel Classification," IEEE Transactions on Knowledge and Data Engineering, vol. 23, no. 7, pp. 1079-1089, 2011.

[15] Y. Yang and J. Jiang, "Adaptive Bi-Weighting Toward Automatic Initialization and Model Selection for HMM-Based Hybrid Meta-Clustering Ensembles," IEEE Transactions on Cybernetics, vol. 49, no. 5, pp. 1657-1668, 2019.

[16] J. M. Moyano, E. G. Galindo, K. Cios, and S. Ventura, "Review of ensembles of multi-label classifiers: Models, experimental study and prospects," Inf. Fusion, vol. 44, no. November, pp. 33-45, 2018.

[17] K. Cho et al., "Learning Phrase Representations using RNN Encoder Decoder for Statistical Machine Translation," in EMNLP, Varna, Bulgaria, 2014, pp. 25-32.

[18] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," CoRR, vol. abs/1412.6980, 2015.

[19] K. C. Chou, "Some remarks on predicting multi-label attributes in molecular biosystems," Molecular Biosystems, vol. 9, pp. 10922-1100, 2013.

[20] M.-L. Zhang and Z.-H. Zhou, "Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization," IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 10, pp. 1338-1351, 2006, doi: 10.1109/TKDE.2006.162.

[21] A. E. Samy, S. R. El-Beltagy, and E. Hassanien, "A Context Integrated Model for Multi-label Emotion Detection," Procedia Computer Science, vol. 142, pp. 61-71, 2018/01/01/ 2018, doi: https://doi.org/10.1016/j.procs.2018.10.461.

[22] J. Zhang, Q. Chen, and B. Liu, "NCBRPred: predicting nucleic acid binding residues in proteins based on multilabel learning," Briefings in bioinformatics, vol. 22, no. 2, 2021.

[23] D. Li, H. Wu, J. Zhao, Y. Tao, and J. Fu, "Automatic Classification System of Arrhythmias Using 12-Lead ECGs with a Deep Neural Network Based on an Attention Mechanism," Symmetry, vol. 12, no. 11, p. 1827, 2020. [Online]. Available: https://www.mdpi.com/2073-8994/12/11/1827.

[24] Hochreiter, S., and J. Schmidhuber. "Long short-term memory." Neural computation. Vol. 9, Number 8, 1997, pp.1735–1780.

[25] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," Neural Computation, vol. 12, no. 10, pp. 2451-2471, 2000, doi: 10.1162/089976600300015015.

[26] K. Zhang, Z. Liu, and L. Zheng, "Short-Term Prediction of Passenger Demand in Multi-Zone Level: Temporal Convolutional Neural Network With Multi-Task Learning," IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 4, pp. 1480-1490, 2020, doi: 10.1109/TITS.2019.2909571.

[27] Srivastava, Nitish, et al. ”Dropout: a simple way to prevent neural networks from overfitting”, JMLR 2014

[28] D. Turnbull, L. Barrington, D. A. Torres, and G. Lanckriet, "Semantic Annotation and Retrieval of Music and Sound Effects," IEEE Transactions on Audio, Speech, and Language Processing, vol. 16, no. 22, pp. 467-476, 2008, doi: 10.1109/TASL.2007.913750.

[29] M. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," Pattern Recognit., vol. 37, no. 9, pp. 1757-1771, 2004.

[30] M.-L. Zhang and Z. Zhou, "ML-KNN: A lazy learning approach to multi-label learning," Pattern Recognit., vol. 40, no. 7, pp. 2038-2048, 2007, doi: 10.1016/j.patcog.2006.12.019.

[31] L. Chen, "Predicting anatomical therapeutic chemical (ATC) classification of drugs by integrating chemical-chemical interactions and similarities," PLoS ONE, vol. 7, no. e35254, 2012.

[32] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," ACM Comp Surv, vol. 31, no. 3, pp. 264-323, 1999.

[33] S. Dubey, S. Chakraborty, S. K. Roy, S. Mukherjee, S. K. Singh, and B. Chaudhuri, "diffGrad: An Optimization Method for Convolutional Neural Networks," IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 11, pp. 4500-4511, 2020.

[34] Janiesch, C., Zschech, P. & Heinrich, K., «Machine learning and deep learning. Electron Markets,» 2021. [Online]. Available: https://doi.org/10.1007/s12525-021-00475-2.

[35] C. Nicholson, «A.I. Wiki- A Beginner's Guide to Important Topics in AI, Machine Learning, and Deep Learning.,» [Online]. Available: https://wiki.pathmind.com/lstm.

[36] Z. C. Lipton, J. Berkowitz, C. Elkan, «A Critical Review of Recurrent Neural Networks,» arXiv:1506.00019, 2015.

[37] M. Phi, «Illustrated Guide to Recurrent Neural Networks,» Towards Data Science, [Online]. Available: https://towardsdatascience.com/illustratedguide-to-recurrent-neural-networks-79e5eb8049c9.

[38] J. Z. K. V. K. Shaojie Bai, «An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling,» arXiv:1803.01271, 2018.

[39] Li Jing et al. πGated orthogonal recurrent units: On learning to forget∫. In: Neural computation 31.4 (2019), pp. 765–783.

[40] Breiman, L. "Random Forests." *Machine Learning* 45, pp. 5−32, 2001.