
**UNIVERSITA' POLITECNICA DELLE MARCHE**

FACOLTA' DI INGEGNERIA

---

Corso di Laurea magistrale Ingegneria Elettronica

# Architettura Deep Learning End-to-End di tipo Target Speaker per il Riconoscimento Vocale Automatico

# End-to-End Target Speaker Deep Learning Architecture for Automatic Speech Recognition

Relatore: Chiar.mo

Prof. Stefano Squartini

Correlatore:

Dott. Samuele Cornell

Tesi di Laurea di:

Alessandro Bacà

A.A. 2024 / 2025

# 1 SOMMARIO

# INTRODUZIONE

In questo capitolo iniziale viene brevemente introdotto il problema dell' **Automatic Speech Recognition** (ASR in breve) e di come è stato affrontato nel tempo fino ad oggi.

## 1.1 AUTOMATIC SPEECH RECOGNITION – ASR

L' Automatic Speech Recognition è un campo di ricerca che studia soluzioni per trascrivere un segnale vocale in forma testuale per mezzo di computers. Gli sviluppi tecnologici recenti in campo Deep Learning e Big Data hanno permesso un forte evoluzione del settore ASR e la diffusione di prodotti e servizi che fanno uso.

Alcuni esempi di applicazioni di tipo Automatic Speech Recognition sono: l'acquisizione di comandi vocali da parte di dispositivi elettronici personali, la loro attivazione per mezzo Wake Word [1], servizi di assistenza vocale, la generazione automatica di sottotitoli o la diarizzazione delle conferenze.

## 1.2 STORIA E SOLUZIONI

Gli algoritmi di Automatic Speech Recognition hanno subito una lunga evoluzione attraverso varie fasi (B.H. Juang, 2004), rese disponibili grazie alle innovazioni della ricerca nella linguistica computazionale, dell'intelligenza artificiale e delle reti neurali artificiali. Qui vengono riportati alcuni contributi innovativi per il loro tempo in questo arco temporale di evoluzione.

I primi modelli di algoritmi ASR erano piuttosto primitivi, il loro funzionamento era basato sull'individuazione di pattern acustici al fine di riconoscere un numero limitatissimo di espressioni vocali manifeste singolarmente e

[1] Una parola chiave che innesca l'attivazione di un sistema automatico, come quando Jean-Luc Picard diceva "Computer" seguito dal comando "…luci" o come l'attivazione di Cortana con "Hey Cortana!".

pronunciate da una sola persona. Si trattava quindi di prime applicazioni Speaker Dependent Small Vocabulary Automatic Speech Recognition.

Nel 1952 i Bell Laboratories hanno prodotto un sistema in grado di riconoscere numeri isolati pronunciati da una singola persona (Davis, Biddulph, & Balashek, 1952). Il meccanismo alla base consisteva nello stimare le principali formanti prodotte nella fonazione delle vocali a partire dallo spettro di potenza del segnale di speech. Il comportamento di queste formanti veniva poi confrontato con dei pettern noti per determinare il numero pronunciato.

Nel 1959 gli omonimi Forgie rimuovono il vincolo del funzionamento per un solo speaker creando un sistema Speaker Independent Small Vocabulary Automatic Speech Recognition in grado di riconoscere dieci vocali indipendentemente dallo speaker che le pronuncia (J. W. Forgie, 1959).

Nel 1959 Fry e Denes hanno creato un sistema basato su fonemi in grado di riconoscere quattro vocali e nove consonanti (D. B. Fry, 1959) impiegando per primi la sintassi statistica al livello di fonema in un sistema di Automatic Speech Recognition. Questa innovazione introduceva informazioni sulle sequenze di fonemi ammesse nella lingua inglese, permettendo di migliorare la precisione nel riconoscimento dei fonemi.

Nel 1962 Sakai e Doshita impiegano per la prima volta un segmentatore di speech che permetteva di analizzare e riconoscere parti differenti della frase in ingresso (J. Sakai). Nei risultati precedenti, l'assenza del segmentatore richiedeva che ogni frase di ingresso avesse un solo elemento da riconoscere. Con il segmentatore era invece possibile riconoscere frasi contenenti più di un elemento. Questo tipo di innovazione si può considerare come il precursore dei metodi Continuous Speech Recognition.

Durante i primi anni del 1970 nuovi contributi di ricerca introducono l'impiego della programmazione dinamica in varie forme, ad oggi ancora una parte importante degli algoritmi in uso.

Nel 1971 il programma di ricerca DARPA Speech Understanding Research, porta alla luce il sistema Harpy (Lowerre, 1986) capace di riconoscere lo speech con un vocabolario di 1011 parole con una precisione ragionevole. Il sistema era un primo grande esempio Large Vocabulary Continuous Speech Recognition. Una importante innovazione di questo algoritmo consisteva nell'eseguire una ricerca lungo un grafo, dove il linguaggio dello speech recognition era rappresentato come una rete connessa, derivata da una rappresentazione lessicale delle parole, con regole di produzione sintattica e regole di terminazione delle parole. Lo speech di ingresso veniva sottoposto ad una analisi parametrica e segmentato, poi la sequenza di speech parametrica segmentata era confrontata con dei template noti determinando una metrica basata sull'Itakura distance (Itakura, 1975). La ricerca a grafo era basata su Beam Search (Beam Search, s.d.), dove in uno schema iterativo, molteplici ipotesi di sequenze di parole riconosciute, in linea con dei vincoli linguistici ed empirici, venivano generate e tra queste venivano poi mantenute quelle che massimizzavano una metrica specifica, di vicinanza ai template noti. Alla fine del processo iterativo, l'ipotesi con la metrica migliore diveniva la sequenza di parole riconosciuta.

Dalla IBM è stato prodotto un sistema chiamato Tangora (F. Jelinek, 1975) di tipo Speaker Dependent Large Vocabulary Continuous Speech Recognition, veniva munito di un modello linguistico, una componente ad oggi ancora in uso. Il modello linguistico era un insieme di regole sintattico-statistiche che descrivevano la probabilità di una sequenza di simboli in un linguaggio (come parole o fonemi) che possono apparire in un segnale di speech. La variante di modello chiamato n-gram era quella più diffusa e definiva la probabilità di occorrenza di una sequenza ordinata di n parole.

Seguendo un altro tipo di filosofia, gli AT&T Bell Laboratories si sono concentrati sulla creazione di un sistema Speaker Independent in grado di supportare le differenze nello speech tra vari speakers e tra vari accenti regionali. Quindi sono stati creati vari tipi di algoritmi di speech clustering per la creazione di pattern sonori di riferimento per le parole nella forma di

templates o di modelli statistici da impiegare su una vasta gamma di tipi di speaker e di accenti. La gestione di una grande variabilità acustica delle varie rappresentazioni di speakers e accenti ha portato alla creazione di una gamma di metriche di misura della distanza.

Negli anni 80 c'è stato un cambio di paradigma che ha portato la direzione di ricerca da un approccio intuitivo di ricerca ed identificazione di pattern noti ad un approccio più rigoroso di modellazione statistica. Questo cambiamento è stato avviato con la diffusione dell'Hidden Markov Model (HMM in breve) (S. E. Levinson, 1983) (Ferguson, 1980). Un HMM è un processo stocastico che modella la variabilità intrinseca del segnale di speech, ma anche la struttura del linguaggio parlato secondo una impostazione a modello statistico.

Con l'avvento del Deep Learning, grazie alla loro capacità di apprendere mapping fortemente non lineari, le reti neurali artificiali profonde hanno iniziato a sostituire varie componenti dei sistemi classici di Automatic Speech Recognition portando alla creazione prima di sistemi ibridi classico-deep learning, poi di sistemi totalmente neurali.

# 2 CONTESTO TEORICO E STRUMENTI SOFTWARE

In questo capitolo vengono formalizzate le problematiche di interesse, sono forniti dei cenni teorici riguardanti importanti elementi del contesto ASR e delle reti neurali artificiali. In fine vengono presentati i principali strumenti software impiegati per la realizzazione del lavoro di tesi.

## 2.1 IL PROBLEMA ASR

Il problema dell'Automatic Speech Recognition (Watanabe S., 2017) consiste nel trascrivere un segnale di speech in forma testuale. Il segnale di speech in ingresso nel dominio del tempo $x(t)$ viene trasformato in una sequenza di speech features $X$ con lunghezza di sequenza $T$ e dimensione del vettore di feature pari a $D$:

$$X = \{x_t \in \mathbb{R}^D | t = 1, \dots, T\}$$

La sequenza di parole in uscita $W$ con lunghezza di sequenza $N \leq T$ è composta da parole $w_n$ prese da un vocabolario $V$, se $n$ indica l'n-esima parola del vocabolario:

$$W = \{w_n \in V | n = 1, \dots, N\}$$

Il riconoscimento o decodifica avviene determinando la sequenza di parole $\widehat{W}$ più probabile sulla base di $X$. Se $V^*$ è l'insieme di tutte le sequenze di parole possibili il problema diventa:

$$\widehat{W} = arg \max_{W \in V^*} p(W|X) \quad (1)$$

La soluzione del problema dipende quindi dal calcolo o dalla stima della probabilità a posteriori $p(W|X)$ e dalla sua massimizzazione nello spazio di $V^*$. Il calcolo della distribuzione viene normalmente approcciato fattorizzando il problema e considerando più o meno assunzioni fino ad ottenere fattori direttamente calcolabili dalle componenti tipiche dei sistemi ASR. Il processo di massimizzazione invece prende il nome di decodifica.

Per misurare le prestazioni dei sistemi ASR vengono comunemente usate due metriche: il Word Error Rate (WER in breve) ed il Character Error Rate (CER in breve). In breve esse indicano rispettivamente il tasso di errore al livello di parola e di carattere nella trascrizione prodotta la sistema ASR, quando questa è confrontata con la trascrizione di riferimento.

## 2.2 Soluzione HMM/DNN

I sistemi HMM/DNN sono sistemi chiamati ibridi in quanto combinano Hidden Markov Model e Deep Neural Network per l'attuazione del riconoscimento. In particolare il sistema HMM stabilisce un legame statistico tra stati nascosti e fonemi che compongono lo speech, mentre la rete neurale artificiale interviene nella stima dello stato nascosto di ogni frame nel sistema HMM.

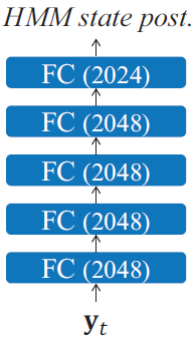



*Figura 1Esempio di Modello Acustico DNN in un sistema HMM/DNN, figura modificata da (Delcroix, 2018)*

Se il sistema HMM ha $J$ stati diversi, la sequenza di stati nascosti è $S = \{s_t \in \{1, \dots, J\} | t = 1, \dots, T\}$ e con essa si può fattorizzare la distribuzione cercata in tre componenti (H. Bourland, 1994):

$$\widehat{W} = arg \max_{W \in V^*} p(W|X) \cong arg \max_{W \in V^*} \sum_{S} p(X|S)p(S|W)p(W)$$

dove le tre distribuzioni $p(X|S)$ , $p(S|W)p(W)$ , $p(W)$ sono chiamate rispettivamente: acoustic model, lexicon model, language model e sono approssimabili con appositi moduli presenti nel sistema.

Questo tipo di soluzione rende il problema risolvibile sfruttando la fattorizzazione di $p(W|X)$ in componenti trattabili e su una serie di assunzioni

di indipendenza condizionale, ma ereditando da una impostazione classica HMM, essa include alcuni aspetti negativi. La creazione di un modello accurato HMM/DNN richiede una varietà di processi specifici per ogni modulo componente, come l'istruzione del modello HMM e delle sue distribuzioni di emissione prima ancora di istruire la DNN. Il funzionamento richiede informazioni linguistiche, che vengono create a mano da soggetti esperti nella forma di pronunciation dictionary per mappare parole a fonemi. Tali informazioni sono pertanto costose, poco pratiche da reperire e suscettibili di errore. In scenari di applicazione realistici, i dati impiegati non riflettono necessariamente le assunzioni di indipendenza condizionale adottate. Infine si tiene conto che la decodifica è relativamente complessa e che ogni modulo componente è istruito separatamente con un proprio criterio e non risponde ad un criterio di ottimizzazione globale, producendo in generale un risultato sub-ottimo. La decodifica viene eseguita con i metodi tipici dei sistemi HMM.

### 2.2.1 Acoustic Model

L'acoustic model $p(X|S)$ è ulteriormente fattorizzabile:

$$p(X|S) = \prod_{t=1}^{T} p(x_t|x_1, \dots, x_{t-1}, S)$$

sotto assunzione di indipendenza condizionale tra ingresso e contesto degli stati nascosti si ha:

$$p(X|S) \cong \prod_{t=1}^{T} p(x_t|s_t) \propto \prod_{t=1}^{T} \frac{p(s_t|x_t)}{p(s_t)}$$

dove la funzione di probabilità per frame temporale $p(x_t|s_t)$ è approssimata dalla distribuzione a posteriori per frame $p(s_t|x_t)/p(s_t)$, che viene stimata con una Deep Neural Network. Questa rete, istruita a classificatore, classifica lo stato nascosto in un dato frame temporale a partire dalle speech feature di quello stesso frame. Nella derivazione del risultato, l'assunzione di indipendenza condizionale impiegata è di tipo forte, perché trascura qualsiasi contesto tra ingresso e stati nascosti. Per alleviare il problema, si può reintrodurre qualche forma di contesto in due modi: si possono usare DNN

ricorrenti, che generano implicitamente il contesto grazie alla ricorrenza, oppure usando DNN non ricorrenti ma includendo alle speech feature di ingresso qualche finestra di contesto anteriore e posteriore.

### 2.2.2 Lexicon Model

La distribuzione viene fattorizzata ulteriormente includendo l'indipendenza condizionale Markoviana:

$$p(S|W) = \prod_{t=1}^{T} p(s_t|s_1, \dots, s_{t-1}, W) \cong \prod_{t=1}^{T} p(s_t|s_{t-1}, W)$$

dove $p(s_t|s_{t-1}, W)$ è calcolabile considerando la probabilità di transizione di stato del modello HMM e convertendo $W$ in stati nascosti, attraverso una rappresentazione a fonemi, per mezzo di un pronunciation dictionary.

### 2.2.3 Language Model

Analogamente alla distribuzione precedente viene eseguita una fattorizzazione con assunzione di indipendenza condizionale markoviana meno rigida:

$$p(W) = \prod_{n=1}^{N} p(w_n|w_1, \dots, w_{n-1}) \cong \prod_{n=1}^{N} p(w_n|w_{n-m-1}, \dots, w_{n-1})$$

## 2.3 Soluzione Connectionist Temporal Classification

Un importante vincolo in questo tipo di soluzione sequence to sequence è il monotonic alignment del mapping tra ingresso e uscita, senza il quale non è possibile usare questo approccio. Anche la trattazione CTC come quella precedente HMM/DNN fa uso di assunzioni markoviane e non sfrutta a pieno i benefici di un sistema ASR istruito end-2-end, ma a differenza di questa non necessita dei moduli specifici di pronunciation dictionary e della costruzione del sistema HMM.

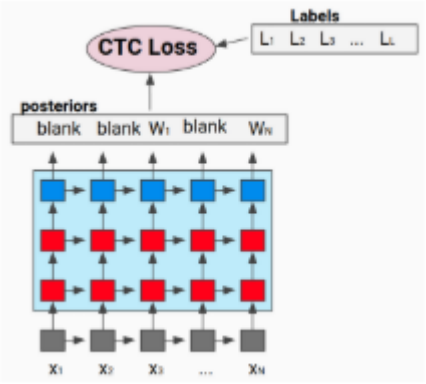



*Figura 2 Esempio di Architettura CTC, immagine modificata da (Ravanelli, s.d.)*

Secondo la formulazione CTC (Watanabe S., 2017) dato un alfabeto di lettere distinte definito dall'insieme $U$, una sequenza di lettere lunga $L$ è descritta come:

$$C = \{c_l \in U | l = 1, \dots, L\}$$

Definita la sequenza di lettere aumentata $C'$ contenente un simbolo extra chiamato black symbol $< b >$ nelle posizioni ad indice $l$ dispari e una lettera quando l'indice è pari:

$$\begin{aligned} C' &= \{< b >, c_1, < b >, c_2, \dots, < b >, c_l, < b >\} \\ &= \{c'_l \in U \cup \{< b >\} | l = 1, \dots, 2L + 1\} \end{aligned}$$

Si introduce la sequenza di lettere $Z$ per frame con aggiunta del blank symbol:

$$Z = \{Z_t \in U \cup \{< b >\} | i = 1, \dots, T\}$$

Si fattorizza la distribuzione a posteriori di interesse $p(C|W)$ in due componenti e si assume l'indipendenza condizionale:

$$p(C|X) = \sum_Z p(C|Z, X)p(Z|X) \cong \sum_Z p(C|Z)p(Z|X)$$

in questo caso i due fattori $p(C|Z)$, $p(Z|X)$ si chiamano rispettivamente letter model e acoustic model. L'indipendenza condizionale assunta in questo caso $p(C|Z, X) \cong p(C|Z)$ è invece ritenuta ragionevole.

### 2.3.1 Letter Model

La fattorizzazione ed una assunzione di indipendenza condizionale portano a:

$$p(C|Z) = \frac{p(Z|C)p(C)}{p(Z)} = \prod_{t=1}^{T} p(z_t|z_1, \dots, z_{t-1}, C)\frac{p(C)}{p(Z)} \cong \prod_{t=1}^{T} p(z_t|z_{t-1}, C)\frac{p(C)}{p(Z)}$$

dove nell'ordine $p(z_t|z_{t-1}, C)$, $p(C)$, $p(Z)$ sono rispettivamente: la probabilità di transizione di stato, il modello linguistico basato su lettere e la probabilità a priori.

Impiegando un finite state transducer da lettere a parole è possibile inglobare un modello linguistico basato su parole durante la decodifica.

La probabilità di transizione di stato $p(z_t|z_{t-1}, C)$ dipende dal tipo di transizione di stato:

$$p(z_t|z_{t-1}, C) \propto \begin{cases} 1 \; con \; z_t = c'_l \; e \; z_{t-1} = c'_l \; \forall l \\ 1 \; con \; z_t = c'_l \; e \; z_{t-1} = c'_{l-1} \; \forall l \\ 1 \; con \; z_t = c'_l \; e \; z_{t-1} = c'_{l-2} \; \forall l \; pari \\ 0 \; altrimenti \end{cases}$$

dove nel primo caso viene indicata una transizione sullo stesso stato, nel secondo caso la transizione coinvolge due stati differenti, invece nel terzo caso la transizione da $c'_{l-2}$ a $c'_l$ con $l$ pari indica una transizione da una lettera ad un'altra saltando un black symbol. La forma della probabilità di transizione di stato testimonia la necessità di un allineamento monotono, ovvero quando $z_{t-1} = c'_m$ allora $z_t = c'_l$ con $l \geq m$.

### 2.3.2 Acoustic Model

Analogamente al caso ibrido si segue una simile fattorizzazione ulteriore:

$$p(Z|X) = \prod_{t=1}^{T} p(z_t|z_1, \dots, z_{t-1}, X) \cong \prod_{t=1}^{T} p(z_t|X)$$

dove la distribuzione condizionata a posteriori per frame $p(z_t|X)$ condizionata a tutti gli ingressi può essere modellata con una rete ricorrente di tipo BLSTM con uscita adattata alla dimensione corretta $|U| + 1$ (la dimensione

dell'alfabeto più il blank symbol) da uno strato lineare con bias e convertita in distribuzione di probabilità con l'operazione Softmax:

$$h_t = BLSTM_t(X)$$

$$p(z_t|X) = Softmax\big(BiasedLinearLayer(X)\big)$$

### 2.3.3 Funzione Obiettivo

Grazie alla fattorizzazione delle due componenti, il problema assume la forma:

$$p(C|X) \cong p_{ctc}(C|X)\frac{p(C)}{p(Z)} = \sum\nolimits_Z \prod\nolimits_{t=1}^{T} p(z_t|z_{t-1}, C)\, p(z_t|X)\frac{p(C)}{p(Z)}$$

dove $p_{ctc}(C|X)$ è la funzione obiettivo:

$$p_{ctc}(C|X) \triangleq \sum\nolimits_Z \prod\nolimits_{t=1}^{T} p(z_t|z_{t-1}, C)\, p(z_t|X) \qquad (2)$$

Sebbene la sommatoria si estenda ad ogni possibile $Z$, la mole di calcolo viene drasticamente ridotta con algoritmi a programmazione dinamica grazie alle proprietà markoviane.

## 2.4 Soluzione con Meccanismo di Attenzione

Il sistema in esame è strutturato in tre parti, un encoder, un meccanismo di attenzione ed un decoder. L'encoder trasforma le feature di ingresso in un vettore nascosto, il meccanismo di attenzione mette in evidenza il contesto rilevante proveniente dal vettore nascosto prodotto dall'encoder. Il decoder stima uno alla volta i simboli della sequenza di uscita in modo autoregressivo sfruttando anche il contesto fornito dal meccanismo di attenzione.

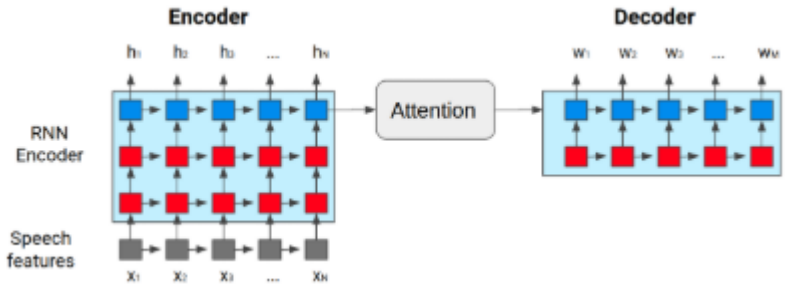



*Figura 3 Esempio di Architettura Encoder Decoder con Attenzione, immagine modificata da (Ravanelli, s.d.)*

Rispetto ai due approcci precedenti, il sistema basato su attenzione (Watanabe S., 2017) non impiega alcuna assunzione di indipendenza condizionale e stima direttamente la probabilità a posteriori $p(C|X)$. Il sistema non necessita una esplicita separazione nei vari moduli tipici dei metodi precedenti e risolve potenzialmente i cinque aspetti negativi illustrati per i sistemi HMM/DNN. A differenze delle altre due soluzioni che possiedono proprietà di alignment monotono, l'attenzione non rispetta questo vincolo inquanto è libera di fornire alignment irregolari.

$$\mathrm{p}(C|X) = p_{att}(C|X) = \prod_{l=1}^{T} p(c_l|c_1, \dots, c_{l-1}, X)$$

### 2.4.1 Encoder

L'encoder converte i vettori di features $X$ nei vettori nascosti $h_t$ con scansione frame e solitamente viene realizzato su strutture a base BLSTM, spesso sottocampionate nella dimensione del frame temporale per ridurre la complessità computazionale dell'encoder:

$$h_t = Encoder(X) = BLSTM_t(X)$$

### 2.4.2 Attention Mechanism

Il compito del meccanismo di attenzione è quello di determinare quale informazione di contesto è importante, così da somministrare anche il contesto in ingresso al decoder.

Il meccanismo di attenzione produce un vettore di coefficienti $a_{lt}$ chiamati attention weight che rappresenta il soft alignment (una forma di importanza relativa) al vettore nascosto $h_t$ per ogni uscita $c_l$. Il vettore di pesi di attenzione è chiamato alignment. Tutti i vettori nascosti $h_t$ partecipano quindi a una somma pesata con coefficienti di pesatura dati dall'alignment producendo un vettore nascosto chiamato glimpse $r_l$:

$$r_l = \sum_{t=1}^{T} a_{lt} h_t$$

Esistono diversi tipi di attenzione, fra di essi vale la pena di riportare la Location Aware Attention (J. Chorowski D. B., 2015). Questo tipo di attenzione prende in ingresso il vettore nascosto $h_t$ dell'encoder, il vettore nascosto del decoder al tempo di simbolo precedente $q_{l-1}$ ed una serie di feature convoluzionali estratte dall'alignment $a_{l-1}$ al tempo di simbolo precedente:

$$f_t = K * a_{l-1}$$

$$e_{lt} = g^T \tanh\big(LinearLayer(q_{l-1}) + LinearLayer(h_t) + BIasedLinearLayer(f_t)\big)$$

$$a_l = \mathrm{Softmax}(e_{lt})$$

dove $*$ indica la convoluzione monodimensionale nella dimensione del tempo di frame con i parametri convoluzionali $K$ tali da produrre l'insieme di $T$ features convoluzionali $\{f_t\}_{t=1}^{T}$.

### 2.4.3 Decoder

Il decoder è un sistema autoregressivo spesso realizzato con LSTM o GRU. Il decoder prende in ingresso il contesto di attenzione $r_l$, il vettore nascosto del decoder $q_{l-1}$ al tempo di simbolo precedente ed il simbolo precedentemente emesso $c_{l-1}$, che può essere fornito con codifica one-hot vector o attraverso embedding. L'uscita del decoder è quindi dimensionalmente adattata alla dimensione del vocabolario con uno strato lineare con bias ed è trasformata nella distribuzione voluta con l'operazione softmax:

$$q_l = LSTM_l(r_l, q_{l-1}, c_{l-1})$$

$$Decoder(\dots) = Softmax\left(BiasedLinearLayer\left(LSTM_l(\dots)\right)\right)$$

### 2.4.4 Funzione Costo e Decodifica

La funzione costo del modello di attenzione è calcolata approssimativamente con la probabilità a posteriori sulla sequenza $p_{att}(C|X)$ dove ai tempi di simbolo passati sono forniti i simboli di ground truth $c^*$ invece che quelli predetti dalla rete:

$$p_{att}(C|X) \cong \prod_{l=1}^{L} p(c_l|c_1^*, \dots, c_{l-1}^*, X) \triangleq p_{att}^*(C|X) \quad (3)$$

Come indicato in (W. Chan, 2016) l'uso dei simboli di ground truth è una assunzione forte che allontana il sistema ad apprendere al livello della sequenza e lo focalizza su una classificazione puntuale con i presupposti di uno storico di predizione perfetto. L'effetto di questa assunzione è la creazione di un mismatch tra le condizioni di training ed inferenza dove lo storico di simboli condizionanti la stima del nuovo simbolo sarà obbligatoriamente quello predetto dalla rete. Questa incongruenza permette ad errori saltuari di predizione del simbolo di compromettere facilmente la stima dei successivi. Il problema può essere mitigato scegliendo di fornire con una certa frequenza, l'ultimo simbolo realmente predetto dalla rete invece che il ground truth, nella decodifica di training. La decodifica può essere condotta con un algoritmo di tipo Beam Search.

## 2.5 Soluzione Ibrida CTC/Attention

Questo tipo di soluzione sfrutta i benefici della CTC e del meccanismo di attenzione sia nella fase di training che in quella di decodifica (Watanabe S., 2017). L'anatomia del sistema è ancora quella di tipo encoder decoder con attenzione.

### 2.5.1 Multiobjective Learning

Con il Multiobjective Learning (MOL in breve) (S. Kim, 2017) la funzione costo nell'apprendimento si basa sul criterio di attenzione ed include anche il task di tipo CTC.

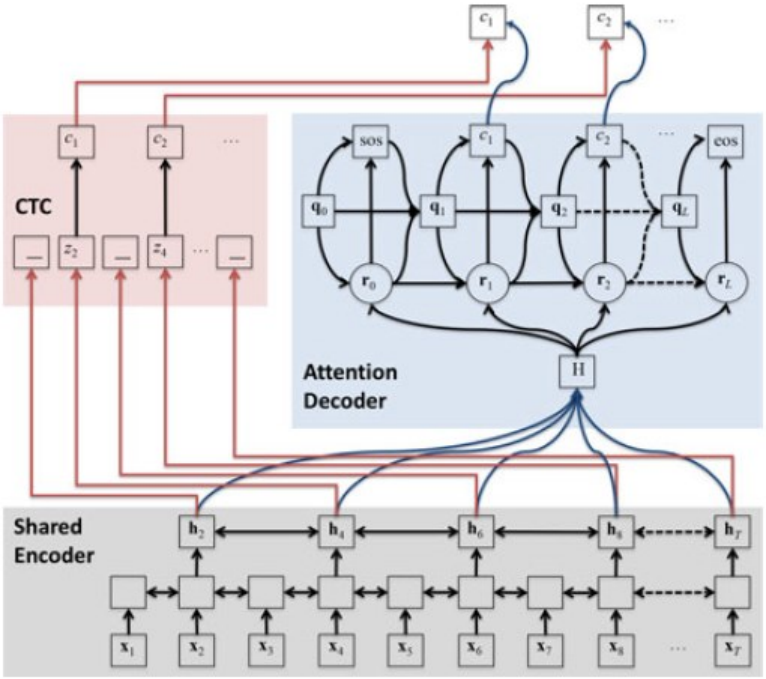



*Figura 4 Esempio Architettura Hybrid CTC/Attention, immagine da (Watanabe S., 2017)*

Il modello encoder è in questo caso condiviso sia dalla componente CTC cha dal meccanismo di attenzione. A differenza del modello con sola attenzione, l'algoritmo forward-backward può imporre un alignment monotono tra speech e sequenze di uscita durante il training. Ovvero, invece che stimare l'alignment desiderato con il solo metodo data-driven dell'attenzione, si ha un aiuto ad accelerare la stima grazie all'algoritmo forward-backward CTC. L'obiettivo da massimizzare è la combinazione lineare delle probabilità logaritmiche di attenzione e CTC dalle equazioni (2-3):

$$\mathcal{L}_{MOL} = \lambda \log p_{ctc}(C|X) + (1-\lambda) \log p^{*}_{att}(C|X)$$

dove $\lambda$ è un parametro ottimizzabile ed è limitato a $0 \leq \lambda \leq 1$. Con questa impostazione il criterio di attenzione è approssimato lettera per lettera mentre il criterio CTC opera al livello di sequenza; quindi, la soluzione MOL può

alleviare il problema della decodifica approssimata menzionata precedentemente, nel sistema basato esclusivamente su attenzione.

### 2.5.2 Decodifica

Il processo di inferenza, come nel caso della sola attenzione, avviene attraverso Beam Search, ma viene tenuto conto delle probabilità CTC quando vengono cercate le ipotesi con alignment migliore allo speech in ingresso. Per illustrare la decodifica congiunta è utile prima introdurre la decodifica basata su attenzione ed alcune tecniche convenzionali che mirano ad alleviarne i problemi di alignment.

#### *2.5.2.1 Decodifica basata su Attenzione con Beam Search*

La decodifica mira a determinare la sequenza di lettere $\hat{C}$ dato un segnale di speech in ingresso $X$ secondo questa relazione:

$$\hat{C} = arg \max_{C \in U^*} \log p(C|X)$$

dove $p(C|X)$ viene calcolata come mostrato precedentemente come nell'equazione (3) mentre $\hat{C}$ viene calcolata con la tecnica Beam Search (Watanabe S., 2017). In breve il Beam Search è una tecnica che genera sequenze di uscita dette ipotesi parziali da sinistra a destra aggiungendo un carattere alla volta, ma mantenendo nel processo solo un numero limitato di sequenze migliori, secondo lo score prodotto da una certa metrica.

Ogni ipotesi inizia con un simbolo speciale detto start of sentence $< sos >$ e per essere considerata correttamente terminata deve concludersi con il carattere speciale end of sentence $< eos >$. Se $L_{max}$ è la massima lunghezza di ipotesi da cercare, posto $\Omega_l$ l'insieme di ipotesi parziali di lunghezza $l$, all'inizio del Beam Search $\Omega_0$ contiene solo $< sos >$. Per $l$ che va da 1 a $L_{max}$ ogni ipotesi parziale in $\Omega_{l-1}$ è estesa accodando alla fine di essa ogni possibile lettera $c$ (incluso $< eos >$), le nuove ipotesi sono quindi raccolte in $\Omega_l$. Lo score $\alpha$ di ogni nuova ipotesi $h$ in $\Omega_l$ è calcolato nel dominio logaritmico come:

$$\alpha(h, X) = \alpha(g, X) + \log p(c|g_{l-1}, X) \qquad (4)$$

dove $g$ è un ipotesi parziale in $\Omega_{l-1}$ ed $h = g \cdot c$ . Se $c$ è il carattere di terminazione $< eos >$, allora $h$ è aggiunto all'insieme di ipotesi complete $\widehat{\Omega}$ e non $\Omega_l$. Alla fine la decodifica culmina nel calcolo molto più elementare nello spazio $\widehat{\Omega}$:

$$\hat{C} = arg \max_{h \in \widehat{\Omega}} \alpha(h, X)$$

e per migliorare l'efficienza di ricerca, $\Omega_l$ mantiene un numero limitato di ipotesi, quelle con score maggiore.

I sistemi ASR basati su attenzione possono essere inclini al compiere errori di inserzione e cancellazione a causa delle proprietà di alignment estremamente flessibili. A causa di questa flessibilità l'attenzione è in grado di attenzionare una porzione qualsiasi del vettore nascosto di encoder per la predizione del simbolo successivo. Nella decodifica il carattere di terminazione $< eos >$ può essere predetto prematuramente portando alla generazione di ipotesi eccessivamente brevi, d'altra parte il decoder può predire il prossimo carattere con grande probabilità attendendo sempre la stessa porzione del vettore nascosto di encoder, producendo quindi lunghe sequenze di ripetizioni dello stesso carattere in uscita.

#### *2.5.2.2 Tecniche convenzionali di decodifica*

Un modo comune (J. Chorowski D. B., 2015) per alleviare il problema dell'alignment sta nel modificare la funzione obiettivo di decodifica facendo in modo che includa un termine addizionale proporzionale alla lunghezza della sequenza di uscita $|C|$ detto length penalty in ragione di un fattore $\gamma$:

$$\hat{C} = arg \max_{C \in U^*} \log\{p(C|X) + \gamma\, |C|\}$$

Il parametro $\gamma$ è ottimizzabile ma non è sufficiente per impedire che vangano selezionate ipotesi eccessivamente corte o eccessivamente lunghe. Un altro approccio efficace è quello di stabilire la lunghezza minima e massima delle ipotesi in relazione alla lunghezza delle speech features in ingresso. Tuttavia, essendo in generale presenti trascrizioni estremamente lunghe o brevi, relativamente alla lunghezza delle speech feature in ingresso, è difficile

trovare il giusto compromesso tra il gestire correttamente casi estremi e prevenire ipotesi con lunghezze irrilevanti.

Un'altra modalità di mitigazione del problema dell'alignment prevede l'aggiunta di un termine ulteriore nella funzione obiettivo di decodifica basato sul termine di coverage (J. Chorowski N. J., 2017) in ragione di un coefficiente proporzionale $\eta$:

$$coverage(C|X) = \sum_{t=1}^{T} Bool2Int\left(\sum_{l=1}^{L} a_{lt} > \tau\right)$$

$$\hat{C} = arg\max_{C \in U^*} \log\{p(C|X) + \gamma\,|C| + \eta \cdot coverage(C|X)\}$$

dove i tre parametri $\gamma, \eta, \tau$ sono ottimizzabili e la funzione $Bool2Int(\cdot)$ mappa una informazione booleana $True/False$ rispettivamene in $1/0$. Il termine di coverage conta sostanzialmente il numero di frame temporali che hanno ricevuto una attenzione cumulativa superiore a una soglia $\tau$ e per come è definito esso favorisce la metrica di un ipotesi quando questa attenziona una parte del vettore nascosto di encoder per la prima volta e non la favorisce ulteriormente quando la stessa parte del vettore è ulteriormente attenzionata. Questo contributo è efficace nell'evitare decodifiche cicliche della stessa sequenza di simboli all'interno di un ipotesi.

Questi due tipi di intervento non sono però in grado di prevenire la prematura predizione del simbolo di terminazione $< eos >$, creando ipotesi eccessivamente brevi e conseguentemente un numero elevato di errori di cancellazione.

Inoltre è difficile stimare dei valori per la terna di parametri $\gamma, \eta, \tau$ che siamo validi per qualsiasi dato di speech da task differenti.

#### *2.5.2.3 Joint Decoding*

Nella decodifica congiunta CTC con Attenzione le probabilità derivanti dai due sistemi $p_{ctc}(C|X), p_{att}(C|X)$ e definite nelle equazioni (2-3), compartecipano sia nel training sia nella fase di inferenza (Watanabe S., 2017).

La funzione obiettivo in fase di decodifica miscela i due contributi in ragione di un coefficiente ottimizzabile $\lambda$:

$$\hat{C} = arg \max_{C \in U^*} \{\lambda \log p_{ctc}(C|X) + (1-\lambda) \log p_{att}(C|X)\}$$

La probabilità CTC impone un alignment monotono che non permette grandi salti o percorsi ciclici sugli stessi frames come illustrato prima. Inoltre può impedire la predizione prematura della terminazione $< eos >$ , di conseguenza è possibile selezionare ipotesi con alignment migliore ed escludere ipotesi irrilevanti senza dover ricorrere a length penalty, coverage o vincoli di lunghezza massima o minima relativi.

Durante la ricerca con Beam Search il decoder calcola lo score per ogni ipotesi parziale, ma essendo il contributo di score CTC calcolato al frame ed il contributo di attenzione sincrono al simbolo di uscita, la combinazione dei due contributi non è banale e si può ricorrere a due metodi: rescoring e one-pass decoding.

#### 2.5.2.3.1 Rescoring

Questo è un metodo composto da due fasi. Nella prima fase viene eseguito Beam Search impiegando il solo contributo di attenzione. Nella seconda fase l'insieme delle ipotesi complete subisce un rescoring usando entrambi i contributi ma con probabilità CTC calcolate con l'algoritmo forward per CTC (A. Graves, 2006). Quindi nella fase di rescoring la decodifica è portata a termine massimizzando la seguente funzione obiettivo:

$$\hat{C} = arg \max_{h \in \widehat{\Omega}} \{\lambda\, \alpha_{ctc}(h, X) + (1-\lambda)\alpha_{att}(h, X)\}$$

$$\alpha_{ctc}(h, X) \triangleq \log p_{ctc}(h|X) \tag{5}$$

$$\alpha_{att}(h, X) \triangleq \log p_{att}(h|X) \tag{6}$$

#### 2.5.2.3.2 One-Pass Decoding

Con questo metodo la probabilità di ogni ipotesi parziale è calcolata con entrambi i contributi e per la CTC viene usato il CTC prefix probability (Graves,

NLTK: The Neural Language Toolkit, 2006) definita come la probabilità cumulativa di tutte le sequenze di simboli che hanno come prefisso $h$:

$$p_{ctc}(h\ldots|X) = \sum_{v\in(U\cup\{<eos>\})^+} p_{ctc}(h\cdot v|X)$$

dove lo score CTC è definito come:

$$\alpha_{ctc}(h,X) \triangleq \log p_{ctc}(h\ldots|X)$$

dove $v$ rappresenta tutte le possibili sequenze di simboli tranne la stringa vuota. Lo score CTC non è calcolabile in modo ricorsivo come in equazione (4) ma può essere calcolato in modo efficiente mantenendo le probabilità in avanti sui frame temporali di ingresso per ogni ipotesi parziale. Questo contributo è poi combinato con quello di attenzione $\alpha_{att}(h,X)$ di equzione (6) per mezzo di $\lambda$.

L'algoritmo di decodifica Beam Search per il metodo One-Pass Decoding è illustrato nella figura seguente.

**procedure** ONEPASSBEAMSEARCH ($X$, $L_{\max}$)
  $\Omega_0 \leftarrow \{<\texttt{sos}>\}$
  $\hat{\Omega} \leftarrow \emptyset$
  **for** $l = 1 \ldots L_{\max}$ **do**
    $\Omega_l \leftarrow \emptyset$
    **while** $\Omega_{l-1} \neq \emptyset$ **do**
      $g \leftarrow$ HEAD($\Omega_{l-1}$)
      DEQUEUE($\Omega_{l-1}$)
      **for each** $c \in \mathcal{U} \cup \{<\texttt{eos}>\}$ **do**
        $h \leftarrow g \cdot c$
        $\alpha(h) \leftarrow \lambda\alpha_{\text{ctc}}(h,X) + (1-\lambda)\alpha_{\text{att}}(h,X)$
        **if** $c =<\texttt{eos}>$ **then**
          ENQUEUE($\hat{\Omega}, h$)
        **else**
          ENQUEUE($\Omega_l, h$)
          **if** $|\Omega_l| > beamWidth$ **then**
            REMOVEWORST($\Omega_l$)
          **end if**
        **end if**
      **end for**
    **end while**
    **if** ENDDETECT($\hat{\Omega}, l$) = `true` **then**
      break ▷ exit for loop
    **end if**
  **end for**
  **return** $\arg\max_{C\in\hat{\Omega}} \alpha(C)$
**end procedure**

*Figura 5 Pseudocodice algoritmo di decodifica CTC/Attenzione one pass decoding, fonte (Watanabe S., 2017)*

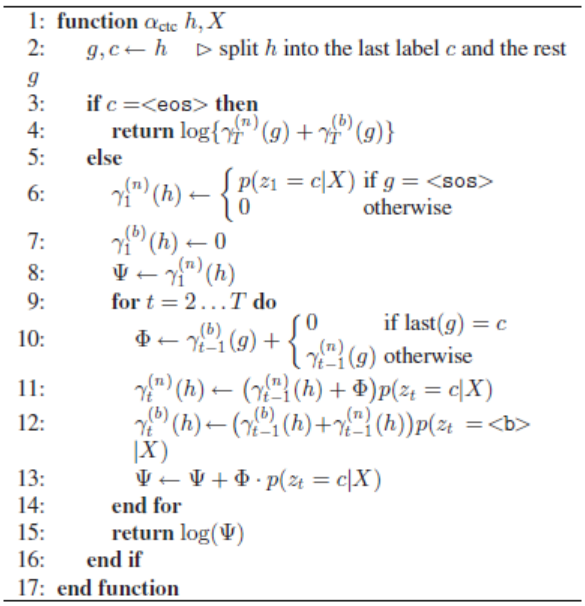
Nell'algoritmo gli insiemi $\Omega_l$, $\widehat{\Omega}$ sono realizzati con strutture di tipo Queue nelle righe $2, 3$ ed accettano rispettivamente le ipotesi parziali di lunghezza $l$ e le ipotesi complete. Ogni ipotesi parziale $g$ in $\Omega_{l-1}$ è estesa con ogni possibile etichetta $c$ nell'insieme di etichette $U$ nelle righe da $4$ a $25$. Ogni ipotesi estesa $h$ subisce scoring nella riga $11$. Appena dopo lo scoring se viene rilevata la terminazione $< eos >$ l'ipotesi è assunta completa ed inserita in $\widehat{\Omega}$ nella riga $13$, altrimenti è inserita in $\Omega_l$ nella riga $15$ eventualmente eliminando l'ipotesi peggiore in $\Omega_l$ nella riga $17$.

La ricerca può essere resa più efficiente attuando un meccanismo di terminazione anticipata quando ci sono scarse possibilità di trovare una ipotesi completa con score maggiore all'aumentare di $l$ in futuro. Questo meccanismo può essere realizzato con la funzione $EndDetect\left(\widehat{\Omega}, l\right)$ nella riga $22$ che ritorna $True$ quando:

$$\sum_{m=0}^{M-1} Bool2Int\left(\left\{\max_{h\epsilon\,\widehat{\Omega}:|h|=l-m} \alpha(h,X) - \max_{h'\epsilon\,\widehat{\Omega}} \alpha(h',X)\right\} < D_{end}\right) = M$$

dove $D_{end}$, $M$ sono soglie predeterminate. Questa relazione ritorna $True$ se gli score di ipotesi completate di recente sono sufficientemente piccole rispetto allo score migliore di tutte le ipotesi completate fino al presente nella procedura di decodifica. Nella sommatoria il primo massimo corrisponde allo score migliore nelle ipotesi completate di recente di lunghezza $l-m$. Il secondo massimo è invece lo score migliore tra tutte le ipotesi complete in $\widehat{\Omega}$. Se la differenza tra i due massimi è inferiore alla soglia $D_{end}$, la funzione $Bool2Int(\cdot)$ ritornerà un $1$ a contribuire alla sommatoria esterna, altrimenti uno $0$. Quindi la sommatoria diviene pari ad $M$ attivando la terminazione prematura quando tutte le differenze sono inferiori alla soglia.

Nella riga $11$ sono calcolati gli score di Attenzione e CTC per ogni ipotesi parziale. Lo score di Attenzione è calcolato con l'equazione (4) mentre lo score CTC richiede una versione modificata di algoritmo forward sincrona al simbolo. L'algoritmo è illustrato nella figura successiva:



1: **function** $\alpha_{\text{ctc}}\ h, X$
2: $g, c \leftarrow h$ ▷ split $h$ into the last label $c$ and the rest $g$
3: **if** $c =$<eos> **then**
4: **return** $\log\{\gamma_T^{(n)}(g) + \gamma_T^{(b)}(g)\}$
5: **else**
6: $\gamma_1^{(n)}(h) \leftarrow \begin{cases} p(z_1 = c|X) & \text{if } g = \text{<sos>} \\ 0 & \text{otherwise} \end{cases}$
7: $\gamma_1^{(b)}(h) \leftarrow 0$
8: $\Psi \leftarrow \gamma_1^{(n)}(h)$
9: **for** $t = 2 \ldots T$ **do**
10: $\Phi \leftarrow \gamma_{t-1}^{(b)}(g) + \begin{cases} 0 & \text{if last}(g) = c \\ \gamma_{t-1}^{(n)}(g) & \text{otherwise} \end{cases}$
11: $\gamma_t^{(n)}(h) \leftarrow (\gamma_{t-1}^{(n)}(h) + \Phi)p(z_t = c|X)$
12: $\gamma_t^{(b)}(h) \leftarrow (\gamma_{t-1}^{(b)}(h) + \gamma_{t-1}^{(n)}(h))p(z_t =$ <b> $|X)$
13: $\Psi \leftarrow \Psi + \Phi \cdot p(z_t = c|X)$
14: **end for**
15: **return** $\log(\Psi)$
16: **end if**
17: **end function**

*Figura 6 Pseudocodice per lo scoring CTC sincrono al simbolo, fonte (Watanabe S., 2017)*

Poste $\gamma_t^{(n)}(h)$ , $\gamma_t^{(b)}(h)$ le probabilità forward delle dell'ipotesi $h$ ai frame temporali $1,..,t$ dove gli apici $(n)$, $(b)$ denotano rispettivamente che tutti i percorsi CTC terminano con simbolo non blank oppure blank $< b >$. Prima di iniziare il Beam Search le due gamma sono inizializzate per ogni $t = 1, \ldots, T$ come:

$$\gamma_t^{(n)}(< sos >) = 0$$

$$\gamma_t^{(b)}(< sos >) = \prod_{\tau=1}^{t} \gamma_{\tau-1}^{(b)}(< sos >) \cdot p(z_\tau =< b > |X)$$

$$\gamma_0^{(b)}(< sos >) = 1$$

È importante notare che il successivo indice temporale $t$ e la lunghezza di ingresso $T$ possono differire da quelli dell'utterance in ingresso $X$ a causa del subsampling temporale nell'encoder.

Nell'algoritmo l'ipotesi $h$ è separata nell'ultima etichetta $c$ ed il testo $g$ in riga 2, se $c$ è la terminazione $< eos >$, viene ritornato il logaritmo della probabilità forward assumendo che l'ipotesi $h$ sia completa in riga $4$. La probabilità in avanti di $h$ è data da:

$$p_{ctc}(h|X) = \gamma_T^{(n)}(g) + \gamma_T^{(b)}(g)$$

Se $c$ non è la terminazione $< eos >$ vengono calcolate le probabilità in avanti $\gamma_t^{(n)}(h), \gamma_t^{(b)}(h)$ e la probabilità di prefisso $\psi = p_{ctc}(h\ldots|X)$ assumendo che $h$ non sia una ipotesi completa. L'inizializzazione e ricorsione di queste probabilità sono descritte nelle righe $6 - 14$. Nella funzione si assume che ogni volta che vengono calcolate le probabilità $\gamma_t^{(n)}(h)$, $\gamma_t^{(b)}(h)$ e $\psi$, le probabilità forward $\gamma_t^{(n)}(g), \gamma_t^{(b)}(g)$ siano già state ottenute con Beam Search poiché $g$ è prefisso di $h$ così che si abbia $|g| < |h|$. Ne consegue che il prefisso e le probabilità forward possono essere calcolate in modo efficiente per ogni ipotesi ed ipotesi parziali con alignment irrilevanti possono essere esclusi dallo score CTC durante il Beam Search.

Quindi il metodo di decodifica congiunto One-Pass Decoding può ridurre il numero di errori di ricerca richiedendo un minore costo computazionale rispetto al metodo di decodifica congiunta con Rescoring.

## 2.6 SOLUZIONE TRANSFORMERS

I Transformers, nati nel contesto del Neural Machine Translation in (A. Vaswani, 2017), hanno attirato molta attenzione nel campo dello Speech Processing e del Natural Language Processing permettendo di ottenere prestazioni notevoli su un ampio spettro di applicazioni, incluso l'Automatic Speech Recognition (L. Dong, 2018), (Q. Song, 2022).

### 2.6.1 Panoramica

Questo tipo di modelli supera le reti ricorrenti tradizionali, che in applicazioni sequence to sequence hanno problemi a gestire sequenze lunghe e sono affette dai fenomeni di esplosione ed evanescenza del gradiente (S. Karita, 2019). Uno dei vantaggi dei Transformers nel campo dello speech sta nel fatto che, mentre le reti ricorrenti osservano solo una piccola parte delle frasi processate, i Transformers la osservano tutta insieme grazie alla loro tipica auto-attenzione (A. Vaswani, 2017). Inoltre il meccanismo di attenzione Multihead (E. Voita, 2019) permette una parallelizzazione efficiente nella fase

di training, rendendo i Transformers ideali per la gestione di grandi datasets, altrimenti meno semplici da gestire nei vari task di Speech Processing. La combinazione di auto-attenzione ed attenzione Multihead rende questi sistemi molto potenti in applicazioni sequence-to-sequence. La natura non intrinsecamente sequenziale dei Transformer porta alla necessità di reintrodurre l'informazione di posizione nelle feature processate; quindi, è incluso un positional encoding in ingresso all'architettura per infondere l'informazione di posizione assoluta o relativa dei tokens nella sequenza.

Per illustrare brevemente le componenti salienti dell'anatomia dei Transformer, viene presa a riferimento la struttura proposta in (A. Vaswani, 2017) mostrata in figura:

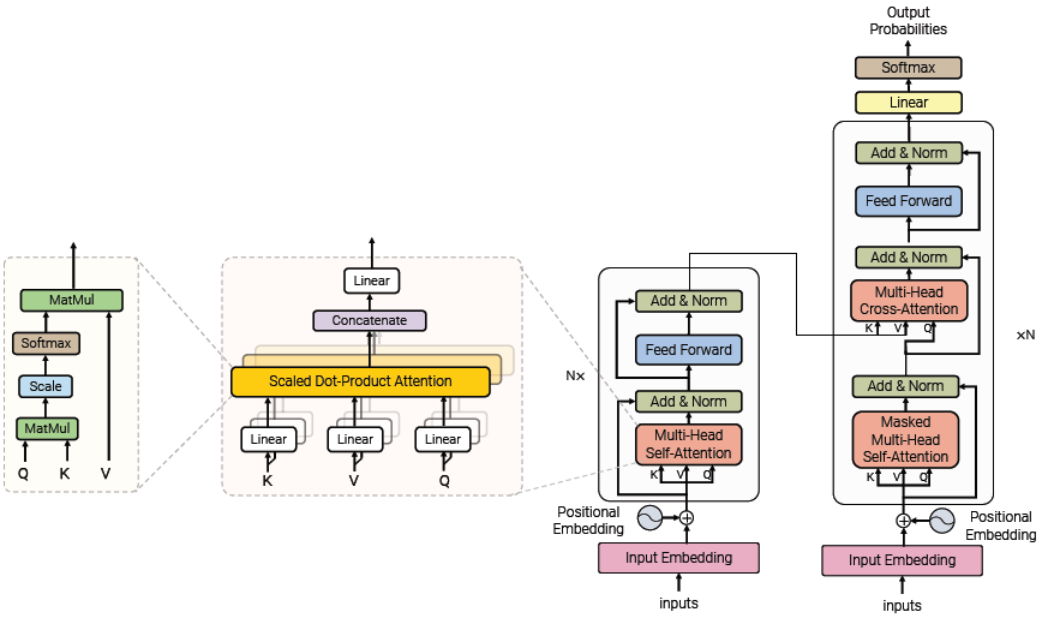



*Figura 7 Esempio Architettura Transformer, fonte (A. Vaswani, 2017)*

La struttura mostrata è di tipo encoder-decoder, entrambi costruiti con una cascata di $N$ blocchi elementari, ognuno con i propri presi apprendibili. Nei blocchi dell'encoder l'auto-attenzione osserva la sequenza di feature in ingresso e ne estrae le informazioni salienti. Il decoder in forma autoregressiva riceve in feedback la sequenza di uscita prodotta fino ad ora e ritardata di un tempo di simbolo poiché inizia con il simbolo di inizio sequenza $< sos >$. Il decoder attenziona questa sequenza con la Masked Multihead

Self-Attention per poi mandare le informazioni appena prodotte insieme alla sequenza di uscita dell'encoder, verso un ulteriore meccanismo di auto-attenzione, dello stesso tipo del primo, ma con ingressi differenti.

### 2.6.2 Self-Attention

Questo sistema mira a cogliere le correlazioni interne a una sequenza di ingresso aggregando informazioni globali da tutta la stessa. Una sequenza di features $X \in \mathbb{R}^{NxD}$ di lunghezza $N$ e dimensione del vettore di feature $D$ viene trasformata nelle matrici di query $Q \in \mathbb{R}^{NxD_k}$, key $K \in \mathbb{R}^{NxD_k}$ e value $V \in \mathbb{R}^{NxD_k}$ con delle trasformazioni lineari:

$$Q = XW^Q$$

$$K = XW^K$$

$$V = XW^V$$

Il prodotto interno tra $Q$ e $K$ è normalizzato scalarmente rispetto a $\sqrt{D_k}$ per permettere al gradiente di operare in una regione meno satura del $Softmax$. Segue l'operazione di $Softmax$ che crea la matrice di attenzione $A \in \mathbb{R}^{NxN}$:

$$A = Softmax\left(\frac{QK^T}{\sqrt{D_k}}\right)$$

ed il risultato dell'auto-attenzione è il vettore $Z$ creato attenzionando la matrice dei value:

$$Z = AV$$

### 2.6.3 Masked Self-Attention

Questo tipo di auto-attenzione è tipica del decoder. La sequenza attenzionata è quella di uscita e per mantenere la proprietà autoregressiva del decoder, unitamente alla traslazione in avanti della sequenza di uscita per mezzo del token di inizio sequenza $< sos >$, viene impedito al sistema di attenzionare informazioni future applicando una maschera $M \in \mathbb{R}^{NxN}$:

$$Z = Softmax\left(\frac{QK^T}{\sqrt{D_k}} \circ M\right)V$$

dove l'operatore $\circ$ indica l prodotto elemento per elemento ed $M$ è una matrice triangolare superiore.

### 2.6.4 Multihead Attention

Invece che calcolare l'attenzione una cola volta, la Multihead Self-Attention (MHSA in breve) sfrutta in modo parallelo diverse istanze di auto-attenzione. Disponendo di $h$ teste di attenzione, ogni auto-attenzione dispone dei propri pesi apprendibili $W^{Q_i}$ , $W^{K_i}$ , $W^{V_i}$ con $i = 0, \ldots, h-1$ usati per proiettare l'ingresso di ogni auto-attenzione su un proprio sottospazio apprendibile ed indipendente:

$$Z_i = Softmax\left(\frac{QW^{Q_i}(KW^{K_i})^T}{\sqrt{D_k/h}}\right)VW^{V_i}$$

quindi i contributi di attenzione delle varie teste sono concatenati e trasformati linearmente alle dimensioni desiderate attraverso la matrice apprendibile $W^O = \mathbb{R}^{hD_k xN}$

$$MHSA(Q, K, V) = [Z_0, Z_1, \ldots, Z_{h-1}]W^O$$

### 2.6.5 Positional Encoding

Questo componente permette di fornire informazione sull'ordinamento delle sequenze ai meccanismi di attenzione, che altrimenti non avrebbero modo di distinguere quale parte di una sequenza viene prima di un'altra. Questo avviene associando ad ogni posizione nella sequenza di ingresso un vettore apprendibile oppure contenente l'informazione posizionale in forma assoluta o relativa.

## 2.7 TARGET SPEAKER ASR

Ipotizzando uno scenario acustico a singolo canale non affetto da rumore e da riverberazione con speakers multipli liberi di parlare in un momento qualsiasi anche sovrapponendosi, i metodi di Automatic Speech Recognition, progettati per gestire un singolo speaker, fallirebbero nel gestire i segmenti

audio a speaker sovrapposti. Data la somiglianza tra i vari segnali, questi sistemi non avrebbero infatti informazioni sufficienti per essere in grado di separare i segnali prodotti dai vari speaker nei domini di rappresentazione noti.

### 2.7.1 Contesto Metodologico

Tre sono gli approcci tipici al problema del riconoscimento a speaker sovrapposti:

- **Blind Source Separation:** (BSS in breve) consiste in una prima fase nel separare le componenti individuali da un segnale mistura[2] nel dominio del tempo (Y. Luo, 2019) (Subakan, 2021) per poi passarle in una seconda fase ad un modello ASR a singolo speaker per la trascrizione. Siccome la fase di separazione non è ottimizzata sotto il criterio ASR, le prestazioni possono essere sub-ottime.
- **Multi-Speaker ASR:** (MS-ASR in breve) sia nei metodi (Chang, 2019) (Sklyar, 2021) (Kanda N. G., 2020) (Guo, 2021) che nelle varianti Speaker Attributed ASR [3](SA-ASR in breve) (Kanda N. Y., 2021) (Kanda N. W., 2022) producono in uscita delle trascrizioni e sono ottimizzate end-to-end con criterio ASR.
- **Target Speaker ASR:** (TS-ASR in breve) mira a trascrivere il segnale vocale di uno specifico speaker detto target speaker disponendo di un suo profilo, spesso ottenibile a partire da una frase ausiliaria, da esso stesso pronunciata.

Una caratteristica comune dei modelli BSS e degli analoghi MS-ASR è la presenza di più rami di uscita, uno per ogni sorgente. I metodi SA-ASR richiedono i profili di tutti gli speaker nella mistura come informazione ausiliaria. Sia i metodi BSS che gli analoghi MS-ASR invece non richiedono alcuna informazione ausiliaria e fra i maggiori fattori limitanti si ha che il

[2] Mistura: indica un segnale di vocale contenente il parlato sovrapposto di più di uno speaker.
[3] Speaker-Attributed ASR: in contesto multi-speaker sovrapposti è un metodo ASR che oltre ad eseguire la trascrizione, attribuisce ogni parte della trascrizione lo speaker a cui è associata.

numero di uscite (quindi di speaker sovrapposti) è fissato a priori e per l'inferenza su lunghe sequenze sono richiesti Permutation Invariant Training (PIT in breve) (Kolbæk, 2017) e Speaker Tracing (Žmolíková, 2019). Inoltre una incongruenza tra il numero di speaker tra le fasi training ed inferenza può ridurne drasticamente le prestazioni. Nel caso di MS-ASR il Serialized Output Training (SOT in breve) può superare alcune di queste limitazioni ma lascia desiderare in termini di prestazioni (Kanda N. G., 2020). In combinazione con SOT, SA-ASR usa informazioni sul profilo degli speaker per migliorare le prestazioni e non essere limitato dal numero fisso di uscite. Tuttavia i metodi di questo tipo assumono la disponibilità dei profili di tutti gli speaker partecipanti alla mistura, questi metodi sono non di meno molto adatti alla trascrizione di conferenze.

Diversamente il metodo TS-ASR (Žmolíková, 2019) (Moriya, 2022) richiede soltanto il profilo del target speaker; quindi, è adatto a quelle situazioni in cui occorre trascrivere un solo speaker di interesse ignorando gli altri. Per come è costruito, questo metodo non necessita di SOT e nemmeno di Speaker Tracing. Tuttavia, nel caso nell'eventualità di trascrizione di speaker multipli e disponendo dei profili necessari, TS-ASR necessiterebbe di un processo di inferenza per ogni speaker.

### 2.7.2 Formulazione TS-ASR

In uno scenario acustico a singolo canale con $N$ speaker differenti ed in generale sovrapposti, se un i-esimo speaker produce il segnale vocale $x_i(n)$ (una sequenza, un segnale nel tempo campionato) ed indicando il target speaker con il pedice $s$, il segnale mistura $y(n)$ è modellabile in questo modo:

$$y[n] = \sum_{i=1}^{N} x_i[n] = x_s[n] + \sum_{i=1, i\neq s}^{N} x_i[n]$$

Una soluzione tipica di questo tipo di problema si presenta di un due fasi: nella prima fase viene stimata una maschera in un dato spazio di rappresentazione, nella seconda fase, questa maschera viene applicata al segnale mistura per

estrarre solo il segnale vocale del target speaker ed il risultato viene elaborato con un modello ASR classico a singolo speaker. In sostanza la soluzione di questo problema equivale a risolverne due: un problema di estrazione del target speaker ed un problema di riconoscimento a singolo speaker.

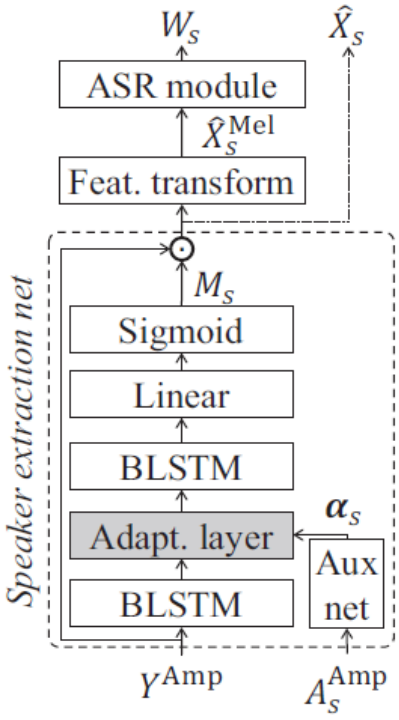



*Figura 8 Esempio di architettura con estrazione target speaker nel dominio tempo frequenza e riconoscimento, fonte (Delcroix M. W., 2019)*

Uno spazio di rappresentazione ampiamente usato (Delcroix, 2018) (Wang, 2018) è il dominio Tempo-Frequenza (TF in breve) nella forma di speech features di tipo log-Mel spectrogram. Lo spettrogramma di tipo log-Mel è ottenuto a partire dal segnale nel dominio del tempo applicando prima una trasformata di Fourier a tempo discreto (DTFT in breve), poi il segnale è elaborato con il banco filtri di Mel. In seguito viene calcolata l'energia del segnale in uscita ad ogni banda passante del banco filtri e ne viene calcolato il logaritmo. Il risultato è una rappresentazione compatta del segnale di partenza nel dominio TF. Il problema riformulato nel nuovo dominio assume la forma seguente:

$$Y(t,f) = X_s(t,f) + \sum_{i=1, i\neq s}^{N} X_i(t,f)$$

Se $M_s(t,f)$ è la maschera di estrazione TF per il target speaker $s$, se $\circ$ indica il prodotto elemento per elemento, allora la stima dello spettrogramma di log-Mel per il target speaker $\widehat{X_s}(t,f)$ si ottiene come:

$$\widehat{X_s}(t,f) = Y(t,f) \circ M_s(t,f)$$

Senza una opportuna guida, il meccanismo di estrazione non è capace di distinguere il target speaker dal resto (Delcroix, 2018). Quindi viene sfruttata una informazione ausiliaria fornita da una registrazione del segnale vocale del solo target speaker. Questa registrazione viene detta frase ausiliaria e viene processata per estrarre da essa un profilo del target speaker, ovvero una sua rappresentazione numerica significativa in uno spazio di rappresentazione. Un esempio di profilo del target speaker è un target speaker embedding.

Il profilo del target speaker va fornito al modello TS-ASR nel modo corretto, l'iniezione di questa informazione nel modello e la sua combinazione con le feature della mistura in ingresso allo stesso viene chiamata fusione. Specialmente quando lo spazio di rappresentazione del profilo e delle feature della mistura sono differenti, il metodo di fusione va selezionato in modo appropriato (Žmolíková, 2019) (Wu, 2019) (Ephrat, 2018).

Nella fase di apprendimento di sistemi TS-ASR può essere utile eseguire pretraining di estrazione per il modulo di estrazione, seguito da fine tuning end-to-end con il sistema completo sul criterio ASR.

## 2.8 STRUMENTI SOFTWARE

Nello sviluppo della parte sperimentale di questa tesi, sono stati necessari vari strumenti software, tra cui un linguaggio di programmazione versatile, un ambiente di sviluppo per il codice e librerie specifiche per lo sviluppo di sistemi Deep Learning / ASR.

### 2.8.1 Linguaggio di Programmazione

Il linguaggio di programmazione scelto è Python nella versione $3.10$, un linguaggio di alto livello orientato ad oggetti di tipo general purpose, scelto

considerando la sua diffusione, gratuità, semplicità di programmazione e sviluppo grazie alla sua somiglianza al forme di pseudocodice, la disponibilità di innumerevoli pacchetti dedicati a funzionalità specifiche, inclusi Machine Learning e Deep Learning.

### 2.8.2 Ambiente di Sviluppo

L'ambiente di sviluppo scelto è il PyCharm Python IDE, poiché è gratuito, semplice da gestire e contiene vari strumenti utili allo sviluppo in Python.

### 2.8.3 Libreria di Sviluppo Deep Learning

Come libreria di Machine Learning è stato scelto PyTorch, poiché gratuito, diffuso e ben documentato. PyTorch è una libreria basata su Torch ed è usato per Computer Vision e Natural Language Processing. Questa libreria fornisce tra l'altro due strumenti molto importanti: il supporto per il calcolo tensoriale, accelerabile tramite GPU, il supporto per la creazione e gestione di Deep Neural Networks accompagnata da un sistema automatico di differenziazione, noto come Autograd.

### 2.8.4 Libreria di Sviluppo ASR

Per semplificare lo sviluppo di sistemi ASR è stata scelta la libreria SpeechBrain, definita come un all-in-one conversational AI toolkit basato su PyTorch. Essa contiene innumerevoli funzionalità per applicazioni Deep Learning inerenti elaborazione del segnale audio ed elaborazione di testi, nonché classe chiamata Brain che astrae vari aspetti della gestione di modelli a reti neurali semplificandone la gestione.

# 3 STATO DELL'ARTE

In questo capitolo viene illustrato un algoritmo di TS-ASR con prestazioni Stato dell'Arte. Il primo riportato è il "Conformer-Based Target-Speaker Automatic Speech Recognition for Single-Channel Audio", un metodo con una architettura moderna basata su Conformer.

## 3.1 ARCHITETTURA CONFORMER

L'architettura Conformer, ovvero Convolution-Augmented Transformer combina efficacemente reti convoluzionali, note per la capacità di modellare dipendenze locali e reti Transformer, note per la capacità di modellare dipendenze globali e ha ottenuto nel 2020 prestazioni Stato dell'Arte in ambito ASR. Il sistema è di tipo encoder decoder ed usa ampiamente metodi di regolarizzazione come dropout, connessioni residue, layernorm e batchnorm.

### 3.1.1 Conformer Encoder

L'encoder è una cascata dei seguenti elementi: un modulo di Data Augmentation, un modulo di convolutional subsampling temporale ed un numero di blocchi Conformer come mostrato nella figura seguente.

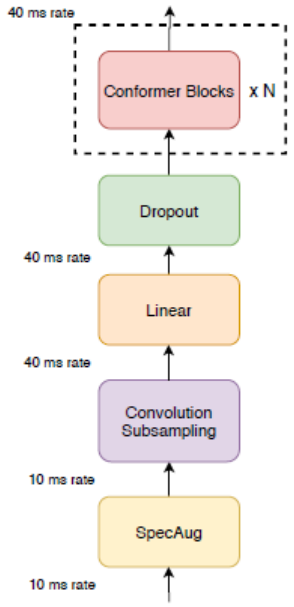



*Figura 9 Struttura Conformer Encoder, fonte (Anmol Gulati, 2000)*

#### *3.1.1.1 Feature Extraction, Data Augmentation*

Il segnale di speech in ingresso viene convertito in feature di tipo log-Mel Spectrogram ad $80$ coefficienti da finestre temporali di $25\,ms$ con scorrimento di $10\,ms$ tra finestre.

Sulle feature estratte viene praticata Data Augmentation con SpecAugment (Park, 2019) con parametro di mask parameter $F = 27$, con $10$ time masks con minimo time-mask ratio $p_s = 0.05$, dove la massima dimensione della time mask è impostata a $p_s$ volte la lunghezza dell'utterance.

#### *3.1.1.2 Convolutional Subsampling Temporale*

Il subsampling temporale è ottenuto con due strati convoluzionali che realizzano un sottocampionamento con fattore $4x$.

#### *3.1.1.3 Conformer Block*

Il Conformer Block è una pila di quattro moduli ognuno con una ulteriore connessione residua tranne l'ultimo, un modulo feed-forward, uno di auto-attenzione, un modulo convoluzionale, un ulteriore modulo feed-forward e layernorm. Le due reti feed-forward sono il risultato di una divisione di una unica rete ispirata alla Macaron-Net (Lu, 2019) in due sistemi più piccoli con pesi dimezzati alla connessione residua.

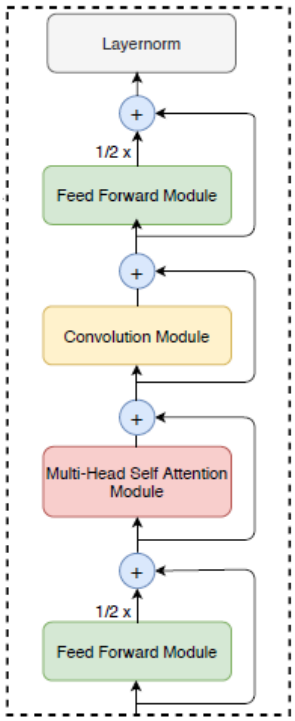



*Figura 10 Struttura Conformer Block, fonte (Anmol Gulati, 2000)*

La catena di elaborazione del blocco Conformer è quindi la seguente:

$$\tilde{x_i} = x_i + \frac{1}{2} FFN(x_i)$$

$$x_i' = \tilde{x_i} + MHSA(\tilde{x_i})$$

$$x_i'' = x_i' + Conv(x_i')$$

$$y_i = Layernorm\left(x_i'' + \frac{1}{2} FFN(x_i'')\right)$$

dove: FFN indica il modulo feed-forward, MHSA indica l'auto-attenzione multi-head e Conv indica il modulo convoluzionale.

Il modulo feed-forward è mostrato nella figura seguente si compone di layernorm, seguito due strati lineari con nel mezzo una attivazione non lineare di tipo Swish (Ramachandran, 2017) con abbinato dropout. Il primo strato lineare espande la dimensione delle feature con coefficiente $4x$ mentre il secondo la riporta alla dimensione originale. Il blocco è in connessione residua con dropout.

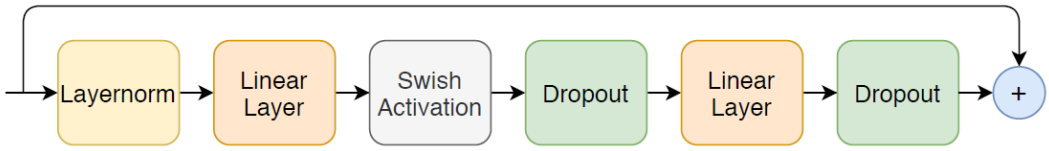



*Figura 11 Struttura modulo feed forward, (Anmol Gulati, 2000)*

Il modulo di auto-attenzione è mostrato nella figura seguente e consiste in un layernorm, seguito da Multihead Attention con Relative Positional Embedding. Il blocco è in connessione residua con dropout. L'auto-attenzione multi-head sfrutta la tecnica del Sinusoidal Relative Positional Encoding (Dai, 2019).

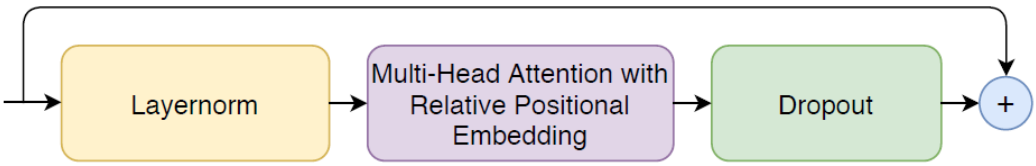



*Figura 12 Struttura modulo auto-attenzione multi head, fonte (Anmol Gulati, 2000)*

Il modulo convoluzionale è mostrato nella figura seguente e consiste in un layernorm, seguito da pointwise convolution, attivazione Gated Linear Unit (Dauphin, 2017) (GLU in breve), 1D depthwise convolution, batchnorm, attivazione non lineare Swish e pointwise convolution. Il blocco è in connessione residua con dropout. La prima pointwise convolution produce una espansione del numero dei canali con fattore $2x$.

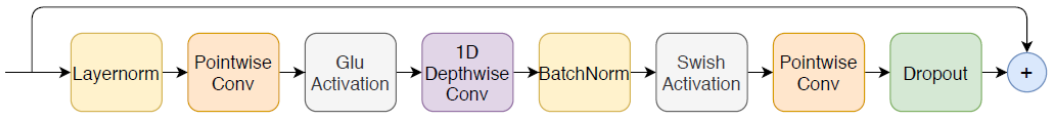



*Figura 13 Struttura modulo convoluzionale, fonte (Anmol Gulati, 2000)*

### 3.1.2 Decoder

Il decoder consiste in un singolo strato LSTM.

### 3.1.3 Regolarizzazione

Il Dropout è usato frequentemente in ogni connessione residua, con $P_{drop} = 0.1$. Un rumore di tipo Variational Noise (Graves, Sequence Transduction with Recurrent Neural Networks, 2012) (K.-C. Jim, 1996) viene introdotto nel modello a scopo di regolarizzazione. Come ulteriore azione regolarizzante

viene usata regolarizzazione $l_2$ con peso $10^{-6}$ su tutti i pesi apprendibili della rete.

### 3.1.4 Modello Linguistico

Un modello linguistico istruito su LibriSpeech language model corpus con trascrizioni aggiunte da LibriSpeech960 è stato impiegato in shallow fusion. Il modello linguistico ha dimensione 4096 con peso di fusione scelto sul dev set attraverso grid search.

## 3.2 CONFORMER-BASED TARGET-SPEAKER ASR FOR SINGLE-CHANNEL AUDIO

L'articolo di riferimento per questo metodo è (Zhang, 2023). Per brevità il nome del sistema sarà abbreviato con l'acronimo CONFTASR. L'architettura proposta è di tipo Single Channel TS-ASR non autoregressiva operante nel dominio tempo-frequenza (TF in breve).

La struttura si compone di tre moduli fondamentali: una TitaNet (Koluguri, 2022) per la creazione del profilo target speaker nella forma di target speaker embedding, un modulo di stima della maschera TF basato su Conformer (Anmol Gulati, 2000) ed in infine un modulo di tipo Single Channel Single-Speaker ASR di tipo Conformer. La topologia impiegata deriva dall'approccio SpeakerBeam (Delcroix, 2018) ma apporta alcune miglioramenti:

- La piccola rete ausiliaria di SpeakerBeam per la creazione del profilo target speaker viene sostituita con TitaNet, una architettura per la creazione di speaker embeddings con prestazioni SOTA nei task di Speaker Verification e Speaker Diarization basata su ContextNet (Han, 2020).
- Sono ottenute prestazioni SOTA per il Target Speaker ASR WER su misture a due speaker su WSJ0-2mix-extr[4] e sono riportati per la prima

[4] https://github.com/xuchenglin28/speaker_extraction

volta in letteratura i risultati per misture a tre speaker su LibriSpeechMix (Kanda N. G., 2020).

- Viene studiato l'effetto di target speaker SNR e della lunghezza della frase ausiliaria sulle prestazioni del modello.

La funzione obiettivo nell'apprendimento è composita e contiene un contributo di tipo Connectionist Temporal Classification (CTC in breve) ed un contributo di tipo scale-invariant spectrogram reconstruction loss per incoraggiare il modello verso una migliore separazione dello spettrogramma del target speaker dalla mistura.

### 3.2.1 Architettura

I tre moduli che compongono l'architettura sono: TitaNet per la generazione del profilo target speaker, MaskNet per la stima della maschera TF di estrazione ed infine l'ASR Conformer.

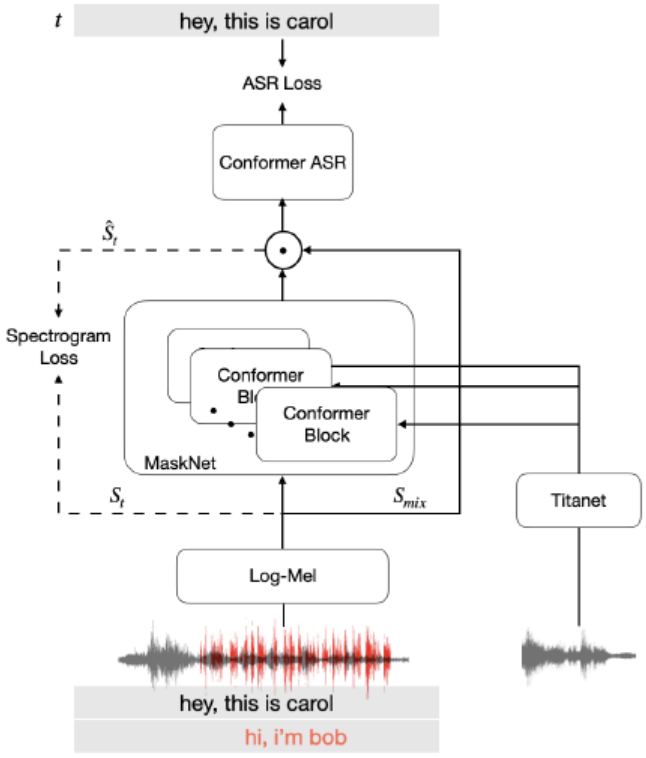



*Figura 14 Architettura CONF-TSASR, fonte (Zhang, 2023)*

Come mostrato nella figura precedente il sistema accetta due ingressi: l'ingresso a sinistra è il segnale mistura contenente i segnali vocali

sovrapposti da speakers differenti incluso il target speaker, sulla destra invece c'è la frase ausiliaria del target speaker. In uscita invece il sistema ritorna la trascrizione dello speech del target speaker ignorando gli altri speakers.

Nel complesso il modello CONF-TSASR possiede $66.1\,M$ di parametri apprendibili ed un totale di $58.4\,M$ di parametri a casa del parziale congelamento della TitaNet.

#### *3.2.1.1 Preprocessing Mistura, Estrazione Features, Data Augmentation*

Il segnale mistura viene prima convertito in features di tipo log-Mel Spectrogram ad $80$ coefficienti su finestre temporali di durata $25ms$ sovrapposte e con scorrimento di $10ms$ . Le feature vengono quindi aumentate al volo applicando SpecAugment (Park, 2019) e vengono sottocampionate nel tempo di frame con fattore $4x$ usando due strati convoluzionali. Le feature così ottenute dalla mistura sono chiamate $S_{mix}$.

#### *3.2.1.2 Preprocessing Frase Ausiliaria, TitaNet*

La frase ausiliaria viene processata con TitaNet, che produce il profilo target speaker nella forma di target speaker embedding a dimensione $192$. Il vettore di embedding viene portato alla dimensione $256$ per mezzo di uno strato lineare e verrà sommato all'ingresso di Conformer Block della MaskNet. I pesi di TitaNet sono inizializzati con i pesi preistruiti disponibili in NVIDIA MeMo toolkit[5]. Durante il training la porzione encoder ContextNet di TitaNet viene congelata e viene fatto apprendere solo il decoder.

#### *3.2.1.3 MaskNet*

Il modulo MaskNet prende in ingresso $S_{mix}$ ed il profilo target speaker per stimare la maschera tempo-frequenza che moltiplicata a $S_{mix}$ produce la stima $\widehat{S_t}$ dello spettrogramma di log-Mel del target speaker. Il modulo contiene $18$ Conformer Blocks, ognuno a dimensione nascosta $256$ e

[5] https://github.com/NVIDIA/NeMo

dimensione feed-forward di $1024$. Il numero di teste nell'attenzione multi-head è pari a $4$. Il modulo convoluzionale ha un kernel size pari a $31$.

#### *3.2.1.4 ASR Conformer*

In ultimo il modulo ASR Conformer prende in ingresso la stima $\widehat{S_t}$ per eseguire la trascrizione. Analogamente alla MaskNet, questo modulo contiene $18$ Conformer Blocks a dimensione $256$ e dimensione feed-forward di $1024$. Il modulo è inizializzato con i pesi preistruiti disponibili in NVIDIA MeMo toolkit.

#### *3.2.1.5 Funzione Costo*

L'intero modello è ottimizzato usando una loss composita a due contributi: il primo contributo è di tipo Connectionist Temporal Classification (CTC in breve), il secondo contributo invece è di tipo spectrogram reconstruction loss ed interviene solo nella fase di training, nella quale si ha accesso al log-mel spectrogram originale $S_t$ del target speaker.

$$\mathcal{L}_{comp} = \alpha\ CTC(\cdot) + (1-\alpha)\ SiSNR(\cdot)$$

Quando viene impiegata la loss composita $\mathcal{L}_{comp}$ l'iperparametro di peso relativo $\alpha$ è selezionato sul validation TS-WER.

La spectrogram reconstruction loss calcola lo Scale-Invariant Signal-to-Noise Ratio (Le Roux, 2019) (SiSNR in breve) tra una versione sovracampionata dello spettrogramma log-Mel stimato $\widehat{S_t}$ e lo spettrogramma log-Mel originale $S_t$. Omettendo per chiarezza visiva i pedici $t$ la metrica $SiSNR$ ci calcola come:

$$SiSNR = 10\log_{10}\left(\frac{\left\|\frac{\hat{S}^T S}{\|S\|^2} S\right\|^2}{\left\|\frac{\hat{S}^T S}{\|S\|^2} S - \hat{S}\right\|^2}\right)$$

L'uso di questa metrica in luogo del semplice $SNR$ sta nel fatto che scalando opportunamente la stima $\widehat{S_t}$ sarebbe possibile ottenere un miglioramento fittizio della metrica senza di fatto apportare un miglioramento percettivo del segnale di speech. In apprendimento il modello a reti neurali regolarizzato con

un contributo di loss basato su $SNR$ non sarebbe vincolato a non apprendere questo metodo di scalatura e potrebbe favorire la riduzione fittizia del contributo di loss sfruttando la scalatura invece che apprendendo un metodo di mascheratura più efficace. La metrica proposta è invece insensibile al fenomeno della scalatura. Un difetto introdotto da questo tipo di approccio consiste nell'ammettere implicitamente che eventuali scalature di $\hat{s}$ non vengano considerate come errori.

La rete neurale sarebbe quindi potenzialmente regolarizzata nella direzione dell'apprendimento di una buona tecnica di mascheratura ma non è punita nello scalare la stima di $\hat{s}$. La dinamica potenzialmente aumentata di $\hat{s}$ è potenzialmente accomodata dal modulo ASR in quanto partecipe all'apprendimento nella fase di training.

### 3.2.2 Dataset, Data Augmentation

CONF-TSASR è stato valutato usando misture a $2$ e $3$ speakers create a partire dai dataset WSJ0 (Garofolo, 1993) e LibriSpeech (Panayotov, 2015). Per usare le misture in contesto TS-ASR un target speaker è selezionato in modo casuale dalle misture ed una frase dello stesso speaker e differente da quelle usate nella mistura viene selezionata come frase ausiliaria.

Le forme di data augmentation impiegate sono: Speed Perturbation (Ko, 2015) e Volume Perturbation (Kanda N. G., 2020). La Speed Perturbation interviene con probabilità $0.3$ su ogni utterance modificandone la velocità ad uno tra questi valori: $95\%, 97.5\%, 100\%, 102.5\%, 105\%$ prima che venga miscelata. Con la Volume Perturbation viene invece scalato il volume della mistura con un fattore casuale scelto tra $0.125$ e $2.0$.

#### *3.2.2.1 WSJ0-2mix-estr e WSJ0-3mix-extr*

Il training set ammonta a $30\ ore$ di misture a due speakers generate selezionando casualmente utterances da speakers diversi nel WSJ0 $si\_tr\_s$ e miscelandole con $SNR$ casuale tra $0$ e $5\ dB$. Nelle misture, l'utterance più breve viene reinserita in testa ed in coda con intervalli di silenzio casuali fino

a corrispondere alla maggiore lunghezza tra le utterance della mistura. Il cross validation set è usato per la selezione degli iperparametri ed è stato realizzato in modo analogo al training set. Il set di evaluation ammonta a $5\ ore$ ed è generato con utterance da $16$ speakers del WSJ0 development set $si\_dt\_05$ ed evaluation set $si\_et\_05$ basati su speakers differenti da quelli di training. Per i dati i evaluation, sono state create $100$ misture a tre speakers. I dati sono sottocampionati ad $8\ kHz$ per ridurre il carico computazionale ed i requisiti di memoria.

I datasets con le misture ottenute a partire da WSJ0 a due e tre speakers sono chiamati rispettivamente WSJ0-2mix-estr e WSJ0-3mix-extr.

#### *3.2.2.2 LibriSpeech2mix, LibriSpeech3mix*

La preparazione di LibriSpeech segue la recipe di Kaldi (D. Povey, 2011), quindi il set $train\_960$ è stato usato per il training, il dev_clean per la selezione degli iperparametri ed il test_clean per il testing. Per ogni utterance in $train\_960$ sono sommate $S-1$ utterance con ritardo casuale tra i tempi di inizio di ognuna di almeno $0.5\ sec$ pur garantendo la sovrapposizione di almeno due utterance. Il volume originale di ogni utterance è stato lasciato inalterato producendo nel complesso un $SNR$ di $0\ dB$. Le frasi ausiliarie sono state create selezionando casualmente in $train\_960$ due utterances non presenti nella mistura ma appartenenti al target speaker ottenendo una durata media di circa $15\ sec$. Il processo è stato eseguito per tre valori si $S$, ovvero $S = \{1,2,3\}$. I dati prodotti da $dev\_clean$ e $test\_clean$ sono stati creati con lo stesso metodo ma applicando un ritardo minimo pari $0$ sec, ovvero permettendo alle frasi nella mistura di partire simultaneamente.

I datasets creati a partire dalle misture LibriSpeech a due e tre speakers sono chiamati LibriSpeech2mix e LibriSpeech3mix.

### 3.2.3 Apprendimento

Per l'apprendimento sono state usate $16$ GPU $V100-23GB$ con batch size globale pari a $64$. Il modello è stato istruito per $60K$ aggiornamenti di

parametri per il dataset WSJ0-mix-extr e $480K$ aggiornamenti di parametri per LibriSpeechMix. L'ottimizzatore impiegato è AdamW (Loshchilov & Hutter, 2019) con Transformer Learning Rate Scheduler. Lo scheduling prevede learning rate di picco pari a $3 * 10^{-4}$ e $0.01$ weight decay con $10K$ e $25K$ passi di warmup con Cosine Annealing a learning rate minimo di $10^{-6}$ rispettivamente per WSJ0-mix-extr e LibriSpeechMix.

### 3.2.4 Risultati WSJ0-mix-extr

| Model | Params, M | N | Loss | Learn Embedding | 2-mix | 3-mix |
|---|---|---|---|---|---|---|
| CONF-TSASR | 85 | 2 | CTC | no | 8.6 | - |
| | | 2 | CTC | yes | 4.8 | - |
| | | 2 | CTC+Spec | yes | **4.2** | - |
| | | 3 | CTC | yes | 5.4 | 13.8 |
| | | 3 | CTC+Spec | yes | **4.8** | **12.4** |
| SpeakerBeam | n/a | 2 | Cross Entropy | yes | 30.6 | - |
| Exformer + Conformer-CTC | 80 | 2 | SiSNR, CTC | no | 13.2 | - |
| Conditional-Conformer-CTC | n/a | 3 | CTC | yes | 19.9** | 34.3** |
| Conformer-CTC | 29 | 1 | CTC | no | 36.7* | 54* |

*Figura 15 Risultati TS-WER su WS0-mix-extr, N indica il numero di speakers nella mistura di train, * indica il WER migliore su trascrizioni individuali, **indica il WER per multi-speaker ASR, fonte (Anmol Gulati, 2000)*

Come termini di comparazione e baseline sono stati inclusi un modello ASR convenzionale (Conformer CTC) istruito su singolo speaker ed usato come baseline, SpeakerBeam (Žmolíková, 2019) istruito su misture a due speakers, Exformer (Wang Z. G., 2022) istruito su misture a due speakers e Conditional-Conformer-CTC (Guo, 2021) istruito su misture fino a tre speakers. SpeakerBeam è un modello TS-ASR che produce direttamente le trascrizioni del target speaker. Exformer è un modello di target speaker source separation basato su SepFormer (Subakan, 2021) quindi richiede un operazione ulteriore per trascrivere l'uscita del modello. Conditional-Conformer-CTC è un modello ASR multi-speaker che usa la conditional speaker chain per trascrivere ogni speaker in successione.

Per rendere il confronto più onesto, Exformer è stato modificato per impiegare la TitaNet preistruita (come per CONF-TSASR) come rete di embedding. Inoltre la model size di Exformer è stata fatta corrispondere a quella della MaskNet di CONF-TSASR. Il modello usato per la trascrizione di Exformer ed il modello ASR convenzionale di baseline sono la stessa cosa. Inoltre gli stessi

pesi di questo modello sono stati usati per inizializzare il modulo ASR di CONF-TSASR.

Come atteso, il modello ASR convenzionale istruito a singolo speaker ha scarse prestazioni su WSJ0-2mix con WER pari a 36.7% mentre SpeakerBeam ed Exformer+Conformer-CTC mostrano rispettivamente TS-WER 30.6% e 13.2% rispettivamente. Conditional-Conformer-CTC ottiene WER di 19.9% e 34.3% per misture a due e tre speakers rispettivamente.

A confronto CONF-TSASR istruito con misture a due speakers ottiene 4.8% TS-WER se istruito con la sola loss CTC e 4.2% quando istruito con la loss composita. Quando istruito con misture a tre speakers CONF-TSASR ottiene 4.8% e 12,4% TS-WER valutandolo su misture a due e tre speakers rispettivamente. Questo suggerisce come che il modello sia in grado di operare correttamente nonostante la presenza di due speakers causanti disturbo. Secondo gli autori il TS-WER ottenuto su WSJ0-2mix-extr è il migliore in letteratura ed i risultati RS-WER ottenuti su WSJ0-3mix-extr sono i primi di questo tipo ad apparire in letteratura.

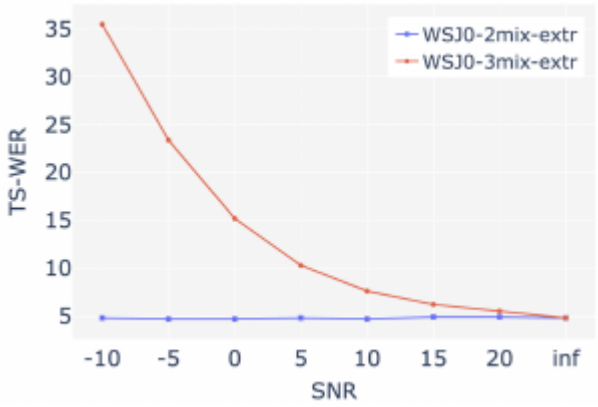



*Figura 16 Studio di sensibilità tra SNR e TS-WER su WS0-mix-extr, fonte (Anmol Gulati, 2000)*

Gli autori riportano come le prestazioni di CONF-TSASR siano più sensibili all'SNR quando la mistura contiene tre speakers sovrapposti rispetto al caso con due speakers.

### 3.2.5 Risultati LibriSpeechMix

| Model | Params, M | L (sec) | 2-mix |
|---|---|---|---|
| CONF-TSASR CTC | 85 | 7.5 | 5.1 |
| | | 15 | 4.6 |
| CONF-TSASR CTC+Spec | 85 | 7.5 | 4.5 |
| | | 15 | **4.2** |
| SOT-Confomer-AED | 129 | 15 | 4.9† |
| t-SOT TT-18 | 82 | 15 | 5.2‡ |
| t-SOT TT-36 | 139 | 15 | 4.4‡ |

*Figura 17 Risultati TS-WER, SA-WER†, permutation invariant SA-WER‡ istruito con misture a due speakers, L indica la lunghezza media della frase ausiliaria, fonte (Anmol Gulati, 2000)*

| Model | L (sec) | 2-mix | 3-mix |
|---|---|---|---|
| CONF-TSASR CTC | 7.5 | 7 | 9.7 |
| | 15 | 6 | 8.4 |
| CONF-TSASR CTC+Spec | 7.5 | 6.3 | 9 |
| | 15 | **5.4** | **7.6** |
| SOT-Confomer-AED | 15 | 6.8† | 9.6† |
| SOT-Confomer-AED SD | 15 | **6.4**† | **8.5**† |

*Figura 18 Risultati TS-WER, permutation invariant SA-WER‡, CONF-TSASR istruito con tre speakers, L indica la lunghezza della frase ausiliaria, SD indica speaker duplication, fonte (Anmol Gulati, 2000)*

Dato che in letteratura non sono presenti risultati TS-ASR su LibriSpeechMix, come riferimenti sono stati usati Conformer-AED (Kanda N. Y., 2021) e la sua versione streaming t-SOT (Kanda N. W., 2022). Le differenze tra CONF-TSASE e questi due modelli impongono alcuni accorgimenti per rendere onesto il confronto. I risultati TS-WER ottenuti su LibriSpeechMix sono resi comparabili allo Speaker Attributed WER trascrivendo lo speech di tutti gli speaker nella mistura uno alla volta come target speaker usando CONF-TSASR. Istruendo il modello con misture a tre speakers si ottengono 5.4% e 7.6% TS-WER su misture a due e tre speakers rispettivamente. Quando invece è istruito su misture a due speakers le prestazioni migliorano a 4.2% TS-WER su LibriSpeech2Mix.

I risultati ottenuti nel caso di misture a due e a tre speakers mostrano che i modelli istruiti con loss composita CTC + SiSNR sono significativamente più performanti di quelli istruiti con la sola CTC.

Quando il sistema è valutato usando solo una frase ausiliaria (con durata media di 7.5 $sec$) per la creazione del profilo target speaker, sia ha leggero degrado delle prestazioni.

# 4 DESCRIZIONE METODO

Questo lavoro di tesi ha avuto inizio tempo addietro con lo studio e la sperimentazione sulle strutture elementari TS-ASR di SpeakerBeam (Delcroix M. Z., 2018) nella sua prima versione a singolo canale basata GMM/HMM Ibrida. Questa è una una architettura di tipo HMM/DNN dove un modello acustico a reti neurali è usato per stimare lo stato nascosto di un modello Hidden Markov Model con distribuzione di emissione di tipo Gaussian Mixture Model. Queta prima fase è stata utile per l'acquisizione di un bagaglio minimo di esperienza nell'affrontare all'atto pratico i sistemi Deep Learning.

L'attività sperimentale è poi virata su un tentativo infruttuoso di fusione tra una modello End-to-End Encoder Decoder con Attenzione Single Speaker ASR con una rete ausiliaria ispirata a (Delcroix M. Z., 2018) per la conversione in TS-ASR. L' encoder impiegato era di tipo CRDNN [6] (ovvero combinante nell'ordine componenti convoluzionali, ricorrenti e DNN) e l'attenzione di tipo Location Aware Attention.

Infine è stato preso a riferimento il metodo (Delcroix M. W., 2019), è su questa parte che si focalizza la seguente trattazione.

## 4.1 ARCHITETTURA DI RIFERIMENTO

Il metodo (Delcroix M. W., 2019) è una versione migliorata del precedente SpeakerBeam (Delcroix M. Z., 2018), ora reso End-to-End con una architettura interamente neurale, una modalità di fusione semplificata, tecniche di regolarizzazione aggiornate e prestazioni superiori.

Il riferimento fornisce due modelli per applicazioni TS-ASR: un Adaptive Encoder ed uno Cascade Connection come mostrati nella figura seguente.

[6] https://speechbrain.readthedocs.io/en/latest/API/speechbrain.lobes.models.CRDNN.html

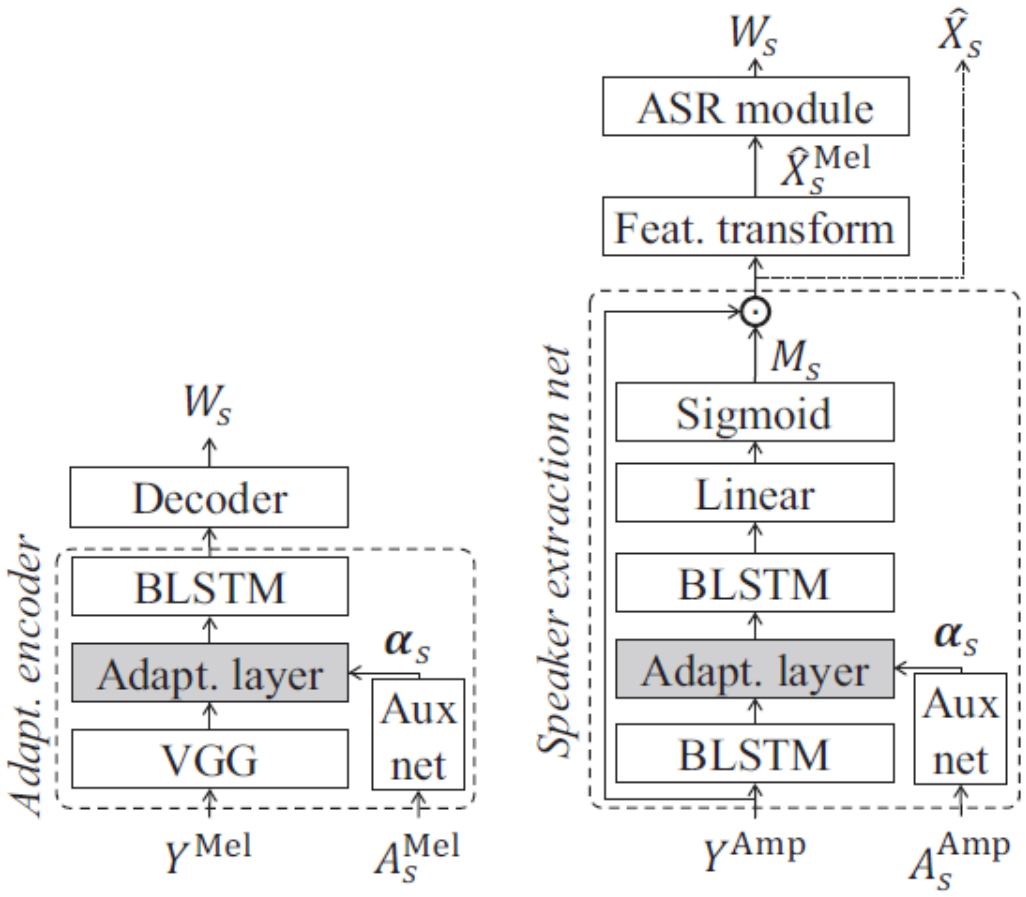



*Figura 19 Modelli E2E TS-ASR, Adaptive Encoder a sinistra, Cascade Connection a destra, fonte (Delcroix M. W., 2019)*

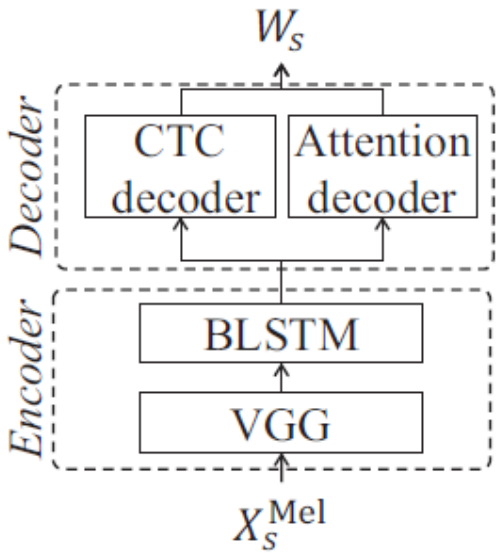



*Figura 20 Modello E2E ASR Baseline, fonte (Delcroix M. W., 2019)*

### 4.1.1 Ingressi e Uscite

Nella nomenclatura degli ingressi ai vari modelli il target speaker speech viene indicato $X$, i segnali mistura sono indicati con $Y$ e le frasi ausiliarie sono indicate con $A$.

Quando i segnali di ingresso sono rappresentati per mezzo di feature log-Mel Spectrogram coefficients, il loro nome include l'apice $Mel$ e la dimensione delle feature conta $80$ coefficienti, con normalizzazione in media e varianza al livello di utterance. Quando invece la rappresentazione si basa su amplitude spectrum coefficients, il nome del segnale ha l'apice $Amp$ e le features hanno dimensione $201$.

Il pedice $S$ sta ad indicare che un ingresso, uscita o variabile nascosta è specifica del target speaker $S$.

L'accento circonflesso $\hat{}$ indica invece che la quantità in esame è stata stimata dal modulo Speaker Extraction Net.

- $\boldsymbol{X_S^{Mel}}$: log-Mel Spectrogram coefficients del target speaker speech
- $\boldsymbol{Y^{Mel}}$ , $\boldsymbol{Y^{Amp}}$ : rispettivamente log-Mel Spectrogram coefficients e amplitude spectrum coefficients del segnale mistura
- $\boldsymbol{A_S^{Mel}}$ **,** $\boldsymbol{A_S^{Amp}}$ **:** rispettivamente log-Mel Spectrogram coefficients e amplitude spectrum coefficients della frase ausiliaria
- $\boldsymbol{a_S}$: profilo target speaker
- $\boldsymbol{M_S}$ : maschera tempo frequenza di estrazione del target speaker speech
- $\boldsymbol{\hat{X}_S}$: stima degli amplitude spectrum coefficients del target speaker speech
- $\boldsymbol{\hat{X}_S^{Mel}}$: stima dei log-Mel Spectrogram coefficients del target speaker speech
- $\boldsymbol{W_S}$: trascrizione del target speaker speech

### 4.1.2 Modello: Baseline E2E ASR

Il modello Baseline E2E ASR (fig. 20) è un modulo end-to-end ASR encoder decoder a singolo speaker. La struttura dell'encoder comprende un modulo CNN ispirato alla configurazione VGG (Watanabe, 2018) (Simonyan, 2014) ed una pila BLSTM. La decodifica è condotta con due decoder paralleli: uno basato su CTC e l'altro su Attenzione.

#### *4.1.2.1 Encoder*

La parte convoluzionale dell'encoder è costituita da $4$ strati convoluzionali con attivazione di tipo ReLU. La prima coppia di strati sfrutta $64$ canali di informazione e la seconda coppia ne sfrutta $128$. Dopo la prima e la seconda coppia di canali è applicato max pooling con fattore di riduzione pari a $2$ lungo le due dimensioni della feature maps. In ogni strato convoluzionale vengono usati kernel a dimensione $3x3$, stride e padding pari a $1$.

| Strato VGG | Input Channels | Output Channels | Kernel Size | Stride | Padding | Ceil Mode |
|---|---|---|---|---|---|---|
| Conv1+ReLU | 1 | 64 | $3x3$ | 1 | 1 | - |
| Conv2+ReLU | 64 | 64 | $3x3$ | 1 | 1 | - |
| Max Pooling | - | - | 2x2 | 2 | - | True |
| Conv3+ReLU | 64 | 128 | $3x3$ | 1 | 1 | - |
| Conv4+ReLU | 128 | 128 | $3x3$ | 1 | 1 | - |
| Max Pooling | - | - | 2x2 | 2 | - | True |

*Tabella 1 Parametrizzazione della componente convoluzionale dell'encoder VGG-BLSTM*

La dimensione temporale dell'uscita della componente VGG dell'encoder è un quarto di quella originale. L'uscita $VGG_{Out}$ del sistema VGG viene ricondotta al formato tensore tipico di PyTorch $[Batch, TimeFrames, Features]$ riordinando le dimensioni del tensore ed appiattendone i canali. Così facendo si ottiene una dimensione di feature pari a $VGG_{Size}$:

$$VGG_{Size} = 128 * N_{Mels}/4 = 128 * 80/4 = 2560$$

$$dim(VGG_{Out}) = [B, TimeFrames/4, VGG_{Size}\ ]$$

La componente ricorrente dell'encoder VGG-BLSTM è una pila BLSTM piramidale con $6$ strati ricorrenti a 1024 neuroni con riproiezione a

dimensione 1024 e subsampling temporale con fattore pari a $2$ sul secondo e terzo strato. Il fattore di subsampling complessivo è pari a $4$. La bidirezionalità delle BLSTM fa sì che si abbiano uscite a dimensione pari al doppio del numero di neuroni delle unità ricorrenti $1024 * 2 = 2048$. Avendo scelto di impiegare strati ricorrenti a 1024 neuroni, viene applicata la riproiezione delle uscite per ricondurle alla dimensione nominale 1024. La riproiezione è eseguita applicando uno strato lineare di dimensioni opportune.

| Strato BLSTM | BLSTM Neurons | Input Size | Output Size | Subsampling Factor |
|---|---|---|---|---|
| BLSTM1 | 1024 | - | - | - |
| Projection1 | - | 2048 | 1024 | - |
| BLSTM2 | 1024 | - | - | - |
| Projection2 | - | 2048 | 1024 | - |
| Subsampling1 | - | - | - | 2 |
| BLSTM3 | 1024 | - | - | - |
| Projection3 | - | 2048 | 1024 | - |
| Subsampling2 | - | - | - | 2 |
| BLSTM4 | 1024 | - | - | - |
| Projection4 | - | 2048 | 1024 | - |
| BLSTM5 | 1024 | - | - | - |
| Projection5 | - | 2048 | 1024 | - |
| BLSTM6 | 1024 | - | - | - |
| Projection6 | - | 2048 | 1024 | - |

*Tabella 2 Parametrizzazione della componente ricorrente dell'encoder VGG-BLSTM*

Il modello è istruito con una funzione costo composita che include sia la loss CTC che la loss di Attenzione. Entrambi i contributi di loss sono impiegati nella decodifica con metodo Hybrid CTC/Attention Joint Decoding One-Pass Decoding illustrato in (cap. 2.5.5).

#### *4.1.2.2 Decoder*

Il decoder CTC è costituito da una funzione di attivazione di tipo Softmax applicata direttamente all'uscita dell'encoder.

Il decoder di attenzione è invece realizzato con uno strato LSTM a $300$ neuroni, attenzione di tipo Location Aware Attention e uno strato di attivazione di tipo Softmax.

### 4.1.3 Modello: Adaptive Encoder

L'Adaptive Encoder (fig. 19 sinistra) è strutturato come un comune encoder per applicazioni End-to-End ASR in cui è stata fusa una rete ausiliaria Auxiliary Network per mezzo di uno strato adibito alla fusione chiamato Adaptation Layer. In ottica TS-ASR la rete ausiliaria esegue l'estrazione del target speaker embedding creando un profilo target speaker $a_S$. Il profilo viene impiegato per fare un operazione che si può vedere come l'equivalente ad una target speaker extraction nel dominio delle feature nascoste. In questo caso la rete di estrazione sarebbe la combinazione del modulo VGG e dell'Adaptation Layer e fornirebbe in uscita direttamente il segnale estratto. Il segnale estratto sarebbe quindi elaborato dal modulo BLSTM terminando le operazioni encoding.

Le parti VGG e BLSTM dell'encoder ed il decoder sono dimensionate in modo identico al modello Baseline E2E ASR; quindi, vengono descritti solo l'Adaptation Layer e l'Auxiliary Network.

#### *4.1.3.1 Adaptation Layer*

L'Adaptation Layer è un modulo che realizza la fusione tra il profilo target speaker e le feature nascoste del modello $h_t^{in}$ producendo in uscita le feature di uscita $h_t^{out}$. In questo caso il modulo è realizzato con la modalità Scaling Activation (Delcroix M. Z., 2019), che modula l'ingresso $h_t^{in}$ direttamente con l'ampiezza di $a_S$ come se operasse da maschera nel dominio delle feature nascoste. Se $\odot$ indica il prodotto element-wise, l'uscita dell'Adaptation Layer è pertanto:

$$h_t^{out} = h_t^{in} \odot a_S$$

Essendo $h^{in} = VGG_{Out}$ , tutti e tre i tensori $h_t^{out}$ , $h_t^{in}$ , $a_S$ devono avere dimensione $VGG_{Size}$.

#### *4.1.3.2 Auxiliary Network*

Questa rete ha il compito di estrarre il profilo target speaker sotto forma di target speaker embedding $a_S$ usando come ingresso le feature log-Mel

Spectrogram coefficients $A_S^{Mel}$ della frase ausiliaria. La struttura è quella di una Sequence Summary Network (Veselý, 2016) (Žmolíková K. D., 2017), ovvero consiste in un modello a rete neurale $G(\cdot)$ seguito da una operazione di averaging temporale. In questo caso $G(\cdot)$ è una Feed-Forward Neural Network. Se il $\tau - esimo$ frame temporale in ingresso è $a_{S,\tau}$ e se la sequenza di ingresso è lunga $T'$ frames temporali, allora si ha:

$$a_S = \frac{1}{T'}\sum_{\tau=1}^{T'} G\left(a_{S,\tau}\right)$$

La Auxiliary Network contiene due strati lineari a 200 neuroni con funzione di attivazione ReLU seguiti da uno strato lineare di uscita senza ulteriore funzione di attivazione. Questo strato adatta l'uscita della rete alla dimensione richiesta $VGG_{Size}$ . La mancanza di una specifica funzione di attivazione su questo strato permette di svincolare $a_S$ da uno specifico range così da permettere ad $a_S$ di agire liberamente sul comportamento del modello.

| Strato | Neurons | Input Size | Output Size |
|---|---|---|---|
| Linear1+ReLU | 200 | 80 | 200 |
| Linear2+ReLU | 200 | 200 | 200 |
| Linear Output | 2560 | 200 | 2560 |

*Tabella 3 Parametrizzazione Auxiliary Network per modello Adaptive Encoder*

### 4.1.4 Modello: Cascade Connection

Il modello Cascade Connection (fig. 19 destra) esegue le operazioni TS-ASR nel dominio tempo-frequenza concatenando una Speaker Extraction Net, un modulo di estrazione di feature deterministico ed un modulo Single Speaker ASR. In questo caso l'Adaptation Layer introduce il profilo target speaker nella Speaker Extraction Net per guidarne l'estrazione nel dominio tempo frequenza.

In termini di numero di parametri totali il Cascade Connection è di dimensioni maggiori rispetto al modello Adaptive Encoder, tuttavia offre una migliore interpretabilità del comportamento del modello, migliori prestazioni e

permette la ricostruzione del target speech signal stimato per valutare le capacità di speech enhancement della Speaker Extraction Net.

#### *4.1.4.1 Modulo Speaker Extraction Net*

Questo modulo è una rete neurale che produce una maschera tempo-frequenza per estrarre il target speaker dalla mistura (Narayanan, 2013) (Wang D. &., 2018). L'estrazione avviene moltiplicando elemento per elemento la maschera prodotta $M_S$ con le feature amplitude spectrum coefficients $Y^{Amp}$ della mistura in ingresso restituendo in uscita gli amplitude spectrum coefficients stimati $\hat{X}_S^{Amp}$:

$$\hat{X}_S^{Amp} = M_S \odot Y^{Amp}$$

Il modulo Speaker Extraction Net consiste in una pila di $3$ strati BLSTM a 512 neuroni con riproiezione. Tra il primo strato BLSTM ed il secondo è inserito l'Adaptation Layer. Seguono uno strato lineare ed una funzione di attivazione di tipo sigmoidale. Lo strato lineare sostituisce adatta la dimensione delle feature di uscita alla dimensione degli amplitude spectrum coefficients della mistura in ingresso, ovvero 201. Infine, l'attivazione sigmoidale limita il range del tensore in uscita nell'intervallo $[0,1]$ producendo la maschera tempo frequenza.

Impiegando una BLST riproiettata al primo strato, cambiano a $512$ le dimensioni delle feature della terna $h_t^{out}$, $h_t^{in}$, $a_S$ dell'Adaptation Layer. Di conseguenza cambia la dimensione dello strato di uscita dell'Auxiliary Network che ora mappa un ingresso a dimensione $200$ in un uscita a dimensione 512 invece che $VGG_{Size} = 2560$ come nell'Adaptive Encoder.

| Strato | Neurons | Input Size | Output Size |
|---|---|---|---|
| BLSTM1 | 512 | 201 | 1024 |
| Proection1 | - | 1024 | 512 |
| Adaptation Layer | - | 512 | 512 |
| BLSTM2 | 512 | 512 | 1024 |
| Proection2 | - | 1024 | 512 |
| BLSTM3 | 512 | 512 | 1024 |

| | | | |
|---|---|---|---|
| Linear | - | 1024 | 201 |
| Sigmoid | - | - | - |

*Tabella 4 Parametrizzazione Speaker Extraction Net del modello Cascade Connection*

#### *4.1.4.2 Modulo Feature Extraction*

Questo modulo converte gli amplitude spectrum coefficients $\hat{X}_S^{Amp}$ stimati con la mascheratura tempo frequenza in log-Mel Spectrogram coefficients $\hat{X}_S^{Mel}$ che verranno usati dal modulo ASR a singolo speaker.

#### *4.1.4.3 Modulo ASR*

Questo modulo è un modello single speaker ASR sulla base delle speech features $\hat{X}_S^{Mel}$ in ingresso.

### 4.1.5 Apprendimento

L'apprendimento segue un metodo Multi-Task Learning (MTL in breve) con una funzione costo $L$ data dalla somma di due contributi di loss $L^{Mix}$ , $L^{Clean}$ corrispondenti a due percorsi di segnale differenti (qualora possibile). I due coefficienti di pesature $\mu, \nu$ determinano la modalità operativa.

$$L = \mu L^{Mix}(Y, W_S, A_S) + \nu L^{Clean}(X_S, W_S)$$

Il contributo di loss $L^{Clean}$ è la loss composita CTC+Attenzione che si ottiene nel caso di percorso di segnale pulito. Nell'apprendimento del modello Adaptive Encoder per percorso di segnale pulito si intende il caso in cui al modello vengono somministrate le feature del segnale pulito di target speaker speech invece che della mistura e viene anche ignorato lo strato di Adaptation Layer come se lo si scavalcasse. Nel caso di apprendimento del modello Cascade Connection anche qui le feature della mistura sono sostituite con quelle del target speaker speech e viene inoltre scavalcato completamente il modulo di Speaker Extraction Net.

Il contributo di loss $L^{Mix}$ è invece la loss composita CTC+Attenzione nel caso nominale in cui le feature di mistura sono somministrate normalmente in ingresso ai modelli in apprendimento e tutte le componenti dei modelli partecipano normalmente all'elaborazione dei segnali.

L'apprendimento con MTL avviene quando $\{\mu = 0.5, \nu = 0.5\}$, invece avviene senza MTL apprendimento quando $\{\mu = 1, \nu = 0\}$.

L'uso dello schema MTL si basa sull'ipotesi che $L^{Clean}$ favorisca l'apprendimento di un metodo migliore di estrazione del target speaker speech. Si suppone infatti che in questo modo, esponendo la parte superiore del modello ad un segnale più pulito, durante la retropropagazione di $L^{Mix}$ le parti inferiori del modello siano maggiormente condizionate a produrre segnali più puliti che in altro modo.

Tutti gli esperimenti sono stati condotti con ESPnet (Watanabe, 2018). Tutti i modelli sono stati istruiti con l'algoritmo di ottimizzazione AdaDelta per $20$ epoche.

Sono stati istruiti due moduli Baseline single speaker E2E ASR: un modello detto Clean Baseline istruito con segnale di speech pulito ed un modello Dominant Baseline istruito sullo speaker dominante nella mistura.

Entrambi i modelli Adaptive Encoder e Cascade Connection sono stati istruiti con dati in Full Overlap.

In ogni esperimento la decodifica è effettuata con l'algoritmo Hybrid CTC/Attention con One Pass Decoding in combinazione con un modello linguistico RNN-LM al livello di parola in shallow fusion. Il modello linguistico è stato istruito sui dati testuali del WS1 corpus (Hori, 2018).

### 4.1.6 Dataset

Il dataset impiegato è il MERL two speaker misture corpus (R. Hershey, 2015) creato dalle utterance a singolo speaker del dataset WSJ0 (Garofolo, 1993) con SNR tra 0 e 5 dB in condizioni full overlap (FO in breve). il training set consiste 20000 misture ed il test set di 3000 misture di speakers differenti da quelli nel training set. Inoltre è stato creato un secondo set con misture simili a quelle del MERL, ma traslando nella mistura l'istante iniziale di una delle utterance che la compongono. Questo secondo set ha lo scopo di creare

condizioni di testing più realistiche di partial overlap (PO in breve). Per entrambi i dataset è stata usata una frequenza di campionamento di $16\ kHz$.

### 4.1.7 Risultati ASR

I risultati proposti riportano il WER ottenuto mediando il TS-WER calcolato su ogni speaker presente nella mistura.

| **Model** | **MTL** | **Full Overlap TS-WER** $[\%]$ | **Partial Overlap TS-WER** $[\%]$ | **Clean Speech WER** |
|---|---|---|---|---|
| Clean baseline | - | 114.7 | 106.7 | 5.5 |
| Dominant Baseline | - | 75.7 | 87.3 | - |
| SpeakerBeam Adaptive Encoder | NO | 21.1 | 16.5 | - |
| | SI | 19.8 | 15.5 | - |
| SpeakerBeam Cascade Connection | NO | 18.4 | **13.6** | - |
| | SI | **18.0** | 15.4 | - |

*Tabella 5 Risultati TS-WER*

Come atteso le due varianti di baseline hanno prestazioni scarse non essendo guidate dal profilo target speaker sia in FO che PO. In condizione PO il WER è comunque alto a causa del riconoscimento dello speaker interferente, che introduce molti errori di inserimento. Invece nel caso di riconoscimento di clean speech, il modello Clean Baseline produce un limite inferiore di WER pari a $5.5\%$.

Entrambi gli schemi SpeakerBeam producono un miglioramento significativo delle prestazioni sia in FO che PO. Il modello Cascade Connection fornisce risultati leggermente migliori.

L'uso di MTL apporta piccoli miglioramenti ovunque tranne nel caso PO con Cascade Connection. Il mancato miglioramento in questa direzione si deve agli errori di inserzione causati dai contributi residui di speaker interferente dopo la mascheratura.

## 4.2 IMPOSTAZIONE SPERIMENTALE

L'obiettivo del lavoro di tesi è stato quello di realizzare un sistema TS-ASR from scratch traendo libera ispirazione dal metodo descritto precedentemente "End-to-end SpeakerBeam for single channel target speech recognition" (Delcroix M. W., 2019). Il lavoro sfruttando il linguaggio di programmazione Python 3.10 e le librerie PyTorch e SpeechBrain specifiche per lo sviluppo Deep Learning ed ASR.

### 4.2.1 Differenze Implementative

La libreria SpeechBrain non supporta direttamente l'algoritmo di decodifica Hybrid CTC/Attention nelle forme Joint Decoding: Rescoring e One Pass Decoding abbinati a Beam Search. Di conseguenza è stata scelta una decodifica basata su attenzione, con attenzione di tipo Location Aware Attention come nel riferimento.

Per compensare almeno parzialmente la potenziale riduzione delle prestazioni conseguenza ad un metodo di decodifica più elementare, viene sperimentata qualche forma di Data Augmentation durante l'apprendimento.

La funzione costo impiegata è di tipo NLL + CTC, ma la componente CTC opera sull'uscita dell'encoder per facilitare l'apprendimento del sistema.

Il dataset scelto per la sperimentazione è LibriSpeech e non WSJ0 come nel riferimento, poiché ampiamente disponibile e di facile impiego.

Di conseguenza anche il modello linguistico ed il tokenizer sono differenti per corrispondere al dataset scelto.

La durata degli esperimenti di apprendimento non è stata fissata a priori a gestita in modalità Early Stopping.

Negli esperimenti è stato incluso un metodo di Learning Rate Annealing.

Lo spazio di soluzioni a questo problema con questa architettura è stato parzialmente valutato sperimentando diverse combinazioni di iperparametri.

### 4.2.2 Preparazione Dataset

Il framework di apprendimento impiegato richiede una operazione detta di preparazione del dataset, il cui scopo è quello di creare un Data Manifest File per ogni split usato, ovvero un file strutturato contenente delle informazioni salienti su ogni utterance disponibile nello split stesso. Le informazioni sono specifiche per il task in esame. I Data Manifest Files sono stati creati in formato json.

### 4.2.3 Preparazione Dataset - Clean Baseline

Nella sperimentazione per il modello Clean Baseline, il dataset impiegato è costruito sui tre split di training [7], development [8] e test [9] di LibriSpeech, separando a sua volta lo split di training originale training e validation con ripartizione relativa $\{80\%, 20\%\}$. La separazione è condotta assicurandosi che non ci sia intersezione tra gli speakers dei due split separati e facendo in modo che il rapporto originale $UPS$ tra il numero di utterance per speaker sia mantenuto costante nei due split separati. I campi informativi riportati sono in questo caso $3$:

- "**wav**": percorso relativo dell'utterance,
- "**n_of_samples**": lunghezza dell'utterance in termini di campioni
- "**words**": trascrizione dell'utterance

### 4.2.4 Preparazione Dataset - Adaptive Encoder

La preparazione per il modello Adaptive Encoder sfrutta tutti i dati disponibili. Per lo split di training sono stati usati i contenuti di train-clean-100, train-clean-360, train-other-500. Per lo split di development sono stati usati: dev-clean, dev-other. Invece per lo split di test sono stati impiegati: test-clean, test-other. Lo split di training è stato separato in training e validation con lo stesso metodo usato prima. I campi informativi inseriti nei Data Manifest File

[7] Train Split: "https://www.openslr.org/resources/12/train-clean-100.tar.gz"

[8] Dev Split: "https://www.openslr.org/resources/12/dev-clean.tar.gz"

[9] Test Split: "https://www.openslr.org/resources/12/test-clean.tar.gz"

sono in questo caso in numero maggiore dato il task TS-ASR. Oltre a quelli mostrati precedentemente ci sono anche i campi:

- "**ad_code**": stringa compressa che codifica quali utterance usare per costruire la frase ausiliaria
- "**int_code**": stringa compressa che codifica quali utterance usare costruire la frase dello speaker interferente
- "**sir**": valore di SIR da impiegare nella miscelazione di target speaker speech e speech interferente.

Per la creazione di “ad_code” vengono sorteggiate varie utterance del target speaker fino a raggiungere una durata di almeno 15 secondi. Poi sono calcolati degli estremi di taglio casuali tali che la durata sia esattamente 15 secondi. La stringa “ad_code” conterrà alla fine delle informazioni compresse su quali utterance impiegare e dove tagliarle una volta concatenate così da ottenere la frase ausiliaria.

Per la creazione di “int_code” viene sorteggiato uno speaker diverso dal target speaker ma proveniente dallo stesso split. Di questo speaker vengono sorteggiate varie utterance fino a raggiungere una durata pari almeno a quella del target speaker speech. Poi sono calcolati degli estremi di taglio casuali tali che la durata sia esattamente quella del target speech. La stringa “int_code” conterrà alla fine delle informazioni compresse su quali utterance impiegare e dove tagliarle una volta concatenate, così da ottenere il segnale di speech interferente in modalità Full Overlap.

### 4.2.5 Pipelines

Per Pipeline si intende una funzione che a partire da un Data Manifest File restituisce un DynamicItemDataset, ovvero un oggetto che racchiude in sé, tutti i metodi necessari alla produzione on demand, del batch di tensori di ingresso e di target per il training di un modello a reti neurali.

Per ogni split preparato è stato creato un DynamicItemDataset dedicato ottenendo i quattro dataset di: training, validation, development e test.

Tutti i dataset prodotti sono stati creati facendo sì che i datapoints che costituiscono ogni batch siano scelti a caso, senza ripetizioni. Nel caso del dataset di training viene fatta una eccezione. I datapoints in questo dataset vengono infatti ordinati in funzione della durata ed raccolti in batch in modo contiguo essi hanno tutti durata temporale simile. In questo modo i batch creati contengono mediamente meno zero padding lungo l'asse temporale rispetto un qualsiasi batch casuale. Come risultato si hanno sequenze con lo stesso contenuto informativo, ma in media più brevi, quindi più veloci da elaborare e con meno requisiti di memoria.

### 4.2.6 Pipelines - Clean Baseline

Per il modello Clean Baseline, la pipeline è di tipo ASR, quindi piuttosto elementare e permette di accedere nell'apprendimento ai seguenti dati di ingresso batch:

- "**id**": identificatore univoco dell'utterance
- "**target_signal**": tensore di clean speech
- "**words**": trascrizione target del clean speech
- "**tokens_bos**": tensore di tokens con in testa il beginning of sentence
- "**tokens_eos**": tensore di tokens con in coda il beginning of sentence
- "**tokens**": tensore target di tokens

### 4.2.7 Pipeline - Adaptive Encoder

In questo caso la pipeline deve fornire maggiori informazioni per task TS-ASR e si diversifica in base all'uso o meno del metodo MTL.

In ottica TS-ASR la pipeline assolve a due nuove funzioni: la creazione dello speech interferente e della frase di adattamento. Entrambi i segnali vengono creati decodificando le stringhe compresse "ad_code" ed "int_code" presenti nei Data Manifest File. Dalla decodifica emergono i nomi delle utterance da concatenare e in quale punto del tempo tagliarle per produrre un segnale di lunghezza corretta. La pipeline carica quindi queste utterance, le concatena e le taglia.

Nella creazione del segnale mistura la pipeline modifica in ampiezza il target speaker speech ed il segnale interferente in accordo con il SIR di miscelazione, poi i due segnali salati vengono sommati.

Ne caso privo di MTL i campi informativi forniti dalla pipeline nel batch sono:

- "id", "words", "tokens_bos", "tokens_eos", "tokens": come nel caso precedente
- "sir": indica il SIR con cui miscelare i segnali nella mistura
- "mixture_signal": il tensore del segnale mistura
- "ad_signal": tensore del segnale mistura

Ne caso con MTL i campi informativi forniti dalla pipeline nel batch sono identici al caso con MTL ma includono anche "target_signal" per accedere al clean speech nel percorso di segnale pulito.

### 4.2.8 Modelli

Ogni modello è stato implementato con una classe propria, contenente il modello stesso ed alcuni metodi primitivi per le routine di training ed evaluation. Le classi create ereditano dalla classe Brain di SpeechBrain espandendone o sovrascrivendone alcune funzionalità. L'espansione è ottenuta con la scrittura di nuovi metodi, mentre la sovrascrittura avviene attraverso l'override del metodo di partenza.

Tra i metodi introdotti in questo lavoro di tesi vale la pena di menzionare "ping_probes", un metodo utile in fase di debug poiché permette di verificare che pesi del modello in apprendimento vengano aggiornati ad ogni passo di ottimizzazione.

#### *4.2.8.1 Tokenizer*

Nel framework impiegato, i testi vengono elaborati usando la codifica Byte Pair Encoding (BPE in breve) a 1000 BPE tokens attraverso un Tokenizer di SpeechBrain già istruito e derivato dall'algoritmo SentencePiece. Essendo una componente relativamente complessa, ne viene usata una versione preistruita disponibile su Hugging Face.

#### *4.2.8.2 Embeddings*

Per semplificare la sperimentazione è stata introdotta la facoltà di impiegare token embedding preistruiti. Gli embedding sono stati estratti da un modello[10] ASR encoder decoder disponibile su Hugging Face.

#### *4.2.8.3 Data Augmentation*

Tra le tecniche di data augmentation disponibili sono state implementate: speed pertub, frequency dropout, time dropout, environmental corruption e double batching (la creazione di un batch comprendente sia i segnali puliti che le versioni aumentate). L'implementazione di questi metodi si basa su TimeDomainSpecAugment di SpeechBrain.

#### *4.2.8.4 Learning Rate Scheduling*

L'apprendimento dei modelli ASR prevede l'impiego di Learning Rate Annealing per mezzo dell'algoritmo NewBobScheduler di SpeechBrain. Questo tipo di Learning Rate Scheduler è usato per ridurre in modo proporzionale il learning rate quando per un certo numero di epoche consecutive, nella fase di validation, il validation WER fallisce nel migliorare.

#### *4.2.8.5 Early Stopping*

L'apprendimento dei modelli ASR viene interrotto sia nel caso in cui siano state processate tutte le epoche richieste, sia quando si verifica Early Stopping. Il metodo di Early Stopping impiegato interrompe l'apprendimento quando per un certo numero di epoche consecutive, nella fase di validation, il validation WER fallisce nel migliorare.

#### *4.2.8.6 Checkpointing*

Al termine di ogni epoca in fase di validation, un metodo di checkpointing si occupa di salvare su disco rigido una copi dei pesi del miglior modello mai valutato nel corso dell'apprendimento. Lo stesso sistema è usato in fase evaluation per caricare dal disco rigido i pesi migliori ed usarli per inizializzare

[10] Embeddings: "www.huggingface.co/speechbrain/asr-crdnn-rnnlm-librispeech/tree/main"

il modello ASR da valutare. La metrica usata per confrontare le varie versioni dei modelli è il validation WER.

#### *4.2.8.7 Modello linguistico*

Nella fase di evaluation viene usato un modello linguistico in shallow fusion. L'architettura del modello è di tipo RNNLM e ne viene usata una versione preistruita di SpeechBrain.

#### *4.2.8.8 Backpropagation*

La retropropagazione viene gestita interamente da PyTorch con l'apposito comando. La sua implementazione è trasparente, poiché realizzata nei metodi di base della classe Brain ed ereditata dai modelli su di essa creati.

### 4.2.9 Modelli - Clean Baseline

La classe creata per questo modello prende il nome di "E2E_ASR_module_for_single_speaker".

Il modello implementa il sistema encoder VGG-BLSTM del riferimento, un decoder LSTM attenzionato con Location Aware Attention ed un modello linguistico RNN-LM in shallow fusion.

La decodifica in fase di validation e di test avviene attraverso Beam Search sul decoder attenzionato ma con parametrizzazioni leggermente diverse. Al fine di rendere la decodifica di validation più breve e quella di test più accurata, il beam size di validation è scelto in modo da essere minore di quello test.

L'algoritmo di ottimizzazione usato è AdaDelta.

L'elaborazione in avanti compiuta dal modello nella fase di training è la seguente:

- Il batch in ingresso "target_signal" subisce eventuale Data Augmentation.

- Dal target_signal eventualmente aumentato vengono estratte le feature log-Mel Spectrogram di nome “feats”
- Avviene la normalizzazione delle feature in media e varianza su base utterance delle features “feats”
- Le feature sono elaborate dall’modulo encoder producendo il tensore “encoded_signal”
- I “tokens_bos” sono embeddati con nome “embedded_tokens_bos”
- Viene attuata la decodifica con il decoder attenzionato prendendo in ingresso “encoded_signal” e ”embedded_tokens_bos” e ritornando in uscita le feature “decoder_outputs”
- Uno strato lineare di uscita “seq_lin” con attivazione logsoftmax converte “decoder_outputs” nelle log-likelihood sequence-to-sequence di predizione “seq_logprobs”
- Uno strato lineare di uscita “ctc_lin” con attivazione logsoftmax converte l’uscita dell’encoder “encoded_signal” nelle “ctc_logprobs”

Nelle fasi di test e validation sono compiute le stesse operazioni ma viene inoltre effettuata una decodifica su decoder attenzionato con beamsearch per ritornare la sequenza di tokens predetti dal sistema.

Per calcolare la funzione costo il modello calcola il contributo di NLL Loss tra le log-likelihood “seq_logprobs” ed il target “tokens_eos”. Poi calcola il contributo di funzione costo CTC sulle log-likelihood CTC “ctc_logprobs” ed il target “tokens”. I due contributi sono infine sommati per ottenere la loss composita.

### 4.2.10 Modelli - Adaptive Encoder

La classe creata per questo modello prende il nome di "E2E_SpeakerBeam_Adaptive_Encoder".

Il modello implementa il sistema encoder VGG-BLSTM con Adaptation Layer e Auxiliary Network del riferimento, un decoder LSTM attenzionato con Location Aware Attention ed un modello linguistico RNN-LM in shallow fusion.

La decodifica in fase di validation e di test avviene attraverso Beam Search sul decoder attenzionato ma con parametrizzazioni leggermente diverse. Al fine di rendere la decodifica di validation più breve e quella di test più accurata, il beam size di validation è scelto in modo da essere minore di quello test.

L'algoritmo di ottimizzazione usato è AdaDelta.

L'elaborazione in avanti compiuta dal modello nella fase di training dipende dall'uso o meno del metodo MTL.

***Elaborazione in Avanti Senza MTL***

Nel caso senza MTL le elaborazioni in avanti compiute dal modello nella fase di training sono le seguenti:

- I "tokens_bos" sono embeddati con nome "embedded_tokens_bos"
- Dal "mixture_signal" eventualmente aumentato vengono estratte le feature log-Mel Spectrogram di nome "mixture_feats"
- Avviene la normalizzazione delle feature in media e varianza su base utterance delle features "mixture_feats"
- Dalla frase di adattamento eventualmente elaborata con Data Augmentation sono estratte le feature di interesse.
- Le feature della mistura sono elaborate dall'modulo encoder producendo il tensore "encoded_mixture"

- Viene attuata la decodifica con il decoder attenzionato prendendo in ingresso “encoded_mixture” e ”embedded_tokens_bos” e ritornando in uscita le feature “decoder_outputs”
- Uno strato lineare di uscita “seq_lin” con attivazione logsoftmax converte “decoder_outputs” nelle log-likelihood sequence-to-sequence di predizione “seq_logprobs”
- Uno strato lineare di uscita “ctc_lin” con attivazione logsoftmax converte l’uscita dell’encoder “encoded_mixture” nelle “ctc_logprobs”

***Elaborazione in Avanti Con MTL***

Oltre alla elaborazione precedente, l’uso di MTL richiede operazioni aggiuntive:

- Dal target_signal eventualmente aumentato vengono estratte le feature log-Mel Spectrogram di nome “target_feats” e vengono normalizzate in media e varianza a livello utterance.
- Le feature sono elaborate dall’modulo encoder producendo il tensore “encoded_target”
- Viene attuata la decodifica con il decoder attenzionato prendendo in ingresso “encoded_target” e ”embedded_tokens_bos” e ritornando in uscita le feature “decoder_outputs”
- Uno strato lineare di uscita “seq_lin” con attivazione logsoftmax converte “decoder_outputs” nelle log-likelihood sequence-to-sequence di predizione “seq_logprobs”

- Uno strato lineare di uscita “ctc_lin” con attivazione logsoftmax converte l’uscita dell’encoder “encoded_mixture” nelle “ctc_logprobs”

***Evaluation***

Nelle fasi di test e validation sono compiute le stesse operazioni ma viene inoltre effettuata una decodifica su decoder attenzionato con beamsearch per ritornare la sequenza di tokens predetti dal sistema.

***Funzione Costo***

Nel caso senza MTL la procedura è la stessa del modello Clean Baseline ma usando le feature di mistura invece che di clean speech.

Quando viene usato MTL, una volta calcolato il contributo il contributo di loss senza MTL, a questo viene sommato un contributo simile, ma prodotto con le feature di clean speech.

### 4.2.11 Ricerca dei Parametri

Data la somiglianza tra il riferimento ed i modelli implementati è stato ritenuto importante valutare qualche combinazione di parametri qualora questi siano ritenuti siano sati ritenuti importanti.

### 4.2.12 Ricerca dei Parametri – Clean Baseline

#### *01_Base_50*

Nella prima configurazione sperimentata è stato scelto un numero di epoche di apprendimento relativamente piccolo di 50 epoche. Impiegano AdaDelta, il comportamento del sistema si suppone essere non troppo sensibile alla scelta del valore di learning rate, che viene impostato prudentemente a $0.6$. Il learning rate annealing usa un fattore di riduzione di $0.8$ con un limite di $3$ epoche fallimentari. Cercando di partire da un punto di mezzo, inizialmente sono stati considerati $4$ strati BLSTM nell’encoder invece che $6$. Il batch size usato è pari a 16, l’unica forma di data augmentation impiegata è la Speed Perturb con velocità relative selezionate casualmente tra 90/100/110. I

moduli di embedding sono istruiti from scratch. Per sfruttare la possibilità della componente VGG dell'encoder di usare un numero maggiore di canali in ingresso, dal clean speech vengono estratte feature contenenti i contributi Delta e DeltaDelta. Per accomodare la dimensionalità aumentata, il primo strato convoluzionale dell'encoder possiede $3$ canali invece che $1$. Nella decodifica con Beam Search, il validation beam size è impostato a 20 ed il test beam size è impostato a 80. La curva di apprendimento ottenuta è riportata in figura. Il WER ottenuto sul development set è pari a 9.92%.

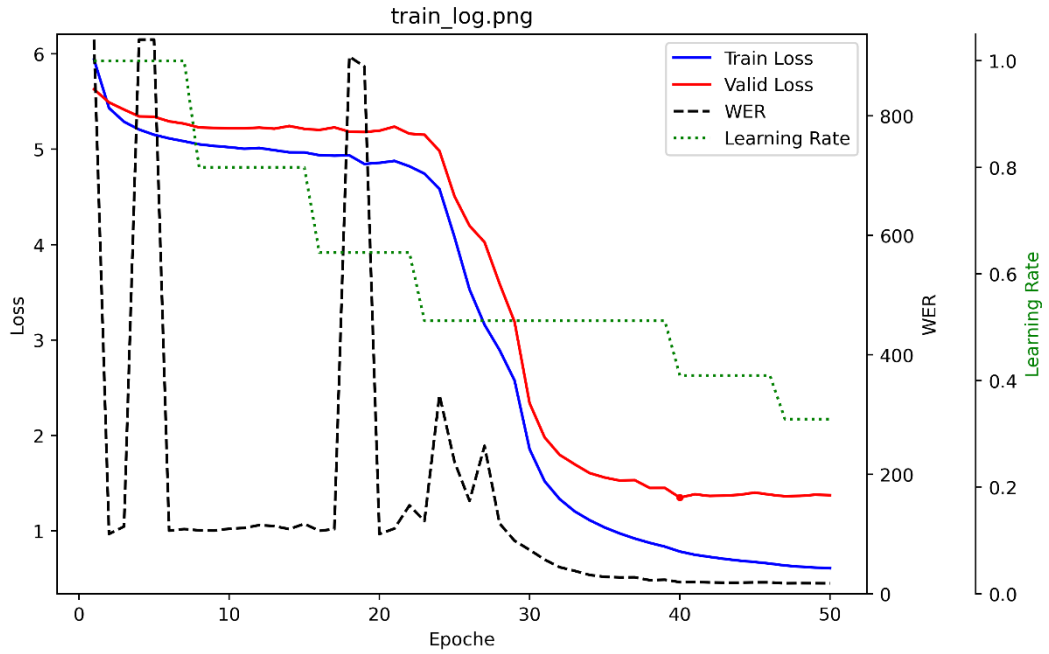



*Figura 21 Apprendimento 01_Base_50*

***02_Base_100***

Dato che la configurazione precedente ha avuto un buon andamento è stato aumentato il numero di epoche di apprendimento portandolo a 100. L'andamento è stato confermato e le prestazioni migliorate leggermente. Il WER ottenuto sul development set è pari a 8.42%.

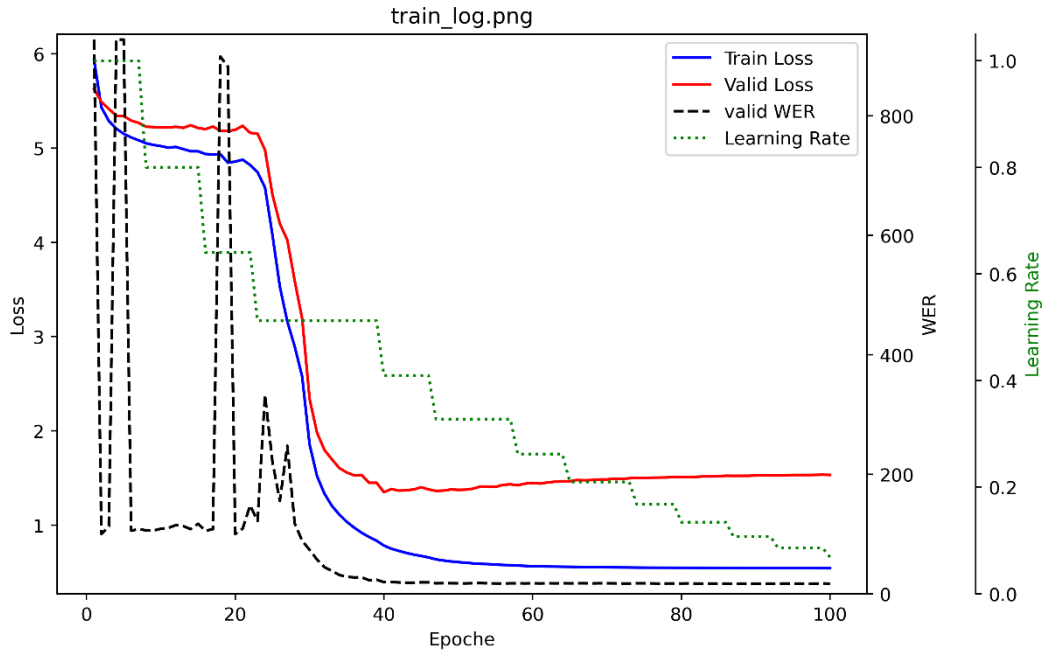



*Figura 22 Apprendimento 02_Base_100*

### *03_Base_100_NoDelta*

Per valutare l'influenza delle feature Delta e DeltaDelta queste sono state tolte in questa configurazione. L'effetto ottenuto non è positivo, il dev WER è pari a 8.73%.

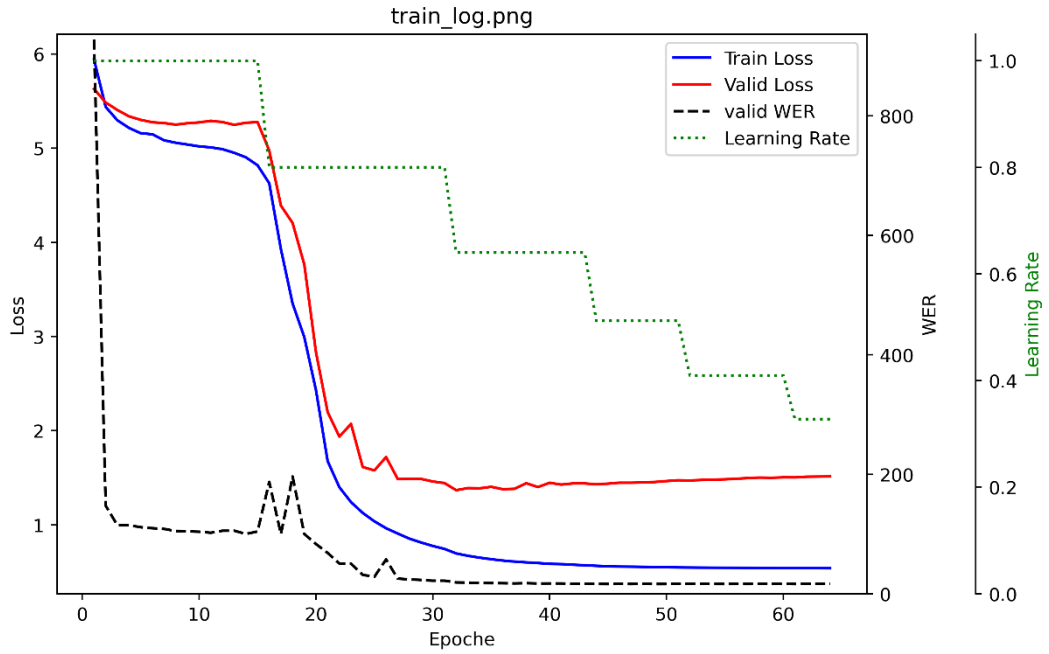



*Figura 23 Apprendimento 03_Base_100_NoDelta*

Per il momento le feature Delta vengono mantenute e viene esplorata l'influenza del numero di strati BLSTM all'encoder.

***04_Base_100_Layer_3***

In questo caso sono usati 3 strati BLSTM all'encoder, il dev WER è pari a 9.03%.

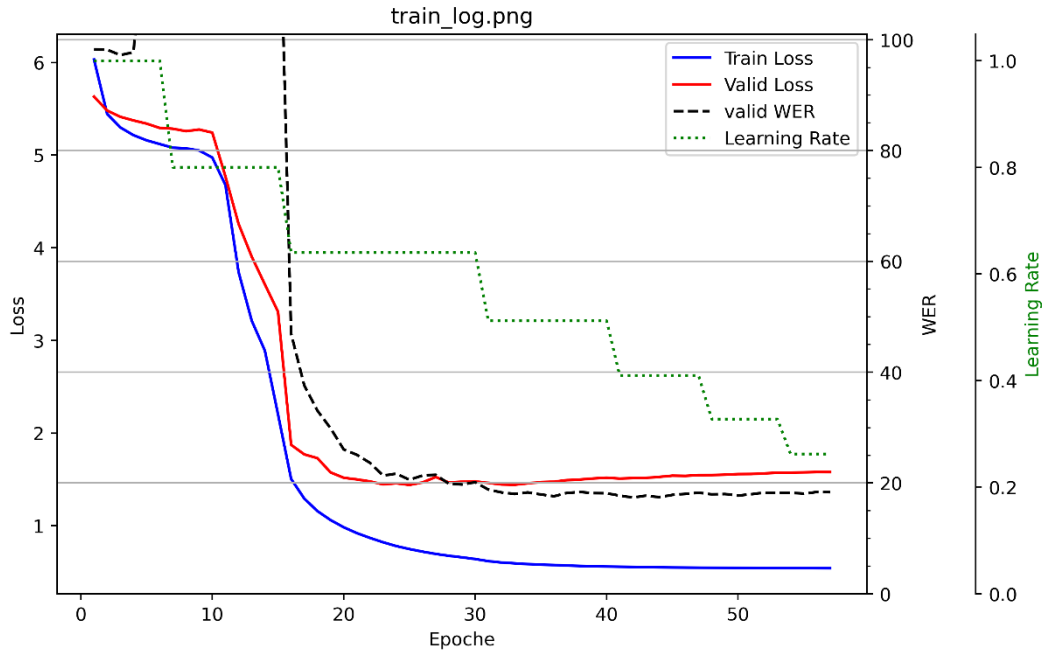



*Figura 24 Apprendimento 04_Base_100_Layer_3*

***05_Base_100_Layer_2***

Il numero di strati BLSTM all'encoder è ridotto a 2, ottenendo un dev WER di 10.19%.

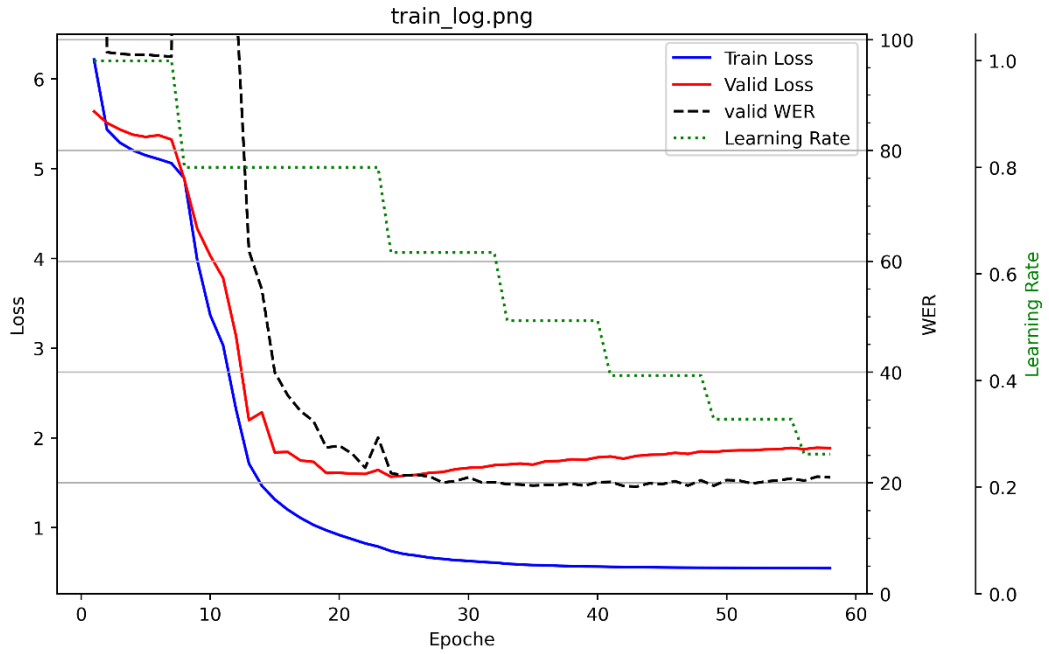



*Figura 25 Apprendimento 05_Base_100_Layer_2*

### *06_Base_100_Layer_5*

Data la tendenza peggiorativa del WER al ridurre del numero di strati, ora vengono portati a 5, ottenendo un dev WER di 100.54%.

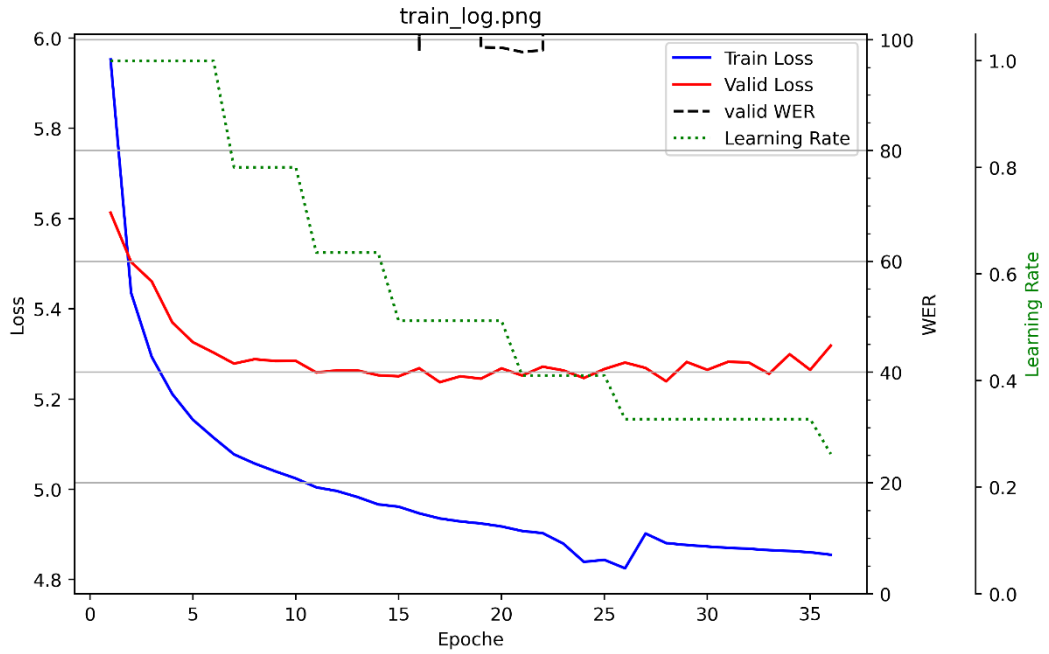



*Figura 26 Apprendimento 06_Base_100_Layer_5*

In questo caso l'apprendimento non è avvenuto, per cercare di favorire la convergenza vengono rimosse le feature Delta e DeltaDelta nel prossimo esperimento.

***07_Base_100_Layer_5_NoDelta***

Le feature Delta e DeltaDelta sono state eliminate ed a causa di errori catastrofici legati ai requisiti di memoria, il batch size è stato ridotto da 16 a 15. Il dev WER ottenuto è del 11.07%.

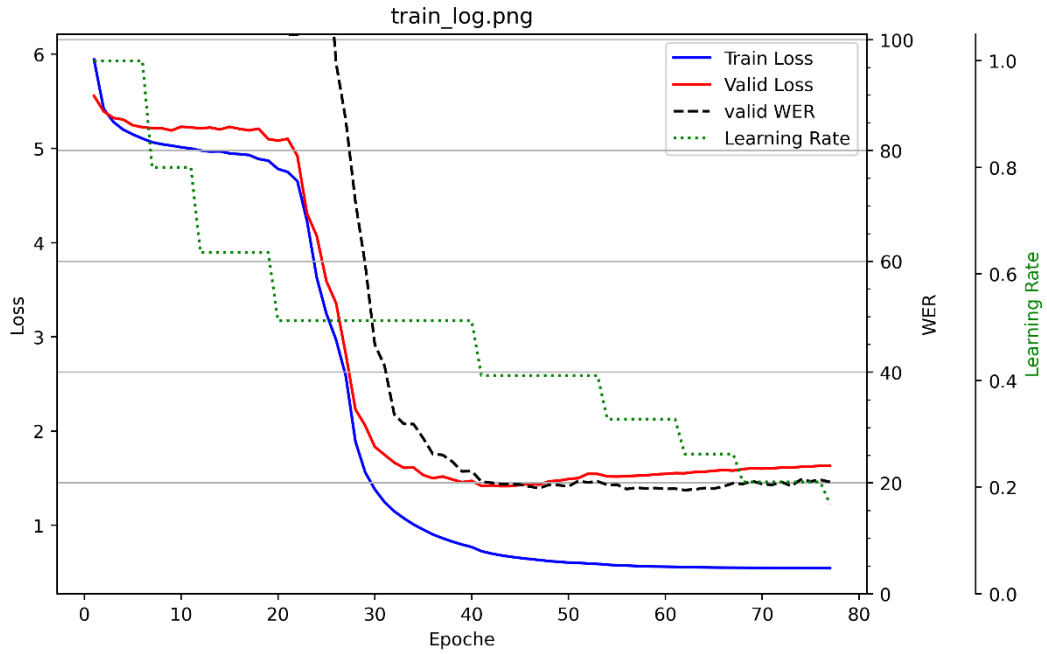



*Figura 27 Apprendimento 07_Base_100_Layer_5_NoDelta*

In questo caso la convergenza è avvenuta, ma non sono migliorate le prestazioni. Nelle sperimentazioni successive vengono di nuovo valutati modelli con meno strati BLSTM di encoder, ma senza l'uso di feature Delta e DeltaDelta.

***08_Base_100_Layer_3_NoDelta***

Qui gli strati BLSTM di encoder sono ridotti a 3 e non vengono usate le feature Delta, DeltaDelta. Il valid WER ottenuto è di 9.92%.

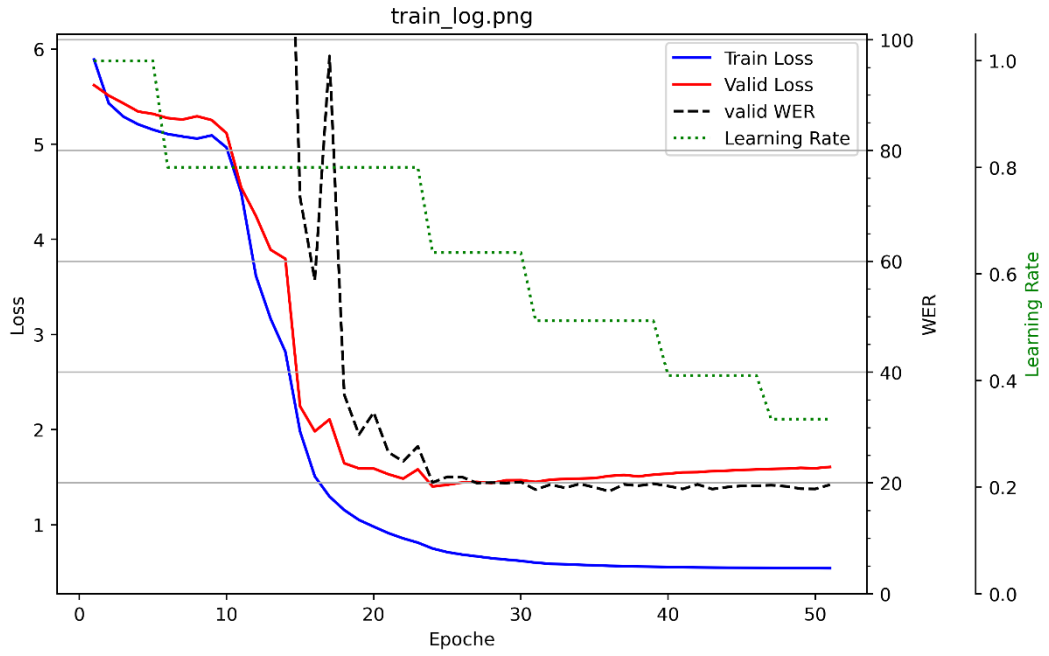



*Figura 28 Apprendimento 08_Base_100_Layer_3_NoDelta*

### *09_Base_100_Layer_2_NoDelta*

Qui gli strati BLSTM di encoder sono ridotti a 2 e non vengono usate le feature Delta, DeltaDelta. Il valid WER ottenuto è di 12.4%.

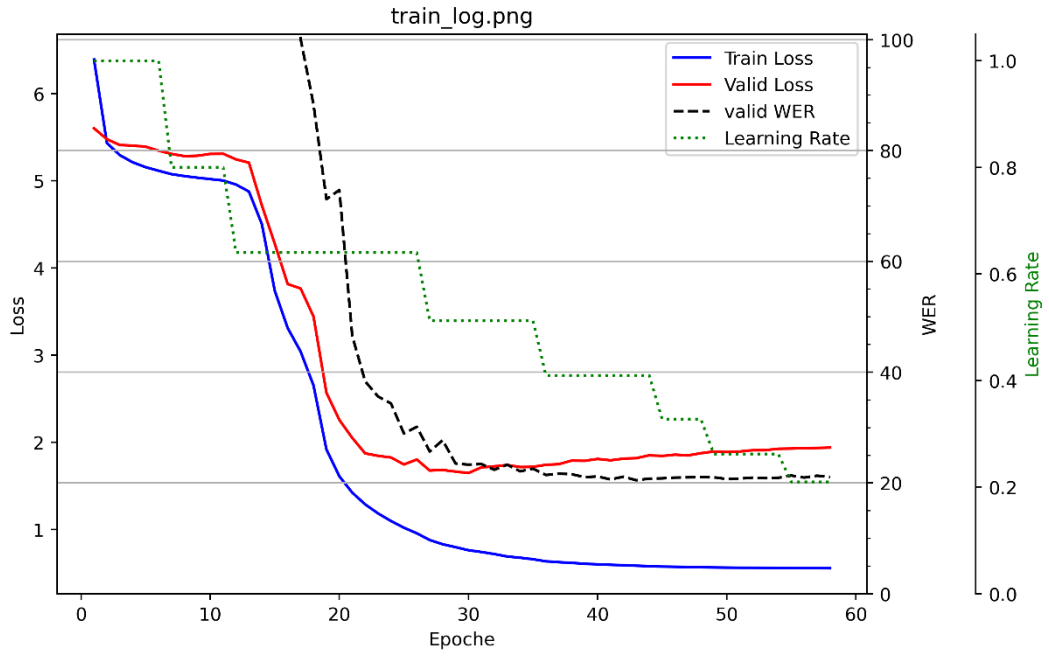



*Figura 29 Apprendimento 09_Base_100_Layer_2_NoDelta*

Fino a questo momento la seconda configurazione sperimentata 02_Base_100 risulta essere quella più performante. Da qui in avanti sono valutati differenti configurazioni di data augmentation.

***10_Base_100_NoAug***

La configurazione corrente è quella di 02_Base_100 ma senza data augmentation. Il dev WER ottenuto è 16.51%.

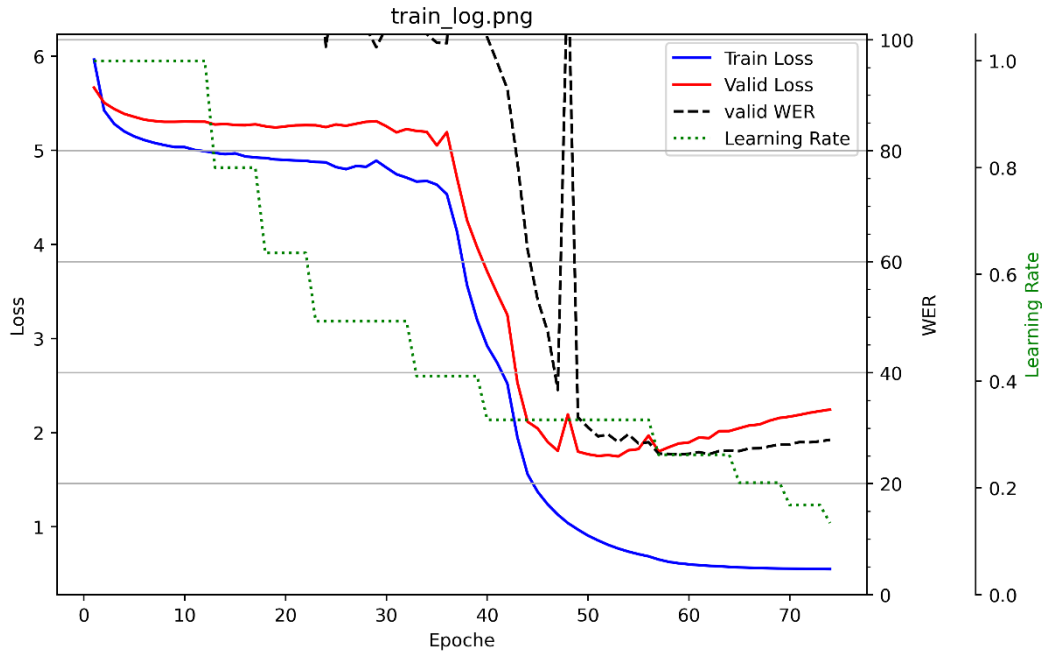



*Figura 30 Apprendimento 10_Base_100_NoAug*

***11_Base_100_DrFreq***

La configurazione corrente prevede solo Drop Frequency (frequency-dropout, parametrizzata di default) come data augmentation. Il dev WER ottenuto è 98.86%.

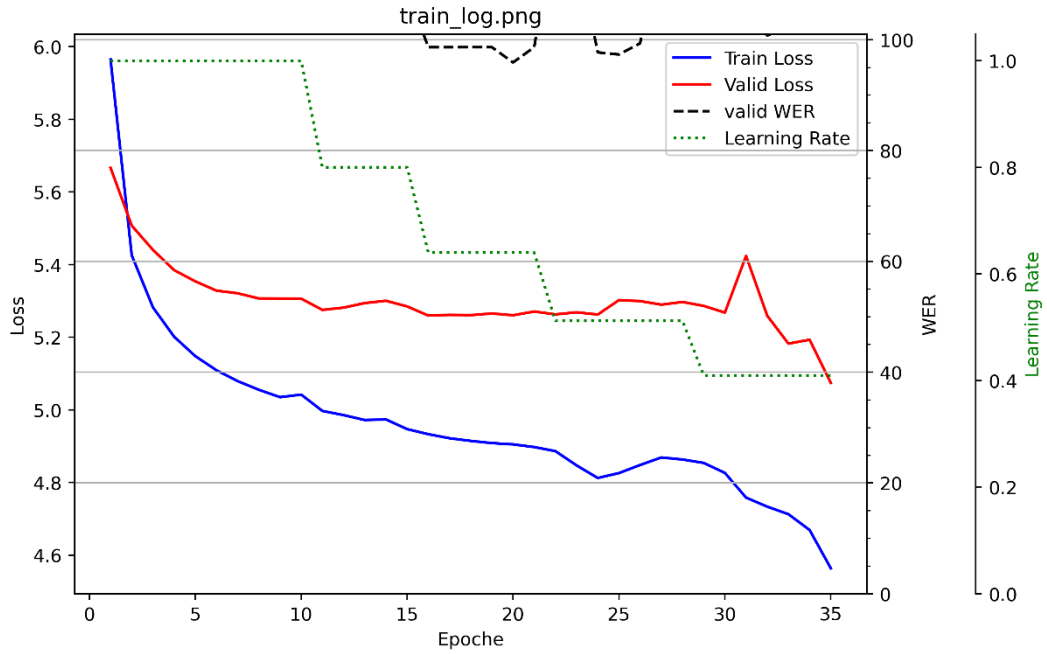



*Figura 31 Apprendimento 11_Base_100_DrFreq*

Il sistema fallisce nell'apprendere terminando per Early Stopping.

### *12_Base_100_DrChunk*

La configurazione corrente prevede solo Drop Chunk (time-dropout, parametrizzata di default) come data augmentation. Il dev WER ottenuto è 98.69%.

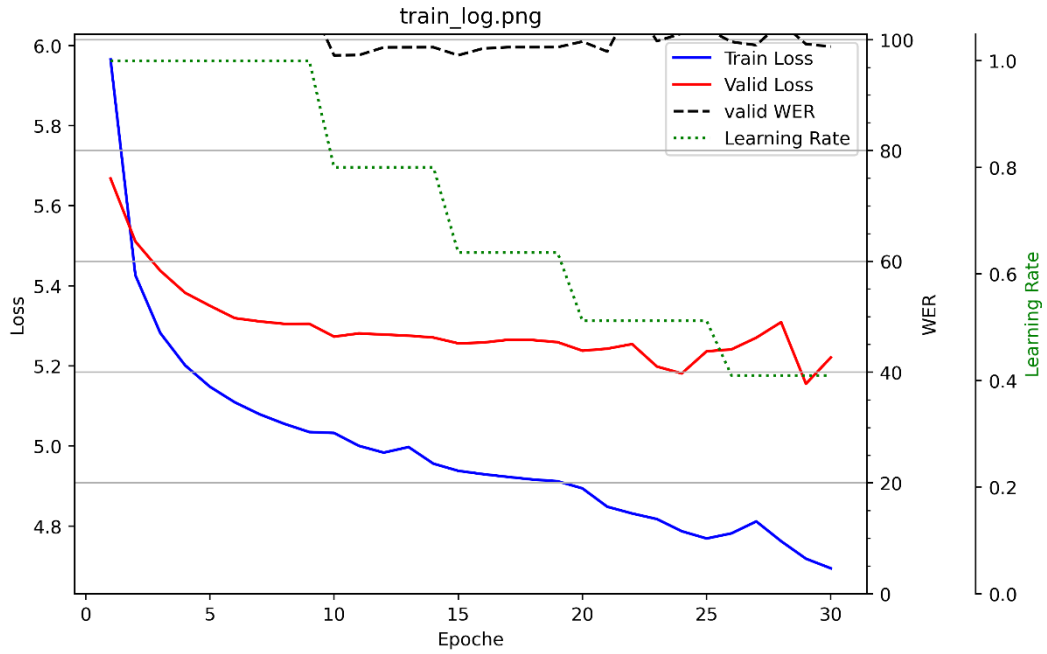



*Figura 32 Apprendimento 12_Base_100_DrChunk*

Il sistema fallisce nell'apprendere terminando per Early Stopping.

### *13_Base_100_EnvC*

La configurazione corrente prevede solo Environmental Corruption (nella forma di rumore additivo, parametrizzata di default) come data augmentation. Il dev WER ottenuto è 111.84%.

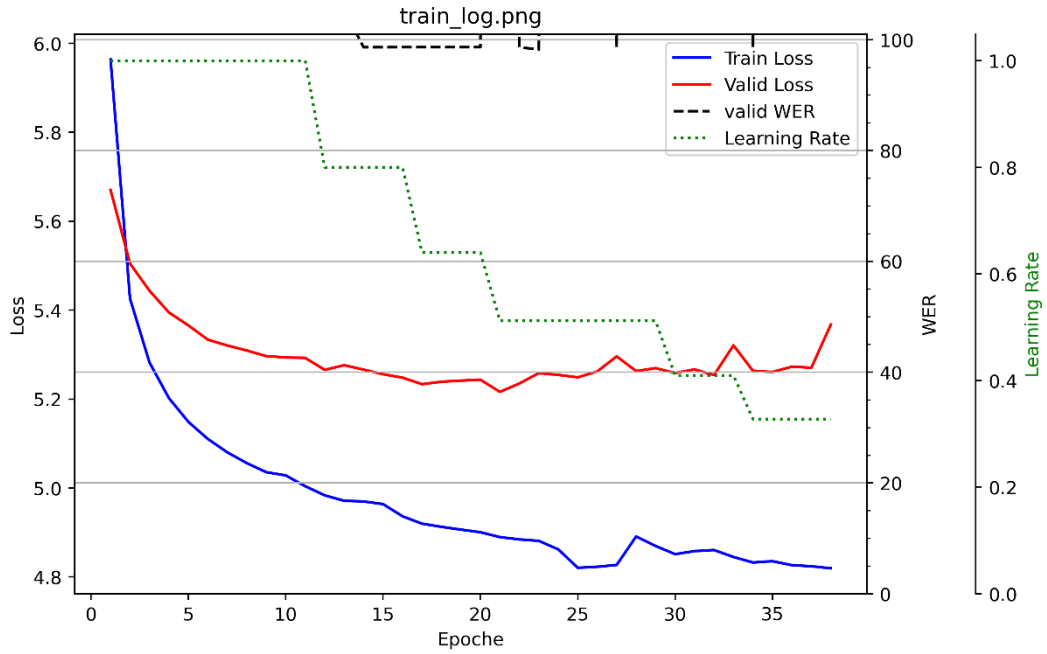



*Figura 33 Apprendimento 13_Base_100_EnvC*

La curva non mostra segni di apprendimento.

### *14_Base_100_EnvC_DB*

La configurazione corrente è identica alla precedente ma include una ulteriore forma di data augmentation, il double batching. Il dev WER ottenuto è 119.6%.

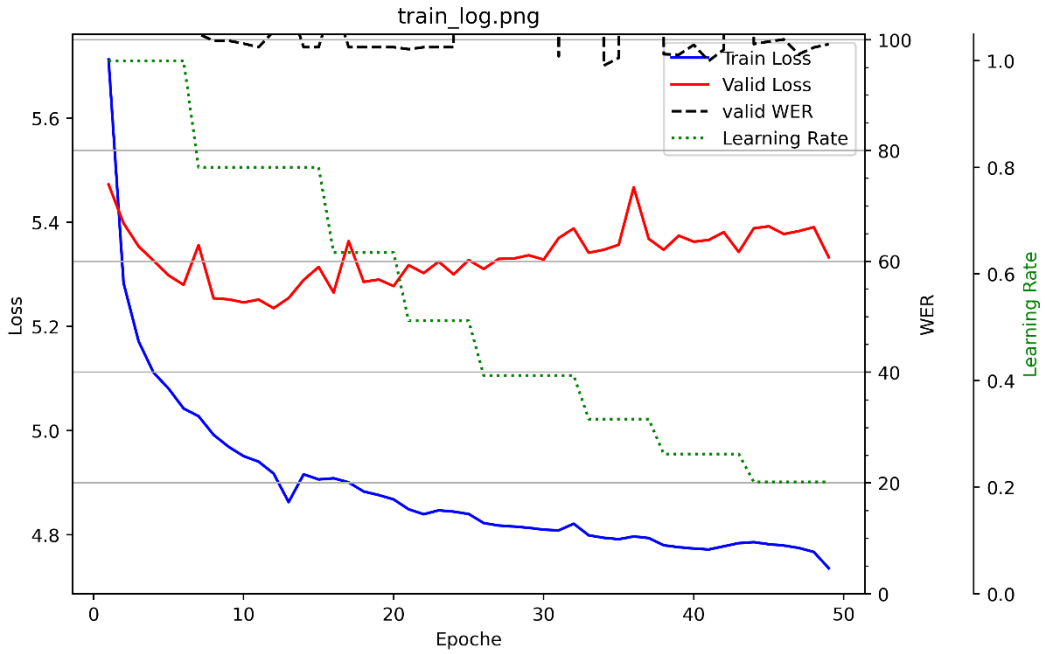



*Figura 34 Apprendimento 14_Base_100_EnvC_DB*

***15_Base_100_FullAug***

Con questa configurazione sono contemporaneamente impiegate tutte e quattro le forme di data augmentation: speed perturb, drop frequency, drop chunk, environmental corruption, double batching. Il de WER ottenuto è 104.21%

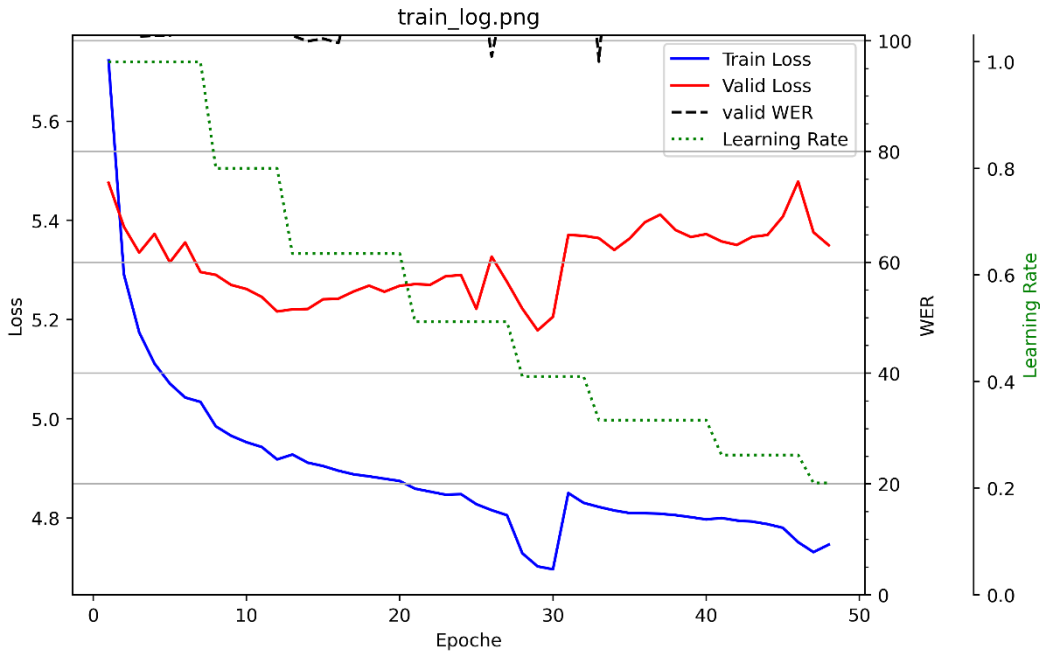



*Figura 35 Apprendimento 15_Base_100_FullAug*

Fino adora l'unica forma di data augmentation con effetto positivo è la speed perturb; quindi, da qui in avanti verrà impiegata solo quella. Fino ad ora è ancora la configurazione 02_Base_100 ad essere la migliore, quindi ora vengono esplorati alcuni parametri di decodifica.

***16_Base_100_BeamSearchOPT***

In questo caso è stata presa la configurazione migliore fino ad ora, ovvero la 02_Base_100 ed è stata eseguita una grid search su uno spazio parametrico ritenuto ragionevolmente grande e con passo fine. Ogni configurazione prodotta dalla grid search è stata applicata al checkpoint del modello 02_Base_100 ed è stata valutata sul dev set.

La migliore configurazione di decodifica ottenuta per il decoder attenzionato è la seguente:

- temperature=1.19
- temperature_lm=1.04
- lm_weight=0.44
- eos_threshold=1.05
- use_max_attention_shift=True
- max_attention_shift=117
- coverage_penalty=1.44

Il dev WER ottenuto è pari a 8.2%.

Il test WER ottenuto è pari a 8.97%.

### 4.2.13 Ricerca dei Parametri – Adaptive Encoder

In questa seconda serie di esperimenti la ricerca di parametri tali da garantire prestazioni accettabili è stata infruttuosa.

Sono state esplorate combinazioni con numero di strati BLSTM di encoder da 1 a 6, con e senza MTL, è stato variato il learning rate un ordine di grandezza alla volta, è stato variato il numero di neuroni presenti negli strati BLSTM da 1024 a 512, è stato incluso ed escludo il subsampling temporale nell'encoder, variato la pazienza nel learning rate annealing da 3 a 6, impiegati strati di embedding preistruiti e non.

Tipicamente il modello non riusciva a convergere o divergeva. Nei casi più fortunati è avvenuto un apprendimento molto scarso.

| Modello | Learning Rate | Learning Rate Annealing | BSLTM Neurons | Strati BLSTM | Sub Sampling | DEV WER [%] |
|---|---|---|---|---|---|---|
| L3 | 1.0 | 0.8/3 | 1024 | 2 | | 41.56% |
| L3 Bis | 1.0 | 0.8/3 | 1024 | 4 | SI | 58.26 |
| L4 (L3) | | | | | | 45.82 |
| L5 (L4) | | | | | | 40.47 |
| L6 (L5) | | | | | | 38.83 |

*Figura 36 Selezione delle configurazioni migliori*

La configurazione migliore è stata ottenuta con i seguenti parametri di base:

- SIR max di miscelazione=5 dB
- Data Augmentation= Speed Perturb
- MTL=False
- Label Smoothing=0.1
- Delta, DeltaDelta=False
- Batch Size=16
- Embedding=Preistruiti

Una volta istruito il modello con 3 strati BLSTM con dev WER del 41.56%, è stato istruito un modello a 4 strati dove i primi 3 sono stati inizializzati con pesi del modello precedente a 3 strati. Cosi facendo sono stati istruiti i modelli a 5 e 6 strati partendo rispettivamente dai pesi dei modelli a 4 e 5 strati.
Il test WER ottenuto con la configurazione L6 (L5) costruita con i 5 strati di L5 (L4) è 39.12%

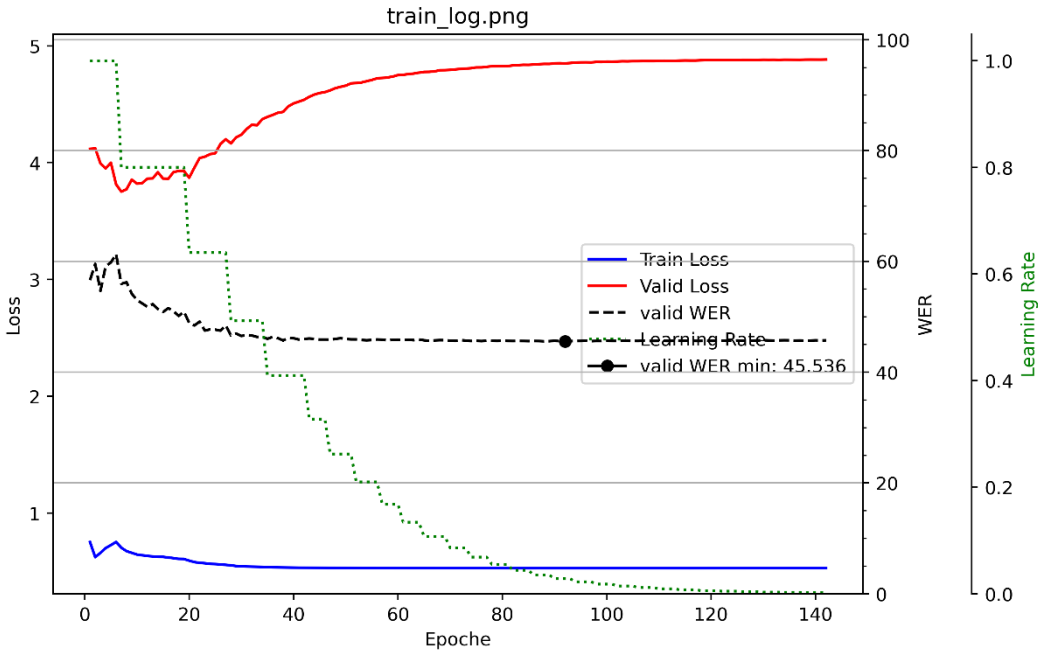



*Figura 37 Apprendimento di L6 preistruito con L5*

## 4.3 RISULTATI

L'implementazione Clean Baseline dell'omonimo modello di SpeakerBeam ottiene un test WER pari a 8.97 %.

L'implementazione Adaptive Encoder dell'omonimo modello di SpeakerBeam ottiene invece un test WER pari a 39.12%.

## 4.4 CONCLUSIONI

Sono stati implementati due modelli per applicazioni ASR.

Il primo modello ispirato alla variante Clean Baseline di SpeakerBeam per applicazioni ASR è stato istruito ed ottimizzato in uno spazio di parametri ritenuto ragionevole. Le prestazioni ottenute si discostano di circa 4.5 punti percentuali rispetto al risultato ottenuto nel riferimento (Delcroix M. W., 2019). Tenendo conto tuttavia delle differenze di implementazione, ed alla luce della datazione del tipo di tecnologia implementata, si può ritenere questo risultato come ragionevolmente accettabile e non privo di margini di miglioramento.

Per il secondo modello ispirato alla variante Adaptive Encoder di SpeakerBeam per applicazioni TS-ASR è stato tentato l'apprendimento con molteplici modalità e configurazioni. Nonostante siano state valutate numerose possibilità non è stata rinvenuta una configurazione che mostrasse un comportamento degno di nota. Le prestazioni ottenute sono insufficienti.

# 5 RIFERIMENTI


*A Gentle Introduction to torch.autograd*. (s.d.). Tratto da pytorch: https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

A. Graves, S. F. (2006). Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. *Proceedings on International Conference on Machine Learning*, 369-376.

A. Vaswani, N. S. (2017). Attention Is All You Need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010.

Anmol Gulati, J. Q.-C. (2000). Conformer: Convolution-augmented transformer for speech recognition. *Interspeech*.

B.H. Juang, L. R. (2004). Automatic Speech Recognition – A Brief History of the Technology.

*Beam Search*. (s.d.). Tratto da Wikipedia: https://en.wikipedia.org/wiki/Beam_search

*BeginnersGuide/Overview*. (s.d.). Tratto da wiki.python.org: https://wiki.python.org/moin/BeginnersGuide/Overview

Chang, X. Q. (2019). End-to-end monaural multi-speaker ASR system without pretraining. *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6256-6260.

*Continuous Speech Recognition*. (s.d.). Tratto da sciencedirect: https://www.sciencedirect.com/topics/engineering/continuous-speech-recognition#:~:text=is%20being%20encoded.-,Continuous%20speech%20recognition,without%20enforced%20pauses%20between%20words.

D. B. Fry, P. D. (1959). The Design and Operation of the Mechanical Speech Recognizer at University College London.

D. Povey, A. G. (2011). The Kaldi speech recognition toolkit. *ASRU*.

Dai, Z. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Dauphin, Y. N. (2017). Language modeling with gated convolutional networks. *International conference on machine learning*, 933-941.

Davis, K. H., Biddulph, R., & Balashek, S. (1952). Automatic Recognition of Spoken Digits.

Delcroix, M. W. (2019). End-to-End SpeakerBeam for Single Channel Target Speech Recognition. *In Interspeech*, 451-455.

Delcroix, M. Z. (2018). Single channel target speaker extraction and recognition with speaker beam. *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 5554-5558.

Delcroix, M. Z. (2019). Compact network for speakerbeam target speaker extraction. *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6965-6969.

E. Voita, D. T. (2019). Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*.

Ephrat, A. M. (2018). Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation. *arXiv preprint arXiv:1804.03619*.

F. Jelinek, L. R. (1975). Design of a Linguistic Statistical Decoder for the Recognition of Continuous Speech.

Ferguson, J. D. (1980). Hidden Markow Model Analysis: An Introducition.

*Formante (acustica)*. (s.d.). Tratto da Wikipedia: https://it.wikipedia.org/wiki/Formante_(acustica)#:~:text=Con%20formante%20si%20intende%20una,mostra%20un%20picco%20di%20ampiezza.

Garofolo, J. S. (1993). *CSR-I (WSJ0) Complete*. Tratto da catalog.ldc.upenn: https://catalog.ldc.upenn.edu/LDC93S6A

Graves, A. (2006). NLTK: The Neural Language Toolkit. *Proc. Joint Conf. Int. Committee Comput. Linguist. Assoc. Comput. Linguist. Interact. Presentat. Sessions*, 69-72.

Graves, A. (2012). *Sequence Transduction with Recurrent Neural Networks*. Tratto da arxiv.org: https://arxiv.org/abs/1211.3711

Guo, P. C. (2021). Multi-speaker ASR combining non-autoregressive conformer CTC and conditional speaker chain. *arXiv preprint arXiv:2106.08595*.

H. Bourland, N. M. (1994). Connectionist Speech Recognition: A Hybrid Approach.

Han, W. Z. (2020). ContextNet: Improving convolutional neural networks for automatic speech recognition with global context. *arXiv preprint arXiv:2005.03191*.

Hayes, K. (2019, 9 16). *why-pycharm-is-becoming-important-for-every-python-programmer*. Tratto da www.webprecious.com: https://www.webprecious.com/why-pycharm-is-becoming-important-for-every-python-programmer/

Hori, T. C. (2018). End-to-end speech recognition with word-based RNN language models. *2018 IEEE spoken language technology workshop (SLT)*, 389-396.

*Introduction to PyTorch*. (s.d.). Tratto da pytorch: https://pytorch.org/tutorials/beginner/introyt/introyt1_tutorial.html

Itakura, F. (1975). Minimum Prediction Residual Principle Applied to Speech Recognition.

J. Chorowski, D. B. (2015). Attention-Based Models for Speech Recognition. *Advances in Neural Information Processing Systems*, 577-585.

J. Chorowski, N. J. (2017). Towards better decoding and language model integration in sequence to sequence models. *Proceedings Interspeech*, 532-527.

J. Sakai, S. D. (s.d.). The Phonetic Typewriter. 1962.

J. W. Forgie, C. D. (1959). Results Obtained from a Vowel Recognition Computer Program.

K.-C. Jim, C. L. (1996). An analysis of noise in recurrent neural networks: convergence and generalization. *IEEE Transactions on neural networks, vol. 7, no. 6,*, 1424–1438.

Kanda, N. G. (2020). Joint speaker counting, speech recognition, and speaker identification for overlapped speech of any number of speakers. *arXiv preprint arXiv:2006.10930*.

Kanda, N. G. (2020). Serialized output training for end-to-end overlapped speech recognition. *arXiv preprint arXiv:2003.12687*.

Kanda, N. W. (2022). Streaming multi-talker ASR with token-level serialized output training. *arXiv preprint arXiv:2202.00842*.

Kanda, N. Y. (2021). End-to-end speaker-attributed ASR with Transformer. *arXiv preprint arXiv:2104.02128*.

Ko, T. P. (2015). Audio augmentation for speech recognition. *Interspeech*, 3586.

Kolbæk, M. Y. (2017). Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 1901-1913.

Koluguri, N. R. (2022). Titanet: Neural model for speaker representation with 1d depth-wise separable convolutions and global context. *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8102-8106.

L. Dong, S. X. (2018). Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 5884–5888.

Le Roux, J. W. (2019). SDR–half-baked or well done? *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 626-630.

Loshchilov, I., & Hutter, F. (2019). DECOUPLED WEIGHT DECAY REGULARIZATION. *ICLR*. Tratto da arxiv.org.

Lowerre, B. (1986). The HARPY Speech Understanding System.

Lu, Y. L. (2019). Understanding and improving transformer from a multi-particle dynamic system point of view. *arXiv preprint arXiv:1906.02762*.

*Modello di Markov nascosto*. (s.d.). Tratto da Wlipedia: https://it.wikipedia.org/wiki/Modello_di_Markov_nascosto

Moriya, T. S. (2022). Streaming target-speaker ASR with neural transducer. *arXiv preprint arXiv:2209.04175*.

Narayanan, A. &. (2013). Ideal ratio mask estimation using deep neural networks for robust speech recognition. *2013 IEEE international conference on acoustics, speech and signal processing*, 7092-7096.

Panayotov, V. C. (2015). Librispeech: an asr corpus based on public domain audio books. *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 5206-5210.

Park, D. S. (2019). Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*.

*PyTorch*. (s.d.). Tratto da wikipedia: https://en.wikipedia.org/wiki/PyTorch

Q. Song, B. S. (2022). Multimodal sparse transformer network for audio-visual speech recognition. *IEEE Transactions on Neural Networks and Learning Systems*.

R. Hershey, J. C. (2015). Deep clustering: Discriminative embeddings for segmentation and separation. *arXiv:1508.04306*.

Ramachandran, P. Z. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Ravanelli, M. e. (s.d.). *ASRfromScratch.ipynb*. Tratto da colab.research.google.com: https://colab.research.google.com/drive/1aFgzrUv3udM_gNJNUoLaHIm78QHtxdIz?usp=sharing#scrollTo=B-0KMtwsCXYn

S. E. Levinson, L. R. (1983). An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition.

S. Karita, N. C. (2019). A comparative study on transformer vs RNN in speech applications. *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 449–456.

S. Kim, T. H. (2017). Joint CTC-Attention based End-to-End Speech Recognition using Multi-task Learning. *Proceedings IEEE Interntional Conference on Acoustics, Speech and Signal Processing*, 4835-4839.

Simonyan, K. &. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sklyar, I. P. (2021). Streaming multi-speaker ASR with RNN-T. *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6903-6907.

*Speaker Diarisation*. (s.d.). Tratto da WIkipedia: https://en.wikipedia.org/wiki/Speaker_diarisation

*Speech Recognition*. (s.d.). Tratto da Speech Processing Book: https://speechprocessingbook.aalto.fi/Recognition/Speech_Recognition.html

*speechbrain*. (s.d.). Tratto da huggingface: https://huggingface.co/speechbrain

*speechbrain*. (s.d.). Tratto da speechbrain: https://speechbrain.github.io/

Subakan, C. R. (2021). Attention is all you need in speech separation. *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 21-25.

Veselý, K. W. (2016). Sequence summarizing neural network for speaker adaptation. *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 5315-5319.

W. Chan, N. J. (2016). Listen, Attend and Spell. *Proceedings IEEE International Conference on Acoustic Speech and Signal Processing*, 4960-4964.

Wang, D. &. (2018). Supervised speech separation based on deep learning: An overview. *IEEE/ACM transactions on audio, speech, and language processing*, 1702-1726.

Wang, Q. e. (2018). Voicefilter: Targeted voice separation by speaker-conditioned spectrogram masking. *arXiv preprint arXiv:1810.04826*.

Wang, Z. G. (2022). Semi-supervised time domain target speaker extraction with attention. *arXiv preprint arXiv:2206.09072*.

Watanabe S., H. T. (2017). Hybrid CTC/Attention Architecture for End-to-End Speech Recognition.

Watanabe, S. H. (2018). ESPnet: End-to-end speech processing toolkit. *arXiv preprint arXiv:1804.00015*.

Wu, J. X. (2019). Time domain audio visual speech separation. *2019 IEEE automatic speech recognition and understanding workshop (ASRU)*, 667-673.

Y. Luo, N. M. (2019). Conv-TasNet: Surpassing ideal time–frequency magnitude masking for speech separation. *IEEE/ACM transactions on audio, speech,*, 1256-1266.

Zhang, Y. P. (2023). Conformer-based target-speaker automatic speech recognition for single-channel audio. *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1-5.

Žmolíková, K. D. (2017). Learning speaker representation for neural network based multichannel speaker extraction. *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 8-15.

Žmolíková, K. D. (2019). Speakerbeam: Speaker aware neural network for target speaker extraction in speech mixtures. *IEEE Journal of Selected Topics in Signal Processing*, 800-814.