Technische Universität München
TUM School of Computation, Information and Technology

Enhancements for Hybrid and End-to-End Speech Recognition Architectures

Tobias Watzel

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. phil. nat. Sebastian Steinhorst

Prüfer der Dissertation:

1. Prof. Dr.-Ing. Gerhard Rigoll
2. Prof. Dr.-Ing. Tim Fingscheidt

Die Dissertation wurde am 09.11.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 26.03.2023 angenommen.

Abstract

Over the years, automatic speech recognition (ASR) in noise-free and single-speaker-dependent environments underwent tremendous progress with uncountable varieties of model architectures. This work examines three well-established model architectures of ASR and introduces related enhancements for improving their performance.

The first model architecture defines a discrete neural quantizer for hybrid approaches in ASR. These quantizers learn a function between high-dimensional feature vectors and discrete labels and are typically not competitive with standard continuous models. A novel training procedure combined with a deep neural network structure can alleviate this negative effect and demonstrate that a discrete neural quantizer can outperform continuous models despite losing information in the quantization procedure.

The second model architecture relies on attentional models with recurrent neural structures extended by additional model components. The appended components are optimized on two variants of time-reversed target labels and provide helpful information for the standard model part of the attentional models. Since the time-reversing procedure does not always produce equally long target label sequences, a novel regularization term is solely integrated into the training process. Therefore, the performance of standard attentional models is improved without increasing the decoding complexity of the model.

The third model architecture refers to recently established self-attentional architectures, which replace recurrent neural structures with self-attentional modules. The modules define strong globally-dependent operations and enable those models to connect information between distant hidden representations or features. However, speech contains valuable local information suppressed by applying such operations. As a result, the suggested approach divides the modules into local and global trainable modules and establishes an effective fusion strategy, leading to enhanced self-attentional model architectures capable of surpassing state-of-the-art approaches.

Zusammenfassung

Über die letzten Jahre hat die automatische Spracherkennung (ASR) in störgeräuscharmen Umgebungen, in denen nur jeweils eine einzelne Person aktiv spricht, einen enormen Fortschritt mit unzähligen Modellarchitekturen erzielt. Diese Arbeit untersucht drei etablierte Modellvarianten der ASR und stellt Erweiterungen für deren Verbesserung und Weiterentwicklung vor.

Die erste dieser Modellvarianten definiert einen diskreten, neuronalen Quantisierer für hybride Ansätze der ASR. Die Quantisierer lernen eine Funktion zwischen hochdimensionalen Merkmalsvektoren und diskreten Labeln und sind auf vielen Ebenen nicht konkurrenzfähig mit kontinuierlichen Systemen. Ein innovatives Trainingsverfahren, welches mit tiefen, neuronalen Netzwerkstrukturen kombiniert wird, kann diesen negativen Effekt verringern und verdeutlichen, dass diskrete, neuronale Quantisier kontinuierliche Systeme trotz eines Informationsverlusts übertreffen können.

Die zweite Modellvariante setzt auf Attention-Modelle mit wiederkehrenden, neuronalen Strukturen, die mit zusätzlichen Systemkomponenten erweitert werden. Diese Komponenten werden auf zwei unterschiedliche, zeitlich umgekehrte Varianten der Ziellabels trainiert und stellen damit nützliche Informationen für die normalen Bestandteile von Attention-Modellen zur Verfügung. Da der zeitliche Umkehrprozess nicht notwendigerweise gleichlange Ziellabelsequenzen erzeugt, wird eine innovative Regularisierung ausschließlich in den Trainingsprozess integriert. Infolgedessen verbessert sich die Leistung normaler Attention-Modellen ohne die Dekodierungskomplexität zu erhöhen.

Die dritte und abschließende Modellvariante dieser Arbeit sind Self-Attention-Modelle, bei denen die wiederkehrenden, neuronalen Strukturen durch Self-Attention-Module ersetzt werden. Die Module definieren stark ausgeprägte, global-abhängige Operationen und erlauben das Verbinden weit entfernter Informationen in verborgenen Repräsentations- und Merkmalssequenzen. Diese Operationen unterdrücken jedoch wertvolle lokale Informationen der Sprache. Aus diesem Grund teilt der vorgeschlagene Ansatz die Standardmodule in optimierbare lokale sowie globale Module ein und etabliert eine effiziente Verschmelzungsstrategie, welche zu verbesserten Self-Attention-Modellen führt, die selbst modernste ASR Ansätze übertreffen können.

Contents

List of Acronyms

AED	attention-based encoder-decoder
AM	acoustic model
AMI	augmented multi-party interaction
ASR	automatic speech recognition
BN	batch normalization
BLSTM	bidirectional long short-term memory
BLSTMP	bidirectional long short-term memory projected
BP	backpropagation
BPE	byte pair encoding
BPTT	backpropagation through time
CART	clustering and regression tree
CE	cross entropy
CHAR	character
CMN	cepstral mean normalization
CNN	convolutional neural network
CTC	connectionist temporal classification
CV	computer vision
DARPA	defense advanced research projects agency
DCT	discrete cosine transformation
DFT	discrete Fourier transformation
DHMM	discrete hidden Markov model
DNN	deep neural network
DNNQ	deep neural network quantizer
DTW	dynamic time warping
E2E	end-to-end
EM	expectation-maximization
GM	grapheme model
GMM	Gaussian mixture model
GPU	graphics processing unit
GRU	gated recurrent unit

HMM	hidden Markov model
KLD	Kullback-Leibler divergence
L2R	left-to-right
LF-MMI	lattice-free maximum mutual information
LIUM	Laboratoire Informatique de'l Universitè du Maine
LM	language model
LN	layer normalization
LSTM	long short-term memory
LSTMP	long short-term memory projected
LVSR	large-vocabulary speech recognition
MFCC	Mel-frequency cepstral coefficient
MHA	multi-head attention
MI	mutual information
MLE	maximum likelihood estimation
MLP	multilayer perceptron
MM	Markov model
MMI	maximum mutual information
MONO	monophone
MSE	mean squared error
NLP	natural language processing
NMT	neural machine translation
NN	neural network
NNVQ	neural network vector quantizer
OOV	out of vocabulary
PDF	probability density function
PM	pronunciation model
PMF	probability mass function
R2L	right-to-left
ReLU	rectified linear unit
RM	Resource Management
RNN	recurrent neural network
RNNT	recurrent neural network transducer
SA	self-attention
SAED	self-attention-based encoder-decoder
SGD	stochastic gradient descent
SOTA	state-of-the-art
TED	technology entertainment design
TED-LIUM-v2	TED-LIUM release 2
TDNN	time-delay neural network
TRI	triphone
VGG	visual geometry group
VQ	vector quantizer
VTLP	vocal tract length perturbation
WER	word error rate

WSJ	Wall Street Journal

List of Symbols

Miscellaneous

e	Euler's number
$\odot$	elementwise multiplication operator
$\oplus$	concatenation operator
$\circledast$	convolutional operator
$(\cdot)^{(t)}$	time index
$(\cdot)_{t_1}^{t_2}$	sub-sequence between t_1 and t_2
$(\cdot)^{[l]}$	layer index
$(\cdot)^{\{k\}}$	parameter index
$(\cdot)^\top$	transpose of a matrix/vector
$(\cdot)^{-1}$	inverse of a matrix
$(\cdot)^*$	optimal sequence
$(\cdot)'$	first differential operator
$\hat{(\cdot)}$	target vector/scalar
$\underbrace{(\cdot)}$	smoothed target vector/scalar
$\overrightarrow{(\cdot)}$	forward reference
$\overleftarrow{(\cdot)}$	backward reference
$\overleftrightarrow{(\cdot)}$	forward-backward reference
$\mathrm{H}(\cdot)$	entropy state function
$\mathrm{I}(\cdot,\cdot)$	mutual information function
$\mathbb{E}(\cdot)$	expectation function
$\mathrm{vec}(\cdot)$	vectorization operator
$\mathrm{diag}(\cdot)$	diagonal matrix
$\mathcal{U}(\cdot,\cdot)$	uniform distribution
$\nabla(\cdot)$	nabla operator
$\lceil\cdot\rceil$	ceiling operator
$\left\|\left\| \cdot \right\|\right\|$	L2 norm

Number Sets

$\mathbb{B}$	set of binary numbers
$\mathbb{C}$	set of complex numbers
$\mathbb{I}$	set of integers
$\mathbb{N}$	set of natural numbers without zero
$\mathbb{P}$	set of probabilities
$\mathbb{R}$	set of real numbers
$\mathbb{R}_-$	set of negative real numbers
$\mathbb{R}_+$	set of positive real numbers

Functions

$\mathcal{A}_{\text{aed}}(\cdot,\cdot,\cdot), \mathcal{A}_{\text{saed}}(\cdot,\cdot,\cdot)$	attention function of AED, and SAED models
$\mathcal{A}_{\text{sa}}(\cdot,\cdot,\cdot)$	self-attention function in SAED models
$C(\cdot,\cdot)$	cost function
$\delta(\cdot,\cdot)$	two-dimensional Kronecker delta
$d(\cdot)$	distance metric function
$\Gamma(\cdot)$	Mel filter bank function
$\mathcal{H}(\cdot)$	representing neural network function
$\mathcal{L}(\cdot)$	liklihood function
$L(\cdot,\cdot)$	loss function
$\Pi(\cdot)$	regularizer function
$\psi(\cdot), \widetilde{\psi}(\cdot)$	standard, and scaled softmax function
$\mathcal{Q}(\cdot,\cdot), \widetilde{\mathcal{Q}}(\cdot,\cdot)$	standard, and viterbi expected liklihood function
$\mathcal{S}(\cdot)$	scoring function
Top-k$(\cdot,\cdot)$	return the indices of the k highest entries
$v(\cdot)$	Mel/ordinary frequency tranformation function
$\zeta(\cdot)$	activation function

Greek

$\boldsymbol{\alpha}^{(t)}, \widetilde{\boldsymbol{\alpha}}^{(t)}, \overline{\boldsymbol{\alpha}}^{(t)}$	standard, viterbi and beam search forward probabilities at time t in HMMs
$\boldsymbol{\alpha}_{\text{att}}^{(o)}$	attention weights at time step o in AED models
$\boldsymbol{\beta}^{(t)}, \widetilde{\boldsymbol{\beta}}^{(t)}$	standard, and viterbi backward probabilities at time t in HMMs
$\boldsymbol{\gamma}^{(t)}, \widetilde{\boldsymbol{\gamma}}^{(t)}$	standard, and viterbi auxiliary forward probabilities at time t in HMMs
γ	regularization parameter in standard losses
$\boldsymbol{\epsilon}$	error in backpropagation
η	learning rate

$\boldsymbol{\Theta}, \boldsymbol{\theta}$	set of network/HMM parameters, single network/HMM parameter
$\boldsymbol{\lambda}$	eigenvalues in SVD
$\boldsymbol{\Lambda}$	center frequency vector
$\boldsymbol{\mu}$	mean of a single Gaussian
$\mu_{\text{train}}, \mu_{\text{dev}}, \mu_{\text{test}}$	metric error on train, dev, and test set
$\boldsymbol{\xi}^{(t)}, \widetilde{\boldsymbol{\xi}}^{(t)}$	standard, and auxiliary transition probabilities at time t in HMMs
$\boldsymbol{\pi}$	initial state probabilities in HMMs
ρ	dropout probability in dropout layers
$\boldsymbol{\Sigma}, \sigma^2$	variance of N-D, and 1-D single Gaussian
$\boldsymbol{\Phi}, \boldsymbol{\phi}$	set of all emission set parameters, single set of emission parameters
$\boldsymbol{\psi}^{(t)}, \overline{\boldsymbol{\psi}}^{(t)}$	most likely state path at time t applying viterbi, or beam search algorithm
$\boldsymbol{\Omega}, \boldsymbol{\omega}$	set of all class labels, single class label

Latin

$\boldsymbol{A}, a$	transition probability matrix, and single transition probability in HMMs
$\boldsymbol{a}$	weighted output of MLPs
$\mathcal{A}$	set of graphemes
$\boldsymbol{B}, b$	emission probability matrix, and single emission probability in HMMs
$\boldsymbol{B}, \boldsymbol{b}, b$	bias matrix, vector, and scalar in DNNs
$\mathcal{B}$	mini-batch
$\boldsymbol{c}, \widetilde{\boldsymbol{c}}$	memory, and auxiliary memory state in LSTMs
$\boldsymbol{c}, c$	Mel cepstrum components, and single Mel cepstrum component
$d, \widetilde{d}$	standard, and auxiliary window size of Gaussian masks $\boldsymbol{G}$
$\boldsymbol{D}$	cost matrix in soft-DTW algorithm
$\mathcal{D}$	dataset
$\boldsymbol{e}^{(o)}$	scoring vectors of attention module $\mathcal{S}_{\text{aed}}$ at output time step o
$\boldsymbol{F}, \boldsymbol{f}^{(t)}$	forget gate sequence, single forget gate output at time t in LSTMs
$\boldsymbol{G}, \boldsymbol{g}^{(o)}, g^{(o)}$	grapheme sequence, representing grapheme vector, and grapheme at output time step o
$\mathcal{G}$	all possible grapheme sequences
$\boldsymbol{H}, \boldsymbol{h}^{(t)}$	sequence of hidden representations, single hidden representation at time step t

$\boldsymbol{I}, i$	image, and single pixel of an image
$\boldsymbol{I}, \boldsymbol{i}$	input gate sequence, and single input gate output at time t in LSTMs
$\mathcal{K}$	ordered set, retaining position of entries
$\boldsymbol{K}, \boldsymbol{k}$	key sequence, and single keys in SAED models
$\boldsymbol{K}, k$	kernel matrix, and single entry in kernel of CNNs
$\boldsymbol{M}, \boldsymbol{m}^{(t)}, m^{(t)}$	output sequence, output vector, and single output at time t of DNNQs
$\overset{*}{\boldsymbol{M}}, \overset{*}{\boldsymbol{m}}^{(t)}, \overset{*}{m}^{(t)}$	firing neuron sequence, firing neuron vector, firing neuron at time step t in DNNQs
$\boldsymbol{O}, \boldsymbol{o}^{(t)}$	output gate sequence, and single gate output at time t in LSTMs
$\boldsymbol{P}, \boldsymbol{p}^{(t)}$	projection sequence, and single projection vecto at t in projected LSTM
$\boldsymbol{Q}, \boldsymbol{q}$	query sequence, and single queries in SAED models
$\boldsymbol{r}$	dropout vector
$\boldsymbol{S}, \boldsymbol{s}^{(t)}$	state sequence, and single state vector at t in HMMs
$\overline{\boldsymbol{S}}, s$	beam search state sequence, and single beam search state vector at t in HMMs
$\mathcal{S}_{\text{loc}}, \mathcal{S}_{\text{glob}}$	local, and global score in SAED models
$\boldsymbol{U}, u$	recurrent weight matrix, and single recurrent weight in RNNs
$\boldsymbol{V}, \boldsymbol{v}$	value sequence, and single value in SAED models
$\boldsymbol{V}, v$	recurrent memory weight matrix, and single recurrent memory weight in LSTMs
$\mathcal{V}$	vocabulary of words $\boldsymbol{w}^{(o)}$
$\boldsymbol{W}, w$	weight matrix, and single weight in DNNs
$\boldsymbol{W}, \boldsymbol{w}^{(o)}$	word sequence, and single word at time o
$\mathcal{W}$	set of all possible word sequences
$\boldsymbol{X}, \boldsymbol{x}^{(t)}, x^{(t)}$	input sequences, input vector, and single input at time t in DNNs
$\mathcal{X}$	set of availible input sequences $\boldsymbol{X}$ in $\mathcal{D}$
$\boldsymbol{Y}, \boldsymbol{y}^{(t)}, y^{(t)}$	output sequences, output vector, and single output at time t in DNNs

List of Figures

Chapter 6: Localness and Fusion Strategies in SAED Models

List of Tables

1

Introduction

The ability to communicate through a complex and defined language separates us, humans, from animals on the planet. The language itself enables humans to communicate most naturally and efficiently. With over 7000 actively spoken languages worldwide [46], humans created various language varieties adapted to location-dependent conditions. Even though languages differ in speed or number of syllables, the average information rate per second remains stable by approximately 39 $\frac{bits}{s}$ [38]. Along the transmission of information, the language also contains implicit psychological, medical, and emotional information, which can be utilized to retrieve medical insights from talking individuals [141].

The interest in automatically recognizing spoken languages and processing them has grown over the last two decades. The development of faster, tinier, and cheaper hardware architectures drives the integration of speech processing algorithms on local devices. Nowadays, smart devices like smartphones can respond to speech commands and long sentences, and perform specific actions, *e.g.*, executing commands in a home automation system [106]. All approaches simplify the interaction between humans and machines since spoken language enables natural communication to share specific information.

Speech processing can be roughly categorized into two major research areas: automatic speech recognition (ASR) and natural language processing (NLP). The objective of ASR is to retrieve a transcript from a given speech signal by applying ASR concepts, in which the alignments between input features and target labels are implicitly learned [22]. In NLP, the output of ASR systems is further processed. The objective is to extract or summarize information from the transcript, translate it into another language, detect speech commands and forward them to succeeding systems to perform specific actions [35].

The following work is established in the research area of ASR, more precise in modeling acoustic models (AMs). Similar to NLP or computer vision (CV) methods, approaches in ASR experienced major performance improvements caused by the resurgence of already well-known neural network (NN) paradigms over the last decade [82]. The increased computational power led to efficient optimization of deep neural network (DNN) structures with multiple layers. Consequently, ASR approaches based on DNNs challenged human speech recognition capabilities. In 2017, Microsoft achieved an important milestone in conversational speech recognition since their ASR systems matched human speech

recognition performance [172].

1.1 Motivation

Even though current ASR systems surpass human speech recognition abilities, several open research questions remain unanswered. Novel ASR architectures, which are modeling AMs, often disregard existing models, as the previous generation no longer achieve state-of-the-art (SOTA) results. This is observable in hybrid approaches [22], attention-based encoder-decoder (AED) models [10], or even in recent self-attention-based encoder-decoder (SAED) approaches [159], despite learning similar posterior distributions for given datasets consisting of feature-target pairs. The following paragraphs briefly introduce novel approaches by describing their differences from non-optimal standard architectures.

Hybrid approaches [22] belong to the most-studied architectures in ASR since they are constructed by hand-crafted modules, which are independently optimized. These modules have predefined in- and outputs and are combined into the final ASR models. Hybrid systems utilize continuous probability distributions modeled by either Gaussian mixture models (GMMs) or DNNs. Although discrete probability distributions are also feasible to apply, they suffer in losing information in the input space since the input feature sequences need to be quantized before being further processed. Therefore, ASR systems utilizing discrete features, *i.e.*, discrete probability distributions, typically are not competitive with continuous ASR systems. Multiple studies [110, 132, 133] have already examined discrete acoustic modeling, and some were even optimistic about surpassing continuous AMs systems [134]. With the recent progress in hardware and software, it is attractive to revise former discrete AM methods. Due to a lack of research over the past years, it remains unclear if discrete AMs could be competitive with continuous AMs if recent computational advantages are applied.

AED models [10] discard the idea of independently optimized modules and instead establish an end-to-end (E2E) approach. They transduce input feature sequences into grapheme sequences employing recurrent neural networks (RNNs). In contrast to hybrid models with their predefined dictionary, pronunciation model (PM), and language model (LM), AED models consist of encoder-decoder architectures and are entirely data-driven, as they learn these modules implicitly from the given data [163]. Therefore, AED models are simple to train since the alignments between input features and graphemes are retrieved implicitly and do not require complex knowledge of the overall system structure. However, it is challenging to identify specific implicitly learned modules for modification. For instance, if the dictionary is extended by unknown words, already optimized models need to be further trained with data including these unknown words. Generally, the training data itself is crucial for robust and powerful ASR systems. Several works have already demonstrated [105, 186] that the generalization of AED models can be improved if additional training data in the form of time-reversed target grapheme sequences are applied. Although the information in the standard and reversed grapheme sequence should be similar, the information extracted by additional decoders improved the decoding results of standard AED architectures [105]. So far, this effect has been

mainly examined in the decoding phase [105]. It is an open research question if a modified cost function could already improve the model during training and eventually transfer this enhancement to the decoding phase.

SAED models [159] replaced the computationally expensive RNNs in AED models to conquer the error accumulation problem of long sequences [105]. These models rely solely on self-attention (SA) mechanisms, evaluating the global importance between two sequences and returning a score for each sequence pair. Since SA modules do not require any recurrent model structures and solely depend on previous model layers, multilayer perceptrons (MLPs) are exclusively employed in SAED models. The SA operations allow the models to connect sequence information far apart, leading to strong global dependencies. Although this global context is beneficial on one side, it suppresses the local information on the other. Several works [111, 145, 149, 174] analyzed the impact of inducing localness into SA modules. However, these studies were primarily conducted in NLP. Their outcome demonstrated that the employment of localness supports the performance of SAED models. Until now, few studies have successfully applied the concept of localness in ASR [174]. Therefore, whether a more complex approach, including localness, could further improve SAED models is unknown.

1.2 Objectives

This work suggests solutions to the previously mentioned open research questions of hybrid [22], AED [10], and SAED models [159]. The contributions are solely executed under laboratory conditions, and a real-time application is not intended. In order to be replicable and comparable, the approaches need to achieve specific properties:

1. The recorded speech data is noise-free.
2. The features are extracted from a single channel.
3. There is no overlapping between speakers, *i.e.*, only one speaker is talking.
4. There is a limitation of newly added parameters to a predefined baseline.
5. The proposed approach surpasses the current SOTA baseline.

Since the contributions are distributed over different AM approaches, particular objectives need to be separately defined for every model class:

- *Hybrid models*: Despite technological progress, it is still challenging to build competitive discrete AMs that surpass continuous AMs. Although Neukirchen *et al.* [110] nearly achieved this objective with their neural network vector quantizer (NNVQ), the final model structure was shallow and had limiting modeling capacity. Moreover, their approach required an expensive iteration through the dataset, which is feasible for small datasets but unfeasible for large datasets. This work aims to tackle the iteration problem by proposing a simple adjustment in the

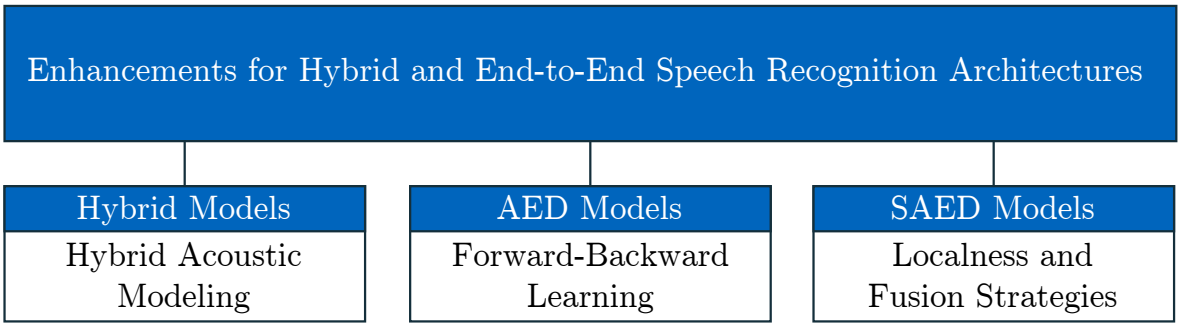


Figure 1.1: A brief overview of the chapters examining and establishing different AM approaches hybrid, AED, and SAED models.

training setup. Therefore, the shallow NNVQs are extendible into deep neural network quantizers (DNNQs), which contain a higher modeling capacity and can be optimized efficiently.

- *AED models*: The approach of Zheng *et al.* in [105] demonstrated that AED models, including additional decoders trained on a time-reversed transcript, returned marginal better results than standard AED models. Although they proposed a simple and efficient way to combine both decoders, they neglected a more precise analysis of combining these decoder losses in the training phase. As a result, the training scheme can be modified, *e.g.*, by integrating a more complex loss. The objective of the subsequent work is to analyze the defined training scheme in [105] and establish an improved loss extension that is solely active in the training phase.

- *SAED models*: SAED models [159] with their SA modules resolved multiple problems of AED approaches where long input and output sequences existed, and valid relations between these sequences faded by vanishing or exploding gradients. The resulting strong global context made it feasible to find relations even between sequence parts located far apart. However, the strong global context dependency also has major disadvantages, as the scoring operation in the SA modules suppresses valuable local information. Despite several works [145, 149, 174] aiming to boost the local context, Nguyen *et al.* [111] recently introduced a novel differential window with individual model branches for local and global scores to achieve localness. Since separate local and global model branches could enhance prior approaches, this work examines the effect of independent local and global model branches for SAED models and establishes effective fusion strategies.

The word error rate (WER) defines the universal and objective metric to compare the performance of AMs. All objectives have to lower the WER metric in common compared to the predefined baseline. LMs are excluded in all the approaches since a further reduction of the WER can be considered, and thorough training on combining AMs and LMs is out of the scope of this work.

1.3 Overview

The structure of this work is depicted briefly in Figure 1.1 and in more detail in the following:

Chapter 2 establishes the general background by introducing mathematical terms, concepts, and fundamentals necessary to understand the following chapters.

Chapter 3 introduces the speech recognition background by defining the task of ASR, related mathematical concepts of ASR, and a presentation of popular datasets.

Chapter 4 examines the potential of discrete ASR systems by considering the latest progress in machine learning and demonstrates the ability of discrete systems to surpass traditional continuous ASR systems.

Chapter 5 analyses the standard AED and SAED models and establishes an extension for utilizing time-reversed information standard architectures.

Chapter 6 demonstrates concepts for localness and corresponding fusion strategies for SAED architectures, leading to models capable of competing with SOTA models.

Chapter 7 concludes by summarizing the key concepts established in this work.

2 General Background

This chapter provides the general background of concepts introduced in subsequent chapters. In the first section, hidden Markov models (HMMs) are established. Then, it is demonstrated how they are utilized in modeling temporal components of sequences and how their parameters are efficiently obtained. Related automatic speech recognition (ASR) formulations are not discussed, as these follow in the next chapter.

In the second section, neural networks (NNs) are introduced. First, standard perceptrons are explained, followed by their activation functions. Then, the standard perceptrons are enhanced, and related variations are discussed. Next, two popular training objectives are defined, including their corresponding cost functions. Based on these cost functions, the backpropagation algorithm and the gradient descent methods are explored, which define the backbone of current optimization techniques for deep neural networks (DNNs). Combining these methods leads to efficient procedures for minimizing the cost functions and proper parameter adjustments of DNNs.

Moreover, the problem of vanishing and exploding gradients is examined, which typically occurs in model optimization and how to overcome these problems. The chapter concludes by analyzing the challenges for training generalized DNNs and suggesting effective strategies to obtain such models. Notice that the following mathematical notations are based on ISO 80000-2 [70].

2.1 Hidden Markov Models

Many applications process data with temporal information, *e.g.*, audio, video, or stock market data. The temporal information in this data contains valuable information, which improves regression and classification tasks. Although it would be simple to assume the i.i.d. of every sample in sequences, the temporal information of adjacent samples would be unused, and precise predictions would be impossible. A standard approach also applied in this work is to employ HMMs [18] for modeling sequential data since these models are well-studied in multiple research domains, *e.g.*, speech recognition [22], action recognition [173], or predicting protein topologies [78].

The theory for *Markov chains* was proposed by Baum *et al.* in 1966 [11] and defined the foundation of statistical models, such as Markov models (MMs) and HMMs. These

models are capable of efficiently modeling sequential data [18] and are introduced below. Thereby, the MMs are further extended to HMMs, which are later applied in ASR approaches.

2.1.1 Model Structure

Let $\boldsymbol{x}^{(t)} \in \mathbb{R}^I$ be a single input vector of size I in a sequence of input vectors $\boldsymbol{X}^{(n)} = \left[\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(T)}\right]$ of length T, where $\boldsymbol{X}^{(n)} \in \mathbb{R}^{I \times T}$. The sequential data $\boldsymbol{X}^{(n)}$ is selected from a finite set $\mathcal{X} = \left\{\boldsymbol{X}^{(n)}\right\}_{n=1}^{N}$ with N elements, whereby T varies according to each sequence $\boldsymbol{X}^{(n)}$. In order to prevent cluttered notation, the superscript n is expelled, and the following formulas are defined for a single input sequence $\boldsymbol{X} \equiv \boldsymbol{X}^{(n)}$.

Although it would be simple to process the inputs $\boldsymbol{x}^{(t)}$ by assuming i.i.d. of every input $\boldsymbol{x}^{(t)}$, the sequential dependence and its temporal information would be lost. Therefore, the i.i.d. assumption has to be relaxed by introducing a standard MM [22]:

$$p\big(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}\big) = \prod_{t=2}^{T} p\big(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-1)}\big). \tag{2.1}$$

Here, $p\big(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}\big)$ defines the joint distribution between the observations $\boldsymbol{x}^{(1)}$, $\ldots, \boldsymbol{x}^{(T)}$, and $p\big(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-1)}\big)$ represents the conditional distribution for $\boldsymbol{x}^{(t)}$ given all previous observations $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-1)}$. This leads to a dependency of the current input $\boldsymbol{x}^{(t)}$ on all preceding inputs from the past. Notice that it is implicitly assumed that $p\big(\boldsymbol{x}^{(1)} | \boldsymbol{x}^{(0)}\big) = p\big(\boldsymbol{x}^{(1)}\big)$ as $\boldsymbol{x}^{(0)}$ is not defined. If $\boldsymbol{x}^{(t)}$ in Equation (2.1) is only conditioned on the previous observation, the *first-order Markov chain* formulation [18] is obtained:

$$p\big(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}\big) = p\big(\boldsymbol{x}^{(1)}\big) \prod_{t=1}^{T} p\big(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}\big). \tag{2.2}$$

Even though higher orders of Markov chains are theoretically feasible, they are out of scope for this thesis since the number of parameters is increasing exponentially [18]. Additionally, the conditional distribution $p\big(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}\big)$ is restricted to be equal in order to be applicable in later chapters. To obtain a more flexible yet parameter-efficient model, discrete latent variables are introduced to each input $\boldsymbol{x}^{(t)}$. Let $s^{(t)} \in \mathbb{N}$ be a latent variable and denoted as a single *state*. Then, the states $s^{(t)}$ can be modeled by K-dimensional binary random variable $\boldsymbol{s}^{(t)} \in \mathbb{B}^K$ with $\mathbb{B} = \{0, 1\}$, where the element at the index k is one, and the remaining elements are zero [119]:

$$s_k^{(t)} = \delta\big(k, s^{(t)}\big) = \begin{cases} 1 & \text{if } k = s^{(t)} \\ 0 & \text{else} \end{cases} \qquad \forall k \in \{1, 2, \cdots, K\}. \tag{2.3}$$

The operator $\delta\big(\,\cdot\,, \cdot\,\big)$ is the Kronecker delta, and K corresponds to the total number of states. Notice that each state $\boldsymbol{s}^{(t)}$ also fulfills the property of a valid probability distribution since its entries lie in an interval $0 \leq s_k^{(t)} \leq 1$ and $\sum_k s_k^{(t)} = 1$.

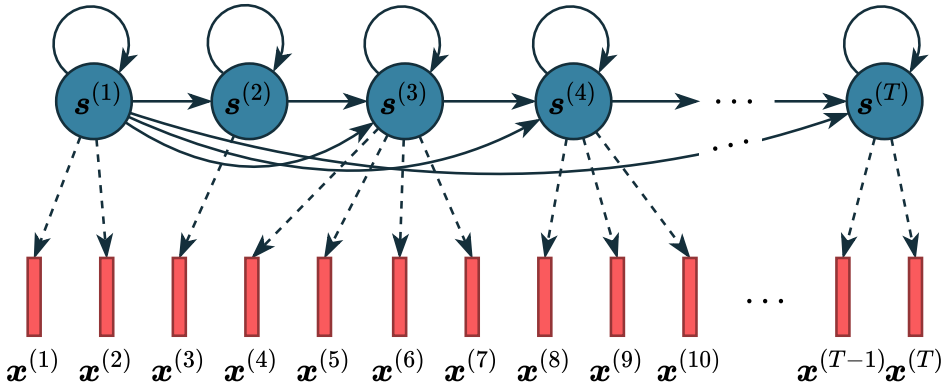


Figure 2.1: The graphical model of an HMM with the corresponding states $\boldsymbol{s}^{(t)}$, which are represented by blue circles. Each input $\boldsymbol{x}^{(t)}$, expressed by the red rectangles, is aligned to a specific state $\boldsymbol{s}^{(t)}$, represented by dotted arrows. The alignment is obtained by a procedure introduced Section 2.1.4.

The introduction of a latent space leads to the formulation of HMMs. In Figure 2.1, a graphical representation of a single HMM is depicted. The latent space also implies a modification of the joint distribution:

$$p\big(\boldsymbol{x}^{(1)},\dots,\boldsymbol{x}^{(T)},\boldsymbol{s}^{(1)},\dots,\boldsymbol{s}^{(T)}\big) = p\big(\boldsymbol{s}^{(1)}\big)\left[\prod_{t=2}^{T} p\big(\boldsymbol{s}^{(t)}|\boldsymbol{s}^{(t-1)}\big)\right]\prod_{t=1}^{T} p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}\big), \tag{2.4}$$

where $p\big(\boldsymbol{s}^{(t)}|\boldsymbol{s}^{(t-1)}\big)$ is the conditional distribution of being in state $\boldsymbol{s}^{(t)}$ given the previous state $\boldsymbol{s}^{(t-1)}$, $p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}\big)$ is the conditional distribution of $\boldsymbol{x}^{(t)}$ for a given state $\boldsymbol{s}^{(t)}$, and $\boldsymbol{S} = [\boldsymbol{s}^{(1)},\dots,\boldsymbol{s}^{(t)},\dots,\boldsymbol{s}^{(T)}]$ is the latent state sequence $\boldsymbol{S} \in \mathbb{B}^{K\times T}$.

In order to fully define HMMs, the given distributions require parameterization. The initial state can be governed by the parameter $\boldsymbol{\pi} \in \mathbb{P}^K$ representing the probability $\pi_k \equiv p\big(s_k^{(1)} = 1\big)$, where the set $\mathbb{P} = \{p \in \mathbb{R} \,|\, 0 \leq p \leq 1\}$. This leads to the parameterized distribution $p\big(\boldsymbol{s}^{(1)}|\boldsymbol{\pi}\big)$:

$$p\big(\boldsymbol{s}^{(1)}|\boldsymbol{\pi}\big) = \prod_{k=1}^{K} \pi_k^{s_k^{(1)}}. \tag{2.5}$$

Notice that $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$ and therefore correspond to probabilities.

The conditional distribution $p\big(\boldsymbol{s}^{(t)}|\boldsymbol{s}^{(t-1)}\big)$ can be parameterized similarly since these states are also modeled by K-dimensional binary random variables. Therefore, the parameter $a_{j,k}$ is introduced with $a_{j,k} \equiv p\big(s_k^{(t)} = 1|s_j^{(t-1)} = 1\big)$. The elements of the matrix $\boldsymbol{A} \in \mathbb{P}^{K\times K}$ represent the *transition probabilities* of the HMM and determine the conditional distribution:

$$p\big(\boldsymbol{s}^{(t)}|\boldsymbol{s}^{(t-1)},\boldsymbol{A}\big) = \prod_{k=1}^{K}\prod_{j=1}^{K} a_{j,k}^{s_j^{(t-1)}s_k^{(t)}}, \tag{2.6}$$

where $\sum_k a_{j,k} = 1$ and $0 \leq a_{j,k} \leq 1$.

Finally, the *emission probabilities* $p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}\big)$ are specified with state-dependent parameters $\boldsymbol{\Phi} = \{\boldsymbol{\phi}^{\{1\}}, \ldots, \boldsymbol{\phi}^{\{k\}}, \ldots, \boldsymbol{\phi}^{\{K\}}\}$:

$$p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}, \boldsymbol{\Phi}\big) = \prod_{k=1}^{K} p\big(\boldsymbol{x}^{(t)}|\boldsymbol{\phi}^{\{k\}}\big)^{s_k^{(t)}}. \tag{2.7}$$

For continuous inputs $\boldsymbol{x}^{(t)}$, candidates such as Gaussian mixture models (GMMs) or DNNs can efficiently parameterize $p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}\big)$. In the case of discrete inputs $\boldsymbol{x}^{(t)}$, conditional tables are applicable.

Based on these three parameterized distributions, HMMs are defined as:

$$p\big(\boldsymbol{X}, \boldsymbol{S}|\boldsymbol{\Theta}\big) = p\big(\boldsymbol{s}^{(1)}|\boldsymbol{\pi}\big)\big[\prod_{t=2}^{T} p\big(\boldsymbol{s}^{(t)}|\boldsymbol{s}^{(t-1)}, \boldsymbol{A}\big)\big] \prod_{t=1}^{T} p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}, \boldsymbol{\Phi}\big), \tag{2.8}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\Phi}\}$ denotes a set of HMM parameters.

2.1.2 Parameter Estimation

The objective of the parameter estimation procedure is to determine the optimal HMM parameters $\boldsymbol{\Theta}^*$ given the input sequence $\boldsymbol{X}$:

$$\boldsymbol{\Theta}^* = \arg\max_{\boldsymbol{\Theta}} p\big(\boldsymbol{X}|\boldsymbol{\Theta}\big). \tag{2.9}$$

This approach is typically known as the maximum likelihood estimation (MLE) approach [18, 64]. The likelihood function $\mathcal{L}\big(\boldsymbol{\Theta}|\boldsymbol{X}\big)$ based on Equation (2.8) is defined:

$$\mathcal{L}\big(\boldsymbol{\Theta}|\boldsymbol{X}\big) = p\big(\boldsymbol{X}|\boldsymbol{\Theta}\big) = \sum_{\boldsymbol{S}} p\big(\boldsymbol{X}, \boldsymbol{S}|\boldsymbol{\Theta}\big) \tag{2.10}$$

$$= \sum_{k^{(1)}=1}^{K} \cdots \sum_{k^{(t)}=1}^{K} \cdots \sum_{k^{(T)}=1}^{K} p\big(\boldsymbol{X}, s_{k^{(1)}}^{(1)}, \ldots s_{k^{(t)}}^{(t)}, \ldots s_{k^{(T)}}^{(T)}|\boldsymbol{\Theta}\big), \tag{2.11}$$

where $k^{(t)}$ defines a specific visiting state in the current state vector $\boldsymbol{s}^{(t)}$ at time t, and $\mathcal{L}\big(\boldsymbol{\Theta}|\boldsymbol{X}\big)$ corresponds to a criterion to verify if a local maximum is reached. A naive approach to determine the likelihood function $\mathcal{L}\big(\boldsymbol{\Theta}|\boldsymbol{X}\big)$ would be to sum over all states $\boldsymbol{S}$ in Equation (2.10), which cannot be parallelized since each state $\boldsymbol{s}^{(t)}$ depends on all the preceding states, and would result in exponentially terms growing w.r.t. to the state sequence length.

2.1.3 Forward-Backward Algorithm

This issue can be resolved by applying the *forward-backward* algorithm [128] combined with a dynamic programming approach [13]. The algorithm defines a recursive scheme to obtain $p\big(\boldsymbol{X}|\boldsymbol{\Theta}\big)$ efficiently. Therefore, Equation (2.10) is rewritten as:

$$p\big(\boldsymbol{X}|\boldsymbol{\Theta}\big) = \sum_{k=1}^{K} p\big(s_k^{(t)}, \boldsymbol{X}|\boldsymbol{\Theta}\big) \quad \forall t \in \{1, 2, \cdots, T\}, \tag{2.12}$$

where $p\big(s_k^{(t)}, \boldsymbol{X}|\boldsymbol{\Theta}\big)$ refers to the probability of reaching the state s_k at the time t, and the input sequence $\boldsymbol{X}$ is emitted by an HMM with its parameters $\boldsymbol{\Theta}$. The probability $p\big(s_k^{(t)}, \boldsymbol{X}|\boldsymbol{\Theta}\big)$ can be separated into two independent probabilities:

$$p\big(s_k^{(t)}, \boldsymbol{X}|\boldsymbol{\Theta}\big) = p\big(s_k^{(t)}, \boldsymbol{X}_1^t, \boldsymbol{X}_{t+1}^T|\boldsymbol{\Theta}\big) = p\big(s_k^{(t)}, \boldsymbol{X}_1^t|\boldsymbol{\Theta}\big)p\big(\boldsymbol{X}_{t+1}^T|s_k^{(t)}, \boldsymbol{X}_1^t, \boldsymbol{\Theta}\big). \tag{2.13}$$

The first joint probability $p\big(s_k^{(t)}, \boldsymbol{X}_1^t|\boldsymbol{\Theta}\big)$ specifies the probability of observing the sub-sequence $\boldsymbol{X}_1^t = [\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}]$ and being in the state $s_k^{(t)}$. The second probability $p\big(\boldsymbol{X}_{t+1}^T|s_k^{(t)}, \boldsymbol{X}_1^t, \boldsymbol{\Theta}\big)$ defines the probability of observing the sub-sequence $\boldsymbol{X}_{t+1}^T = [\boldsymbol{x}^{(t+1)}, \ldots, \boldsymbol{x}^{(T)}]$ and reaching the state $s_k^{(t)}$ based on the already observed sub-sequence $\boldsymbol{X}_1^t$. The probabilities are then denoted as the *forward* probability $\boldsymbol{\alpha}^{(t)} \in \mathbb{P}^K$ [22]:

$$\begin{aligned} \alpha_k^{(t)} &= p\big(s_k^{(t)}, \boldsymbol{X}_1^t|\boldsymbol{\Theta}\big) && (2.14)\\ &= \sum_{j=1}^{K} \alpha_j^{(t-1)} p\big(s_k^{(t)}|s_j^{(t-1)}, \boldsymbol{X}_1^{t-1}, \boldsymbol{\Theta}\big)p\big(\boldsymbol{x}^{(t)}|s_k^{(t)}, s_j^{(t-1)}, \boldsymbol{\Theta}\big), && (2.15) \end{aligned}$$

and the *backward* probability $\boldsymbol{\beta}^{(t)} \in \mathbb{P}^K$ [22]:

$$\begin{aligned} \beta_k^{(t)} &= p\big(\boldsymbol{X}_{t+1}^T|s_k^{(t)}, \boldsymbol{X}_1^t, \boldsymbol{\Theta}\big) && (2.16)\\ &= \sum_{j=1}^{K} \beta_j^{(t+1)} p\big(s_j^{(t+1)}|s_k^{(t)}, \boldsymbol{X}_1^t, \boldsymbol{\Theta}\big)p\big(\boldsymbol{x}^{(t+1)}|s_j^{(t+1)}, s_k^{(t)}, \boldsymbol{\Theta}\big). && (2.17) \end{aligned}$$

The probabilities are further simplified by the three principles below [18, 22]. Notice that not all model parameters in $\boldsymbol{\Theta}$ are dependent on the employed probability distributions and are removed if necessary:

1. For first-order Markov chains, the current state $\boldsymbol{s}^{(t)}$ is only dependent of the previous state $\boldsymbol{s}^{(t-1)}$ and independent of all the preceding states before the time step $t-1$. Since these probabilities are typically not time-dependent, the time index is removed:
$$p\big(s_j^{(t)}|s_k^{(t-1)}, \boldsymbol{X}_1^{t-1}, \boldsymbol{\Theta}\big) = p\big(s_j|s_k, \boldsymbol{A}\big). \tag{2.18}$$

2. The inputs $\boldsymbol{x}^{(t)}$ in the given sequence $\boldsymbol{X}$ are i.i.d. and therefore not correlated:
$$p\big(\boldsymbol{x}^{(t)}|s_j^{(t)}, s_k^{(t-1)}, \boldsymbol{X}_1^{(t-1)}, \boldsymbol{\Theta}\big) = p\big(\boldsymbol{x}^{(t)}|s_j^{(t)}, s_k^{(t-1)}, \boldsymbol{\Phi}^{\{j\}}\big). \tag{2.19}$$

3. For standard HMMs, the emission probabilities are solely state dependent. Therefore, the Equation (2.19) can be simplified into:

$$p\big(\boldsymbol{x}^{(t)}|s_j^{(t)}, s_k^{(t-1)}, \boldsymbol{\Phi}\big) = p\big(\boldsymbol{x}^{(t)}|s_j^{(t)}, \boldsymbol{\Phi}\big). \tag{2.20}$$

The forward recursion $\alpha_k^{(t)}$ is then obtained by:

$$\alpha_k^{(t)} = \left[\sum_{j=1}^{K} \alpha_j^{(t-1)} p\big(s_k|s_j\big)\right] p\big(\boldsymbol{x}^{(t)}|s_k^{(t)}\big), \tag{2.21}$$

and the backward recursion $\beta_k^{(t)}$ by:

$$\beta_k^{(t)} = \left[\sum_{j=1}^{K} \beta_j^{(t+1)} p\big(s_j|s_k\big)\right] p\big(\boldsymbol{x}^{(t+1)}|s_j^{(t+1)}\big), \tag{2.22}$$

where the readability is further improved by excluding the HMM parameters $\boldsymbol{\Theta}$.

Both recursions require an initialization step to determine the next recursion step $t+1$:

$$\alpha_k^{(1)} = p\big(\boldsymbol{x}^{(1)}|s_k^{(1)}\big) p\big(s_k^{(1)}|\boldsymbol{\pi}\big) \quad \forall k \in \{1, 2, \cdots, K\}, \tag{2.23}$$

where $s_k^{(1)}$ defines the initial state governed by $\boldsymbol{\pi}$ distribution, introduced in Equation (2.5). Similarly, the backward recursion is initialized for the time step T:

$$\beta_k^{(T)} = 1 \quad \forall k \in \{1, 2, \cdots, K\}. \tag{2.24}$$

Generally, the likelihood $\mathcal{L}\big(\boldsymbol{\Theta}|\boldsymbol{X}\big)$ in Equation (2.12) is solely obtained by recursions of the forward probabilities $\alpha_k^{(t)}$ and the initialization of the backward probabilities $\beta_k^{(T)}$ at the last time step T:

$$p\big(\boldsymbol{X}|\boldsymbol{\Theta}\big) = \sum_{j=1}^{K} \alpha_j^{(t)} \quad \forall t \in \{1, 2, \cdots, T\}. \tag{2.25}$$

The introduction of $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\beta}^{(t)}$ provides an efficient way to estimate the probability of any state $\boldsymbol{s}^{(t)}$ to any time t. Although the backward probabilities $\boldsymbol{\beta}^{(t)}$ were not applied for $\mathcal{L}\big(\boldsymbol{\Theta}|\boldsymbol{X}\big)$, they are required for optimizing the HMM parameters, which will be covered soon.

2.1.4 Viterbi Criterion

The Viterbi criterion allows an approximation $\widetilde{\mathcal{L}}\big(\boldsymbol{\Theta}|\boldsymbol{X}\big)$ of the entire likelihood $p\big(\boldsymbol{X}|\boldsymbol{\Theta}\big)$ by replacing the *sum* with *max* operators in Equation (2.11) [22]:

$$\widetilde{\mathcal{L}}\big(\boldsymbol{\Theta}|\boldsymbol{X}\big) = \max_{1\le k^{(1)}\le K} \cdots \max_{1\le k^{(t)}\le K} \cdots \max_{1\le k^{(T)}\le K} p\big(\boldsymbol{X}, s_{k^{(1)}}^{(1)}, \ldots, s_{k^{(t)}}^{(t)}, \ldots, s_{k^{(T)}}^{(T)}|\boldsymbol{\Theta}\big). \tag{2.26}$$

The max operators simplify the forward recursion in Equation (2.21) to $\widetilde{\boldsymbol{\alpha}}^{(t)} \in \mathbb{P}^K$:

$$\widetilde{\alpha}_k^{(t)} = \max_{1 \leq j \leq K} \left[\widetilde{\alpha}_j^{(t-1)} p\big(s_k | s_j\big) \right] p\big(\boldsymbol{x}^{(t)} | s_k^{(t)}\big), \tag{2.27}$$

and define an efficient approximation of Equation (2.21). Similarly to the Forward-Backward algorithm, the recursion is calculated up to the last time step T to receive the final approximation $\widetilde{\mathcal{L}}\big(\boldsymbol{\Theta} | \boldsymbol{X}\big)$:

$$\widetilde{\mathcal{L}}\big(\boldsymbol{\Theta} | \boldsymbol{X}\big) = \max_{1 \leq k \leq K} \widetilde{\alpha}_k^{(T)}. \tag{2.28}$$

Notice that the backward recursion is skipped, as the forward recursion $\widetilde{\boldsymbol{\alpha}}^{(t)}$ is sufficient to obtain $\widetilde{\mathcal{L}}\big(\boldsymbol{\Theta} | \boldsymbol{X}\big)$ by setting all entries in $\widetilde{\boldsymbol{\beta}}^{(T)} \in \mathbb{P}^K$ to one.

Besides determining an approximation of $\mathcal{L}\big(\boldsymbol{\Theta} | \boldsymbol{X}\big)$, the Viterbi criterion is normally utilized to return the optimal state sequence $\boldsymbol{S}^*$. Therefore, it is necessary to track the optimal path $\boldsymbol{\psi}^{(t)} \in \mathbb{N}^K$ up to the current time t since, otherwise, the information is lost in applying the recursion:

$$\psi_k^{(t)} = \underset{1 \leq j \leq K}{\arg\max} \left[\widetilde{\alpha}_j^{(t-1)} p\big(s_k | s_j\big) \right] p\big(\boldsymbol{x}^{(t)} | s_k^{(t)}\big) \quad \forall k \in \{1, 2, \cdots, K\}. \tag{2.29}$$

After reaching the final recursion step $\widetilde{\boldsymbol{\alpha}}^{(T)}$, a back-tracking procedure retrieves the optimal state sequence $\boldsymbol{S}^*$, where the first entry of $\boldsymbol{S}^*$ is initialized with:

$$\boldsymbol{s}^{*(t)} = 0 \quad \forall t \in \{1, 2, \cdots, T\}. \tag{2.30}$$

Then, the state $\boldsymbol{s}^{*(t)}$ with the highest probability is found in the last time step T:

$$k^{*(T)} = \underset{1 \leq k \leq K}{\arg\max} \widetilde{\alpha}_k^{(T)} \tag{2.31}$$

$$\boldsymbol{s}^{*(T)} = \boldsymbol{s}_{k^{*(T)}} \tag{2.32}$$

The optimal state $\boldsymbol{s}^{*(T)}$ defines the initial state of the back-tracking procedure, in which Equation (2.29) is employed to determine the previous states iteratively:

$$k^{*(t-1)} = \psi_{k^{*(t-1)}}^{(t)} \quad \forall t \in \{T, T-1, \cdots, 2\} \tag{2.33}$$

$$s^{*(t-1)} = s_{k^{*(t-1)}} \quad \forall t \in \{T, T-1, \cdots, 2\} \tag{2.34}$$

2.1.5 Baum-Welch Training

Finally, the actual training based on Equation (2.9) of the HMM parameters is achieved by either the *Baum-Welch* [12] or Viterbi training [84]. They differ in Equation (2.11), where the Baum-Welch training applies the entire sum, whereby the Viterbi training replaces the sums with mathematical *max* operators. Both procedures iteratively optimize the

parameters utilizing the expectation-maximization (EM) algorithm [43], which maximizes Equation (2.10) until a local, predefined criterion is reached.

The log-likelihood $p\big(\boldsymbol{X}|\boldsymbol{\Theta}\big)$ defined in Equation (2.9) cannot be optimized directly. Therefore, an iterative procedure is required, where new parameters $\boldsymbol{\Theta}$ raise the log-likelihood $p\big(\boldsymbol{X}|\boldsymbol{\Theta}\big)$. Such an approach is established as:

$$\ln p\big(\boldsymbol{X}|\boldsymbol{\Theta}\big)^{(\tau)} \geq \ln p\big(\boldsymbol{X}|\boldsymbol{\Theta}_{\text{old}}\big)^{(\tau-1)}, \tag{2.35}$$

where τ refers to the current iteration step in the optimization. In each iteration step τ, the objective is to obtain new HMM parameters $\boldsymbol{\Theta}$, which return a higher likelihood than the old model parameters $\boldsymbol{\Theta}_{\text{old}}$. The Baum-Welch training [12] corresponds to a modified version of the EM algorithm [43] and specifies an efficient way to achieve this objective. Therefore, an expectation function $\mathcal{Q}\big(\boldsymbol{\Theta}, \boldsymbol{\Theta}_{\text{old}}\big)$ is defined in the *E-step*. However, solely the input sequence $\boldsymbol{X}$ is directly observable, whereas the latent sequence $\boldsymbol{S}$ is hidden. A solution is obtained if the expectation of the entire likelihood $\ln p\big(\boldsymbol{X}, \boldsymbol{S}|\boldsymbol{\Theta}\big)$ is defined w.r.t. to the posterior distribution of the latent variables $p\big(\boldsymbol{S}|\boldsymbol{X}, \boldsymbol{\Theta}_{\text{old}}\big)$ [18]:

$$\begin{aligned} \mathcal{Q}\big(\boldsymbol{\Theta}, \boldsymbol{\Theta}_{\text{old}}\big) &= \mathbb{E}_{p(\boldsymbol{S}|\boldsymbol{X}, \boldsymbol{\Theta}_{\text{old}})} \ln p\big(\boldsymbol{X}, \boldsymbol{S}|\boldsymbol{\Theta}\big) && (2.36) \\ &= \sum_{\boldsymbol{S}} p\big(\boldsymbol{S}|\boldsymbol{X}, \boldsymbol{\Theta}_{\text{old}}\big) \ln p\big(\boldsymbol{X}, \boldsymbol{S}|\boldsymbol{\Theta}\big). && (2.37) \end{aligned}$$

The notation is typically simplified by introducing two auxiliary posterior distributions, where the marginal posterior distribution $\boldsymbol{\gamma}^{(t)} \in \mathbb{P}^K$ is specified as:

$$\gamma_k^{(t)} = p\big(s_k^{(t)}|\boldsymbol{X}, \boldsymbol{\Theta}_{\text{old}}\big) = \frac{p\big(s_k^{(t)}, \boldsymbol{X}|\boldsymbol{\Theta}_{\text{old}}\big)}{p\big(\boldsymbol{X}|\boldsymbol{\Theta}_{\text{old}}\big)}, \tag{2.38}$$

and the joint posterior distribution $\boldsymbol{\xi}^{(t)} \in \mathbb{P}^{K \times K}$:

$$\xi_{k,j}^{(t)} = p\big(s_j^{(t+1)}, s_k^{(t)}|\boldsymbol{X}, \boldsymbol{\Theta}_{\text{old}}\big) = \frac{p\big(s_k^{(t)}, s_j^{(t+1)}, \boldsymbol{X}|\boldsymbol{\Theta}_{\text{old}}\big)}{p\big(\boldsymbol{X}|\boldsymbol{\Theta}_{\text{old}}\big)}. \tag{2.39}$$

Both distributions can be efficiently determined with the forward and backward probabilities [128] defined in Equations (2.21) and (2.22). By including both probabilities, the margin $\gamma_k^{(t)}$ is rewritten in the following way:

$$\gamma_k^{(t)} = \frac{p\big(s_k^{(t)}, \boldsymbol{X}_1^t, \boldsymbol{X}_{t+1}^T|\boldsymbol{\Theta}_{\text{old}}\big)}{p\big(\boldsymbol{X}|\boldsymbol{\Theta}_{\text{old}}\big)} = \frac{\alpha_k^{(t)} \beta_k^{(t)}}{\sum_{l=1}^{K} \alpha_l^{(t)} \beta_l^{(t)}}. \tag{2.40}$$

Similarly, the joint margin $\xi_{k,j}^{(t)}$ is redefined:

$$\xi_{k,j}^{(t)} = \frac{p\big(s_k^{(t)}, s_j^{(t+1)}, \boldsymbol{X}_1^t, \boldsymbol{X}_{t+1}^T|\boldsymbol{\Theta}_{\text{old}}\big)}{p\big(\boldsymbol{X}|\boldsymbol{\Theta}_{\text{old}}\big)} \tag{2.41}$$

$$= \frac{\alpha_k^{(t)} p\big(s_j|s_k, \boldsymbol{\Theta}_{\text{old}}\big) p\big(\boldsymbol{x}^{(t+1)}|s_j^{(t+1)}\big) \beta_j^{(t+1)}}{\sum_{k=1}^{K}\sum_{j=1}^{K} \alpha_k^{(t)} p\big(s_j|s_k, \boldsymbol{\Theta}_{\text{old}}\big) p\big(\boldsymbol{x}^{(t+1)}|s_j^{(t+1)}\big) \beta_j^{(t+1)}}. \tag{2.42}$$

The likelihood $p\big(\boldsymbol{X}, \boldsymbol{S}|\boldsymbol{\Theta}\big)$ can then be inserted into $\mathcal{Q}\big(\boldsymbol{\Theta}, \boldsymbol{\Theta}_{\text{old}}\big)$ and simplified by the auxiliary margins $\boldsymbol{\gamma}^{(t)}$ and $\boldsymbol{\xi}^{(t)}$:

$$\begin{aligned}\mathcal{Q}\big(\boldsymbol{\Theta}, \boldsymbol{\Theta}_{\text{old}}\big) &= \sum_{\boldsymbol{S}} \big(\ln\big\{p\big(\boldsymbol{s}^{(1)}|\boldsymbol{\pi}\big)\big\} + \ln\big\{\prod_{t=2}^{T} p\big(\boldsymbol{s}^{(t)}|\boldsymbol{s}^{(t-1)}, \boldsymbol{A}\big)\big\} \qquad (2.43)\\ &\quad + \ln\big\{\prod_{t=1}^{T} p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}, \boldsymbol{\Phi}\big)\big\}\big) p\big(\boldsymbol{S}|\boldsymbol{X}, \boldsymbol{\Theta}_{\text{old}}\big)\\ &= \sum_{k=1}^{K} \gamma_k^{(1)} \ln \pi_k + \sum_{t=2}^{T-1}\sum_{j=1}^{K}\sum_{k=1}^{K} \xi_{k,j}^{(t)} \ln a_{j,k} + \sum_{t=1}^{T}\sum_{k=1}^{K} \gamma_k^{(t)} \ln p\big(\boldsymbol{x}^{(t)}|\boldsymbol{\phi}^{\{k\}}\big), \qquad (2.44)\end{aligned}$$

which finalizes the E-step of the EM algorithm [43]. Notice that the first E-step requires an initialization of the model parameters. The *M-step* then maximizes the expectation function $\mathcal{Q}\big(\boldsymbol{\Theta}, \boldsymbol{\Theta}_{\text{old}}\big)$ w.r.t. the model parameter $\boldsymbol{\Theta} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\Phi}\}$ and considers $\gamma_k^{(t)}$ and $\xi_{k,j}^{(t)}$ as constant terms. The initial state distribution $\boldsymbol{\pi}$ is obtained utilizing a proper Lagrange multiplier [16]:

$$\sum_{k=1}^{K} \gamma_k^{(1)} \ln \pi_k + \lambda\big(\sum_{k=1}^{K} \pi_k - 1\big), \tag{2.45}$$

and setting the derivatives w.r.t. π_k to zero, leading to the maximization of $\pi_{\text{MLE},k}$:

$$\hat{\pi}_{\text{MLE},k} = \frac{-\gamma_k^{(1)}}{\lambda} = \frac{\gamma_k^{(1)}}{\sum\limits_{j=1}^{K} \gamma_j^{(1)}} \quad \forall k \in \{1, 2, \cdots, K\}, \tag{2.46}$$

with $\lambda = \sum_{j=1}^{K} \gamma_j^{(1)}$, λ being the constant Lagrange multiplier, and $\big(\,\cdot\,\big)_{\text{MLE}}$ defining the obtained variable $\big(\,\cdot\,\big)$ through the approach of MLE. A similar procedure is applied to determine the transition probabilities governed by $\boldsymbol{A}_{\text{MLE}}$:

$$a_{\text{MLE},j,k} = \frac{\sum\limits_{t=1}^{T-1} \xi_{k,j}^{(t)}}{\sum\limits_{l=1}^{K}\sum\limits_{t=1}^{T-1} \xi_{k,l}^{(t)}} = \frac{\sum\limits_{t=1}^{T-1} \xi_{k,j}^{(t)}}{\sum\limits_{t=1}^{T-1} \gamma_k^{(t)}} \quad \forall k, j \in \{1, 2, \cdots, K\}. \tag{2.47}$$

The maximization of $\mathcal{Q}\big(\boldsymbol{\Theta}, \boldsymbol{\Theta}_{\text{old}}\big)$ w.r.t. $\boldsymbol{\phi}^{\{k\}}$ depends on the utilized emission probability distribution. For a single multidimensional Gaussian distribution $\mathcal{N}\big(\boldsymbol{x}|\boldsymbol{\phi}^{\{k\}}\big)$, where $p\big(\boldsymbol{x}|\boldsymbol{\phi}^{\{k\}}\big) = \mathcal{N}\big(\boldsymbol{x}|\boldsymbol{\phi}^{\{k\}}\big)$ and $\boldsymbol{\phi}^{\{k\}} = \{\boldsymbol{\mu}^{\{k\}}, \boldsymbol{\Sigma}^{\{k\}}\}$ gather the mean vector $\boldsymbol{\mu}^{\{k\}} \in \mathbb{R}^I$ and

the co-variance matrix $\boldsymbol{\Sigma}^{\{k\}} \in \mathbb{R}^{I \times I}$, the statements resemble a standard MLE approach for Gaussian models, whereby the responsibility margin $\gamma_k^{(t)}$ is included, resulting in:

$$\boldsymbol{\mu}_{\text{MLE}}^{\{k\}} = \frac{\sum_{t=1}^{T} \gamma_k^{(t)} \boldsymbol{x}^{(t)}}{\sum_{t=1}^{T} \gamma_k^{(t)}}, \tag{2.48}$$

and

$$\boldsymbol{\Sigma}_{\text{MLE}}^{\{k\}} = \frac{\sum_{t=1}^{T} \gamma_k^{(t)} \big(\boldsymbol{x}^{(t)} - \boldsymbol{\mu}_{\text{MLE}}^{\{k\}}\big)\big(\boldsymbol{x}^{(t)} - \boldsymbol{\mu}_{\text{MLE}}^{\{k\}}\big)^T}{\sum_{t=1}^{T} \gamma_k^{(t)}}. \tag{2.49}$$

The mean vector $\boldsymbol{\mu}_{\text{MLE}}^{\{k\}}$ and the co-variance matrix $\boldsymbol{\Sigma}_{\text{MLE}}^{\{k\}}$ are state-dependent parameters.

2.1.6 Viterbi Training

The Viterbi training [84] refers to an approximation of the Baum-Welch training, in which the Viterbi criterion is applied. Thereby, the approximated forward probabilities $\widetilde{\boldsymbol{\alpha}}^{(t)}$ and backward probabilities $\widetilde{\boldsymbol{\beta}}^{(t)}$ lead to a modification of the standard auxiliary margins $\boldsymbol{\gamma}^{(t)}$ into:

$$\widetilde{\gamma}_k^{(t)} = \frac{\widetilde{\alpha}_k^{(t)} \widetilde{\beta}_k^{(t)}}{\sum_{l=1}^{K} \widetilde{\alpha}_l^{(t)} \widetilde{\beta}_l^{(t)}} = \frac{\widetilde{\alpha}_k^{(t)}}{\sum_{l=1}^{K} \widetilde{\alpha}_l^{(t)}} \quad \forall k \in \{1, 2, \cdots, K\}, \quad \forall t \in \{1, 2, \cdots, T\} \tag{2.50}$$

and $\boldsymbol{\xi}^{(t)}$ into $\widetilde{\boldsymbol{\xi}}^{(t)}$ into:

$$\widetilde{\xi}_{k,j}^{(t)} = \frac{\widetilde{\alpha}_k^{(t)} p\big(s_j | s_k\big) p\big(\boldsymbol{x}^{(t+1)} | s_j^{(t+1)}\big) \widetilde{\beta}_j^{(t+1)}}{\sum_{k=1}^{K} \sum_{j=1}^{K} \widetilde{\alpha}_k^{(t)} p\big(s_j | s_k\big) p\big(\boldsymbol{x}^{(t+1)} | s_j^{(t+1)}\big) \widetilde{\beta}_j^{(t+1)}} \quad \substack{\forall k,j \in \{1,2,\cdots,K\}, \\ \forall t \in \{1,2,\cdots,T-1\}.} \tag{2.51}$$

These approximated margins are then employed in the E-step of the EM algorithm [43] and result in a modified expectation function $\widetilde{\mathcal{Q}}\big(\boldsymbol{\Theta}, \boldsymbol{\Theta}_{\text{old}}\big)$, which is $\mathcal{Q}\big(\boldsymbol{\Theta}, \boldsymbol{\Theta}_{\text{old}}\big)$ and sufficient for practical applications.

2.1.7 Beam Search

Although the Viterbi criterion described in Section 2.1.4 allows an efficient procedure for obtaining the most likely latent sequence $\boldsymbol{S}^*$, it requires iterating K times through all entries of $\widetilde{\alpha}_k$, corresponding to the number of entire states $s_k^{(t)}$ in each time step t. Even though it is computationally manageable for low-dimensional latent variables, *e.g.*, a

low number of individual states, it is infeasible for thousands of individual states, as the computational cost would rise excessively.

In 1980, a pruning solution was proposed by Erman *et al.* [48], who suggested solely utilizing a limited number of concurrent paths. Instead of considering all K most likely paths, they applied a beam of the K_{best} most probable paths at the current time t, where K_{best} often refers to the beam size of the beam. Therefore, an ordered set $\mathcal{K}^{(t)}$ can be defined and returned by a mathematical function Top-k$\big(\,\cdot\,,\cdot\,\big)$. The function determines the indices $i^{(l)}$ of the K_{best} highest entries of a regarding vector in descending order:

$$\mathcal{K}^{(t)} = \text{Top-k}\big(\overline{\boldsymbol{\alpha}}^{(t)}, K_{\text{best}}\big) \tag{2.52}$$

where $\mathcal{K}^{(1)} = \{i^{(1)}, \ldots, i^{(l)}, \ldots, i^{(K_{\text{best}})}\}$, and $\overline{\boldsymbol{\alpha}}^{(t)} \in \mathbb{P}^K$ specifies a pruned version of $\widetilde{\boldsymbol{\alpha}}^{(t)}$, where K_{best} entries are unequal to zero, and the remaining entries are zero. Notice that for $t = 1$, the ordered set $\mathcal{K}^{(1)} = \text{Top-k}\big(\widetilde{\boldsymbol{\alpha}}^{(1)}, K_{\text{best}}\big)$ is determined on $\widetilde{\boldsymbol{\alpha}}^{(1)}$, leading to the subsequent iterations:

$$\overline{\alpha}_k^{(t)} = \max_{\forall j \in \mathcal{K}^{(t-1)}} \big[\overline{\alpha}_j^{(t-1)} p\big(s_k|s_j\big)\big] p\big(\boldsymbol{x}^{(t)}|s_k^{(t)}\big) \quad \forall k \in \{1, 2, \cdots, K\}, \tag{2.53}$$

The most likely pruned path is obtained following Section 2.1.4:

$$\overline{\psi}_k^{(t)} = \arg\max_{\forall j \in \mathcal{K}^{(t-1)}} \big[\overline{\alpha}_j^{(t-1)} p\big(s_k|s_j\big)\big] p\big(\boldsymbol{x}^{(t)}|s_k^{(t)}\big) \quad \forall k \in \{1, 2, \cdots, K\}. \tag{2.54}$$

The recursions $\overline{\boldsymbol{\alpha}}^{(t)}$ and $\overline{\boldsymbol{\psi}}^{(t)}$ are calculated up to the time T before applying the backtracking algorithm to retrieve the most likely pruned state sequence $\overline{\boldsymbol{S}}^*$.

2.2 Neural Networks

The following section covers the theoretical background of NNs, also named as DNNs for deeper network structures. Therefore, the concept of perceptrons is introduced, which defines the base unit of NNs. These perceptrons build the foundations of three main structures: The multilayer perceptrons (MLPs), the convolutional neural networks (CNNs), and the recurrent neural networks (RNNs). Besides the mentioned DNN architectures, various activation functions are discussed to achieve non-linear model properties. These DNNs are optimized by stochastic gradient descent in order to generalize to specific data distributions, where the training objective is specified by various loss functions. The corresponding network weight gradients are obtained by backpropagation, where the objective training error between the output of the network and ground truth is determined and then backpropagated through the entire network from the output back to the input layer. Deeper network structures have to cope with issues such as vanishing or exploding gradients, which hinder the training success and lead to non-optimal convergence. Along the standard network training schemes, the robustness of DNNs has to be ensured. Therefore, the section concludes this chapter by presenting multiple regularization strategies, such as batch normalization (BN) and dropout, to prevent overfitting.

2.2.1 The Perceptron

The concept of the perceptron, illustrated in Figure 2.2, was invented by Frank Rosenblatt in 1958 [136]. Inspired by the neurons in the human brain, he sought to develop models for representing these neurons. Standard perceptrons are typically composed of J different units or outputs a_j. Each output a_j is obtained by a weighted sum of the elements x_i of an input $\boldsymbol{x} \in \mathbb{R}^I$. The bias b_j terms allow the perceptron to shift each output a_j and can

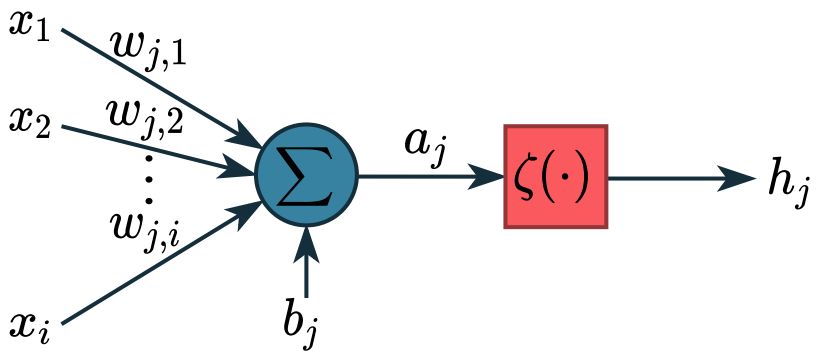


Figure 2.2: The concept of a single perceptron, consisting of multiple output units a_j. Each output unit a_j, shifted by the bias b_j, is generated by a weighted sum of the input $\boldsymbol{x}$. The i-th element of the input $\boldsymbol{x}$ is connected to the j-th unit of the perceptron by a weight $w_{j,i}$. In order to achieve a non-linear property for higher modeling capacity, an activation function $\zeta(\,\cdot\,)$ is employed for each output a_j, leading to the final output h_j.

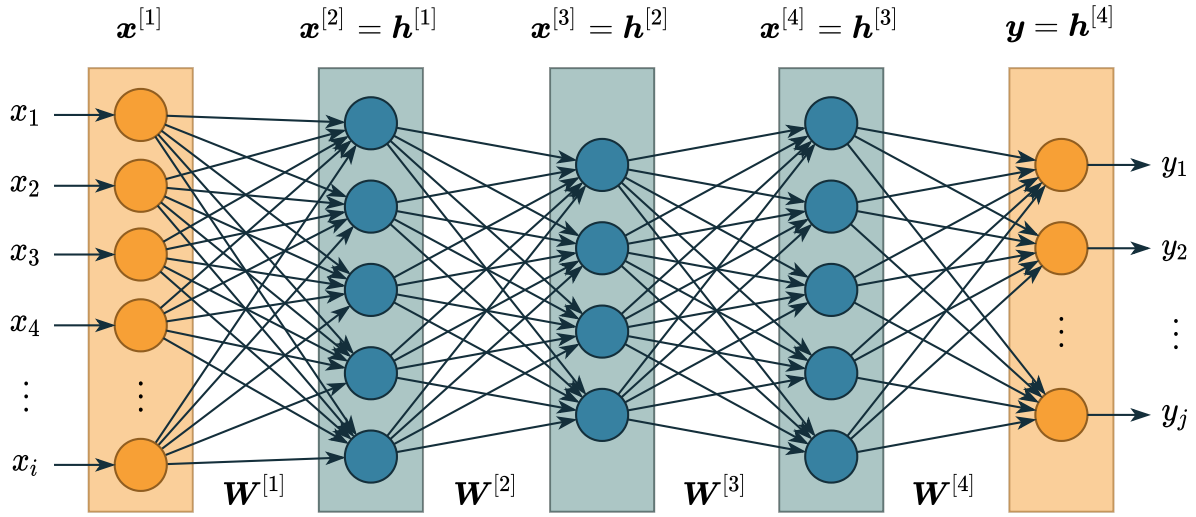


Figure 2.3: The architecture of an MLP, where multiple perceptrons are stacked into layers. The input $\boldsymbol{x}^{[1]} = \boldsymbol{x}$ and the output $\boldsymbol{h}^{[4]} = \boldsymbol{y}$ are highlighted with orange boxes, whereby the hidden layers are marked with bluish boxes with their outputs $\boldsymbol{h}^{[1]}$, $\boldsymbol{h}^{[2]}$, and $\boldsymbol{h}^{[3]}$. The input $\boldsymbol{x}^{[1]}$ is forwarded through the entire MLP to generate the response $\boldsymbol{h}^{[4]} = \boldsymbol{y}$. For simplifications, the bias terms b_j are excluded.

also be interpreted as a threshold before the output a_j is emittable.

The connection of the i-th element of the input $\boldsymbol{x}$ to the j-th unit of the perceptron is specified by the weight $w_{j,i}$, which can be modified depending on the importance of the connection. The final output h_j is retrieved by applying the activation function $\zeta(\,\cdot\,)$ to each unit output a_j, resulting in non-linear perceptron properties.

Multiple perceptrons can be stacked into L layers to generate MLPs, depicted in Figure 2.3. The output of each layer l can be specified in a layer-wise definition:

$$h_j^{[l]} = \zeta^{[l]}\big(a_j^{[l]}\big) = \zeta^{[l]}\Big(\sum_{i=1}^{I^{[l]}} w_{j,i}^{[l]} x_i^{[l]} + b_j^{[l]}\Big) \quad \forall l \in \{1, 2, \cdots, L\}, \tag{2.55}$$

where $(\,\cdot\,)^{[l]}$ specifies the layer l to which the variable or function refers, and L defines the total number of layers, including the input and output layers. The Equation (2.55) can also be rewritten utilizing matrix formulation:

$$\boldsymbol{h}^{[l]} = \zeta^{[l]}\big(\boldsymbol{a}^{[l]}\big) = \zeta^{[l]}\big(\boldsymbol{W}^{[l]}\boldsymbol{x}^{[l]} + \boldsymbol{b}^{[l]}\big) \quad \forall l \in \{1, 2, \cdots, L\}, \tag{2.56}$$

with $\boldsymbol{h}^{[l]}, \boldsymbol{a}^{[l]}, \boldsymbol{b}^{[l]} \in \mathbb{R}^{J^{[l]}}$, $\boldsymbol{x}^{[l]} \in \mathbb{R}^{I^{[l]}}$, and $\boldsymbol{W}^{[l]} \in \mathbb{R}^{J^{[l]} \times I^{[l]}}$. Each layer output $\boldsymbol{h}^{[l]}$ is then fed as the subsequent input $\boldsymbol{x}^{[l+1]}$ of the next layer:

$$\boldsymbol{x}^{[l+1]} = \boldsymbol{h}^{[l]} \quad \forall l \in \{1, 2, \cdots, L\}, \tag{2.57}$$

where the first layer is denoted as the input layer, setting the input $\boldsymbol{x}$ as $\boldsymbol{x}^{[1]} = \boldsymbol{x}$, and the last layer $\boldsymbol{y} = \boldsymbol{h}^{[L]}$ with $\boldsymbol{y} \in \mathbb{R}^{J^{[L]}}$ is termed the output layer. The finalized MLP

consists of multiple layers l, including non-linear functions ζ [18]. The MLPs can be interpreted as a non-linear function $\mathcal{H}_{\boldsymbol{\Theta}}(\,\cdot\,)$, capable of approximating any arbitrary function [69]:

$$\mathcal{H}_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \boldsymbol{y} = \boldsymbol{h}^{[L]} = \boldsymbol{W}^{[L]}\boldsymbol{x}^{[L]} + \boldsymbol{b}^{[L]} \tag{2.58}$$

$$= \boldsymbol{W}^{[L]}\big(\zeta^{[L-1]}\big(\boldsymbol{W}^{[L-1]} \dots \zeta^{[1]}\big(\boldsymbol{W}^{[1]}\boldsymbol{x}^{[1]} + \boldsymbol{b}^{[1]}\big) \dots \boldsymbol{b}^{[L-1]}\big)\big) + \boldsymbol{b}^{[L]}, \tag{2.59}$$

where activation functions ζ are commonly not applied in the output layer $\boldsymbol{y}$, *i.e.*, $\zeta^{[L]}\big(\boldsymbol{a}^{[L]}\big) = \boldsymbol{a}^{[L]}$, to obtain unbounded output values. The ability to approximate any arbitrary function $\mathcal{H}_{\boldsymbol{\Theta}}(\boldsymbol{x})$ by MLPs was already demonstrated by Hornik *et al.* in 1989. They proved in their theoretical work [69] that even MLPs with single hidden layers are sufficient to achieve this objective with high precision. Even though these were promising theoretical results, there are technical limitations in practice. For instance, memory resources are finite, resulting in limited layer units and total layers.

Over the last years, the number of layers, *i.e.*, the depth of DNNs, steadily increased as more computational resources became available. However, the vanishing gradient problem, covered in Section 2.2.9, temporally limited the total depth of these architectures. In 2016, He *et al.* established a solution for this issue [65] by introducing residual connections and enabled DNN architectures with over 1000 layers. In general, depth significantly reduces the number of total network parameters without losing the model capacity of DNNs. Instead of increasing the number of units per layer, the number of layers increases, leading to a higher level of abstraction without modifying the total number of model parameters.

2.2.2 Activation Functions

Activation functions are required to obtain non-linear MLPs, essential to separate data samples in the hyperspace, where linear hyperplanes are insufficient. Besides the non-linearity property, MLPs without applying activation functions cannot learn higher levels of abstraction as these networks are transformable into standard perceptrons. For instance, such transformation can be demonstrated by an MLP with a single hidden layer:

$$\boldsymbol{y} = \boldsymbol{h}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{x}^{[2]} + \boldsymbol{b}^{[2]} \tag{2.60}$$

$$= \boldsymbol{W}^{[2]}\big(\boldsymbol{W}^{[1]}\boldsymbol{x}^{[1]} + \boldsymbol{b}^{[1]}\big) + \boldsymbol{b}^{[2]} \tag{2.61}$$

$$= \boldsymbol{W}^{[2]}\boldsymbol{W}^{[1]}\boldsymbol{x}^{[1]} + \boldsymbol{W}^{[2]}\boldsymbol{b}^{[1]} + \boldsymbol{b}^{[2]} \tag{2.62}$$

$$= \widetilde{\boldsymbol{W}}\boldsymbol{x}^{[1]} + \widetilde{\boldsymbol{b}}, \tag{2.63}$$

where the weight matrices are combined into $\widetilde{\boldsymbol{W}} = \boldsymbol{W}^{[2]}\boldsymbol{W}^{[1]}$ and the bias terms into $\widetilde{\boldsymbol{b}} = \boldsymbol{W}^{[2]}\boldsymbol{b}^{[1]} + \boldsymbol{b}^{[2]}$. The final MLP formulation corresponds to the definition of standard perceptrons in Equation (2.56).

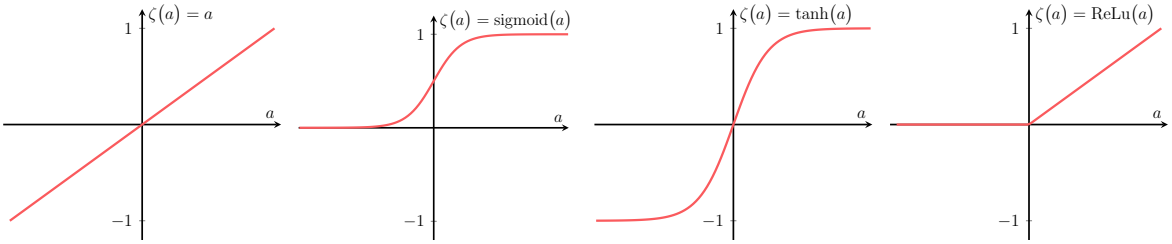


Figure 2.4: Various activation functions, utilized in the DNN layers. The index j is omitted for better readability. The first graph on the left depicts the *identity* or linear function, which passes the output a unaltered to h. In the second graph, the *sigmoid* function is plotted, inspired by the firing neurons of the human brain. The third graph illustrates the *tanh* function, a rescaled version of the sigmoid function. The last graph presents one of the most popular activation functions in computer vision, the *rectified linear unit (ReLU)* function.

Therefore, activation functions need to be employed to achieve the advantage of multiple layers in MLPs. In Rosenblatt's original work [136], a unit step function was employed as an activation function:

$$h = \zeta\big(a\big) = \begin{cases} 0 & \text{if } a < 0 \\ 1 & \text{if } a > 0, \end{cases} \tag{2.64}$$

where a single unit output $a = a_j$ of an MLP is considered, and the corresponding layer indices l are excluded to improve readability.

In recent approaches, a wide range of activation functions have been proposed. The most established activation functions are introduced in the following and are depicted in Figure 2.4. The simplest type of candidate functions are identities, also known as linear layers:

$$\zeta\big(a\big) = a, \tag{2.65}$$

where the activation functions are not modifying the weighted sum a. Identities are typically employed in the output layers of MLPs.

Other candidates are logistic functions, such as the *sigmoid* functions, which correspond to a soft version of the unit step in Equation (2.64):

$$\zeta\big(a\big) = \frac{1}{1 + \exp\big(-a\big)}, \tag{2.66}$$

where $\exp\big(\cdot\big) = \mathrm{e}^{(\cdot)}$ with e referring to Euler's number

The *tanh* functions define rescaled versions of the sigmoid functions:

$$\zeta\big(a\big) = \frac{\exp\big(a\big) - \exp\big(-a\big)}{\exp\big(a\big) + \exp\big(-a\big)}. \tag{2.67}$$

The tanh functions differ primarily in their output range, with $0 < \zeta\big(a\big) < 1$ for the *sigmoid* and $-1 < h < 1$ for the *tanh* functions. Initially, logistic functions were inspired

by the action potential of the human brain [68], as neurons require a certain amount of time for activation in contrast to the direct activations in unit step functions.

The last candidate, the ReLU [108] functions, are determined as:

$$\zeta\big(a\big) = \max\big(0, a\big) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{else,} \end{cases} \tag{2.68}$$

and correspond to the most applied activation functions in current approaches [65, 153, 154]. The function combines two important properties of activation functions: non-linearity and non-vanishing gradients induced by the sparsity of ReLUs. Therefore, very deep MLPs are obtainable in reduced training time [52].

Apart from the most popular activation functions, various other activation functions exist, such as leaky ReLUs [99], gated linear units [41], exponential linear units [37], or Swish units [129], which return slightly better results in task-dependent approaches.

2.2.3 Recurrency

The MLPs can efficiently model any arbitrary function $\mathcal{H}_{\boldsymbol{\Theta}}\big(\boldsymbol{x}\big)$. However, their predictions are limited to single inputs $\boldsymbol{x}^{(t)}$, which do not consider temporal information. Even though it is feasible to utilize these input sequences $\boldsymbol{X}$ by ignoring their temporal information, valuable information would be discarded.

Standard Recurrent Neural Network An extension of standard MLPs capable of considering temporal information are recurrent neural networks (RNNs), where early variations were already described in 1990 [47] and a few years later in [73]. RNNs consist of shared weights and *recurring* model structures, where the hidden representations $\boldsymbol{h}^{[l](t-1)}$ of the previous time step $t-1$ are redirected to the input at time t, and the model weights are shared along the time axis:

$$\boldsymbol{h}^{[l](t)} = \zeta^{[l]}\big(\boldsymbol{W}^{[l]}\boldsymbol{x}^{[l](t)} + \boldsymbol{U}^{[l]}\boldsymbol{h}^{[l](t-1)} + \boldsymbol{b}^{[l]}\big) \quad \forall l \in \{1, 2, \cdots, L\}, \forall t \in \{1, 2, \cdots, T\}. \tag{2.69}$$

The hidden representation $\boldsymbol{h}^{[l](t-1)}$ of the previous time step $t-1$ is often referred to as a hidden state of an RNN [183], which implicitly condenses the temporal component into the fixed-length representation $\boldsymbol{h}^{[l](t-1)}$, weighted by the recurrent weight matrix $\boldsymbol{U} \in \mathbb{R}^{J^{[l]} \times J^{[l]}}$. In practice, RNNs are unfolded T times. The result of the unfolding process of the model is depicted in Figure 2.5, where the RNNs are considered standard MLPs with an additional input $\boldsymbol{h}^{[l](t-1)}$ in each time step t.

Similar to MLPs, RNNs can generate an output $\boldsymbol{y}^{(t)}$ for each time step t:

$$\boldsymbol{y}^{(t)} = \boldsymbol{h}^{[L](t)} = \boldsymbol{W}^{[L]}\boldsymbol{x}^{[L](t)} + \boldsymbol{b}^{[L]} \quad \forall t \in \{1, 2, \cdots, T\}, \tag{2.70}$$

if $\boldsymbol{x}^{[l+1](t)} = \boldsymbol{h}^{[l](t)}$ is set for layer one up to layer $L-1$, including the current time step t.

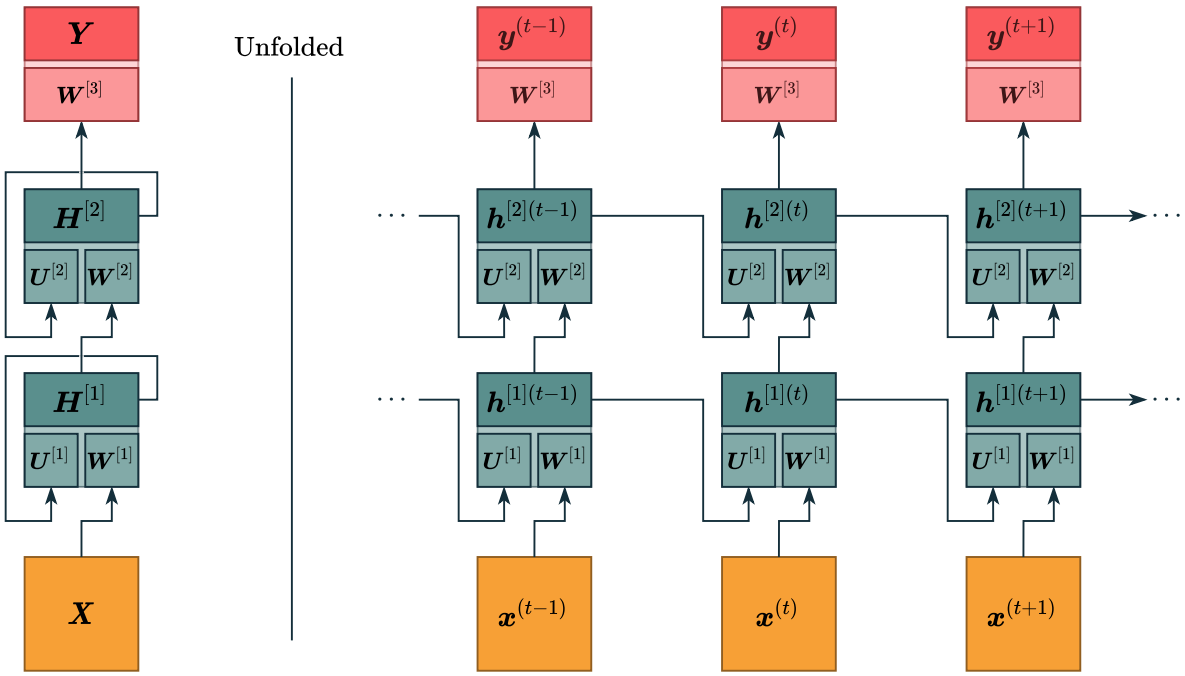


Figure 2.5: The architecture of an RNN. In practice, the RNN is unfolded, where the RNN is copied T times, corresponding to the length of the input sequence $\boldsymbol{X} \in \mathbb{R}^{I \times T}$. The unfolded RNN is then handled as a standard MLP for each time step t. However, the hidden unit output $\boldsymbol{h}^{[l](t-1)}$ has to be stored for the next time step $t+1$.

Long Short-Term Memory Although the recurrence of standard RNNs allows the implicit utilization of temporal information, they lack in several aspects [183]:

1. In long input sequences $\boldsymbol{X}$, RNNs tend to be vulnerable to vanishing gradients since they resemble very deep MLPs, which is caused by the recurrency. The problem of vanishing gradients is explained in Section 2.2.9.

2. The hidden state $\boldsymbol{h}^{[l](t)}$ cannot be stored for a specific time step t and then applied in later time steps, which is required if valuable information needs to be transferred from the beginning to the end of the sequence $\boldsymbol{X}$.

3. If hidden states $\boldsymbol{h}^{[l](t-1)}$ are memorized and radical changes between the input $\boldsymbol{x}^{(t)}$ and $\boldsymbol{x}^{(t+1)}$ occur, a reset or forget mechanism is required to avoid over-dominant impacts.

4. Sometimes, the hidden states $\boldsymbol{h}^{[l](t)}$ contain barely new information. Therefore, it would be beneficial to skip or lower the impact of the current hidden representation $\boldsymbol{h}^{[l](t)}$ on future representations.

In 1997, Hochreiter *et al.* proposed an extension of standard RNNs. They established the long short-term memory networks (LSTMs) [67], which tackled all the aspects mentioned above. In their approach, they created a cell with a memory state. The cell is built out

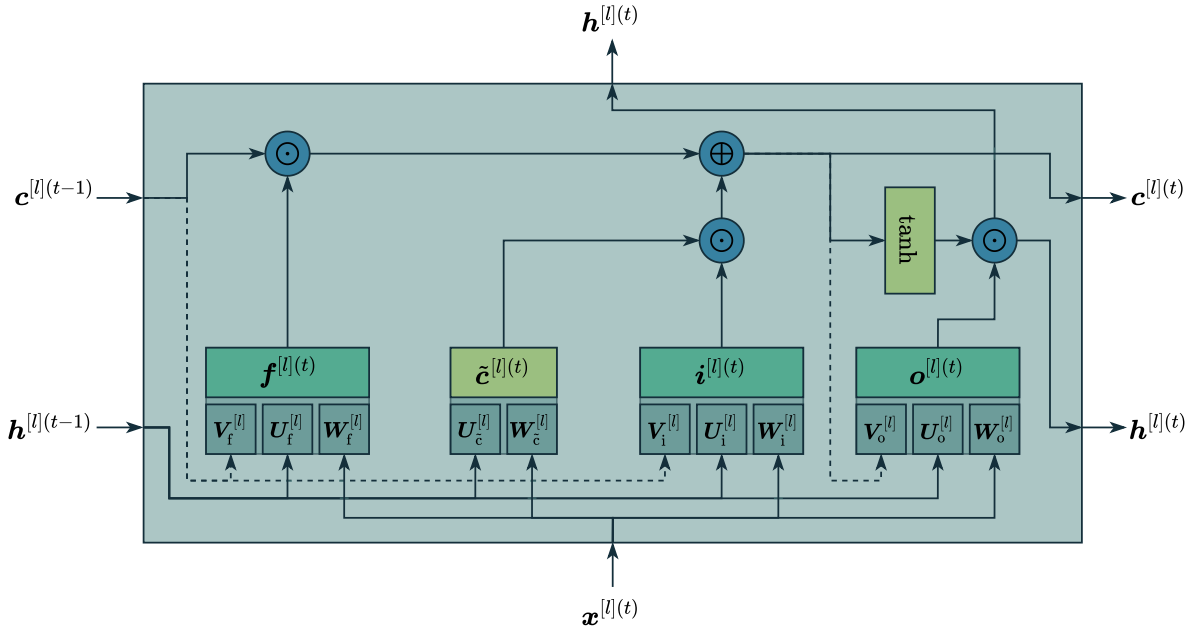


Figure 2.6: The architecture of LSTMs with peephole connections. The LSTMs are governed by four gate mechanisms: the forget gate $\boldsymbol{f}^{[l](t)}$, the input gate $\boldsymbol{i}^{[l](t)}$, the output gate $\boldsymbol{o}^{[l](t)}$, and an auxiliary memory state $\widetilde{\boldsymbol{c}}^{[l](t)}$. These gates are trainable MLPs and control the internal memory state $\boldsymbol{c}^{[l](t)}$ based on input sequences $\boldsymbol{X}$. In the forget, input, and output gates, sigmoid activation functions are applied, represented by darker greenish boxes around the gate functions. In the auxiliary memory state, the tanh activation functions are utilized, depicted by brighter greenish boxes. The dashed lines refer to the *peephole* connections [50].

of several gate mechanisms, represented by perceptrons, which flexibly process the input $\boldsymbol{x}^{[l](t)}$ and the hidden state $\boldsymbol{h}^{[l](t-1)}$ in each layer l. A few years later, the standard LSTM approach was modified to utilize the memory state in the gate modules of the model [50]. Both LSTM variations are depicted in Figure 2.6, in which the dashed lines refer to the modifications in LSTMs with peephole connections.

The input gate $\boldsymbol{i}^{[l](t)} \in \mathbb{P}^{J^{[l]}}$, the forget gate $\boldsymbol{f}^{[l](t)} \in \mathbb{P}^{J^{[l]}}$, and the output gate $\boldsymbol{o}^{[l](t)} \in \mathbb{P}^{J^{[l]}}$ is defined as:

$$\boldsymbol{i}^{[l](t)} = \mathrm{sigmoid}\big(\boldsymbol{W}_{\mathrm{i}}^{[l]}\boldsymbol{x}^{[l](t)} + \boldsymbol{U}_{\mathrm{i}}^{[l]}\boldsymbol{h}^{[l](t-1)} + \boldsymbol{V}_{\mathrm{i}}^{[l]}\boldsymbol{c}^{[l](t-1)} + \boldsymbol{b}_{\mathrm{i}}^{[l]}\big) \quad \substack{\forall l \in \{1,2,\cdots,L\} \\ \forall t \in \{1,2,\cdots,T\},} \tag{2.71}$$

$$\boldsymbol{f}^{[l](t)} = \mathrm{sigmoid}\big(\boldsymbol{W}_{\mathrm{f}}^{[l]}\boldsymbol{x}^{[l](t)} + \boldsymbol{U}_{\mathrm{f}}^{[l]}\boldsymbol{h}^{[l](t-1)} + \boldsymbol{V}_{\mathrm{f}}^{[l]}\boldsymbol{c}^{[l](t-1)} + \boldsymbol{b}_{\mathrm{f}}^{[l]}\big) \quad \substack{\forall l \in \{1,2,\cdots,L\} \\ \forall t \in \{1,2,\cdots,T\},} \tag{2.72}$$

$$\boldsymbol{o}^{[l](t)} = \mathrm{sigmoid}\big(\boldsymbol{W}_{\mathrm{o}}^{[l]}\boldsymbol{x}^{[l](t)} + \boldsymbol{U}_{\mathrm{o}}^{[l]}\boldsymbol{h}^{[l](t-1)} + \boldsymbol{V}_{\mathrm{o}}^{[l]}\boldsymbol{c}^{[l](t)} + \boldsymbol{b}_{\mathrm{o}}^{[l]}\big) \quad \substack{\forall l \in \{1,2,\cdots,L\} \\ \forall t \in \{1,2,\cdots,T\},} \tag{2.73}$$

where $\boldsymbol{W}_{\mathrm{i}}^{[l]}, \boldsymbol{W}_{\mathrm{f}}^{[l]}, \boldsymbol{W}_{\mathrm{o}}^{[l]} \in \mathbb{R}^{J^{[l]} \times I^{[l]}}$, $\boldsymbol{U}_{\mathrm{i}}^{[l]}, \boldsymbol{U}_{\mathrm{f}}^{[l]}, \boldsymbol{U}_{\mathrm{o}}^{[l]} \in \mathbb{R}^{J^{[l]} \times J^{[l]}}$, and $\boldsymbol{V}_{\mathrm{i}}^{[l]}, \boldsymbol{V}_{\mathrm{f}}^{[l]}, \boldsymbol{V}_{\mathrm{o}}^{[l]} \in \mathbb{R}^{J^{[l]} \times J^{[l]}}$ are weight matrices, $\boldsymbol{b}_{\mathrm{i}}^{[l]}, \boldsymbol{b}_{\mathrm{f}}^{[l]}, \boldsymbol{b}_{\mathrm{o}}^{[l]} \in \mathbb{R}^{J^{[l]}}$ are the biases, and the terms regarding $\boldsymbol{c}^{[l](t-1)}$ refer to the newly added peephole connections proposed in [50]. All three gates are standard perceptrons whose activation functions are set to *sigmoid* functions.

In order to memorize essential aspects at a specific time step t in the sequence, a

memory state is added. Similar to the gate perceptrons in Equations (2.71) to (2.73), an additional perceptron, which represents an auxiliary memory state $\widetilde{\boldsymbol{c}}^{[l](t)} \in \{c \in \mathbb{R} \,|\, -1 \leq c \leq 1\}^{J^{[l]}}$, is employed:

$$\widetilde{\boldsymbol{c}}^{[l](t)} = \tanh\left(\boldsymbol{W}_{\widetilde{\mathrm{c}}}^{[l]} \boldsymbol{x}^{[l](t)} + \boldsymbol{U}_{\widetilde{\mathrm{c}}}^{[l]} \boldsymbol{h}^{[l](t-1)} + \boldsymbol{b}_{\widetilde{\mathrm{c}}}^{[l]}\right) \quad \forall l \in \{1, 2, \cdots, L\}, \forall t \in \{1, 2, \cdots, T\}. \tag{2.74}$$

In contrast to the gates defined above, the auxiliary memory states $\widetilde{\boldsymbol{c}}^{[l](t)}$ do not have access to the memory states $\boldsymbol{c}^{[l](t)}$, and their activation functions are set to tanh functions. The remaining parameters $\boldsymbol{W}_{\widetilde{\mathrm{c}}}^{[l]} \in \mathbb{R}^{J^{[l]} \times I^{[l]}}$ and $\boldsymbol{U}_{\widetilde{\mathrm{c}}}^{[l]} \in \mathbb{R}^{J^{[l]} \times I^{[l]}}$ refer to weight matrices and $\boldsymbol{b}_{\widetilde{\mathrm{c}}}^{[l]} \in \mathbb{R}^{J^{[l]}}$ to a bias vector. Since the final memory state $\boldsymbol{c}^{[l](t)} \in \mathbb{R}^{J^{[l]}}$ requires options for skipping, updating, or forgetting the current memory state $\boldsymbol{c}^{[l](t)}$, it is specified as:

$$\boldsymbol{c}^{[l](t)} = \boldsymbol{f}^{[l](t)} \odot \boldsymbol{c}^{[l](t-1)} + \boldsymbol{i}^{[l](t)} \odot \widetilde{\boldsymbol{c}}^{[l](t)} \quad \forall l \in \{1, 2, \cdots, L\}, \forall t \in \{1, 2, \cdots, T\}, \tag{2.75}$$

where $\boldsymbol{f}^{[l](t)}$ and $\boldsymbol{i}^{[l](t)}$ correspond to the forget and input gates, respectively, and the operator $\odot$ defines the Hadamard product, *i.e.*, the elementwise product of two vectors or matrices. These gate perceptrons generate the current memory state $\boldsymbol{c}^{[l](t)}$ by either addressing more importance on past memory states $\boldsymbol{c}^{[l](t-1)}$ or by addressing more relevance to the current input $\widetilde{\boldsymbol{c}}^{[l](t)}$, which leads to a reduction of the forget gates $\boldsymbol{f}^{[l](t)}$, or an increase of the input gates $\boldsymbol{i}^{[l](t)}$, respectively.

Finally, the hidden state $\boldsymbol{h}^{[l](t)} \in \{c \in \mathbb{R} \,|\, -1 \leq c \leq 1\}^{J^{[l]}}$ is obtained by a combination of the output gate $\boldsymbol{o}^{[l](t)}$ and the memory state $\boldsymbol{c}^{[l](t)}$:

$$\boldsymbol{h}^{[l](t)} = \boldsymbol{o}^{[l](t)} \odot \tanh\left(\boldsymbol{c}^{[l](t)}\right) \quad \forall l \in \{1, 2, \cdots, L\}, \forall t \in \{1, 2, \cdots, T\}, \tag{2.76}$$

where the activation functions are set to *tanh* functions. The composition of $\boldsymbol{h}^{[l](t)}$ also demonstrates the advantage of LSTMs, where stored information from the past can be stored in the memory state $\boldsymbol{c}^{[l](t)}$, forwarded over long temporal distances, and then actively considered in the current hidden state $\boldsymbol{h}^{[l](t)}$. Notice that the outputs $\boldsymbol{y}^{(t)} = \boldsymbol{h}^{[L](t)}$ for the current time step t can be obtained by the similar output procedure of standard RNNs (see Equation (2.70)).

Along LSTMs, gated recurrent units (GRUs) exist [31], offering another extension approach for RNNs. Even though LSTMs exist longer than GRUs, a recent study could not identify major performance improvements in employing GRUs over LSTMs [36].

Long Short-Term Memory Projected A further extension of standard LSTMs [50] is long short-term memory projected networks (LSTMPs). These networks were proposed by Sak *et al.* [139], who applied projection layers to reduce the dimension of the hidden state $\boldsymbol{h}^{[l](t)}$. Thereby, the LSTMPs were forced to employ their parameters more efficiently, yielding a significant performance improvement. The simple yet efficient approach introduced a recurrent projection layer $\boldsymbol{p}^{[l](t)}$:

$$\boldsymbol{p}^{[l](t)} = \boldsymbol{W}_{\mathrm{p}}^{[l]} \boldsymbol{h}^{[l](t)} \quad \forall l \in \{1, 2, \cdots, L\}, \forall t \in \{1, 2, \cdots, T\}, \tag{2.77}$$

where $\boldsymbol{p}^{[l](t)} \in \mathbb{R}^{\widetilde{J}^{[l]}}$ refers to a dimension-reduced projection of $\boldsymbol{h}^{[l](t)} \in \mathbb{R}^{J^{[l]}}$, *i.e.*, $\widetilde{J}^{[l]} < J^{[l]}$, and $\boldsymbol{W}_{\mathrm{p}}^{[l]} \in \mathbb{R}^{\widetilde{J}^{[l]} \times J^{[l]}}$ to the regarding projection matrix. In their experiments, satisfying results were typically obtained for $\widetilde{J}^{[l]} = \frac{J^{[l]}}{4}$ [29]. The projection $\boldsymbol{p}^{[l](t-1)}$ then replaced the hidden state $\boldsymbol{h}^{[l](t-1)}$ of standard LSTMs in Equations (2.71) to (2.74):

$$\boldsymbol{i}^{[l](t)} = \mathrm{sigmoid}\big(\boldsymbol{W}_{\mathrm{i}}^{[l]} \boldsymbol{x}^{[l](t)} + \widetilde{\boldsymbol{U}}_{\mathrm{i}}^{[l]} \boldsymbol{p}^{[l](t-1)} + \boldsymbol{V}_{\mathrm{i}}^{[l]} \boldsymbol{c}^{[l](t-1)} + \boldsymbol{b}_{\mathrm{i}}^{[l]}\big) \quad \substack{\forall l \in \{1,2,\cdots,L\} \\ \forall t \in \{1,2,\cdots,T\},} \tag{2.78}$$

$$\boldsymbol{f}^{[l](t)} = \mathrm{sigmoid}\big(\boldsymbol{W}_{\mathrm{f}}^{[l]} \boldsymbol{x}^{[l](t)} + \widetilde{\boldsymbol{U}}_{\mathrm{f}}^{[l]} \boldsymbol{p}^{[l](t-1)} + \boldsymbol{V}_{\mathrm{f}}^{[l]} \boldsymbol{c}^{[l](t-1)} + \boldsymbol{b}_{\mathrm{f}}^{[l]}\big) \quad \substack{\forall l \in \{1,2,\cdots,L\} \\ \forall t \in \{1,2,\cdots,T\},} \tag{2.79}$$

$$\boldsymbol{o}^{[l](t)} = \mathrm{sigmoid}\big(\boldsymbol{W}_{\mathrm{o}}^{[l]} \boldsymbol{x}^{[l](t)} + \widetilde{\boldsymbol{U}}_{\mathrm{o}}^{[l]} \boldsymbol{p}^{[l](t-1)} + \boldsymbol{V}_{\mathrm{o}}^{[l]} \boldsymbol{c}^{[l](t)} + \boldsymbol{b}_{\mathrm{o}}^{[l]}\big) \quad \substack{\forall l \in \{1,2,\cdots,L\} \\ \forall t \in \{1,2,\cdots,T\},} \tag{2.80}$$

$$\widetilde{\boldsymbol{c}}^{[l](t)} = \tanh\big(\boldsymbol{W}_{\widetilde{\mathrm{c}}}^{[l]} \boldsymbol{x}^{[l](t)} + \widetilde{\boldsymbol{U}}_{\widetilde{\mathrm{c}}}^{[l]} \boldsymbol{p}^{[l](t-1)} + \boldsymbol{b}_{\widetilde{\mathrm{c}}}^{[l]}\big) \quad \substack{\forall l \in \{1,2,\cdots,L\} \\ \forall t \in \{1,2,\cdots,T\},,} \tag{2.81}$$

and also reduced the number of parameters in the weight matrices $\widetilde{\boldsymbol{U}}_{\mathrm{i}}^{[l]}, \widetilde{\boldsymbol{U}}_{\mathrm{f}}^{[l]}, \widetilde{\boldsymbol{U}}_{\mathrm{o}}^{[l]} \in \mathbb{R}^{J^{[l]} \times \widetilde{J}^{[l]}}$. Deep LSTMPs are retrieved by following the standard RNN output procedure in Equation (2.70).

Bidirectional Long Short-Term Memory Projected Standard LSTMs offer an efficient approach for generating hidden representations $\boldsymbol{h}^{[l](t)}$ relying on past temporal context. However, if the entire input sequence $\boldsymbol{X}$ exists, it is also beneficial to employ future context for the current hidden state $\boldsymbol{h}^{[l](t)}$. In 1997, Schuster *et al.* proposed the first bidirectional RNN architecture [142], which utilized the past and future context. Later, Graves *et al.* transferred this idea to bidirectional long short-term memory networks (BLSTMs), which were applied to phoneme classification [58]. A major drawback of BLSTMs is caused by a large increase in model parameters since two individual LSTMs are required to consider the past and future context independently, which results in a doubled number of parameters compared to standard LSTMs.

This issue is resolved by first applying LSTMPs for the past context with the projected hidden states $\overrightarrow{\boldsymbol{p}}^{[l](t)}$:

$$\overrightarrow{\boldsymbol{p}}^{[l](t)} = \overrightarrow{\boldsymbol{W}}_{\mathrm{p}}^{[l]} \overrightarrow{\boldsymbol{h}}^{[l](t)} \quad \forall l \in \{1, 2, \cdots, L\}, \forall t \in \{1, 2, \cdots, T\}, \tag{2.82}$$

and then additional LSTMPs for the future context with the projected hidden representations $\overleftarrow{\boldsymbol{p}}^{[l](t)}$ [57],

$$\overleftarrow{\boldsymbol{p}}^{[l](t)} = \overleftarrow{\boldsymbol{W}}_{\mathrm{p}}^{[l]} \overleftarrow{\boldsymbol{h}}^{[l](t)} \quad \forall l \in \{1, 2, \cdots, L\}, \forall t \in \{1, 2, \cdots, T\}, \tag{2.83}$$

where $\overrightarrow{\boldsymbol{p}}^{[l](t)}, \overleftarrow{\boldsymbol{p}}^{[l](t)} \in \mathbb{R}^{\widetilde{J}^{[l]}}$, and $\overrightarrow{\boldsymbol{W}}_{\mathrm{p}}^{[l]}, \overleftarrow{\boldsymbol{W}}_{\mathrm{p}}^{[l]} \in \mathbb{R}^{\widetilde{J}^{[l]} \times J^{[l]}}$. The hidden states $\overrightarrow{\boldsymbol{h}}^{[l](t)}$ and $\overleftarrow{\boldsymbol{h}}^{[l](t)}$ are determined following Equations (2.78) to (2.81), and the symbols $\overrightarrow{(\cdot)}$ and $\overleftarrow{(\cdot)}$ refer to the parameters or outputs following the past and future projected hidden state sequence, respectively.

In order to return the final and deep bidirectional long short-term memory projected (BLSTMP) architecture, the projected hidden states of the current layer l are concatenated and fed as the new input to the subsequent forward and backward LSTMPs layers:

$$\boldsymbol{x}^{[l+1](t)} = \overleftrightarrow{\boldsymbol{p}}^{[l](t)} = \overrightarrow{\boldsymbol{p}}^{[l](t)} \oplus \overleftarrow{\boldsymbol{p}}^{[l](t)} \quad \forall l \in \{1, 2, \cdots, L\}, \tag{2.84}$$

where $\oplus$ defines a concatenate operator between two vectors or matrices. The concatenation between $\overrightarrow{\boldsymbol{p}}^{[l](t)} \oplus \overleftarrow{\boldsymbol{p}}^{[l](t)}$ results in the vector $\overleftrightarrow{\boldsymbol{p}}^{[l](t)} \in \mathbb{R}^{2\tilde{J}^{[l]}}$. The final BLSTMP output $\boldsymbol{y}^{(t)}$ for the time step t is obtained by combining $\overrightarrow{\boldsymbol{p}}^{[L](t)}$ and $\overleftarrow{\boldsymbol{p}}^{[L](t)}$ in the final layer L:

$$\boldsymbol{y}^{(t)} = \overrightarrow{\boldsymbol{W}}^{[L]} \overrightarrow{\boldsymbol{p}}^{[L](t)} + \overleftarrow{\boldsymbol{W}}^{[L]} \overleftarrow{\boldsymbol{p}}^{[L](t)} + \boldsymbol{b}^{[L]} \quad \forall t \in \{1, 2, \cdots, T\}. \tag{2.85}$$

2.2.4 Convolution

Even though MLPs and RNNs can approximate any arbitrary function [69], the size of the inputs $\boldsymbol{x}^{[l](t)}$ to each layer l strongly influences the required computational resources for calculating the corresponding matrix operations. Since MLPs and RNNs can solely process one-dimensional input data $\boldsymbol{x} \in \mathbb{R}^I$, higher-dimensional data structures, such as images $\boldsymbol{I} \in \mathbb{R}^{M \times N}$, are typically vectorized if they need to be used in these models. Therefore, a function $\mathrm{vec}\big(\,\cdot\,\big)$ is defined as:

$$\mathrm{vec}\big(\boldsymbol{I}\big) = \big(i_{1,1}, i_{1,2}, \ldots, i_{2,1}, i_{2,2}, \ldots, i_{M,N-1}, i_{M,N}\big)^\top, \tag{2.86}$$

where $\boldsymbol{I}$ refers to a two-dimensional image with its entries $i_{m,n}$. However, even if small images are vectorized, MLPs are fed by high-dimensional input vectors $\boldsymbol{x} = \mathrm{vec}\big(\boldsymbol{I}\big)$ of high dimension, *i.e.*, $\boldsymbol{x} \in \mathbb{R}^{MN}$.

As MLPs are infeasible to process high-dimensional data efficiently without applying dimension-reducing techniques beforehand, and small input shifts in $\boldsymbol{x}$ lead to major parameter changes in training, CNNs suggest an alternative way of processing high-dimensional data [83]. Thereby, CNNs apply three key concepts: parameter sharing, translation equivariant, and sparse connections [82, 183].

A convolutional layer of CNNs can be derived from standard perceptrons [183] if $\boldsymbol{X} \in \mathbb{R}^{I \times J}$ refers to two-dimensional inputs and $\boldsymbol{H} \in \mathbb{R}^{I \times J}$ to two-dimensional hidden representations of a perceptron:

$$h_{i,j} = \zeta\big(b_{i,j} + \sum_o \sum_p w_{i,j,o,p} x_{o,p}\big) \quad \forall i \in \{1, 2, \cdots, I\}, \forall j \in \{1, 2, \cdots, J\} \tag{2.87}$$

$$= \zeta\big(b_{i,j} + \sum_h \sum_w \widetilde{k}_{i,j,h,w} x_{i+h,j+w}\big) \quad \forall i \in \{1, 2, \cdots, I\}, \forall j \in \{1, 2, \cdots, J\}, \tag{2.88}$$

where the subscripts $\big(o, p\big)$ re-indexed by $\big(o = i + h, p = j + w\big)$. The initial weight matrix is replaced by a kernel $\widetilde{\boldsymbol{K}} \in \mathbb{R}^{I \times J \times \widetilde{H} \times \widetilde{W}}$ defined as $\widetilde{\boldsymbol{K}}_{i,j,h,w} = \boldsymbol{W}_{i,j,i+h,j+w}$, and

$b_{i,j}$ refers to an entry of the two-dimensional bias matrix $\boldsymbol{B} \in \mathbb{R}^{I \times J}$. The kernel matrix $\widetilde{\boldsymbol{K}}_{i,j,h,w}$ exists for each position $\big(h, w\big)$ in the input matrix $\boldsymbol{X}$ and is spatial-dependent on regions $\big(i, j\big)$. For instance, unique position-dependent kernels are learned if similar features are placed at different positions in the input $\boldsymbol{X}$. Since this is highly inefficient caused of the number of required parameters, a single kernel $\widetilde{\boldsymbol{K}}$ is shared along each position $\big(i, j\big)$:

$$h_{i,j} = \zeta\big(b + \sum_{h} \sum_{w} \widetilde{\boldsymbol{K}}_{h,w} x_{i+h,j+w}\big) \quad \forall i \in \{1, 2, \cdots, I\}, \forall j \in \{1, 2, \cdots, J\}, \tag{2.89}$$

yielding in a single scalar bias $b \in \mathbb{R}$ and a dimensional-reduced kernel $\widetilde{\boldsymbol{K}} \in \mathbb{R}^{\widetilde{H} \times \widetilde{W}}$. In general, the Equation (2.89) already defines the *convolutional* operation between a kernel $\widetilde{\boldsymbol{K}}$ and an input $\boldsymbol{X}$. Furthermore, the utilization of a shared kernel $\widetilde{\boldsymbol{K}}$ induces translation equivariance, where a specific shift in the input $\boldsymbol{X}$ equals a similar shift in the output $\boldsymbol{H}$. Notice that the concept of equivariance is not present in MLPs, as a minor modification in the input would lead to major changes in the output.

So far, there is no advantage in utilizing convolutional layers compared to standard MLPs since the size of the kernel $\widetilde{\boldsymbol{K}}$ is not limited and could be equal to the size of the input $\boldsymbol{X}$. In practice, processing two-dimensional data, such as images, requires detecting specific local features, such as edges. Since these features are locally bounded, smaller kernels $\boldsymbol{K} \in \mathbb{R}^{H^{[l]} \times W^{[l]}}$ are sufficient to detect those:

$$h_{i,j}^{[l]} = \zeta^{[l]}\big(b^{[l]} + \sum_{h=1}^{H^{[l]}} \sum_{w=1}^{W^{[l]}} k_{h,w}^{[l]} x_{i+h,j+w}^{[l]}\big) \quad \begin{smallmatrix} \forall\ 1 \leq i \leq I^{[l]} - H^{[l]} + 1, \\ \forall\ 1 \leq j \leq J^{[l]} - W^{[l]} + 1, \\ \forall l \in \{1, 2, \cdots, L\}, \end{smallmatrix} \tag{2.90}$$

with $H^{[l]} \ll I^{[l]}$ and $W^{[l]} \ll J^{[l]}$ being the dimensions of the reduced kernel $\boldsymbol{K}$, which greatly decreases the number of required kernel parameters. Notice that in the formulation above, the stride length is set to one, and no zero-padding is applied. The stride length s refers to every position $\big(si, sj\big)$ of input $\boldsymbol{X}^{[l]}$ in which the convolution operation is employed. The zero-padding specifies an operation to sustain the dimensions of the output $h_{i,j}^{[l]}$. Therefore, outer rows and columns of $\boldsymbol{X}^{[l]}$ are filled with zero values added. If the zero-padding procedure is skipped, the dimension reduction of $h_{i,j}^{[l]}$ can be observed in Equation (2.90), where the dimensions of $\boldsymbol{H}^{[l]} \in \mathbb{R}^{(I^{[l]} - H^{[l]} + 1 < I^{[l]}) \times (I^{[l]} - W^{[l]} + 1 < J^{[l]})}$ are reduced compared to the dimensions of $\boldsymbol{X}^{[l]} \in \mathbb{R}^{I^{[l]} \times J^{[l]}}$.

The finalized Equation (2.90) corresponds to the standard definition of a convolutional layer in CNNs [171, 183]. Typically, CNNs are further extended for multiple input and output channels. For instance, a single-layer CNN with one input channel and $Z^{[l]}$ output channels would yield the kernel dimensions $\boldsymbol{K} \in \mathbb{R}^{Z^{[l]} \times 1 \times H^{[l]} \times W^{[l]}}$.

Besides the standard CNNs, other variants can be found in dilated convolutions [177], where sparse kernels are employed, or deformable convolutions [40], where each kernel weight can be shifted continuously instead of symmetric kernel shifts.

Pooling A drawback of the hidden representations $h_{i,j}^{[l]}$ of convolutional layers refers to the missing translation invariance property, which can be achieved by pooling layers

summarizing the statistics of $h_{i,j}^{[l]}$ [82]. The most popular pooling layers are *max-pooling* layers [171]:

$$h_{i,j}^{[l]} = \max_{1\leq o\leq O^{[l]}} \max_{1\leq p\leq P^{[l]}} x_{O^{[l]}i+o,P^{[l]}j+p}^{[l]} \quad \begin{matrix}\forall i\in\left\{0,1,\cdots,\left\lceil\frac{I^{[l]}}{O^{[l]}}\right\rceil-1\right\},\forall j\in\left\{0,1,\cdots,\left\lceil\frac{J^{[l]}}{P^{[l]}}\right\rceil-1\right\},\\ \forall l\in\{1,2,\cdots,L\},\end{matrix} \tag{2.91}$$

and *average-pooling* layers:

$$h_{i,j}^{[l]} = \frac{1}{O^{[l]}P^{[l]}} \sum_{o=1}^{O^{[l]}} \sum_{p=1}^{P^{[l]}} x_{O^{[l]}i+o,P^{[l]}j+p}^{[l]} \quad \begin{matrix}\forall i\in\left\{0,1,\cdots,\left\lceil\frac{I^{[l]}}{O^{[l]}}\right\rceil-1\right\},\forall j\in\left\{0,1,\cdots,\left\lceil\frac{J^{[l]}}{P^{[l]}}\right\rceil-1\right\},\\ \forall l\in\{1,2,\cdots,L\}.\end{matrix} \tag{2.92}$$

In these layers, the input $\boldsymbol{X}^{[l]}$ is divided into $\left\lceil\frac{I^{[l]}}{O^{[l]}}\right\rceil \times \left\lceil\frac{J^{[l]}}{P^{[l]}}\right\rceil$ non-overlapping subregions $\boldsymbol{X}_{\text{sub}} \in \mathbb{R}^{O^{[l]}\times P^{[l]}}$, with $\lceil\cdot\rceil$ defining the ceiling operation. Each subregion $\boldsymbol{X}_{\text{sub}}$ is then represented either by the maximum value, *i.e.*, in max-pooling layers, or by an average value, *i.e.*, in the average pooling layer, of these subregions. In this process, the information on the precise input position is lost, and solely the information of an approximated position is retained, leading to the translation invariance property.

2.2.5 Training Criterion

The sections above presented the fundamental layer types of several current state-of-the-art architectures. All model parameters representing standard DNNs have to be optimized or trained based on a predefined *training criterion*. In *supervised training*, where labeled datasets $\mathcal{D} = \left\{(\boldsymbol{X}^{(n)}, \hat{\boldsymbol{Y}}^{(n)})\right\}_{n=1}^{N}$ exist, the optimization criterion can be defined either as a regression or a classification task.

Regression Task The regression tasks can be considered curve-fitting tasks, where continuous functions have to be determined, which can represent the datasets $\mathcal{D}$. The target label $\hat{\boldsymbol{y}}^{(t)} \in \hat{\boldsymbol{Y}}^{(n)}$ in the datasets $\mathcal{D}$ are continuous vectors of any size J, leading to continuous outputs $\boldsymbol{Y}$ of the obtained function $\mathcal{H}_{\boldsymbol{\Theta}}$ with equal dimensions J [18].

Classification Task In classification tasks, model functions $\mathcal{H}_{\boldsymbol{\Theta}}$ classify inputs $\boldsymbol{x}^{(t)} \in \boldsymbol{X}^{(n)}$ and assign them to a particular category, where the number of categories is finite [18]. The target labels $\hat{y}^{(t)} \in \hat{\boldsymbol{Y}}^{(n)}$ in the dataset $\mathcal{D}$ are typically discrete and one-hot encoded to obtain $\hat{\boldsymbol{y}}^{(t)} \in \mathbb{B}^J$ with $\|\hat{\boldsymbol{y}}^{(t)}\| = 1$ (compare to Equation (2.3))

Generally, the training criterion is defined as follows:

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} L\big(\mathcal{H}_{\boldsymbol{\Theta}}\big(\boldsymbol{x}\big), \hat{\boldsymbol{y}}\big), \tag{2.93}$$

where $\boldsymbol{\Theta}$ gathers all the model parameters, *i.e.*, all weights $w_{j,i}^{[l]}$ and bias $b_j^{[l]}$, $\boldsymbol{\Theta}^*$ represents the optimal model parameters based on a loss function $L\big(\boldsymbol{y}, \hat{\boldsymbol{y}}\big)$.

2.2.6 Loss

The loss functions define a training criterion to determine optimal model parameters $\boldsymbol{\Theta}$ by minimizing its function value. The impact of the loss function on the network parameters, *i.e.*, the gradients of the loss w.r.t. $\boldsymbol{\Theta}$, are derived by the backpropagation (BP) algorithm, introduced in the following subsection. Therefore, the chosen loss functions need to be differentiable to apply the BP algorithm.

The Mean Squared Error Loss Function The most popular loss function candidate in the mean squared error (MSE) loss $L_{\mathrm{MSE}}(\,\cdot\,,\cdot\,)$, utilized in several regression tasks since it defines an intuitive and straightforward training criterion:

$$L_{\mathrm{MSE}}(\boldsymbol{y},\hat{\boldsymbol{y}}) = \frac{1}{J}\sum_{j=1}^{J}\big(y_j-\hat{y}_j\big)^2, \tag{2.94}$$

where J specifies the dimension of the DNN outputs $\boldsymbol{y}$ and the target label $\hat{\boldsymbol{y}}$. The gradients of the loss w.r.t. the outputs $\boldsymbol{y}$ of the DNN are obtained by:

$$\frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{y}} = \frac{2}{J}(\boldsymbol{y}-\hat{\boldsymbol{y}}), \tag{2.95}$$

corresponding to a scaled subtraction between the outputs $\boldsymbol{y}$ and the target labels $\hat{\boldsymbol{y}}$.

The Kullback-Leibler Divergence Loss Function The Kullback-Leibler divergence (KLD) loss [80] defines an information-theoretic training criterion that measures the information loss caused by approximated distributions in comparison to the true target distributions [82]:

$$L_{\mathrm{KLD}}(\boldsymbol{y},\hat{\boldsymbol{y}}) = \sum_{j=1}^{J}\hat{y}_j\ln\left(\frac{\hat{y}_j}{y_j}\right) = \sum_{j=1}^{J}\hat{y}_j\big(\ln\hat{y}_j-\ln y_j\big) \tag{2.96}$$

It has a non-negative value property and is often referred to as a distance measurement between two distributions, even though asymmetric properties exist since generally $\widetilde{L}_{\mathrm{KLD}}(\boldsymbol{y},\hat{\boldsymbol{y}}) \neq \widetilde{L}_{\mathrm{KLD}}(\hat{\boldsymbol{y}},\boldsymbol{y})$ [82].

Since the KLD loss functions require probability distributions, the network outputs $\boldsymbol{y}$ are commonly transformed into probability distributions by *softmax* function $\psi(\,\cdot\,)$ [82], and the target labels $\hat{\boldsymbol{y}}$ are one-hot encoded (compare to Equation (2.3)):

$$\widetilde{y}_j = \psi(\boldsymbol{y})_j = \frac{\exp\big(y_j\big)}{\sum_{k=1}^{J}\exp\big(y_k\big)} \quad \forall j\in\{1,2,\cdots,J\}, \tag{2.97}$$

with $\widetilde{\boldsymbol{y}}$ being the transformed output $\boldsymbol{y}$. As a result, the standard KLD loss function is modified [183]:

$$\widetilde{L}_{\mathrm{KLD}}(\boldsymbol{y},\hat{\boldsymbol{y}}) = \sum_{j=1}^{J}\hat{y}_j\left(\ln\hat{y}_j-\ln\frac{\exp\big(y_j\big)}{\sum_{k=1}^{J}\exp\big(y_k\big)}\right) \tag{2.98}$$

$$= \sum_{j=1}^{J} \hat{y}_j \ln \hat{y}_j + \ln \sum_{j=1}^{J} \exp\big(y_j\big) - \sum_{k=1}^{J} \hat{y}_k y_k, \tag{2.99}$$

where $\widetilde{L}_{\mathrm{KLD}}$ refers to the modified KLD loss function, and $\mathrm{H}(\hat{\boldsymbol{y}}) = \sum_{j=1}^{J} \hat{y}_j \ln \hat{y}_j$ defines the entropy measurement of the target labels $\hat{\boldsymbol{y}}$. Although it would be feasible to apply standard outputs $\boldsymbol{y}$ instead of the transformed outputs $\widetilde{\boldsymbol{y}}$ in the KLD loss function, the gradients of $\widetilde{L}_{\mathrm{CE}}$ w.r.t. $\boldsymbol{y}$ are simplified significantly [183]:

$$\frac{\partial L_{\mathrm{KLD}}\big(\boldsymbol{y}, \hat{\boldsymbol{y}}\big)}{\partial \boldsymbol{y}} = \psi\big(\boldsymbol{y}\big) - \hat{\boldsymbol{y}}. \tag{2.100}$$

Notice that the gradients w.r.t. to $\boldsymbol{y}$ of the entropy $\frac{\partial}{\partial \boldsymbol{y}} \mathrm{H}\big(\hat{\boldsymbol{y}}\big) = \frac{\partial}{\partial \boldsymbol{y}} \sum_{j=1}^{J} \hat{y}_j \ln \hat{y}_j = 0$ since the target labels $\hat{\boldsymbol{y}}$ are independent of the model parameters. In practice, the *softmax* is prone to overflow and underflow, which need to be considered [82].

The Cross Entropy Loss Function The cross entropy (CE) loss function is derived from the enhanced KLD loss function. Since the target labels $\hat{\boldsymbol{y}}$ are not altered during training, the target label distribution can be assumed constant. Therefore, the entropy of the dataset is $\mathrm{H}\big(\hat{\boldsymbol{y}}\big) = 0$, and the KLD loss function $\widetilde{L}_{\mathrm{KLD}}\big(\boldsymbol{y}, \hat{\boldsymbol{y}}\big)$ is simplified into:

$$\widetilde{L}_{\mathrm{CE}}\big(\boldsymbol{y}, \hat{\boldsymbol{y}}\big) = -\sum_{j=1}^{J} \hat{y}_j \ln \frac{\exp\big(y_j\big)}{\sum_{k=1}^{J} \exp\big(y_k\big)} = \ln \sum_{j=1}^{J} \exp\big(y_j\big) - \sum_{k=1}^{J} \hat{y}_k y_k, \tag{2.101}$$

whereby the DNN outputs $\boldsymbol{y}$ are transformed by the softmax function ψ. The gradients of the KLD and CE loss functions are equal, as the entropy $\mathrm{H}\big(\hat{\boldsymbol{y}}\big)$ of the target labels remains constant in training.

Besides the three fundamental loss functions introduced above, several other domain-related loss functions exist. In ASR, popular loss functions are the connectionist temporal classification (CTC) loss [55] or the lattice-free maximum mutual information (MMI) loss [125]. The CTC loss function is further discussed in Section 3.1.2, as it is employed in multiple state-of-the-art (SOTA) approaches of ASR.

2.2.7 Backpropagation

Based on the training criteria defined in Section 2.2.5 and the introduction of multiple loss functions L in Section 2.2.6, the training objective of DNNs is specified. However, methods for minimizing chosen loss functions L and efficiently computing the required gradients of loss functions L w.r.t. all model parameters were not discussed so far. These gradients are required to perform the gradient descent in Section 2.2.8.

In 1982, Werbos introduced the first efficient BP algorithm for DNNs [167], although he had already presented related base concepts in 1974 [166]. Nowadays, the gradients for all modern DNN architectures are determined by the BP algorithm.

The concept of the BP algorithms requires the definition of loss functions $L\big(\boldsymbol{y}, \hat{\boldsymbol{y}}\big)$ (see also Section 2.2.6), where the outputs $\boldsymbol{y}$ are retrieved by feeding the DNNs with the inputs $\boldsymbol{x}$ (compare to Section 2.2.1), and the target labels $\hat{\boldsymbol{y}}$ are obtained from a given dataset $\mathcal{D}$. Then, an initial error $\boldsymbol{\epsilon}^{[L]} \in \mathbb{R}^{J^{[L]}}$ is determined in the output layer L [82, 112]:

$$\boldsymbol{\epsilon}^{[L]} = \nabla_{\boldsymbol{y}} L\big(\boldsymbol{y}, \hat{\boldsymbol{y}}\big) \odot \zeta'^{[L]}\big(\boldsymbol{a}^{[L]}\big), \tag{2.102}$$

where $\nabla_{\boldsymbol{y}} = \big(\frac{\partial}{\partial y_1}, \ldots, \frac{\partial}{\partial y_j}\big)^\top$ specifies the partial derivative operator w.r.t. the variable in the subscript and $\big(\,\cdot\,\big)'$ defines the first derivative. Then, the error $\boldsymbol{\epsilon}^{[L]}$ is backpropagated to the preceding layers l:

$$\boldsymbol{\epsilon}^{[l]} = \big(\boldsymbol{W}^{[l+1]\top} \boldsymbol{\epsilon}^{[l+1]}\big) \odot \zeta'^{[l]}\big(\boldsymbol{a}^{[l]}\big) \quad \forall l \in \{L-1, L, \cdots, 1\}, \tag{2.103}$$

leading to error vectors $\boldsymbol{\epsilon}^{[l]} \in \mathbb{R}^{J^{[l]}}$ in each layer l. The corresponding gradients of the loss L w.r.t. all the model weights are derived by:

$$\frac{\partial L\big(\boldsymbol{y}, \hat{\boldsymbol{y}}\big)}{\partial w_{j,i}^{[l]}} = h_i^{[l-1]} \boldsymbol{\epsilon}_j^{[l]} \quad \forall l \in \{L-1, L, \cdots, 1\}, \tag{2.104}$$

and w.r.t. all the model biases by:

$$\frac{\partial L\big(\boldsymbol{y}, \hat{\boldsymbol{y}}\big)}{\partial b_j^{[l]}} = \boldsymbol{\epsilon}^{[l]} \quad \forall l \in \{L-1, L, \cdots, 1\}, \tag{2.105}$$

where $\boldsymbol{a}^{[0]}$ corresponds to the input $\boldsymbol{x} = \boldsymbol{a}^{[0]}$.

In general, the BP algorithm can be interpreted as an efficient approach to backpropagate gradients from the output to the input layer by applying the chain rule multiple times [112]:

$$\frac{\partial L\big(\boldsymbol{y}, \hat{\boldsymbol{y}}\big)}{\partial w_{j,i}^{[l]}} = \sum_{j^{[L]}} \sum_{j^{[L-1]}} \cdots \sum_{j^{[l+1]}} \frac{\partial L\big(\boldsymbol{y}, \hat{\boldsymbol{y}}\big)}{\partial h_{j^{[L]}}^{[L]}} \frac{\partial h_{j^{[L]}}^{[L]}}{\partial h_{j^{[L-1]}}^{[L-1]}} \cdots \frac{\partial h_{j^{[l+1]}}^{[l+1]}}{\partial h_j^{[l]}} \frac{\partial h_j^{[l]}}{\partial w_{j,i}^{[l]}}. \tag{2.106}$$

Backpropagation Through Time In 1988, Werbos extended the BP algorithm to the backpropagation through time (BPTT) algorithm [168] to be applicable for RNNs (compare to Section 2.2.3). Similar to the standard BP, the output sequence $\boldsymbol{Y}$ is generated by feeding the input sequence $\boldsymbol{X}$ into the RNN, obtaining the gradients of the loss $\widetilde{L}\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big)$ by:

$$\frac{\partial \widetilde{L}\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big)}{\partial w_{j,i}^{[l]}} = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial L\big(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}\big)}{\partial w_{j,i}^{[l]}} \tag{2.107}$$

$$= \frac{1}{T}\sum_{t=1}^{T}\left(\sum_{j^{[L]}}\sum_{j^{[L-1]}}\cdots\sum_{j^{[l+1]}}\frac{\partial L\big(\boldsymbol{y}^{(t)},\hat{\boldsymbol{y}}^{(t)}\big)}{\partial h_{j^{[L]}}^{[L](t)}}\frac{\partial h_{j^{[L]}}^{[L](t)}}{\partial h_{j^{[L-1]}}^{[L-1](t)}}\cdots\frac{\partial h_{j^{[l+1]}}^{[l+1](t)}}{\partial h_{j}^{[l+1](t)}}\frac{\partial h_{j}^{[l](t)}}{\partial w_{j,i}^{[l]}}\right). \tag{2.108}$$

All the gradients are determined following the standard BP algorithm, except for the last gradient $\partial h_j^{[l](t)}/\partial w_{j,i}^{[l]}$, as $h_j^{[l](t)}$ also depends on the previous hidden state $h_j^{[l](t-1)}$ caused by the recurrent connection. As a result, the last gradient consists of separate gradients. The first gradient represents the standard gradient of the hidden state $h_j^{[l](t)}$ w.r.t. the weights $w_{j,i}^{[l]}$ of current layer l at the time t. The second gradient corresponds to the extended gradient introduced by the BPTT algorithm [183] and considers the dependency on the previous time step $t-1$:

$$\frac{\partial h_j^{[l](t)}}{\partial w_{j,i}^{[l]}} = \frac{\partial h_j^{[l](t)}}{\partial w_{j,i}^{[l]}} + \sum_{m=1}^{t-1}\left(\prod_{n=m+1}^{t}\frac{\partial h_j^{[l](n)}}{\partial h_j^{[l](n-1)}}\right)\frac{\partial h_j^{[l](m)}}{\partial w_{j,i}^{[l]}}. \tag{2.109}$$

In practice, the formulation is determined by unfolding the RNNs, and performing a forward pass for each time step t (compare to Figure 2.5). Then, the BP algorithm is utilized to acquire all time-dependent layer gradients.

2.2.8 Stochastic Gradient Descent

The gradient descent procedure [82] employs the retrieved gradients for minimizing the predefined loss functions L, which results in modifying the model parameters. Therefore, the introduced loss functions L are reformulated as cost functions $C\big(\,\cdot\,,\cdot\,\big)$, utilizing multiple input samples from the given dataset $\mathcal{D}$ instead of single tuples $\big(\boldsymbol{x},\hat{\boldsymbol{y}}\big)\in\mathcal{D}$:

$$C\big(\boldsymbol{Y},\hat{\boldsymbol{Y}}\big) = \frac{1}{N}\sum_{n=1}^{N} L\big(\mathcal{H}_{\boldsymbol{\Theta}}\big(\boldsymbol{x}^{(n)}\big),\hat{\boldsymbol{y}}^{(n)}\big). \tag{2.110}$$

In practice, employing the entire dataset $\mathcal{D}$ is infeasible, caused of the limited memory resources. For this reason, randomly sampled subsets $\mathcal{B}\in\mathcal{D}$ are applied, resulting in the stochastic gradient descent (SGD) procedure:

$$C_{\mathrm{SGD}}\big(\boldsymbol{Y}_{\mathcal{B}},\hat{\boldsymbol{Y}}_{\mathcal{B}}\big) = \frac{1}{B}\sum_{n=1}^{B} L\big(\mathcal{H}_{\boldsymbol{\Theta}}\big(\boldsymbol{x}^{(n)}\big),\hat{\boldsymbol{y}}^{(n)}\big), \tag{2.111}$$

with $\big(\boldsymbol{x}^{(n)},\hat{\boldsymbol{y}}^{(n)}\big)\in\mathcal{B}$ and $\big(\,\cdot\,\big)_{\mathcal{B}}$ referring to the values in the current mini-batch $\mathcal{B}$, whose size is set by B. The mini-batches $\mathcal{B}$ are typically sampled without replacement from a given dataset $\mathcal{D}$ until the batches cannot be filled. This condition concludes the current epoch, which specifies an iteration through the entire dataset $\mathcal{D}$.

The following SGD procedure is initiated by the parameter initialization of the DNN for the first iteration step $\tau=1$. Then, each iteration step $\tau+1$ consists of first obtaining the set of model parameter gradients $\mathcal{G}^{(\tau+1)}$ for each mini-batch $\mathcal{B}$ by the BP algorithm:

$$\mathcal{G}^{(\tau+1)} = \frac{\partial C_{\text{SGD}}\left(\boldsymbol{Y}_{\mathcal{B}}, \hat{\boldsymbol{Y}}_{\mathcal{B}}\right)}{\partial \boldsymbol{\Theta}^{(\tau)}} = \left\{ \frac{\partial C_{\text{SGD}}\left(\boldsymbol{Y}_{\mathcal{B}}, \hat{\boldsymbol{Y}}_{\mathcal{B}}\right)}{\partial w_{j,i}^{[l](\tau)}}, \frac{\partial C_{\text{SGD}}\left(\boldsymbol{Y}_{\mathcal{B}}, \hat{\boldsymbol{Y}}_{\mathcal{B}}\right)}{\partial b_{j}^{[l](\tau)}} \right\} \quad \begin{matrix} \forall\ 1 \leq i \leq I^{[l]}, \\ \forall\ 1 \leq j \leq J^{[l]} \\ \forall l \in \{1, 2, \cdots, L\}, \end{matrix} \tag{2.112}$$

followed by a descent step. As the notation would be cluttered by including all gradients, the descent step is demonstrated for a single gradient vector $\boldsymbol{g}^{(\tau+1)} \in \mathbb{R}^D$ with $\boldsymbol{g} \in \mathcal{G}^{(\tau+1)}$, where $\boldsymbol{g}^{(\tau+1)}$ is the gradient of a single network parameter $\boldsymbol{\theta}^{(\tau+1)} \in \mathbb{R}^D$ with $\boldsymbol{\theta}^{(\tau+1)} \in \boldsymbol{\Theta}$ and is obtained by the BP algorithm. Since these gradients of SGD are prone to noise caused by the low sample size B, momentum terms $\boldsymbol{v}$ are added to the SGD formulation to smooth the gradients by preventing sudden changes [127].

The momentum vector $\boldsymbol{v}^{(\tau)} \in \mathbb{R}^D$ is updated by:

$$\boldsymbol{v}^{(\tau+1)} = \alpha \boldsymbol{v}^{(\tau)} - \eta \boldsymbol{g}^{(\tau+1)}, \tag{2.113}$$

with $\alpha \in \mathbb{P}$ being a hyperparameter to modify the impact of the momentum on the gradient vectors $\boldsymbol{g}^{(\tau)}$, and $\eta \in \mathbb{R}_+$ defines the learning rate of each gradient descent step and requires it to be positive. The optimization process slows down for lower learning rates η, and the risk of getting stuck in a high local minimum is increased. In contrast, larger learning rates η speed up the optimization process with the risk of noisy gradients overshooting local minimas.

The descent step is finalized by updating the corresponding network parameter $\boldsymbol{\theta}^{(\tau+1)}$ by:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} + \boldsymbol{v}^{(\tau+1)}. \tag{2.114}$$

The descending procedure is repeated until a stopping criterion is reached.

Over the years, the SGD has been steadily improved and extended, primarily to receive more robust gradients. Popular candidates are found in Adagrad [45], Adadelta [180], and Adam [76]. Adagrad adapts the learning rate by tracking the inputs $\boldsymbol{X}$. If frequent inputs $\boldsymbol{x}$ are observed, the learning rate η is decreased, whereas higher learning rates η are set for more infrequent inputs $\boldsymbol{x}$ [45]. The Adadelta optimizer corresponds to an extension of Adagrad [180]. Here, a gradient window is added to restrict past gradients. Additionally, gradients were replaced by decaying running averages to improve efficiency. The Adam optimizer utilizes two bias-corrected momentum vectors with exponential decay factors, efficiently compensating for noisy gradients and not requiring any learning rate adaptions [76].

2.2.9 Vanishing and Exploding Gradients

The BP algorithm efficiently determines the gradients of loss functions L w.r.t. the model parameters $\boldsymbol{\Theta}$. However, typical problems of DNN and RNN optimization with BP are *vanishing* and *exploding gradients*. Vanishing gradients refer to events in DNN training where the training procedure experiences sudden halts, and further loss changes are not observable. Exploding gradients define the opposite events, where the value of the loss L

abruptly exceeds a numerical representation. Both issues are already known since 1993 when Bengio *et al.* [14] described problems in training RNNs with very long sequences. They identified that the vanishing and exploding gradient problem is caused by the norm of the gradients, which either decreases to zero or increases to infinity, respectively.

A precise theoretical analysis with a regarding solution was introduced by Pascanu *et al.* [117] a few years later. They demonstrated that the problem of vanishing and exploding gradients is observable in RNN layers l, an eigendecomposition:

$$\boldsymbol{W}^{[l]} = \boldsymbol{V}^{[l]}\text{diag}\big(\boldsymbol{\lambda}^{[l]}\big)\boldsymbol{V}^{[l]^{-1}}, \tag{2.115}$$

is applied, where $\boldsymbol{W}^{[l]} \in \mathbb{R}^{I^{[l]} \times I^{[l]}}$ is assumed to be a square matrix, $\boldsymbol{V}^{[l]} \in \mathbb{R}^{I^{[l]} \times I^{[l]}}$ describes a square matrix, $\big(\cdot\big)^{-1}$ defines the inverse matrix operator, $\text{diag}\big(\boldsymbol{\lambda}^{[l]}\big) \in \mathbb{R}^{I^{[l]} \times I^{[l]}}$ refers to a diagonal matrix, and $\boldsymbol{\lambda}^{[l]} \in \mathbb{R}^{I^{[l]}}$ are the eigenvalues of $\boldsymbol{W}^{[l]}$. If RNNs are then unfolded t times, the matrix $\boldsymbol{W}^{[l]}$ is reused t times (compare to Section 2.2.3) to obtain the gradients by the BPTT algorithm [82]:

$$\big(\boldsymbol{W}^{[l]}\big)^t = \boldsymbol{V}^{[l]}\text{diag}\big(\boldsymbol{\lambda}^{[l]}\big)^t\boldsymbol{V}^{[l]^{-1}}. \tag{2.116}$$

As a result, the gradients are scaled by the eigenvalues $\text{diag}\big(\boldsymbol{\lambda}^{[l]}\big)^t$, and vanish for eigenvalues smaller than one and explode for eigenvalues larger than one [82].

Pascanu *et al.* [117] suggested a gradient-clipping procedure to prevent exploding gradients. An exemplary single gradient $\boldsymbol{g}^{(\tau+1)}$ in the current descent step $\tau + 1$ is rescaled if its norm exceeds a threshold β:

$$\boldsymbol{g}^{(\tau+1)} = \frac{\beta\boldsymbol{g}^{(\tau)}}{\big\|\boldsymbol{g}^{(\tau)}\big\|} \quad \text{if} \quad \big\|\boldsymbol{g}^{(\tau)}\big\| > \beta, \tag{2.117}$$

with $\big\|\cdot\big\|$ referring to the L2 vector norm. The hyperparameter β need to be found heuristically, whereby β values corresponding to half of the average norm lead to regular training [117]. The gradient-clipping approach requires minor computational overhead and yields more stable training as the gradients $\boldsymbol{g}^{(\tau)}$ are bounded.

For vanishing gradients, several strategies are applicable. Glorot *et al.* [52] recommended utilizing ReLU activation functions, as they induce sparsity with "dead", unrecoverable neurons and lead to simplification, which need not be considered in the BP algorithm. Other strategies proposed skip connections [65], allowing the gradient flow to skip specific layers and BN layers for normalizing the current mini-batch $\mathcal{B}$, which are covered in Section 2.2.11.

2.2.10 Model Generalization

The established methods and algorithms laid the foundations for constructing and training DNN architectures. However, solely focusing on obtaining the optimal parameters is insufficient, as it is essential to attain model parameters $\boldsymbol{\Theta}$ that also perform well on unseen training data [82].

Commonly, the generalization ability of $\mathcal{H}_{\boldsymbol{\Theta}}$ is measured by splitting a labeled dataset $\mathcal{D}$ into three non-overlapping sets: the training set *train* $\mathcal{D}_{\text{train}}$, the development set *dev* $\mathcal{D}_{\text{dev}}$, and the testing set *test* $\mathcal{D}_{\text{test}}$, where $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{dev}}, \mathcal{D}_{\text{test}} \subseteq \mathcal{D}$. In the training phase, the model is optimized on $\mathcal{D}_{\text{train}}$ and evaluated on $\mathcal{D}_{\text{dev}}$. The test set $\mathcal{D}_{\text{test}}$ is excluded from the optimization, and only employed in the final evaluation. Depending on the objective, a metric is defined to measure the error between the model predictions $\boldsymbol{Y}$ and the target labels $\hat{\boldsymbol{Y}}$. For each data set, the *training* μ_{train}, *dev* μ_{dev}, and *test errors* μ_{test} are obtained. The test error μ_{test} is also referred to as the *generalization error* [82] since it specifies the generalization capabilities of DNNs on unseen data. Notice that this statement is only valid if and only if all the samples of the sets are i.i.d. and drawn from the same underlying data-generating distribution [82].

Therefore, training DNNs on given datasets $\mathcal{D}_{\text{train}}$ is more challenging than simply minimizing chosen cost functions. Instead, it is essential to carefully tune the model parameters until training and generalization errors are minimized since such specifications lead to models performing well on unseen data. This concept is known as the *bias-variance trade-off* [18], where it is essential to achieve a balance between expected model predictions, including strong biases or strong variances. The model bias, which is obtained for all sets of $\mathcal{D}$, refers to the average error between the model predictions and the true underlying regression function of the datasets in $\mathcal{D}$. The model variance specifies the sensitivity around the average prediction errors [18]. If the concept of the bias-variance trade-off is ignored, DNN models are prone to either *over-* or *underfitting*. Such conditions describe a problematic relationship between the size of a dataset $\mathcal{D}_{\text{train}}$ and the degree of freedom introduced by the model parameters [18, 82].

The overfitting condition is achieved if the model bias tends to be irrelevant, leading to variance-governed network functions $\mathcal{H}_{\boldsymbol{\Theta}}$. The number of model parameters is so large that DNNs can memorize the entire training dataset $\mathcal{D}_{\text{train}}$ rather than determining a generalized network function $\mathcal{H}_{\boldsymbol{\Theta}}$. Besides the underlying data-generation distribution, several other generating distributions are implicitly learned and overlap with the true distribution [82]. In DNN training, overfitting is typically identified if the training error μ_{train} continuously decreases while the dev error μ_{dev} stagnates or rises.

The underfitting condition describes bias-dominated network functions $\mathcal{H}_{\boldsymbol{\Theta}}$. The number of utilized model parameters is too low to derive generalized model functions $\mathcal{H}_{\boldsymbol{\Theta}}$ for representing the underlying data-generating distribution. Commonly, such models tend to be less complex in their model architecture and are infeasible to learn complex and challenging patterns [82]. A similar effect is observable in the MLP from Equation (2.63), which does not apply any activation functions. As a result, it cannot determine non-linear class separation borders in the feature space. In DNN optimization, underfitting occurs if the training error μ_{train} does not decrease even after several training iteration steps.

2.2.11 Model Regularization

Nowadays, model regularization is essential in current DNN applications since these models are usually representing variance-dominated model functions $\mathcal{H}_{\boldsymbol{\Theta}}$. Consequently, they are prone to overfit the given training dataset $\mathcal{D}_{\text{train}}$, as their exceeding number of

parameters leads to a high degree of freedom in the model functions $\mathcal{H}_{\boldsymbol{\Theta}}$. In general, model regularization approaches are directly applied in the cost functions C, modifying the layers of DNNs, augmenting the training dataset $\mathcal{D}_{\text{train}}$ on-the-fly, or limiting the optimization time of DNN models. In the following, training strategies are discussed that optimize the training process of DNN architectures to obtain variance-balanced network parameters of $\mathcal{H}_{\boldsymbol{\Theta}}$.

Weight Decay Regularization methods, which directly alter the cost functions C of DNNs, are introduced by additional regularization terms $\Pi\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big)$ to the functions:

$$\widetilde{C}\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big) = C\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big) + \gamma \Pi\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big), \tag{2.118}$$

with $\widetilde{C}\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big)$ defining the extended cost function $C\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big)$ and γ referring to the weight factor of the regularization Π.

The *weight decay* belongs to the most popular regularization functions in DNN training. It was initially proposed by Hoerl *et al.* [82] for linear regression models and was then transferred into the domain of DNNs. Weight decay extends the cost functions by another objective, represented by the regularization term $\Pi\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big) = \sum_{l=1}^{L-1} \big\|\boldsymbol{W}^{[l]}\big\|$, leading to the modified cost function $\widetilde{C}$:

$$\widetilde{C}\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big) = C\big(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\big) + \gamma \sum_{l=1}^{L-1} \big\|\boldsymbol{W}^{[l]}\big\|, \tag{2.119}$$

where $\big\|\boldsymbol{W}^{[l]}\big\|$ corresponds to the Frobenius norm of a weight matrix $\boldsymbol{W}^{[l]}$ of the layer l. The declining norm of the weight matrices $\boldsymbol{W}^{[l]}$ constrains the degree of freedom in the weight parameters and prevents overfitting.

Interestingly, similar regularization effects are achievable by adding noise to the input sequence $\boldsymbol{X}$, demonstrated by Bishop [17] in 1995. His method corresponded to a more efficient approach, as no additional resources are required for determining the gradients of the regularization terms Π.

Batch Normalization The batch normalization (BN) referred to layer-dependent regularization methods and was proposed by Ioffe & Szegedy to reduce the *internal covariate shift* in DNNs [71], which yielded regularization effects and faster optimization speed of DNNs. According to their vague definition, the covariate shift defines a concept of continuously altering input layer distributions, which leads to slower model training induced by unstable parameter estimations.

In general, the BN layers are applied after the hidden representations $\boldsymbol{h}^{[l](n)}$ of the current layer l to conquer the internal covariate shift of the next layer $l+1$. In the training phase, BN layers derive the first- and second-moment statistics of the hidden representations $\boldsymbol{H}_{\mathcal{B}}^{[l](n)} \in \mathbb{R}^{J^{[l]} \times B}$ in the current mini-batch $\mathcal{B}$:

$$\boldsymbol{\mu}_{\mathcal{B}}^{[l]} = \frac{1}{B} \sum_{n=1}^{B} \boldsymbol{h}_{\mathcal{B}}^{[l](n)}, \tag{2.120}$$

with $\boldsymbol{\mu}_{\mathcal{B}}^{[l]} \in \mathbb{R}^{J^{[l]}}$ specifying the mean and $\boldsymbol{h}_{\mathcal{B}}^{[l](n)} \in \mathbb{R}^{J^{[l]}}$ referring to a sample of the hidden representation batch $\boldsymbol{H}_{\mathcal{B}}^{[l](n)}$. The variance ${\boldsymbol{\sigma}_{\mathcal{B}}^{[l]}}^2$ of the current output is determined by:

$$\boldsymbol{\sigma}_{\mathcal{B}}^{[l]^2} = \frac{1}{B}\sum_{n=1}^{B}\left(\boldsymbol{h}_{\mathcal{B}}^{[l](n)} - \boldsymbol{\mu}_{\mathcal{B}}^{[l]}\right)^2 = \frac{1}{B}\sum_{n=1}^{B}\left(\boldsymbol{h}_{\mathcal{B}}^{[l](n)} - \boldsymbol{\mu}_{\mathcal{B}}^{[l]}\right) \odot \left(\boldsymbol{h}_{\mathcal{B}}^{[l](n)} - \boldsymbol{\mu}_{\mathcal{B}}^{[l]}\right), \tag{2.121}$$

where $\boldsymbol{\sigma}_{\mathcal{B}}^{[l]} \in \mathbb{R}^{J^{[l]}}$. Then, the BN layers normalize the representations $\boldsymbol{h}_{\mathcal{B}}^{[l](n)}$ of the current mini-batch $\mathcal{B}$ in the following:

$$\widetilde{\boldsymbol{h}}_{\mathcal{B}}^{[l](n)} = \kappa^{[l]} \frac{\boldsymbol{h}_{\mathcal{B}}^{[l](n)} - \boldsymbol{\mu}_{\mathcal{B}}^{[l]}}{\sqrt{{\boldsymbol{\sigma}_{\mathcal{B}}^{[l]}}^2 + \boldsymbol{\varepsilon}^{[l]}}} + \boldsymbol{\nu}^{[l]}, \tag{2.122}$$

where $\kappa^{[l]}$ and $\nu^{[l]}$ are trainable parameters with $\boldsymbol{\nu}^{[l]} = \nu^{[l]}\mathbb{1}^{[l]} \in \mathbb{R}^{J^{[l]}}$, the heuristic parameter ε to prevent numerical instabilities with $\boldsymbol{\varepsilon}^{[l]} = \varepsilon^{[l]}\mathbb{1}^{[l]} \in \mathbb{R}^{J^{[l]}}$, and the statement $\mathbb{1}^{[l]} \in \mathbb{R}^{J^{[l]}}$ defines vectors of ones. During the inference, each BN layer derives the first and second-moment statistics from the entire training dataset $\mathcal{D}_{\text{train}}$ and applies whereby an unbiased variance ${\boldsymbol{\sigma}_{\mathcal{B}}^{[l]}}^2 = \frac{B}{B-1}{\boldsymbol{\sigma}_{\mathcal{B}}^{[l]}}^2$. In practice, the required population means and variances are already retrieved by moving average and moving variance in the training process.

Despite the enormous success of BN layers in various research domains, it remains unclear how BN improves the generalization of DNNs, and why larger learning rates η can be set without observing divergent training outcomes. The original authors of BN [71] emphasized that the covariate shift is mainly responsible for the beneficial effect of BN, even though they owed a clear explanation of the concept of covariate shift.

Recently, several studies have questioned the idea of covariate shift and established alternative explanations of how BN layers modify the training process [19, 96, 140]. Santurkar *et al.* [140] discovered that BN does not necessarily reduce such covariant shifts. Instead, they argued that BN layers smooth the optimization landscape and ensure more predictive and stable gradients. Therefore, larger learning rates η can be selected, resulting in reduced optimization durations. Another work from Luo *et al.* [19] analyzed the regularizing effect of BN on the gradients in DNN training. They identified that the concept of BN implicitly regularizes large parameter updates by reducing the magnitude of the layer activation functions. They also established a theoretical framework for explaining BN layers, where they explicitly demonstrated the regularization effect of BN layers [96].

Layer Normalization The concept of layer normalization (LN) was proposed by Ba *et al.* [6] to eliminate several drawbacks of BN layers. For instance, BN layers solely depend on the samples of the current mini-batch $\mathcal{B}$, which leads to highly correlated value shifts in the next layer if the hidden representations of the previous layer differ tremendously. Moreover, BN layers cannot be applied in real-time applications since the

statistics of a mini-batch size $B = 1$ are not representative. And finally, it is unclear how BN is employed in RNNs.

Therefore, Ba *et al.* [6] proposed the concept of LN, where the output units $\boldsymbol{a}^{[l]}$ of the current layer l are normalized by the mean $\mu_{\mathcal{L}}^{[l]}$:

$$\mu_{\mathcal{L}}^{[l]} = \frac{1}{J^{[l]}} \sum_{j=1}^{J^{[l]}} a_j^{[l]}, \tag{2.123}$$

and the variance ${\sigma_{\mathcal{L}}^{[l]}}^2$:

$${\sigma_{\mathcal{L}}^{[l]}}^2 = \frac{1}{J^{[l]}} \sum_{j=1}^{J^{[l]}} \left(a_j^{[l]} - \mu_{\mathcal{L}}^{[l]}\right)^2, \tag{2.124}$$

where $\left(\,\cdot\,\right)_{\mathcal{L}}$ refers to the parameters related to the LN method. The current hidden representations $\boldsymbol{h}^{[l](n)}$ are then normalized by the LN formulation:

$$\widetilde{h}_j^{[l](n)} = \zeta \left(\kappa^{[l]} \frac{a^{[l](n)} - \mu_{\mathcal{L}}^{[l]}}{\sqrt{{\sigma_{\mathcal{L}}^{[l]}}^2 + \boldsymbol{\varepsilon}^{[l]}}} + \nu_j^{[l]} \right) \quad \forall j \in \{1, 2, \cdots, J^{[l]}\}, \tag{2.125}$$

Dropout Another popular regularization technique is *dropout*, introduced by Srivastava *et al.* [66, 150] in 2014. In the training phase, dropout layers set the hidden representations $\boldsymbol{h}^{[l](n)}$ with a probability of ρ to zero and leave the remaining representations unmodified. This procedure results in noisy network layers since the hidden representations $\boldsymbol{h}^{[l](n)}$ are dropped out by the random variable $\boldsymbol{r}^{[l]} \in \mathbb{B}^{J^{[l]}}$ sampled from a Bernoulli probability distribution $r_j^{[l]} \sim \text{Bern}\big(\rho\big)$. As a result, DNNs cannot rely on specific connections represented by the weight matrices $\boldsymbol{W}^{[l]}$ and are forced to derive more generalizing model parameters. Consequently, induced noise heavily influences the gradients, yielding an increased training time.

In the inference phase, dropout is not applied since induced noise is not intended. Additionally, all weight matrices $\boldsymbol{W}^{[l]}$ need to be scaled by $\widetilde{\boldsymbol{W}}^{[l]} = \rho \boldsymbol{W}^{[l]}$ to prevent a wrong scaling during the inference phase.

In practice, dropout is implemented as the *inverted dropout* procedure [66, 119, 150], where dropout layers are applied on the current layer output $\boldsymbol{h}^{[l]}$, and the dropout probability ρ is replaced with the keep probability $\widetilde{\rho} = 1 - \rho$. Combining both aspects results in the following statement:

$$\widetilde{\boldsymbol{h}}^{[l]} = \frac{\boldsymbol{r}^{[l]} \odot \boldsymbol{h}^{[l]}}{\widetilde{\rho}^{[l]}}, \tag{2.126}$$

where optimal values for $\widetilde{\rho}^{[l]}$ are in the range of $[0.5, 0.8]$. An advantage of these modifications is that the rescaling operation of the model weights is transferred from the inference to the training phase, as the expectations $\mathbb{E}\big(\,\cdot\,\big)$ of the standard and altered hidden representations $\boldsymbol{h}^{[l]}$ remain unchanged (*i.e.*, $\mathbb{E}\big(\widetilde{\boldsymbol{h}}^{[l]}\big) = \mathbb{E}\big(\boldsymbol{h}^{[l]}\big)$) [183]. Therefore,

the optimized model weights can be directly utilized in the evaluation phase, and no further adjustment is required.

Even though dropout layers generate solely marginal computational overhead, they require heuristic parameters $\widetilde{\rho}^{[l]}$, which depends on the applied model architecture. In recent years, Kingma *et al.* introduced *variational dropout* layers [20], where individual layer-dependent dropout parameters $\widetilde{\rho}^{[l]}$ are learned in the optimization phase.

Label Smoothing In the established loss and cost functions from Section 2.2.6, given target vectors $\hat{\boldsymbol{y}}$ are always assumed. In classification tasks, the categories are typically scalar class values, *i.e.*, $\hat{y} \in \mathbb{N}$, and are transformed into representing category vectors. A standard approach to transforming these class labels $\hat{y}$ into representing target vectors $\hat{\boldsymbol{y}} \in \mathbb{B}^{J^{[L]}}$ defines the one-hot encoding procedure in Equation (2.3), which makes them applicable in the presented loss and cost functions.

Although the approach of one-hot encoded target labels $\hat{\boldsymbol{y}}$ allows an efficient encoding procedure, DNN models trained on these labels are prone to overfitting. Since models continuously learn to assign the entire probability to a single neuron y_j in the output layer in training, the generalization capability of DNNs is restricted [155]. As a result, substantial value differences between the single firing neuron and the remaining neurons exist and lead to overconfident model predictions [155].

A solution for this overfitting scenario was given by Szegedy *et al.* [155], where they combined the original one-hot encoded target label distribution $\hat{\boldsymbol{y}}$ with a fixed uniform distribution $\mathcal{U}(\,\cdot\,)$ represented by the vector $\boldsymbol{u} \in \mathbb{R}^{J^{[L]}}$:

$$\hat{\tilde{\boldsymbol{y}}} = \big(1 - \iota\big)\hat{\boldsymbol{y}} + \iota \boldsymbol{u}, \tag{2.127}$$

which led to smoothed target label distribution $\hat{\tilde{\boldsymbol{y}}}$. The impact of the selected uniform distribution $\boldsymbol{u} = \mathcal{U}\big(1, J^{[L]}\big)$, in which each entry was assigned to $u_j = \frac{1}{J^{[L]}}$, was modified by the smoothing parameters ι, where Szegedy *et al.* [155] achieved reasonable results with $\iota = 0.1$. The resulting smooth target label distribution $\hat{\tilde{\boldsymbol{y}}}$ prevents overfitting by establishing a positive lower value bound ($\iota = 0.1$), which avoids huge value differences between each output neuron.

Early Stopping In contrast to regularization methods, which directly modify either the cost function C of DNN or the model architecture itself, *early stopping* provides a simple yet efficient regularization approach without any training overhead. Thereby, early stopping is integrated into the training process by defining a specific moment in the overall training where the optimization has to be stopped to prevent overfitting [18]. Since the training error μ_{train} does not provide any information on a pending overfitting process, the validation error μ_{dev} on $\mathcal{D}_{\text{dev}}$ is a reliable measurement to determine the training stop. After each epoch, which defines a complete iteration through $\mathcal{D}_{\text{train}}$ and feeds to the DNN, the validation error μ_{dev} is obtained. If μ_{dev} stops declining for several predefined epochs (in practice, one to three epochs), the overall optimization of the DNN is stopped.

Dataset Augmentation Typically, DNNs benefit from utilizing more training data as they are then less prone to overfitting. However, in practice, it is challenging to obtain large labeled datasets since the sole collection of training data is not sufficient. The collected data need to also be labeled, which is difficult to automate and requires human surveillance.

An alternative approach is offered by *dataset augmentation*, where the training dataset $\mathcal{D}_{\text{train}}$ is extended on-the-fly by defining plausible transformations of the input sample $\boldsymbol{x}$ [82], whereby the target value $\hat{\boldsymbol{y}}$ remains unaltered. One of the most straightforward transformations is adding noise to the input $\boldsymbol{X}$ [146]. Therefore, the DNN is challenged to denoise the input $\boldsymbol{X}$ to predict the correct target label $\hat{\boldsymbol{y}}^{(n)}$.

3

Speech Background

This chapter establishes the theoretical foundation of automatic speech recognition (ASR). The first section introduces the general mathematical formulation for defining ASR, followed by describing the decoding process to obtain the best hypothesis for a given feature sequence and specifying the ASR metric for the evaluation phase. The second section covers the extraction of the Mel-frequency cepstral coefficients (MFCCs) and log Mel features, which serve as the input features for all ASR approaches in this thesis. The third section elaborates on popular augmentation techniques for generating additional training data. The subsequent section discusses various types of target labels in ASR. In the fifth section, the two popular datasets TED-LIUM release 2 (TED-LIUM-v2) and LibriSpeech are presented, on which the established approaches are evaluated. The chapter concludes with the current state-of-the-art (SOTA) results on the TED-LIUM-v2 and LibriSpeech.

3.1 Formulation of Automatic Speech Recognition

In this section, the central problem of ASR is defined, followed by an introduction of corresponding mathematical formulations. The formulations allow a redefinition of the problem in the context of *hybrid* and *end-to-end (E2E)* ASR. Both theoretical concepts build the foundation of the contributions introduced in Chapters 4 to 6.

3.1.1 The Formulation for Hybrid Speech Recognition

The central problem of hybrid ASR is defined by mapping an input sequence $\boldsymbol{X} \in \mathbb{R}^{I \times T}$ to a word sequence $\boldsymbol{W} \in \mathcal{W}$, whereby $\mathcal{W}$ is a set gathering all possible word sequences. The word sequence $\boldsymbol{W} = \left[\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(o)}, \ldots, \boldsymbol{w}^{(O)}\right]$ of length O with $\boldsymbol{w}^{(o)} \in \mathbb{R}^{J}$ consists of multiple words $\boldsymbol{w}^{(o)} \in \mathcal{V}$, where $\mathcal{V}$ corresponds to a lexicon set of known words $\boldsymbol{w}^{(o)}$. Multiple representing word sequences $\boldsymbol{W}$ with higher and lower probability exist for a given input sequence $\boldsymbol{X}$. In a mathematical context, the decision of selecting the best fitting word sequence $\boldsymbol{W}$ from $\mathcal{W}$ for a given input sequence $\boldsymbol{X}$ can be specified with Bayes decision theory [22, 163]:

$$\boldsymbol{W}^* = \arg\max_{\boldsymbol{W} \in \mathcal{W}} p\big(\boldsymbol{W}|\boldsymbol{X}\big), \tag{3.1}$$

where $\boldsymbol{W}^*$ refers to the most probable word sequence $\boldsymbol{W}$ for a given input sequence $\boldsymbol{X}$, and $p\big(\boldsymbol{W}|\boldsymbol{X}\big)$ corresponds to a posterior distribution. Consequently, the key problem of hybrid ASR lies in efficiently modeling the posterior distribution $p\big(\boldsymbol{W}|\boldsymbol{X}\big)$.

In hybrid speech recognition, $p\big(\boldsymbol{W}|\boldsymbol{X}\big)$ is factorized into three distinct distributions by applying Bayes' theorem [22]:

$$\boldsymbol{W}^* = \arg\max_{\boldsymbol{W} \in \mathcal{W}} \frac{p\big(\boldsymbol{X}|\boldsymbol{W}\big)p\big(\boldsymbol{W}\big)}{p\big(\boldsymbol{X}\big)} = \arg\max_{\boldsymbol{W} \in \mathcal{W}} p\big(\boldsymbol{X}|\boldsymbol{W}\big)p\big(\boldsymbol{W}\big) = \arg\max_{\boldsymbol{W} \in \mathcal{W}} p\big(\boldsymbol{X}, \boldsymbol{W}\big) \tag{3.2}$$

$$= \arg\max_{\boldsymbol{W} \in \mathcal{W}} \sum_{\boldsymbol{S}} p\big(\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{S}\big) = \arg\max_{\boldsymbol{W} \in \mathcal{W}} \sum_{\boldsymbol{S}} p\big(\boldsymbol{X}|\boldsymbol{W}, \boldsymbol{S}\big)p\big(\boldsymbol{S}, \boldsymbol{W}\big) \tag{3.3}$$

$$\approx \arg\max_{\boldsymbol{W} \in \mathcal{W}} \sum_{\boldsymbol{S}} \underbrace{p\big(\boldsymbol{X}|\boldsymbol{S}\big)}_{\text{AM}} \underbrace{p\big(\boldsymbol{S}|\boldsymbol{W}\big)}_{\text{PM}} \underbrace{p\big(\boldsymbol{W}\big)}_{\text{LM}}, \tag{3.4}$$

where $p\big(\boldsymbol{X}|\boldsymbol{S}\big)$ corresponds to the acoustic model (AM), $p\big(\boldsymbol{S}|\boldsymbol{W}\big)$ to the pronunciation model (PM), and $p\big(\boldsymbol{W}\big)$ to the language model (LM). Each model is independently optimized and does not exchange information with other model components. Notice that in Equation (3.2), $p\big(\boldsymbol{X}\big)$ is excluded as it is independent of $\boldsymbol{W}$. In Equation (3.3), the joint probability distribution $p\big(\boldsymbol{X}, \boldsymbol{W}\big)$ is extended by the state sequence $\boldsymbol{S}$ of an hidden Markov model (HMM), which does not alter the original joint distribution, as $\boldsymbol{S}$ is marginalized out of $p\big(\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{S}\big)$ by summing over all states $\boldsymbol{s}^{(t)} \in \boldsymbol{S}$. Moreover, Equation (3.4) is approximated by $p\big(\boldsymbol{X}|\boldsymbol{S}\big) \approx p\big(\boldsymbol{X}|\boldsymbol{W}, \boldsymbol{S}\big)$ since $\boldsymbol{S}$ corresponds to a direct representation of $\boldsymbol{W}$ [163].

Acoustic Model In hybrid ASR, $p\big(\boldsymbol{X}|\boldsymbol{S}\big)$ specifies the likelihood of an AM, commonly modeled by Gaussian mixture models (GMMs), multilayer perceptrons (MLPs), or recurrent neural networks (RNNs). Generally, $p\big(\boldsymbol{X}|\boldsymbol{S}\big)$ is simplified by following Section 2.1.1 [163]:

$$p\big(\boldsymbol{X}|\boldsymbol{S}\big) = \prod_{t=1}^{T} p\big(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)}, \dots, \boldsymbol{x}^{(1)}, \boldsymbol{S}\big) \tag{3.5}$$

$$\approx \prod_{t=1}^{T} p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}\big) \propto \prod_{t=1}^{T} \frac{p\big(\boldsymbol{s}^{(t)}|\boldsymbol{x}^{(t)}\big)}{p\big(\boldsymbol{S}\big)}, \tag{3.6}$$

where the chain rule of probability theory is applied in Equation (3.5). As the likelihood for the current input $\boldsymbol{x}^{(t)}$ is only dependent on the current state $\boldsymbol{s}^{(t)}$, it can be further reduced to a framewise likelihood function $p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}\big)$ [163]. By employing Bayes' theorem once more, the framewise likelihood function $p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}\big)$ can be approximated by the pseudo-likelihood [22] consisting of the framewise posterior distribution $p\big(\boldsymbol{s}^{(t)}|\boldsymbol{x}^{(t)}\big)$ normalized

by $p(\boldsymbol{S})$. This approximation is valid as the $p(\boldsymbol{X})$ remains constant during optimizing Equation (3.1).

In the past, the framewise likelihood $p(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)})$, also known as the emission probability in the HMM framework (compare Equation (2.20)), was modeled by GMMs. The likelihood $p(\boldsymbol{X}|\boldsymbol{S})$ was maximized using the Baum-Welch algorithm (see Section 2.1.5). Nowadays, the posterior distribution $p(\boldsymbol{S}|\boldsymbol{X})$ is learned by deep neural networks (DNNs) as these networks possess large modeling capacities, superior to GMMs. In the decoding phase, the likelihood $p(\boldsymbol{X}|\boldsymbol{S})$ is approximated by the pseudo-likelihood $\frac{p(\boldsymbol{S}|\boldsymbol{X})}{p(\boldsymbol{S})}$.

Pronunciation Model The PMs are defined by the conditional distribution $p(\boldsymbol{S}|\boldsymbol{W})$ and represent the state transitions of HMMs for a given word sequence $\boldsymbol{W}$ [163]. The distribution is reformulated by employing the chain rule, and the conditional dependence of the current state $\boldsymbol{s}^{(t)}$ on all preceding states $\boldsymbol{s}^{(t-1)}, \dots, \boldsymbol{s}^{(1)}$ is approximated by a first-order Markov chain (compare Section 2.1.1):

$$p(\boldsymbol{S}|\boldsymbol{W}) = \prod_{t=1}^{T} p(\boldsymbol{s}^{(t)}|\boldsymbol{s}^{(t-1)}, \dots, \boldsymbol{s}^{(1)}, \boldsymbol{W}) \tag{3.7}$$

$$\approx \prod_{t=1}^{T} p(\boldsymbol{s}^{(t)}|\boldsymbol{s}^{(t-1)}, \boldsymbol{W}). \tag{3.8}$$

In contrast to the standard state transitions of HMMs in Equation (2.6), the state transitions $p(\boldsymbol{s}^{(t)}|\boldsymbol{s}^{(t-1)}, \boldsymbol{W})$ are conditioned on the word sequence $\boldsymbol{W}$. The sequence $\boldsymbol{W}$ is typically converted to a deterministic phoneme sequence, in which each phoneme is modeled by individual 3-state HMMs[1]. Thus, the HMM state transitions in $p(\boldsymbol{s}^{(t)}|\boldsymbol{s}^{(t-1)}, \boldsymbol{W})$ represent phoneme sequences for a given word sequences $\boldsymbol{W}$ and also ensure monotonic alignments by solely depending on past states $\boldsymbol{s}^{(t-1)}$ [163].

Language Model The prior distribution $p(\boldsymbol{W})$ is modeled by LMs. Similar to PMs, the chain rule of probability theory is applied to $p(\boldsymbol{W})$ in the first step:

$$p(\boldsymbol{W}) = \prod_{o=1}^{O} p(\boldsymbol{w}^{(o)}|\boldsymbol{w}^{(o-1)}, \dots, \boldsymbol{w}^{(1)}) \tag{3.9}$$

$$\approx \prod_{o=1}^{O} p(\boldsymbol{w}^{(o)}|\boldsymbol{w}^{(o-1)}, \dots, \boldsymbol{w}^{(o-O_{\mathrm{LM}}-1)}), \tag{3.10}$$

followed by approximating the conditional dependence of the current $\boldsymbol{w}^{(o)}$ on all the preceding words $\boldsymbol{w}^{(o-1)}, \dots, \boldsymbol{w}^{(1)}$ by an O_{LM}-order Markov chain, where settings with $O_{\mathrm{LM}} = 4$ returned satisfying results in prior studies [98]. Notice that in practice, $p(\boldsymbol{W})$

[1]The definition of a phoneme is covered in Section 3.4.1.

requires a power scaling, *i.e.*, $\ln(p(\boldsymbol{W}))$[15], since AMs are commonly overconfident due to their strongly correlated framewise likelihoods $p(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)})$ [123].

Finally, the AMs, PMs, and LMs are combined into the hybrid ASR formulation.

3.1.2 The Formulation for End-To-End Speech Recognition

End-to-end (E2E) speech recognition reformulates the problem of ASR by altering the problem of finding the best fitting word sequence $\boldsymbol{W}$ into obtaining the best fitting grapheme sequence $\boldsymbol{G} \in \mathcal{G}$ for a given input sequence $\boldsymbol{X} \in \mathcal{X}$:

$$\boldsymbol{G}^* = \arg\max_{\boldsymbol{G}\in\mathcal{G}} p(\boldsymbol{G}|\boldsymbol{X}), \tag{3.11}$$

where $\mathcal{G}$ corresponds to a set collecting all possible grapheme sequences. The grapheme sequence $\boldsymbol{G} = \left[\boldsymbol{g}^{(1)}, \ldots, \boldsymbol{g}^{(o)}, \ldots, \boldsymbol{g}^{(O)}\right]$ of the length of O with $\boldsymbol{g}^{(o)} \in \mathbb{R}^J$ is composed of several graphemes $\boldsymbol{g}^{(o)} \in \mathcal{A}$, where $\mathcal{A}$ defines a set of graphemes, *e.g.*, the English alphabet.

In the subsequent paragraph, the problem of E2E speech recognition is similarly factorized as in Equation (3.1) to demonstrate the transition from hybrid to E2E models in ASR. Therefore, the connectionist temporal classification (CTC) [55] is first explained before discussing current E2E models with attention mechanisms modeling Equation (3.1) directly [163].

Since hybrid ASR approaches require expert knowledge for modeling AMs, PMs, and LMs, a single, combining model would simplify the training tremendously. In general, RNNs are capable of such tasks. However, they have two major drawbacks [55]. Firstly, they require framewise aligned training data, typically generated by optimized hybrid ASR models. Secondly, the standard objective functions for training MLPs or RNNs were solely defined for separate training samples in the past and led to independent predictions for each training sample. In 2006, Graves *et al.* introduced the CTC objective function [55] and resolved the mentioned drawbacks. As a result, the training of RNN was feasible without requiring framewise-aligned training data.

CTC Approach The CTC objective function provides a solution for modeling $p(\boldsymbol{G}|\boldsymbol{X})$ efficiently and is inspired by hybrid ASR approaches. Similarly, $p(\boldsymbol{G}|\boldsymbol{X})$ can be factorized in multiple distributions, whereby $p(\boldsymbol{U}|\boldsymbol{G})$ is modeled by a grapheme model (GM) instead of a PM, and $p(\boldsymbol{G})$ is governed by a grapheme-based LM instead of a word-based LM [163]:

$$\begin{aligned}
\boldsymbol{G}^* &= \arg\max_{\boldsymbol{G}\in\mathcal{G}} p(\boldsymbol{G}|\boldsymbol{X}) && (3.12)\\
&= \arg\max_{\boldsymbol{G}\in\mathcal{G}} \sum_{\boldsymbol{U}} p(\boldsymbol{G},\boldsymbol{U}|\boldsymbol{X}) = \arg\max_{\boldsymbol{G}\in\mathcal{G}} \sum_{\boldsymbol{U}} p(\boldsymbol{G}|\boldsymbol{U},\boldsymbol{X}) p(\boldsymbol{U}|\boldsymbol{X}) && (3.13)\\
&\approx \arg\max_{\boldsymbol{G}\in\mathcal{G}} \sum_{\boldsymbol{U}} p(\boldsymbol{G}|\boldsymbol{U}) p(\boldsymbol{U}|\boldsymbol{X}) = \arg\max_{\boldsymbol{G}\in\mathcal{G}} \sum_{\boldsymbol{U}} \frac{p(\boldsymbol{U}|\boldsymbol{G})p(\boldsymbol{G})}{p(\boldsymbol{U})} p(\boldsymbol{U}|\boldsymbol{X}) && (3.14)
\end{aligned}$$

$$= \arg\max_{\boldsymbol{G}\in\mathcal{G}} \sum_{\boldsymbol{U}} \underbrace{p\big(\boldsymbol{U}|\boldsymbol{X}\big)}_{\text{AM}} \underbrace{p\big(\boldsymbol{U}|\boldsymbol{G}\big)}_{\text{GM}} \underbrace{p\big(\boldsymbol{G}\big)}_{\text{LM}} \frac{1}{p\big(\boldsymbol{U}\big)}. \tag{3.15}$$

Instead of adding the state sequence $\boldsymbol{S}$ in the hybrid setup (compare Equation (3.3)), the sequence $\boldsymbol{U} = \big[\boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(\widetilde{o})}, , \ldots, \boldsymbol{u}^{(\widetilde{O})}\big] = \big[\varnothing, \boldsymbol{g}^{(1)}, \varnothing, \ldots, \varnothing, \boldsymbol{g}^{(o)}, \varnothing, \ldots, \varnothing, \boldsymbol{g}^{(O)}, \varnothing\big]$ is introduced in Equation (3.13), where $\boldsymbol{u}^{(\widetilde{o})} \in \widetilde{\mathcal{A}}$. The auxiliary sequence $\boldsymbol{U}$ corresponds to the augmented sequence $\boldsymbol{G}$, which incorporates a blank symbol $\varnothing$ into the grapheme set $\widetilde{\mathcal{A}} \equiv \mathcal{A} \cup \{\varnothing\}$. The original $\boldsymbol{G}$ is augmented by placing blank symbols $\varnothing$ between each grapheme $\boldsymbol{g}^{(o)}$ and at the beginning and the ending of $\boldsymbol{G}$. The blank symbols $\varnothing$ allow collapsing of repetitive graphemes and the detection of grapheme boundaries [163]. Moreover, in Equation (3.14), the dependency of the GM on the input $\boldsymbol{X}$ is eliminated by assuming conditional independence, *i.e.*, $p\big(\boldsymbol{G}|\boldsymbol{U}\big) \approx p\big(\boldsymbol{G}|\boldsymbol{U}, \boldsymbol{X}\big)$, which is a valid simplification to separate the distributions of AM, GM, and LM [163]. In the original CTC approach, $\frac{1}{p(\boldsymbol{U})}$ is not considered [55] and will be ignored in further discussions.

The final CTC formulation Equation (3.15) resembles a similar hybrid ASR formulation, where $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$ is factorized into three separated distributions:

$$\boldsymbol{G}^* \approx \arg\max_{\boldsymbol{G}\in\mathcal{G}} \sum_{\boldsymbol{U}} \prod_{\widetilde{o}=1}^{\widetilde{O}} p\big(\boldsymbol{u}^{(\widetilde{o})}|\boldsymbol{X}\big) p\big(\boldsymbol{u}^{(\widetilde{o})}|\boldsymbol{u}^{(\widetilde{o}-1)}, \boldsymbol{G}\big) \frac{p\big(\boldsymbol{G}\big)}{p\big(\boldsymbol{U}\big)}. \tag{3.16}$$

The resulting formulation seems infeasible to calculate since it requires summing over all possible auxiliary grapheme sequence $\boldsymbol{U}$. However, Graves *et al.* [55] demonstrated that the summation in $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$ is feasible by utilizing the Markov assumption, which enabled them to apply dynamic programming algorithms as the forward-backward algorithm (compare to Section 2.1.3). Since a detailed definition and explanation of the corresponding CTC cost function would be out of scope for this thesis, the interested reader is referred to the original work from Graves *et al.* [55].

Attentional Approach In order to approximate $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$, the conditional independence assumption is heavily exploited in the CTC approach. As a result, strong biases occur and lead to questionable approximations of $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$. In recent years, the interest in purely data-driven E2E approaches, capable of directly estimating $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$ without assuming any conditional independence, has steadily increased:

$$\boldsymbol{G}^* = \arg\max_{\boldsymbol{G}\in\mathcal{G}} p\big(\boldsymbol{G}|\boldsymbol{X}\big) = \arg\max_{\boldsymbol{G}\in\mathcal{G}} \prod_{t=o}^{O} p\big(\boldsymbol{g}^{(o)}|\boldsymbol{g}^{(o-1)}, \ldots, \boldsymbol{g}^{(1)}, \boldsymbol{X}\big). \tag{3.17}$$

Current SOTA approaches utilize either attention [34] or a self-attention [159] mechanisms to model the probabilistic chain in $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$. The probabilistic chain in Equation (3.17) considers the entire input sequence $\boldsymbol{X}$ in every conditional distribution $p\big(\boldsymbol{g}^{(o)}|\boldsymbol{g}^{(o-1)}, \ldots, \boldsymbol{g}^{(1)}, \boldsymbol{X}\big)$. However, modeling the posterior distribution $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$ without any constraints has significant drawbacks: non-monotonic alignments. Therefore, grapheme predictions $\boldsymbol{g}^{(o)}$ at the beginning of $\boldsymbol{G}$ may depend on inputs $\boldsymbol{x}^{(t)}$ from the end of $\boldsymbol{X}$. In hybrid ASR and the CTC approach, monotonic alignments are enforced by

restricting the framewise likelihood $p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}\big)$ of the PM and the framewise posterior distribution $p\big(\boldsymbol{u}^{(t)}|\boldsymbol{X}\big)$ of the GM solely depending on $\boldsymbol{s}^{(t-1)}$ or $\boldsymbol{u}^{(t-1)}$, respectively.

3.1.3 Decoding

To evaluate ASR approaches on specific datasets, the optimized ASR models generate hypothesis transcripts for given input sequences $\boldsymbol{X}$. The obtained hypothesis transcript is then tested against an ASR metric, introduced in Section 3.1.4. Typically, hypothesis transcripts are obtained by the *decoding* procedure, which aims to either solve Equation (3.1) to obtain the most probable word sequence $\boldsymbol{W}^*$ or Equation (3.11) to retrieve the most probable grapheme sequence $\boldsymbol{G}^*$ for a given input sequence $\boldsymbol{X}$.

For hybrid ASR models, the most probable word sequence $\boldsymbol{W}^*$ is retrieved by the Viterbi algorithm (compare to Section 2.1.4) [175]. The Viterbi algorithm determines the most probable state sequence $\boldsymbol{S}^*$ for a given input sequence $\boldsymbol{X}$, then transforms the state sequence $\boldsymbol{S}^*$ into the optimal word sequence $\boldsymbol{W}^*$ utilizing the PM (see Section 3.1.1). In practice, the exhaustive Viterbi decoding is replaced by the beam search algorithm [48] (compare Section 2.1.7), corresponding to the greedy version of the Viterbi decoding. The beam search algorithm allows a trade-off between required computational resources and the accuracy of obtained state sequences $\boldsymbol{S}^*$, as only the K_{best} paths are stored in the Viterbi algorithm.

In E2E models, the greedy beam search algorithm is also applied for decoding, as the Viterbi algorithm is intractable for a large grapheme set $\mathcal{A}$. The standard decoding procedure is initialized by feeding the special start-of-sentence token <sos> [170] into the model. Then, relying on <sos> and the input sequence $\boldsymbol{X}$, the grapheme probability distribution $p\big(\boldsymbol{g}^{(2)}|\text{<sos>}, \boldsymbol{X}\big)$ for the next time step $t = 2$ is generated, and the K_{best} paths for $p\big(\boldsymbol{g}^{(2)}|\text{<sos>}, \boldsymbol{X}\big)$ are stored. The decoding procedure is iteratively repeated until the auxiliary end-of-sequence token <eos> is detected or a maximal predefined length for the output grapheme sequence is reached. The hypothesis transcript is generated in the final step by selecting the most probable path to obtain the most probable grapheme sequence $\boldsymbol{G}^*$ [170].

3.1.4 Evaluation Metric

In ASR, the performance of ASR models is typically evaluated in the word error rate (WER) metric. Therefore, the number of reference words $N_{\text{ref}} \in \mathbb{N}$ from a given transcript is counted, and the hypothesis transcript from an optimized model is aligned to a reference hypothesis. Next, the number of substituted words $N_{\text{sub}} \in \mathbb{N}$, deleted words $N_{\text{del}} \in \mathbb{N}$, and inserted words $N_{\text{ins}} \in \mathbb{N}$ are determined and inserted into the WER metric function [123]:

$$\text{WER}\big(N_{\text{sub}}, N_{\text{del}}, N_{\text{ins}}, N_{\text{ref}}\big) = 100\,\frac{N_{\text{sub}} + N_{\text{del}} + N_{\text{ins}}}{N_{\text{ref}}}, \tag{3.18}$$

where the WER is measured in %. The Levenshtein distance [87] is generally applied to obtain the minimal N_{sub}, N_{del}, and N_{ins} required to transform the hypothesis transcript

into the reference transcript.

3.2 Speech Features

In all domains of pattern recognition, it is essential to generate robust and uncorrelated features for recognition tasks. Over the years, MFCC features [42] became the most popular feature type in several audio domains, such as voice recognition [107], keyword spotting [184], and speaker recognition [156]. These features are hand-crafted in multiple subsequent steps, corresponding to specific operations to obtain uncorrelated and robust features. Recently, neural-based features generated by neural networks (NNs), *e.g.*, wav2vec features [7], challenge such hand-crafted features since neural-based features allow entirely E2E approaches. Since all established approaches in this thesis utilize MFCC features, the required steps for generating them are explained in the following (compare to Figure 3.1).

3.2.1 Pre-Emphasis

In the pre-emphasis phase, higher frequencies of the recorded speech signal $\widetilde{\boldsymbol{X}} \in \mathbb{I}^{1\times\widetilde{T}}$ of length $\widetilde{T}$ with $\mathbb{I} = \{x \in \mathbb{N} \,|\, 1 \leq x \leq 2^{16}\}$ are emphasized to compensate for the negative spectral slope caused by the human speech production system [122]. Generally, pre-emphasis filters transform the recorded speech signal $\widetilde{\boldsymbol{X}}$ into pre-emphasized signals $\mathring{\boldsymbol{X}} \in \mathbb{I}^{1\times\widetilde{T}}$ by applying the transfer function $H\big(z\big) \in \mathbb{C}$ [130]:

$$H\big(z\big) = 1 - 0.95z^{-1}. \tag{3.19}$$

3.2.2 Windowing

In the windowing phase, the pre-emphasized speech signal $\mathring{\boldsymbol{X}}$ is split into T segments $\overline{\boldsymbol{X}} = [\overline{\boldsymbol{x}}^{(1)}, \ldots, \overline{\boldsymbol{x}}^{(t)}, \ldots, \overline{\boldsymbol{x}}^{(T)}]$ with $\overline{\boldsymbol{X}} \in \mathbb{I}^{\mathring{I}\times T}$ and $\overline{\boldsymbol{x}}^{(t)} \in \mathbb{I}^{\mathring{I}}$. The speech signal can be assumed quasi-stationary for adequate segment lengths $\mathring{I}$, *i.e.*, varying solely gradually in time [130]. Typically, segment sizes of 20 ms (corresponds to $\mathring{I} = 400$ for a speech signal sampled with 16 kHz) are sufficient for assuming stationary signals with stable acoustic characteristics [130]. In practice, the segments are retrieved utilizing a sliding window of 20 ms, which operates over the signal with a shift of 10 ms, causing an overlap of 10 ms. As the segments $\overline{\boldsymbol{x}}^{(t)}$ contain sudden changes at their boundaries, which would result in edge effects in the subsequent discrete Fourier transformation (DFT) procedure, the boundaries in each segment $\overline{\boldsymbol{x}}^{(t)}$ are smoothed by a window vector $\overline{\boldsymbol{w}} \in \mathbb{P}^{\mathring{I}}$ [130].

For a single segment $\overline{\boldsymbol{x}} = \overline{\boldsymbol{x}}^{(t)}$, the windowing procedure is executed as:

$$\boldsymbol{q} = \overline{\boldsymbol{w}} \odot \overline{\boldsymbol{x}}. \tag{3.20}$$

In the popular MFCC features, *Hamming windows* are commonly employed [113]:

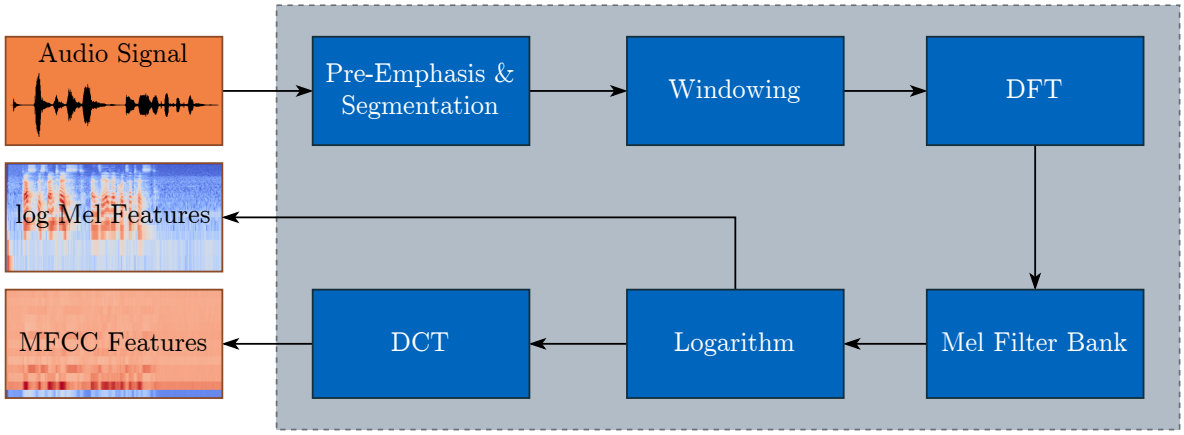


Figure 3.1: The procedure for generating the MFCC and log Mel features. The audio signal is pre-emphasized, segmented, and windowed before the DFT is applied. Then, the power spectrum components are passed through the Mel-filter banks and the logarithm is employed. The resulting components correspond to the log Mel features. If the DCT is utilized on the log Mel spectrum, the final MFCC features are obtained.

$$\overline{w}_l = 0.54 - 0.46 \cos\left(\frac{2\pi l}{\mathring{I}}\right) \quad \forall l \in \{1, 2, \cdots, \mathring{I}\}, \tag{3.21}$$

with $\boldsymbol{q} \in \mathbb{I}^{\mathring{I}}$ referring to the segment $\overline{\boldsymbol{x}}$ after applying the Hamming window. Notice that the following steps are defined for a single windowed segment to avoid cluttered notation.

3.2.3 Discrete Fourier Transformation

In the next step, each windowed segment $\boldsymbol{q}$ is transformed into the frequency domain by a DFT to retrieve its components $\boldsymbol{Q} \in \mathbb{C}^{\mathring{I}}$ [130]:

$$Q_k = \sum_{l=1}^{\mathring{I}} q_l \exp\left(\frac{-\mathrm{j}2\pi(l-1)(k-1)}{\mathring{I}}\right) \quad \forall k \in \{1, 2, \cdots, \mathring{I}\}, \tag{3.22}$$

where $\mathring{I}$ also defines the number of points utilized for the DFT, and j specifies the imaginary unit.

3.2.4 Mel Spectrum

The log Mel spectrum components $\boldsymbol{C} \in \mathbb{R}^M$ are obtained by passing the power spectrum components $|Q_k|^2$ through the Mel-filter bank $\Gamma_m(k)$, followed by taking the logarithm [130]:

$$C_m = \ln\left(\sum_{k=1}^{\mathring{I}} |Q_k|^2 \Gamma_m(k)\right) \quad \forall m \in \{1, 2, \cdots, M\}, \tag{3.23}$$

where $\Gamma_m(k)$ is defined as:

$$\Gamma_m(k) = \begin{cases} 0 & \text{if } k < \Lambda_{m-1} \\ \frac{2(k-\Lambda_{m-1})}{\Lambda_m - \Lambda_{m-1}} & \text{if } \Lambda_{m-1} \leq k \leq \Lambda_m \\ \frac{2(\Lambda_{m+1}-k)}{\Lambda_{m+1}-\Lambda_m} & \text{if } \Lambda_m < k \leq \Lambda_{m+1} \\ 0 & \text{if } k > \Lambda_{m+1}. \end{cases} \tag{3.24}$$

with $\mathbf{\Lambda} \in \mathbb{R}_+^M$ referring to the center frequencies of the Mel-filter bank $\Gamma_m(k)$. The center frequency vector $\mathbf{\Lambda}$ is obtained by [72]:

$$\Lambda_m = v^{-1}\left(v(F_{\min}) + \frac{v(F_{\max}) - v(F_{\min})}{M-1}(m-1)\right) \quad \forall m \in \{1, 2, \cdots, M\}, \tag{3.25}$$

with $F_{\min} \in \mathbb{R}_+$ and $F_{\max} \in \mathbb{R}_+$ corresponding to the minimal and maximal frequency, respectively, and M refers to the number of filters in $\Gamma_m(k)$. In practice, the minimum frequency is set to $F_{\min} = 300$ and the maximal frequency to $F_{\max} = 8000$ for speech signals sampled by 16 kHz [72]. Moreover, the transformation function $v(\,\cdot\,)$:

$$f_{\text{Mel}} = v(f_{\text{ord}}) = 1125 \ln\left(1 + \frac{f_{\text{ord}}}{700}\right), \tag{3.26}$$

and the inverse transformation function $v^{-1}(\,\cdot\,)$:

$$f_{\text{ord}} = v^{-1}(f_{\text{Mel}}) = 700\left(\exp\left(\frac{f_{\text{Mel}}}{1125}\right) - 1\right), \tag{3.27}$$

are required in the center-frequency vector $\mathbf{\Lambda}$. The transformation function $v(f_{\text{ord}})$ converts the ordinary frequency f_{ord} to the human-perceived frequency f_{Mel} in the Mel scale, and $v^{-1}(\,\cdot\,)$ transforms the Mel frequency f_{Mel} back to the ordinary frequency f_{ord}. Initially, the Mel scale was introduced by Stevens *et al.* [151], who analyzed frequency scales that better represent human hearing. They discovered that human hearing is superior to differentiating minor changes for low frequencies up to 1kHz and that this effect disperses for higher frequencies.

Furthermore, the logarithm of the Mel spectrum is taken to separate the actual speech signal from the impulse response of the vocal tract. If it is assumed that the actual speech signal is convolved with the impulse response of the vocal tract, a transformation in the frequency domain leads to a multiplication of both spectra. Thus, both spectra can be separated in the logarithm domain, where the multiplicative spectra are transformed into additive spectra. The additive spectral proportion of the vocal tract in the logarithm domain is removed in a subsequent procedure.

3.2.5 Discrete Cosine Transformation

The near-optimal compression property of the discrete cosine transformation (DCT) [4] is utilized to decorrelate the former Mel spectrum caused by the overlapping triangle

filters of the Mel-filter bank $\Gamma_m(k)$. The number of total MFCCs is typically reduced since most information is condensed in the first coefficients. Thereby, satisfying results are already obtained for the first 13 coefficients [124].

Finally, the Mel cepstrum [21] is retrieved by the DCT as:

$$\boldsymbol{c} = c_v = \sum_{m=1}^{M} C_m \cos\left(\frac{\pi(v-1)(m+0.5)}{M}\right) \quad \forall v \in \{1, 2, \cdots, V\}, \tag{3.28}$$

where V refers to the number of MFCCs, which can be lower or equal to the number of Mel-filter banks M.

Recently, the employment of the DCT and reducing the number of MFCCs have become less critical as rising computational resources simplify the processing of high-dimensional data. Nowadays, the DCT step is skipped to retrieve log Mel features $C_m \in \mathbb{R}^M$, or the number of total MFCCs is set to the maximal number of filter banks $V = M$.

3.2.6 Cepstral Mean Normalization

Transforming the Mel spectrum into the logarithm domain leads to a separation of the spectral proportion of the vocal track and the actual speech signal. If the transfer function of the vocal track is assumed constant on average, it is removable by normalization techniques. The cepstral mean normalization (CMN), also known as cepstral mean subtraction, belongs to the most popular normalization approaches for MFCCs [86]:

$$\overline{\boldsymbol{c}}^{(t)} = \boldsymbol{c}^{(t)} - \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{c}^{(t)}, \tag{3.29}$$

since it is simple to apply and only requires a marginal computational overhead. There have also been variations of cepstral mean normalization (CMN), where the CMN method is solely applied to speech-representing MFCCs and silenced MFCCs are excluded. A detailed study was conducted by Westphal, who analyzed the impact of several normalization techniques [169].

3.2.7 Dynamic Mel-Frequency Cepstral Coefficient Features

Even though the generated and normalized MFCCs $\overline{\boldsymbol{c}}^{(t)}$ efficiently condense the recorded speech signal into a few decorrelated coefficients, the resulting feature vectors solely represent static speech characteristics for a fixed time frame and disregard any dynamics of the signal. The first approach considering the dynamics of MFCCs was proposed by Furui [49], who improved the performance of his models by applying the dynamic features. The dynamics of normalized MFCCs are the velocity, which is denoted as the delta MFCCs $\Delta\overline{\boldsymbol{c}}^{(t)}$:

$$\Delta\overline{\boldsymbol{c}}^{(t)} = \frac{\sum_{n=1}^{2} n\left(\overline{\boldsymbol{c}}^{(t+n)} - \overline{\boldsymbol{c}}^{(t-n)}\right)}{2\sum_{n=1}^{N} n^2}, \tag{3.30}$$

and the acceleration referred to as the delta-delta MFCCs $\Delta^2\overline{\boldsymbol{c}}^{(t)}$:

$$\Delta^2\overline{\boldsymbol{c}}^{(t)} = \frac{\sum_{n=1}^{2} n\left(\Delta\overline{\boldsymbol{c}}^{(t+n)} - \Delta\overline{\boldsymbol{c}}^{(t-n)}\right)}{2\sum_{n=1}^{N} n^2}. \tag{3.31}$$

The normalized static MFCCs $\overline{\boldsymbol{c}}^{(t)}$, the delta MFCCs $\Delta\overline{\boldsymbol{c}}^{(t)}$, and the delta-delta MFCCs $\Delta^2\overline{\boldsymbol{c}}^{(t)}$ are concatenated to obtain the feature inputs $\boldsymbol{x}^{(t)} = \overline{\boldsymbol{c}}^{(t)} \oplus \Delta\overline{\boldsymbol{c}}^{(t)} \oplus \Delta^2\overline{\boldsymbol{c}}^{(t)}$ with $\boldsymbol{x}^{(t)} \in \mathbb{R}^{3V}$. Note that the approximation of the derivatives in Equations (3.30) and (3.31) is based on finite differences, explained in more detail in [104].

3.3 Speech Feature Augmentation

In order to avoid overfitting, input features are augmented by applying predefined transformations, leading to artificially increased training datasets. The augmentation is commonly employed at three distinct locations: directly on the raw speech signal $\widetilde{\boldsymbol{X}}$ [116], before retrieving the Mel spectrum by modifying either the Mel-filter bank $\Gamma\big(k\big)$ [72] or the power spectrum $|\boldsymbol{Q}|^2$ [74] or on the final MFCCs $\boldsymbol{c}^{(t)}$ [116]. Recent augmentation techniques for ASR models are introduced in the following section.

3.3.1 Speed Perturbation

In the past, common augmentation strategies combined the vocal tract length perturbation (VTLP) [72] and tempo perturbation [74] to increase the number of speech samples. The VTLP modifies the Mel-filter banks $\Gamma_m\big(k\big)$ (compare Section 3.2.4), where the center-frequency vector $\boldsymbol{\Lambda}$ is continuously transformed by randomly applying a linear warping along the frequency axis [72]. The tempo perturbation approach alters the speed rate of the signal directly in the power spectrum components $|Q_k|^2$ and leaves the pitch and spectral envelope of the signal unchanged [74].

A more straightforward approach was proposed by Ko *et al.* [77], who constructed additional data by modifying the speed of speech signal $\widetilde{\boldsymbol{X}}$. They employed the two additional speed factors of 90% and 110% of the original speech signal and adjusted the pitch before generating the MFCCs. The evaluation revealed that their simple speed perturbation approach outperformed VTLP and tempo perturbation and combinations of these techniques without increasing the implementation effort [77].

3.3.2 SpecAugment

An alternative augmentation technique SpecAugment was introduced by Park *et al.* [116], which is directly applicable to the normalized log Mel features $\boldsymbol{C}^{(t)}$ or normalized MFCC features $\boldsymbol{c}^{(t)}$. Initially, three deformations of the spectrum were introduced: *time warping*, *frequency masking*, and *time masking*. Recent works [8, 187] omitted the time warping as it is computationally expensive. Therefore, solely the time and frequency masking are discussed, and the interested reader is referred to [116].

Time Masking Time masking defines masks in which t_Δ adjacent input features $\boldsymbol{x}^{(t)}$ of an input sequence $\boldsymbol{X}$ are set to zero. The beginning of the mask is specified by the parameter t_0, which is drawn from uniform distribution $t_0 \sim \mathcal{U}\big(1, T\big)$. The mask length is set by the parameter t_Δ, drawn from uniform distribution $t_\Delta \sim \mathcal{U}\big(0, T_\Delta\big)$, and the maximum mask length is bounded by T_Δ. Generally, the time masking process is applied n_{tm} times on the same input spectrum, where n_{tm} is chosen by a uniform distribution $n_{\text{tm}} \sim \mathcal{U}\big(1, N_{\text{tm}}\big)$, and the number of repetitive mask operations are constrained by N_{tm} [116].

Frequency Masking The frequency masking procedure applies a mask by setting i_Δ consecutive entries in all input features $\boldsymbol{x}^{(t)}$ in $\boldsymbol{X}$ to zero. The starting position i_0 and the length i_Δ of the mask are drawn from the uniform distributions $i_0 \sim \mathcal{U}\big(1, I\big)$ and $i_\Delta \sim \mathcal{U}\big(0, I_\Delta\big)$, respectively, where I refers to the feature dimension of the feature sequence $\boldsymbol{X} \in \mathbb{R}^{I \times T}$ and i_Δ is bounded by I_Δ. The frequency masking procedure is also repeated n_{fm} times, determined by a uniform distribution $n_{\text{fm}} \sim \mathcal{U}\big(1, N_{\text{fm}}\big)$, and the masking procedure is limited to maximal repetitions of N_{fm} [116].

3.4 Speech Labels

The training criterion of AMs in ASR systems is defined as a standard classification task (compare to Section 2.2.5), where AMs are trained on a labeled dataset $\mathcal{D}_{\text{train}}$, followed by an evaluation of their generalization capabilities represented by the unseen dataset $\mathcal{D}_{\text{test}}$. Since robust and uncorrelated features are required to obtain stable results, MFCC features $\boldsymbol{c}^{(t)}$ (compare to Section 3.2) are utilized. Depending on the model setup, these features can slightly differ in their dimensions if delta $\Delta\overline{\boldsymbol{c}}^{(t)}$ and delta-delta MFCC features $\Delta^2\overline{\boldsymbol{c}}^{(t)}$ are appended or solely log Mel coefficients $\boldsymbol{C}^{(t)}$ are utilized by skipping the DCT.

Three popular options are available for target labels: Phonemes, characters, or byte pair encoding (BPE) units. In the following, these label types are introduced and compared utilizing the exemplary word combination:

Automatic speech recognition.

3.4.1 Phonemes

Phonemes are phonetic base units of spoken words [158] and offer a general pronunciation definition for words in any language. Although spoken words slightly differ for every speaker due to their anatomy or dialect, people can understand each other as long as the phonemes are closely related, recognizable, and the word to recognize is already known [158].

Monophone In ASR, single phonemes are typically denoted as *monophones (MONOs)*, and either modeled by three-state HMMs [143] or as stand-alone DNNs [161]. Instead of

Table 3.1: The 39 phonemes in the English language with corresponding examples and translations [2].

Phoneme	Example	Translation	Phoneme	Example	Translation
AA	odd	AA D	L	lee	L IY
AE	at	AE T	M	me	M IY
AH	hut	HH AH T	N	knee	N IY
AO	ought	AO T	NG	ping	P IH NG
AW	cow	K AW	OW	oat	OW T
AY	hide	HH AY D	OY	toy	T OY
B	be	B IY	P	pee	P IY
CH	cheese	CH IY Z	R	read	R IY D
D	dee	D IY	S	sea	S IY
DH	thee	DH IY	SH	she	SH IY
EH	Ed	EH D	T	tea	T IY
ER	hurt	HH ER T	TH	theta	TH EY T AH
EY	ate	EY T	UH	hood	HH UH D
F	fee	F IY	UW	two	T UW
G	green	G R IY N	V	vee	V IY
HH	he	HH IY	W	we	W IY
IH	it	IH T	Y	yield	Y IY L D
IY	eat	IY T	Z	zee	Z IY
JH	gee	JH IY	ZH	seizure	S IY ZH ER
K	key	K IY			

modeling high-dimensional output spaces with hundreds of thousands of English words, each word is converted to its MONO representation. The English language distinguishes between 39 phonemes [2] summarized in Table 3.1 with corresponding examples. As a result, the output space of ASR models is highly reduced from thousands of words to only 39 MONOs. The representative MONO sequences for every word are obtained by predefined lexicons (*e.g.*, the CMUdict [2]), which offer mappings between words and MONOs and vice versa. The application of the CMUdict on the exemplary word sequence leads to the following MONO sequence:

AO T AH M AE T IH K S P IY CH R EH K AH G N IH SH AH N.

Triphone Even though MONOs are reducing the training complexity by reducing the output space of ASR models, they lack consistency compared to words [85]. The inconsistency is commonly reduced by employing context-dependent phonemes, also known as *triphones (TRIs)* [143], where three adjacent MONOs are combined into a single TRI. This leads to 59 319 unique TRI combinations, which fortunately do not all exist in the English language. Additionally, the number of TRIs is reduced by clustering and regression tree (CART) approaches, where similar TRIs are clustered together and

tied to single representative TRIs [176]. The exemplary word sequence is translated to the TRI sequence:

AO-T+AH T-AH+M AH-M+AE M-AE+T AE-T+IH T-IH+K IH-K+S K-S+P
S-P+IY P-IY+CH IY-CH+R CH-R+EH R-EH+K EH-K+AH K-AH+G
AH-G+N G-N+IH N-IH+SH IH-SH+AH SH-AH+N.

Notice that the TRIs notation is inspired by Young *et al.* [175], who specified TRI as *L-X+R*, where *L* refers to the left-content MONO, *X* to the center-content MONO, and *R* to the right-content MONO.

3.4.2 Characters

MONOs and TRIs still belong to the most popular target label types in ASR and achieve SOTA results [98]. However, they always require a lexicon for mapping between phonemes and words, and if a specific word mapping is unknown, the phoneme sequence is incorrectly recognized. This typical issue in phoneme-based ASR approaches known as the out of vocabulary (OOV) problem. The problem describes a condition in which a predicted phoneme sequence cannot be mapped to a corresponding word, as the translation does not exist in the lexicon. A common strategy consists in increasing the lexicon size to reduce the number of missing words, though the strategy is prone to fail for special person or city names.

With the rising popularity of DNNs, there has been an increased interest in E2E approaches based on RNNs, which directly convert input features into character (CHAR) sequences. Until 2006, these approaches required a pre-segmentation of the training data and post-processing to retrieve the final word sequence, as RNNs were only trained on independent pre-segmented training samples [55]. In 2006, Graves *et al.* introduced the CTC approach, enabling RNN-based E2E approaches to directly apply CHAR labels for unsegmented sequential data such as MFCC feature sequences [55]. Therefore, a lexicon was not further required, and the issue of OOV was solved. The exemplary word sequence is mapped into the following CHAR sequence:

A U T O M A T I C S P E E C H R E C O G N I T I O N.

3.4.3 BPE Units

The byte pair encoding (BPE) units [144] belong to predominant word segmentation algorithms in E2E ASR. The BPE algorithm is typically applied by a tokenizer, *e.g.*, the popular Sentencepiece tokenizer [79], which determines the optimal, representative tokens given the training dataset transcript. The number of tokes is commonly limited by heuristically chosen values. Each word is then represented by those tokes, constructed by CHARs, punctuation, or special graphemes. The tokenized exemplary word sequence is represented by the obtained tokes as:

Table 3.2: A summary of the characteristics of the TED-LIUM-v2 dataset according to the obtained splits. The symbol # is read as the *number of*.

Characteristics of TED-LIUM-v2				
		# Speakers		
Split	# Hours	# Female	# Male	# Total Speakers
Train	206.8	398	844	1242
Dev	1.7	1	7	8
Test	3.1	2	9	11

_A UT OM AT IC _SP E E CH _RE C O G N ITION,

where the tokes are derived by the Sentencepiece tokenizer, which is trained on the transcript of the TED-LIUM-v2 [138], and the maximum number of the tokens is set to 500.

3.5 Popular ASR Datasets

In ASR, models are evaluated under various conditions, which differ in the level of difficulty. The most straightforward condition corresponds to single-speaker and noise-free environments. More challenging conditions are represented by the overlapping speech from various speakers [164] or speech interfered with by general noise [89]. In this thesis, all established approaches assume single-speaker and clean speech data.

Over the years, several open-source clean-speech datasets were introduced, steadily increasing in size. The increase in size involved a consistently higher demand for computational power to efficiently train ASR models on these datasets. Thereby, three well-known datasets were established in the speech community: the TED-LIUM-v2 [138], a dataset with up to 250 h of speech data, followed by LibriSpeech [115] with nearly 1000 h of speech data, and GigaSpeech [27] with 10 000 h labeled speech data. Since solely the TED-LIUM-v2 and LibriSpeech datasets are utilized in the subsequent experiments, they are discussed in more detail in the following sections.

3.5.1 TED-LIUM Release 2

The TED-LIUM release 2 (TED-LIUM-v2) [138] dataset corresponds to a small-sized dataset, requiring limited resources in the form of a single graphics processing unit (GPU). The dataset incorporates several talks from the American media company technology entertainment design (TED) [5], which invites famous and non-famous speakers worldwide to give lectures with subtitles under the slogan "ideas worth spreading" [5].

As all these talks are subtitled, Rousseau *et al.* (from the Laboratoire Informatique de'l Universitè du Maine (LIUM)) decided to create the TED-LIUM-v2 [138] in 2014,

Table 3.3: The characteristics of the LibriSpeech dataset with regarding female and male speakers and their corresponding recording time for each split. The symbol # is read as the *number of*.

Characteristics of LibriSpeech				
		# Speakers		
Split	# Hours	# Female	# Male	# Total Speakers
Train-other-500	496.7	564	602	1166
Train-clean-360	363.6	439	482	921
Train-clean-100	100.6	125	126	251
Train-combined	960.9	1128	1210	2338
Dev-clean	5.4	20	20	40
Dev-other	5.3	16	17	33
Test-clean	5.4	20	20	40
Test-other	5.1	17	16	33

where they collected 1495 TED lectures with a mean duration of 10 min and 12 s and held by 1242 unique speakers. These lectures were then filtered and split into 92 976 speech segments with their corresponding transcript. This process led to the *train* split of the overall dataset with a total of 207 h transcript speech data, whereby 141 h are male and 66 h female speakers. A similar process was repeated for the *dev* and *test*, resulting in 507 and 1155 speech segments, respectively. Furthermore, the most relevant of the overall 2.6 million words were summarized in a phonetized dictionary with 159 849-word variants and 152 213 unique words. All characteristics of the dataset are depicted in Table 3.2.

3.5.2 LibriSpeech

LibriSpeech [115] defines a dataset of mid-range size, which requires more computational demanding resources. Panayotov *et al.* derived the dataset by combining the books of Project Gutenberg [63] and the speech data of the audiobooks from LibriVox [102], where both groups are volunteer-driven projects which rely on public domain books. The Project Gutenberg was founded by Michael Hart in 1971 to "encourage the creation and distribution of ebooks" [63], where volunteers collect public domain books, digitalize and provide them as ebooks to the public. In 2005, Hugh McGuire extended the Project Gutenberg by founding LibriVox [102]. LibriVox is also a volunteer-driven project that records public domain books from Project Gutenberg and releases the recordings as public domain audiobooks.

On the foundation of Public Gutenberg and LibriVox, Panayotov *et al.* established the dataset LibriSpeech in 2015 [115], in which they began to gather 8000 public domain audiobooks (in 2015) from LibriVox [102] with the corresponding meta-data and retrieved

the according transcript in the ebooks from Project Gutenberg [63]. Then, a four-stage process was applied to obtain good speech-to-text alignments. First, the transcript of all ebooks was preprocessed to be consistent, followed by training a simple bigram LM for each ebook. In parallel, MFCCs [42] were extracted from the audiobooks recorded by over 9000 unique speakers. For the lexicon, they relied on the 134 000 words of CMUdict [2] provided by Carnegie Mellon University. An initial HMM-GMM AM was trained on the 100 h VoxForge dataset [1] made available by volunteers, who manually aligned their recordings to the corresponding transcript. Combining the optimized AM and the LM of each book, they decoded every audiobook to determine a base alignment. Then, the Smith-Waterman alignment algorithm [148] was employed to obtain an ideal local alignment. The alignments with the highest similarity were stored and split into segments of 35 s lengths, whereas the remaining ones were discarded. Inaccurate alignments with transpositions, substitutions, deletions, or insertions of the reader were filtered out [115]. The accurate 35 s segments were further split into shorter alignments if the silence exceeded 0.3 s. For the dev and test data, a cut was permitted at the end of sentences.

Finally, the retrieved base corpus of around 1200 h of speech data was partitioned into a disjoint train set of *train-clean-100*, *train-clean-360*, and *train-other-500*, a dev set of *dev-clean* and *dev-other*, and a test set of *test-clean* and *test-other*. A summary of all splits is depicted in Table 3.3.

3.6 SOTA Results on Popular ASR Datasets

This section gathers current SOTA results on TED-LIUM-v2 and LibriSpeech, compared to the approaches later established in Chapters 4 to 6. Since the performance of all ASR approaches differs significantly, several reasons are discussed.

3.6.1 SOTA Results on TED-LIUM-v2

It is questionable if general comparisons of current SOTA approaches [60–62, 75, 81, 181, 187] on the TED-LIUM-v2 dataset are sufficient to determine the overall best model architecture since only a limited number of novel approaches have been evaluated on the TED-LIUM-v2 dataset. Based on Table 3.4, hybrid approaches seem to define the superior model class, even though different AMs and training schemes are applied. For instance, in 2017, Han *et al.* utilized a combination of time-delay neural networks (TDNNs) and long short-term memory networks (LSTMs) [61], and later in 2019, a multi-stride transformer AM [62] relied on lattice-free maximum mutual information (LF-MMI) as an objective function. On the other hand, Zhou *et al.* selected a standard multi-layer bidirectional long short-term memory (BLSTM) as AM, optimized with the cross entropy (CE) objective, and extended the optimization by discriminative sequence training [51]. Therefore, it is challenging to define a fair comparison between different models, and it seems that all existing architectures have the right to exist.

The comparison in Table 3.4 demonstrates that the introduced methods are not competitive with recently proposed SOTA approaches. However, it is vital to put the

Table 3.4: An overview of relevant SOTA results for the TED-LIUM-v2 [138] reported for recent approaches and established approaches of this thesis. All results are given in WER (%).

TED-LIUM-v2							
Paper	Year	AM	Labels	CTC	LM	Dev	Test
Han *et al.* [62]	2019	Hybrid	TRI	✗	✓	7.7	8.0
Han *et al.* [61]	2017	Hybrid	TRI	✗	✓	7.1	7.7
Zhou *et al.* [187]	2020	Hybrid	TRI	✗	✓	**5.1**	**5.6**
Karita *et al.* [75]	2019	RNN	TRI	✗	✓	9.3	8.1
Kürzinger *et al.* [81]	2020	Attention	BPE	✓	✓	10.7	10.7
Zeyer *et al.* [181]	2019	Attention	TRI	✗	✓	10.3	8.8
Guo *et al.* [60]	2021	Self-Attention	BPE	✓	✗	9.3	8.7
Guo *et al.* [60]	2021	Self-Attention	BPE	✓	✓	8.6	7.2
Watzel *et al.* [13†]	2019	Hybrid	TRI	✗	✗	27.1	27.1
Watzel *et al.* [12†]	2020	Attention	CHAR	✗	✗	15.7	15.9
Watzel *et al.* [10†]	2020	Self-Attention	BPE	✗	✗	14.5	13.2
Watzel *et al.* [11†]	2021	Self-Attention	BPE	✓	✗	**9.8**	**9.1**

results in context to the examined and proposed methods in [10†, 11†, 12†, 13†]. A closer look into Chapters 4 to 6 reveals that all approaches did not aim to surpass current SOTA models and examined the effects of specific modifications regarding the overall model parameters. The utilized architectures were simplified versions of SOTA models, where the AMs were reduced in total size, and auxiliary losses were omitted. In addition, none of these methods integrated an LM into the final evaluation, which typically further reduces the WER on the evaluation sets (see Zeyer *et al.* [182]). Therefore, a fair comparison of [10†, 11†, 12†, 13†] to the most current SOTA approaches is not given.

3.6.2 SOTA Results on LibriSpeech

Nowadays, the LibriSpeech [115] belongs to the most popular noise-free ASR datasets and has been intensively evaluated over the years. Thereby, novel model architectures were proposed, surpassing existing methods and questioning older approaches. However, besides the model architectures, these novel approaches also introduced innovative training methods, boosting the performance of traditional ASR models. The latest SOTA models [60, 61, 98, 114, 179, 182], evaluated on LibriSpeech, are listed in Table 3.5 and compared to the novel training method [12†] introduced in Chapter 5. The approach of Watzel *et al.* [12†] is not competitive with the SOTA approaches. However, this direct comparison can not be seen as fair, as the authors did not aim to surpass SOTA attentional models, which is observable in their applied model structure and training scheme. Compared to Zeyer *et al.* [182], who utilized a model architecture similar to Watzel *et al.* [12†], Watzel *et al.* constructed their encoder by four LSTM layers with 1024 units and did not pre-train them. The output layer size of the decoder was set

Table 3.5: Relevant SOTA results for LibriSpeech [138] reported in recent approaches and compared to established approaches of this thesis. Note that approaches utilizing additional data or not fully covering all evaluation sets are excluded from this comparison. All results are stated in WER (%).

LibriSpeech									
Paper	Year	AM	Labels	CTC	LM	Dev		Test	
						Clean	Other	Clean	Other
Zeghidour *et al.* [179]	2017	CNN	CHAR	✗	✓	3.2	10.1	3.4	11.2
Pan *et al.* [114]	2020	CNN	TRI	✓	✓	**1.6**	**4.2**	**1.8**	**4.5**
Han *et al.* [61]	2017	Hybrid	TRI	✗	✓	3.0	8.8	3.6	8.7
Lüscher *et al.* [98]	2019	Hybrid	TRI	✗	✓	1.9	4.5	2.3	5.0
Zeyer *et al.* [182]	2018	Attention	BPE	✓	✗	4.9	14.4	4.9	15.4
Zeyer *et al.* [182]	2018	Attention	BPE	✓	✓	4.3	12.9	4.4	13.5
Guo *et al.* [60]	2021	Self-Attention	BPE	✓	✓	1.9	4.9	2.1	4.9
Watzel *et al.* [12†]	2020	Attention	BPE	✗	✗	7.2	20.1	7.3	20.6

to 100 since they utilized 100 BPE units, and the overall training was not guided by the CTC loss [55]. In contrast, Zeyer *et al.* [182] employed six encoder layers with 2024 LSTM units and deployed an encoder pre-training. Moreover, they created 10 000 BPE units, adapted their decoder output to this size, and supported the overall model by the CTC loss [55].

Other exciting aspects are the novel model architectures in general. The results in Table 3.5 reveal that a particular model architecture alone does not lead to the best ASR approach. The three leading approaches [60, 98, 114], which do not utilize external data and fully report the results on all evaluation sets, rely on three different model architectures. The AM applied by Lüscher *et al.* [98] corresponds to a standard HMM-DNN model, further optimized by discriminative sequence training [51] based on generated lattices. In the approach of Guo *et al.* [60], the AM is represented by a Conformer [59], an extension of a transformer model [159]. And Pan *et al.* [114] built up their AM on a variant of 1D-convolutional neural networks (CNNs) organized in a multistream setup. All these architectures differ fundamentally, and their performance gap is negligible. This indicates that each model has the potential to reduce the WER on LibriSpeech further, and no architecture should receive preferential treatment.

4

Hybrid Acoustic Modeling

In this chapter, the alternative approach from Watzel *et al.* [13†] for discrete acoustic models (AMs) is established, demonstrating the potential of deep neural networks (DNNs) as discrete AMs compared with traditional continuous models, such as Gaussian mixture models (GMMs). The first section introduces discrete and continuous automatic speech recognition (ASR) formulations and elaborates on the strengths and weaknesses of these approaches. The second section discusses related methods comparable to the later introduced deep neural network quantizer (DNNQ), then lays the mathematical foundation for the established procedure. The fourth section specifies the overall training setup by defining the model architecture, the training scheme, and the decoding approach. The final results are examined in the fifth section, where several configurations exhaustively analyze the performance of DNNQ and continuous GMMs. The last section concludes the chapter by giving an outlook on future works and the potential of discrete ASR models.

4.1 Introduction

Over the years, hybrid ASR models steadily reduced their word error rate (WER) on several clean speech ASR datasets as the Wall Street Journal (WSJ) dataset [120], containing solely clean speech with one speaker and without any noise, and on the more challenging datasets, such as the augmented multi-party interaction (AMI) dataset [23], where several office meeting scenarios were recorded with multiple speakers talking and additional background noise from the office itself.

Hybrid ASR often distinguishes between applying discrete and continuous AMs, which differentiate in the employed input sequence $\boldsymbol{X}$. In continuous AMs, the high-dimensional inputs $\boldsymbol{x}^{(t)} \in \mathbb{R}^I$ are typically uncountable and from an input sequence $\boldsymbol{X} \in \mathbb{R}^{I \times T}$. Therefore, generating the AM likelihood $p\big(\boldsymbol{X}|\boldsymbol{S}\big)$ requires probability density functions (PDFs), which specify functions transforming any input $\boldsymbol{x}^{(t)} \in \mathbb{R}^I$ into continuous sampling spaces. A well-known PDF model is GMMs [22], where a single Gaussian from a GMM is depicted in Figure 4.1. In the training phase, the statistics for each state s_k with its corresponding GMMs are gathered, and the parameters are

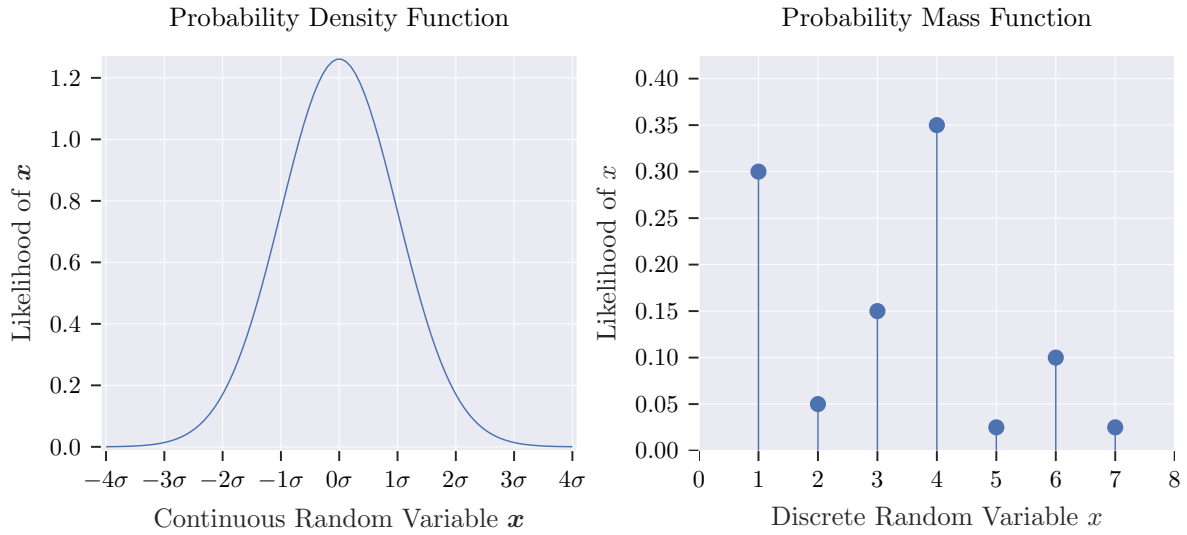


Figure 4.1: On the left, the PDF of a single Gaussian with its parameters $\mu = 0$ and $\Sigma = 0.1$ is plotted, whereby PDFs are typically applied in continuous AMs. Notice that the likelihood $p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}\big)$ can exceed values of one. The PMF of a discrete random variable $x^{(t)}$ is depicted on the right, commonly utilized in discrete AMs. In contrast to PDFs, the likelihood $p\big(x^{(t)}|\boldsymbol{s}^{(t)}\big)$ is limited to values between zero and one in PMFs.

derived by maximum likelihood estimation (MLE). The GMMs can also be replaced by strong DNN discriminators, modeling the posterior $p\big(\boldsymbol{S}|\boldsymbol{X}\big)$. In the decoding phase, the posterior is then transformed into the pseudo-likelihoods specified in Equation (3.6).

Discrete AMs rely on discrete random variables, where the inputs $x^{(t)} \in \mathbb{N}$ are from an input sequence $\boldsymbol{X} \in \mathbb{N}^{1\times T}$ and are countable, as the number of individual inputs is commonly limited. Since the emitted likelihood $p\big(\boldsymbol{X}|\boldsymbol{S}\big)$ relies on discrete values, probability mass functions (PMFs) are applied for each state s_k. The PMFs can be seen as look-up tables, where the statistics of occurring discrete inputs $x^{(t)}$ are counted and normalized by the total number of state visits. Then, the corresponding emission probabilities $p\big(\boldsymbol{X}|\boldsymbol{S}\big)$ are retrieved by selecting the value at the index $x^{(t)}$ of a look-up table [22]. Notice that in contrast to continuous PDFs, where the likelihood $p\big(\boldsymbol{x}^{(t)}|\boldsymbol{s}^{(t)}\big)$ can exceed values of one, the PMF likelihood $p\big(x^{(t)}|\boldsymbol{s}^{(t)}\big)$ is restricted to values equal to or smaller than one, which can be observed in Figure 4.1.

In 2016, the performance of hybrid ASR systems relying on continuous AMs and employing clean speech datasets pared with human recognition performance [172]. Even though this milestone demonstrated the superiority of DNNs in ASR, their training approaches are computationally expensive and require tremendous resources and labeled datasets. In the past, alternative approaches with discrete AMs were quite popular since they reduced the complexity of ASR models by quantizing the continuous input features $\boldsymbol{x}^{(t)} \in \mathbb{R}^I$ into discrete feature $x^{(t)} \in \mathbb{N}$ before being further processed. Generally, quantization is achieved by applying vector quantizers (VQs) with corresponding optimized centroids, also denoted as codebooks. These codebooks are determined by unsupervised algorithms such as the k-means algorithm [18]. The VQs determine the minimal distance

of $\boldsymbol{x}^{(t)}$ to all the classes in the codebooks and assigns the corresponding class label $x^{(t)}$ with the minimal distance.

Nowadays, discrete AMs have been more or less disregarded, as each quantization procedure leads to a loss of information. As a result, ASR models based on discrete AMs are inferior to continuous AMs and produce higher WERs.

Despite the disadvantage of the quantization procedure in discrete AMs, multiple discrete approaches were established in the past [25, 110, 133–135, 137, 178]. However, they did not apply a fair comparison to continuous AMs or did not evaluate large-vocabulary speech recognition (LVSR) datasets. This chapter establishes multiple novel contributions initially proposed by Watzel *et al.* [13†]:

- Methods for modeling discrete AMs are revisited and reimplemented in current state-of-the-art (SOTA) toolkits.
- A novel deep neural network quantizer (DNNQ) for discrete AMs is introduced, which can be trained on arbitrary output layer sizes.
- A novel cost sampling procedure is proposed, which provides an efficient training process for DNNQs.
- A fair comparison of discrete and continuous AMs is given by evaluating the LVSR dataset TED-LIUM release 2 (TED-LIUM-v2) [138].
- Exhaustive evaluations of the DNNQs reveal that discrete AMs, represented by DNNQs, outperform continuous GMM systems.

4.2 Related Work

The first detailed examination of vector quantization procedure in speech encoding applications was done by Makhoul *et al.* [101]. They analyzed the performance of VQs for encoding speech signals and derived the theoretical limits of such quantization procedures. Moreover, they analyzed common problems in practical environments, such as optimal VQ architectures, efficient implementations, and performance in practice.

A few years later, Yu *et al.* applied VQs in phoneme recognition and compared several supervised clustering approaches for determining corresponding codebooks. They assumed improved VQs would lead to higher recognition rates in the subsequent discrete hidden Markov models (DHMMs) [178]. However, they did not observe a decrease in the WER even though the VQs returned better phoneme recognition rates.

Another related approach was established by Le Cerf *et al.*, who utilized multilayer perceptrons (MLPs) as quantization networks [25]. They quantized the inputs $\boldsymbol{x}^{(t)}$ by selecting the output with the highest probability as corresponding class labels instead of employing MLPs as probability estimators. Furthermore, they provided several extensions of their approach, in which multiple outputs based on descending order of the output probabilities were applied, MLPs were considered as fuzzy VQs, or individual MLPs, for processing the base features, the delta, the delta-delta, and the energy features.

In all former approaches, the VQ quantization procedure relied on minimizing the distortion of each input $\boldsymbol{x}^{(t)}$ to its class label $x^{(t)}$. An alternative approach was proposed by Rigoll in 1994 [133], in which he laid the mathematical foundation for maximum mutual information (MMI) MLPs, also known as neural network vector quantizers (NNVQs), where he utilized the information-theoretic MMI criterion as a distance measurement for obtaining the proper NNVQ weights. His evaluation revealed that NNVQs are superior to standard VQs, whose codebooks were retrieved by classical k-means training [100]. Although NNVQs seemed to represent strong alternatives to standard approaches, they contained three drawbacks:

1. The standard backpropagation approach was not applicable, as the quantization of inputs $\boldsymbol{x}^{(t)}$ to class labels $x^{(t)}$ is achieved by selecting the number of the maximum output of the NNVQs, corresponding to the mathematical non-differentiable max operator.

2. It remained unclear if NNVQs are competitive with continuous approaches since the evaluation of the novel architecture was solely compared with VQs relying on codebooks derived by the k-means algorithm.

3. The model was evaluated in a phoneme recognition task, where 18 Japanese phonemes had to be recognized, and it was unclear how well the novel NNVQ model would perform on LVSR datasets.

Two years later, the latter two drawbacks were eliminated by Rigoll *et al.* [135], who trained their novel NNVQs on the 1000-word Resource Management (RM) dataset introduced by the defense advanced research projects agency (DARPA) in 1988 [126]. Rigoll *et al.* split their single NNVQ into four separate NNVQs and applied spliced feature vectors to increase the feature context, leading to three NNVQs trained on the base, delta, and delta-delta features, respectively, and a single NNVQ on the log energy and its two derivatives. The final evaluation revealed that combined NNVQs were superior to k-means-determined VQs but could not surpass continuous AM systems modeled by GMMs.

Shortly after the first evaluation on a larger dataset, Neukirchen and Rigoll presented a solution for the non-differentiable max operator [134]. They utilized the scaled and differentiable softmax function on NNVQ outputs, leading to an approximation of the max operator if a proper scaling factor was selected and enabled the application of a standard backpropagation (BP) approach (compare Section 2.2.7). The evaluation of the novel NNVQ on the RM dataset also indicated that, for the first time, discrete AMs were slightly superior to continuous AMs represented by GMMs. Additionally, a theoretical analysis of novel NNVQs in the context of GMMs was also given [110], where their differentiable solution led to a generalization of NNVQs, resulting in network architectures with arbitrary output layer sizes.

A few years later, the NNVQs were further evaluated on the challenging WSJ dataset [120] that contained over 400 h of labeled speech data with 47 M words and belonged to the largest speech datasets at this time. Even though Rottland *et al.*

expected promising results, the obtained results revealed that discrete AMs represented by NNVQs were not competitive with continuous GMM AMs.

4.3 Proposed Method

This section is divided into the theory of standard NNVQs, the extension of NNVQs into DNNQs and their training scheme, and the training procedure of discrete hidden Markov models (HMMs). First, the theory of the NNVQs is established [133, 134]. Then, the framework of the NNVQs is further extended to the novel architecture and training scheme of DNNQs. The last part describes the integration of the quantized outputs $x^{(t)}$ into the framework of discrete HMMs.

4.3.1 Theory of Standard NNVQ

The theory of training NNVQs by the mutual information (MI) criterion is derived by following the approaches in [133, 134], where a more general definition of the emission probabilities from Equation (2.7) is defined:

$$\boldsymbol{\Theta}^{*} = \arg\max_{\boldsymbol{\Theta}} \left\{ \ln p\big(\boldsymbol{X}|\boldsymbol{\Omega},\boldsymbol{\Theta}\big) \right\} = \arg\max_{\boldsymbol{\Theta}} \left\{ \sum_{t=1}^{T} \ln p\big(\boldsymbol{x}^{(t)}|\boldsymbol{\omega}^{(t)},\boldsymbol{\Theta}\big) \right\} \tag{4.1}$$

$$= \arg\max_{\boldsymbol{\Theta}} \left\{ \sum_{t=1}^{T} \sum_{j=1}^{\overset{*}{J}^{[L]}} \ln p\big(\boldsymbol{x}^{(t)}, m_j^{(t)}|\boldsymbol{\omega}^{(t)},\boldsymbol{\Theta}\big) \right\}, \tag{4.2}$$

where $\boldsymbol{m}^{(t)} = \mathcal{H}_{\boldsymbol{\Theta}}(\boldsymbol{x}^{(t)})$ with $\boldsymbol{m}^{(t)} \in \mathbb{R}^{\overset{*}{J}^{[L]}}$ refers to the output of NNVQs, and $\overset{*}{J}^{[L]}$ can be arbitrarily chosen depending on the task to solve. The model $\boldsymbol{\Theta}$ can represent any arbitrary DNN architectures, and $\boldsymbol{\Omega} = [\boldsymbol{\omega}^{(1)}, \cdots, \boldsymbol{\omega}^{(t)}, \cdots, \boldsymbol{\omega}^{(T)}]$ are class labels for the corresponding input sequence $\boldsymbol{X}$, where monophones (MONOs) or triphones (TRIs) states are popular target label options in ASR.

In order to simplify the subsequent notations, the parameters $\boldsymbol{\Theta}$ are omitted in the next step, and the likelihood $p\big(\boldsymbol{x}^{(t)}|\boldsymbol{\omega}^{(t)},\boldsymbol{\Theta}\big)$ is rewritten as:

$$p\big(\boldsymbol{x}^{(t)}|\boldsymbol{\omega}^{(t)},\boldsymbol{\Theta}\big) = p\big(\boldsymbol{x}^{(t)}|\boldsymbol{\omega}^{(t)}\big) = \sum_{j=1}^{\overset{*}{J}^{[L]}} p\big(\boldsymbol{x}^{(t)}, m_j^{(t)}|\boldsymbol{\omega}^{(t)}\big) \tag{4.3}$$

$$= \sum_{j=1}^{\overset{*}{J}^{[L]}} p\big(\boldsymbol{x}^{(t)}|m_j^{(t)},\boldsymbol{\omega}^{(t)}\big) p\big(m_j^{(t)}|\boldsymbol{\omega}^{(t)}\big) = \sum_{j=1}^{\overset{*}{J}^{[L]}} p\big(\boldsymbol{x}^{(t)}|m_j^{(t)}\big) p\big(m_j^{(t)}|\boldsymbol{\omega}^{(t)}\big), \tag{4.4}$$

where $p\big(\boldsymbol{x}^{(t)}|m_j^{(t)},\boldsymbol{\omega}^{(t)}\big)$ is reduced to $p\big(\boldsymbol{x}^{(t)}|m_j^{(t)}\big)$, as the inputs $\boldsymbol{x}^{(t)}$ are commonly considered conditionally independent of the classes $\boldsymbol{\omega}^{(t)}$ [133]. Then, the likelihood

$p\big(\boldsymbol{x}^{(t)}|m_j^{(t)}\big)$ and the conditional probability $p\big(m_j^{(t)}|\boldsymbol{\omega}^{(t)}\big)$ are reformulated by utilizing Bayes' theorem, yielding:

$$p\big(\boldsymbol{x}^{(t)}|m_j^{(t)}\big) = \frac{p\big(m_j^{(t)}|\boldsymbol{x}^{(t)}\big)p\big(\boldsymbol{x}^{(t)}\big)}{p\big(m_j^{(t)}\big)}, \tag{4.5}$$

and:

$$p\big(m_j^{(t)}|\boldsymbol{\omega}^{(t)}\big) = \frac{p\big(\boldsymbol{\omega}^{(t)}|m_j^{(t)}\big)p\big(m_j^{(t)}\big)}{p\big(\boldsymbol{\omega}^{(t)}\big)}. \tag{4.6}$$

Both terms are substituted into Equation (4.4):

$$p\big(\boldsymbol{x}^{(t)}|\boldsymbol{\omega}^{(t)}\big) = \sum_{j=1}^{\overset{*}{J}{}^{[L]}} \frac{p\big(m_j^{(t)}|\boldsymbol{x}^{(t)}\big)p\big(\boldsymbol{x}^{(t)}\big)}{p\big(m_j^{(t)}\big)} \frac{p\big(\boldsymbol{\omega}^{(t)}|m_j^{(t)}\big)p\big(m_j^{(t)}\big)}{p\big(\boldsymbol{\omega}^{(t)}\big)} \tag{4.7}$$

$$= \frac{p\big(\boldsymbol{x}^{(t)}\big)}{p\big(\boldsymbol{\omega}^{(t)}\big)} \sum_{j=1}^{\overset{*}{J}{}^{[L]}} p\big(\boldsymbol{\omega}^{(t)}|m_j^{(t)}\big)p\big(m_j^{(t)}|\boldsymbol{x}^{(t)}\big) = \frac{p\big(\boldsymbol{x}^{(t)}\big)}{p\big(\boldsymbol{\omega}^{(t)}\big)} p\big(\boldsymbol{\omega}^{(t)}|\overset{*}{m}{}^{(t)}\big), \tag{4.8}$$

and the standard quantization procedure of VQs is applied to obtain simplified conditional probability $p\big(\boldsymbol{\omega}^{(t)}|\overset{*}{m}{}^{(t)}\big)$:

$$p\big(m_j^{(t)}|\boldsymbol{x}^{(t)}\big) = \begin{cases} 1 & \text{if } m_j^{(t)} = \overset{*}{m}{}^{(t)} \\ 0 & \text{else} \end{cases} \qquad \forall j \in \{1, 2, \cdots, J^{[L]}\}, \tag{4.9}$$

where $p\big(m_j^{(t)}|\boldsymbol{x}^{(t)}\big)$ refers to the posterior distribution given an input $\boldsymbol{x}^{(t)}$. Notice that the posterior distribution has values of one at the index j of the firing neuron $\overset{*}{m}{}^{(t)}$ solely and zero otherwise.

Finally, $p\big(\boldsymbol{x}^{(t)}|\boldsymbol{\omega}^{(t)}\big)$ is substituted into the general likelihood formulation defined in Equation (4.2):

$$\boldsymbol{\Theta}^* = \arg\max_{\boldsymbol{\Theta}} \Big\{ \sum_{t=1}^{T} \ln \frac{p\big(\boldsymbol{x}^{(t)}\big)}{p\big(\boldsymbol{\omega}^{(t)}\big)} p\big(\boldsymbol{\omega}^{(t)}|\overset{*}{m}{}^{(t)}\big) \Big\} \tag{4.10}$$

$$= \arg\max_{\boldsymbol{\Theta}} \Big\{ \sum_{t=1}^{T} \ln p\big(\boldsymbol{x}^{(t)}\big) - \sum_{t=1}^{T} \ln p\big(\boldsymbol{\omega}^{(t)}\big) + \sum_{t=1}^{T} \ln p\big(\boldsymbol{\omega}^{(t)}|\overset{*}{m}{}^{(t)}\big) \Big\} \tag{4.11}$$

$$= \arg\max_{\boldsymbol{\Theta}} \Big\{ - \sum_{t=1}^{T} \ln p\big(\boldsymbol{\omega}^{(t)}\big) + \sum_{t=1}^{T} \ln p\big(\boldsymbol{\omega}^{(t)}|\overset{*}{m}{}^{(t)}\big) \Big\}, \tag{4.12}$$

whereby $\sum_{t=1}^{T} \ln p\big(\boldsymbol{x}^{(t)}\big)$ is omitted, as it does not depend on the parameters $\boldsymbol{\Theta}$. The remaining sums can be expressed as expectations $\mathbb{E}\big(\ln p\big(\boldsymbol{\Omega}\big)\big)$ and $\mathbb{E}\big(\ln p\big(\boldsymbol{\Omega}|\overset{*}{\boldsymbol{M}}\big)\big)$:

$$\boldsymbol{\Theta}^* = \arg\max_{\boldsymbol{\Theta}} \big\{ -\mathbb{E}(\ln p(\boldsymbol{\Omega})) + \mathbb{E}\big(\ln p(\boldsymbol{\Omega}|\overset{*}{\boldsymbol{M}})\big)\big\} \tag{4.13}$$

$$= \arg\max_{\boldsymbol{\Theta}} \big\{ \underbrace{\mathrm{H}(\boldsymbol{\Omega}) - \mathrm{H}(\boldsymbol{\Omega}|\overset{*}{\boldsymbol{M}})}_{\mathrm{I}(\boldsymbol{\Omega},\overset{*}{\boldsymbol{M}})} \big\}, \tag{4.14}$$

and further reformulated into the entropies $\mathrm{H}(\boldsymbol{\Omega})$ and $\mathrm{H}(\boldsymbol{\Omega}|\overset{*}{\boldsymbol{M}})$ by considering that the expectation of ln probabilities can be defined as entropies. The last expression leads to the information-theoretic criterion of the MI $\mathrm{I}(\boldsymbol{\Omega}, \overset{*}{\boldsymbol{M}})$ [133], where the MI $\mathrm{I}(\boldsymbol{\Omega}, \overset{*}{\boldsymbol{M}})$ between the two discrete label sequences $\boldsymbol{\Omega}$ and $\overset{*}{\boldsymbol{M}}$ is maximized to retrieve proper parameters $\boldsymbol{\Theta}$.

4.3.2 Training Procedure of NNVQs

The goal of the training procedure of NNVQs is to determine the proper parameters $\boldsymbol{\Theta}^*$ based on training criterion:

$$\boldsymbol{\Theta}^* = \arg\max_{\boldsymbol{\Theta}} \big\{\mathrm{H}(\boldsymbol{\Omega}) - \mathrm{H}(\boldsymbol{\Omega}|\boldsymbol{M})\big\}, \tag{4.15}$$

which defines a supervised training task of maximizing the MI $\mathrm{I}(\boldsymbol{\Omega}, \boldsymbol{M})$[133]. Notice that standard NNVQ outputs are considered, which are not satisfying the VQ quantization formulation in Equation (4.9). Moreover, the fixed target labels are set to $\boldsymbol{\Omega} = \hat{\boldsymbol{Y}}$, which originates from a given dataset $\mathcal{D}$, and the target label sequence $\hat{\boldsymbol{Y}}$ is typically obtained by the one-hot encoding strategy defined in Equation (2.3). As the target vectors $\hat{\boldsymbol{y}}^{(t)}$ are fixed, the entropy $\mathrm{H}(\boldsymbol{\Omega})$ remains constant during optimization, and the training criterion in Equation (4.15) is simplified into:

$$\boldsymbol{\Theta}^* = \arg\max_{\boldsymbol{\Theta}} \big\{ -\mathrm{H}(\hat{\boldsymbol{Y}}|\boldsymbol{M})\big\}. \tag{4.16}$$

The conditional entropy $\mathrm{H}(\hat{\boldsymbol{Y}}|\boldsymbol{M})$ is then reformulated in terms of probabilities:

$$\mathrm{H}(\hat{\boldsymbol{Y}}|\boldsymbol{M}) = -\sum_{t=1}^{T}\sum_{i=1}^{J^{[L]}}\sum_{j=1}^{\overset{*}{J}^{[L]}} p\big(\hat{y}_i^{(t)}, m_j^{(t)}\big) \ln p\big(\hat{y}_i^{(t)}|m_j^{(t)}\big) \tag{4.17}$$

$$= -\sum_{t=1}^{T}\sum_{i=1}^{J^{[L]}}\sum_{j=1}^{\overset{*}{J}^{[L]}} p\big(\hat{y}_i^{(t)}, m_j^{(t)}\big) \ln \frac{p\big(\hat{y}_i^{(t)}, m_j^{(t)}\big)}{\sum_{k=1}^{J^{[L]}} p\big(\hat{y}_k^{(t)}, m_j^{(t)}\big)}. \tag{4.18}$$

The joint probability distribution $p\big(\hat{\boldsymbol{Y}}, \boldsymbol{M}\big)$ can be represented by $p\big(\hat{\boldsymbol{Y}}, \overset{*}{\boldsymbol{M}}\big)$ if the standard quantization procedure from Equation (4.15) is assumed, where the firing neuron $\overset{*}{m}^{(t)} = \mathcal{H}_{\boldsymbol{\Theta}}(\boldsymbol{x}^{(t)})$ is one and the remaining output neurons are zero. Additionally,

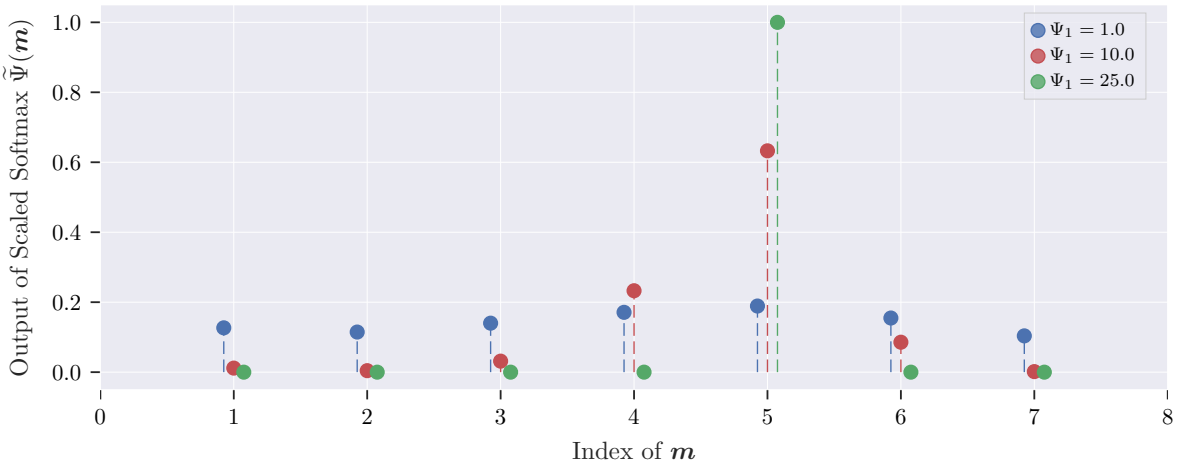


Figure 4.2: The result of a scaled softmax functions $\widetilde{\psi}(\boldsymbol{m})$ with the three different scaling values $\Psi_1 = 1.0$, $\Psi_2 = 10.0$, and $\Psi_3 = 25.0$ for given NNVQ output $\boldsymbol{m} = (0.5, 0.4, 0.6, 0.8, 0.9, 0.7, 0.3)^\top$. For $\Psi_1 = 1.0$, the scaled softmax corresponds to the standard softmax function $\psi(\boldsymbol{m}) = \widetilde{\psi}(\boldsymbol{m})$. In the case of the scaling factor $\Psi_3 = 25.0$, a close approximation of the max operator is obtained, resulting in the dominant firing neuron $\overset{*}{m} = 5$, or $\overset{*}{m}$ being one-hot encoded as $\overset{*}{\boldsymbol{m}} = (0, 0, 0, 0, 1, 0, 0)^\top$.

if the one-dimensional target labels $\hat{y}^{(t)}$ instead of the one-hot encoded vectors $\hat{\boldsymbol{y}}^{(t)}$ are considered, the joint probability distribution $p\big(\hat{\boldsymbol{Y}}, \overset{*}{\boldsymbol{M}}\big)$ can be efficiently derived:

$$\mathrm{H}\big(\hat{\boldsymbol{Y}} | \overset{*}{\boldsymbol{M}}\big) = -\sum_{t=1}^{T} p\big(\hat{y}^{(t)}, \overset{*}{m}^{(t)}\big) \ln \frac{p\big(\hat{y}^{(t)}, \overset{*}{m}^{(t)}\big)}{\sum_{\tau=1}^{T} p\big(\hat{y}^{(\tau)}, \overset{*}{m}^{(t)}\big)}, \tag{4.19}$$

Initially, Rigoll approximated the quantization procedure of VQs in his MLPs by setting the maximum output $m_j^{(t)}$ in $\boldsymbol{m}^{(t)} \in \mathbb{R}^{\overset{*}{J}^{[L]}}$ to one and the remaining outputs to zero [133], leading to a non-differentiable max operation. This operator was later replaced by Rigoll and Neukirchen, who introduced a scaled and differentiable softmax function $\widetilde{\psi}$ [134]:

$$\overset{*}{\boldsymbol{m}}^{(t)} \approx \widetilde{\psi}\big(\boldsymbol{m}^{(t)}\big)_j = \frac{\exp\left(\Psi m_j^{(t)}\right)}{\sum_{k=1}^{J} \exp\left(\Psi m_k^{(t)}\right)} \quad \forall j \in \{1, 2, \cdots, \overset{*}{J}^{[L]}\}. \tag{4.20}$$

with $\overset{*}{\boldsymbol{m}}^{(t)}$ refers to an approximation of $\overset{*}{m}^{(t)}$ being one-hot encoded (compare Equation (2.3)) and Ψ to a heuristic parameter for scaling the enhanced softmax $\widetilde{\psi}$. In Figure 4.2, the scaled softmax function $\widetilde{\psi}$ is depicted for multiple scaling factors Ψ, whereby $\Psi_1 = 1.0$ is equivalent to a standard softmax function, and $\Psi_3 = 25.0$ corresponds to an approximated max operation, and the NNVQs can be assumed to follow the quantization formulation of standard VQs. If higher or lower values are chosen for Ψ, NNVQs generate spikier or smoother outputs $\widetilde{\psi}\big(\boldsymbol{m}^{(t)}\big)$, leading to stronger and weaker approximations of $\overset{*}{m}^{(t)}$ being one-hot encoded.

4.3.3 The Extended Training Procedure for DNNQs

Although Rigoll and Neukirchen [134] described a procedure to obtain the gradients of $\mathrm{H}\big(\hat{\boldsymbol{Y}}|\overset{*}{\boldsymbol{M}}\big)$ w.r.t. the weights of the MLPs, their approach required iterating through the entire dataset $\mathcal{D}$ to gather the joint probability $p\big(\hat{\boldsymbol{Y}}, \overset{*}{\boldsymbol{M}}\big)$. Moreover, they did not employ standard cost functions, and their optimization relied solely on minimizing $\mathrm{H}\big(\hat{\boldsymbol{Y}}|\overset{*}{\boldsymbol{M}}\big)$.

Watzel *et al.* established an alternative training procedure in their work [13†], in which they applied the standard cross entropy (CE) cost function $C_{\mathrm{CE}}\big(\hat{\boldsymbol{Y}}_{\mathcal{B}}, \overset{*}{\boldsymbol{M}}_{\mathcal{B}}\big)$ (compare Section 2.2.6) for implicitly maximizing the MI $\mathrm{I}\big(\hat{\boldsymbol{Y}}, \overset{*}{\boldsymbol{M}}\big)$ instead of explicitly minimizing $\mathrm{H}\big(\hat{\boldsymbol{Y}}|\overset{*}{\boldsymbol{M}}\big)$ [160]. They obtained the CE cost function $C_{\mathrm{CE}}\big(\hat{\boldsymbol{Y}}_{\mathcal{B}}, \overset{*}{\boldsymbol{M}}_{\mathcal{B}}\big)$ for small mini-batches $\mathcal{B}$ of size B and prevented inefficient iterations over the entire dataset $\mathcal{D}$ which was essential for modifying the standard NNVQs into DNNQs by increasing the number of hidden layers.

However, utilizing CE cost functions $C_{\mathrm{CE}}\big(\hat{\boldsymbol{Y}}_{\mathcal{B}}, \overset{*}{\boldsymbol{M}}_{\mathcal{B}}\big)$ is more complex than it seems, as it requires equal dimensions $\overset{*}{J}^{[L]} \overset{!}{=} J^{[L]}$ between the DNNQ outputs $\boldsymbol{m}^{(t)} \in \mathbb{R}^{\overset{*}{J}^{[L]}}$ and the one-hot encoded target labels $\hat{\boldsymbol{y}}^{(t)} \in \mathbb{R}^{J^{[L]}}$. Since the dimensions of the DNNQ output $\boldsymbol{m}^{(t)}$ can be arbitrarily set, similar to standard VQs, a dimension mismatch $\overset{*}{J}^{[L]} \neq J^{[L]}$ would occur. Additionally, the MI $\mathrm{I}\big(\hat{\boldsymbol{Y}}, \overset{*}{\boldsymbol{M}}\big)$ heavily depends on the number $\overset{*}{J}^{[L]}$ of unique emitting labels corresponding to the output dimensions of the DNNQ. The modeling capability of DNNQs is limited if small output dimensions $\overset{*}{J}^{[L]}$ are chosen, as the theoretical limit of the MI depends on the number of unique emitting output labels $\overset{*}{\boldsymbol{m}}^{(t)}$.

The approach of Watzel *et al.* [13†] introduced a novel two-stage training scheme, which enabled them to employ CE cost functions $C_{\mathrm{CE}}\big(\hat{\boldsymbol{Y}}_{\mathcal{B}}, \overset{*}{\boldsymbol{M}}_{\mathcal{B}}\big)$ for any arbitrary DNNQ output dimension $\overset{*}{J}^{[L]}$. Therefore, the joint probability $p\big(\hat{\boldsymbol{Y}}, \overset{*}{\boldsymbol{M}}\big)$ is obtained and conditioned on $\overset{*}{\boldsymbol{M}}$ in the first stage:

$$\boldsymbol{P} \equiv \frac{p\big(\hat{\boldsymbol{Y}}_{\mathcal{B}}, \overset{*}{\boldsymbol{M}}_{\mathcal{B}}\big)}{p\big(\overset{*}{\boldsymbol{M}}_{\mathcal{B}}\big)} = p\big(\hat{\boldsymbol{Y}}_{\mathcal{B}}|\overset{*}{\boldsymbol{M}}_{\mathcal{B}}\big), \tag{4.21}$$

where $\boldsymbol{P} \in \mathbb{P}^{J^{[L]} \times \overset{*}{J}^{[L]}}$ represents the conditional probability distribution $p\big(\hat{\boldsymbol{Y}}_{\mathcal{B}}|\overset{*}{\boldsymbol{M}}_{\mathcal{B}}\big)$. The matrix $\boldsymbol{P}$ is determined by following the approach from Rigoll and Neukirchen [134]:

$$p_{j,i} = \frac{\varepsilon_{\mathrm{prob}} + \sum_{b=1}^{B} \delta\big(j, \hat{y}^{(b)}\big)\delta\big(i, \overset{*}{m}^{(b)}\big)}{\varepsilon_{\mathrm{prob}}\overset{*}{J}^{[L]} + \sum_{b=1}^{B} \delta\big(i, \overset{*}{m}^{(b)}\big)} \quad \begin{matrix} \forall j \in \{1,2,\cdots,J^{[L]}\} \\ \forall i \in \{1,2,\cdots,\overset{*}{J}^{[L]}\} \end{matrix}, \tag{4.22}$$

with $\varepsilon_{\mathrm{prob}}$ referring to a heuristic parameter for smoothing the values in $\boldsymbol{P}$. If a proper mini-batch size B is chosen, the matrix $\boldsymbol{P}$ can approximate the conditional probability distribution of the entire dataset $\mathcal{D}$ in each descent step, *i.e.*, $p\big(\hat{\boldsymbol{Y}}|\overset{*}{\boldsymbol{M}}\big) \approx \boldsymbol{P}$.

In the second stage, the DNNQ outputs $\overset{*}{\boldsymbol{m}}^{(t)}$ are mapped from $\mathbb{P}^{\overset{*}{J}^{[L]}}$ into the label space $\mathbb{B}^{J^{[L]}}$ by:

$$\widetilde{\overset{*}{\boldsymbol{m}}}^{(t)} = \boldsymbol{P}\overset{*}{\boldsymbol{m}}^{(t)} \tag{4.23}$$

with $\widetilde{\overset{*}{\boldsymbol{m}}}^{(t)} \in \mathbb{P}^{J^{[L]}}$ specifying the transformed DNNQ outputs $\overset{*}{\boldsymbol{m}}^{(t)}$. The linear transformation in Equation (4.23) can be interpreted as a marginal probability, representing $\overset{*}{\boldsymbol{m}}^{(t)} \in \mathbb{P}^{\overset{*}{J}^{[L]}}$ in the label space $\mathbb{B}^{J^{[L]}}$ [13†]. As a result, DNNQs with arbitrary output layer sizes are trainable, as their output can be mapped to any lower or higher dimensional label space $\mathbb{B}^{J^{[L]}}$.

4.3.4 The Discrete Hidden Markov Model Training

In standard HMMs, the likelihood $p\big(\boldsymbol{X}|\boldsymbol{S},\boldsymbol{\Phi}\big)$ for specific states s_k is modeled by PDFs, governed by the parameters $\boldsymbol{\Phi}$. Since the DNNQs generate discrete labels $\overset{*}{m}^{(t)}$, represented by their one-hot encoded outputs $\overset{*}{\boldsymbol{m}}^{(t)}$, the discrete likelihood $p\big(\overset{*}{\boldsymbol{M}}|\boldsymbol{S},\boldsymbol{\Phi}\big)$ in DHMMs is modeled by PMFs [22].

The DHMMs parameters are iteratively updated in the MLE procedure, which relies on a given Viterbi state sequence $\boldsymbol{S}^*$ (compare Section 2.1.4). In contrast to standard HMMs, the training of DHMMs is heavily simplified, as the auxiliary margins $\widetilde{\gamma}_k^{(t)}$ and $\widetilde{\xi}_{l,k}^{(t)}$ correspond to deterministic counts [109]:

$$\widetilde{\gamma}_k^{(t)} = \delta\big(s^{*(t)}, s_k^{(t)}\big) \quad \forall k \in \{1, 2, \cdots, K\}, \tag{4.24}$$

and:

$$\widetilde{\xi}_{l,k}^{(t)} = \widetilde{\gamma}_l^{(t)}\delta\big(s^{*(t+1)}, s_k^{(t)}\big) \quad \forall l,k \in \{1, 2, \cdots, K\}, \tag{4.25}$$

which are utilized to update the PMFs representing the likelihoods $p\big(\overset{*}{\boldsymbol{m}}^{(t)}|s_k^{(t)},\boldsymbol{\Phi}^{\{k\}}\big)$ for each state $s_k^{(t)}$. The PMFs can be gathered in a matrix $\boldsymbol{B} \in \mathbb{P}^{K\times\overset{*}{J}^{[L]}}$, where each row corresponds to the likelihood $p\big(\overset{*}{\boldsymbol{m}}^{(t)}|\boldsymbol{s}_k,\boldsymbol{\Phi}^{\{k\}}\big)$:

$$b_{\mathrm{MLE},k,j} = \frac{\sum_{t=1}^{T}\delta\big(s^{*(t)}, s_k^{(t)}\big)\delta\big(j, \overset{*}{m}_j^{(t)}\big)}{\sum_{t=1}^{T}\delta\big(s^{*(t)}, s_k^{(t)}\big)} \quad \begin{matrix}\forall k\in\{1,2,\cdots,K\}\\ \forall j\in\{1,2,\cdots,\overset{*}{J}^{[L]}\}\end{matrix}. \tag{4.26}$$

The transition probabilities are updated following the procedure of standard HMMs (compare Equation (2.6)):

$$a_{\mathrm{MLE},l,k} = \frac{\sum_{t=1}^{T-1}\delta\big(s^{*(t)}, s_l^{(t)}\big)\delta\big(s^{*(t+1)}, s_k^{(t)}\big)}{\sum_{t=1}^{T-1}\delta\big(s^{*(t)}, s_k^{(t)}\big)} \quad \forall l,k \in \{1, 2, \cdots, K\}. \tag{4.27}$$

Notice that the number of bins in the normalized histograms, *i.e.*, the likelihood $p\big(\overset{*}{\boldsymbol{m}}^{(t)}|\boldsymbol{s}_k,\boldsymbol{\Phi}^{\{k\}}\big)$, depends on the number of unique DNNQ labels $\overset{*}{m}^{(t)}$.

4.4 Experimental Setup

The evaluation of DNNQs was conducted by Watzel *et al.* [13†] on the publicly available dataset TED-LIUM-v2 [138]. The 207 h transcribed speech dataset belongs to the class

of small-sized datasets and is already pre-split into the train, dev, and test sets. The DNNQ models are implemented in the first version of the TensorFlow toolkit [3], which corresponds to one of the most popular toolkits in machine learning. It provides several pre-implemented functions in Python and C++ for building up DNNs, which can then be efficiently trained on graphics processing units (GPUs). The pre-processing procedure of the TED-LIUM-v2 dataset, the decoding procedure for the optimized DNNQ models, and the final evaluation are accomplished with the Kaldi toolkit [124]. The well-known toolkit gathers all the required modules for training SOTA approaches in ASR, executed by pre-defined scripts.

The experimental section is divided into four parts. First, the pre-processing procedure of the TED-LIUM-v2 is described, which was employed by Watzel *et al.* Then, the architecture of DNNQs is explained, including the applied regularization techniques. In the third part, the novel training scheme from Watzel *et al.* is introduced. Finally, the last part presents the decoding procedure.

4.4.1 Pre-Processing of the TED-LIUM-v2 Dataset

In the first step, Watzel *et al.* [13†] extracted 13-dimensional Mel-frequency cepstral coefficients (MFCCs) $\boldsymbol{c}^{(t)} \in \mathbb{R}^{13}$ for each 25 ms segment $\overline{\boldsymbol{x}}^{(t)} \in \mathbb{I}^{400}$, where the first coefficient was replaced by the log energy of this segment[1]. Then, they normalized the extracted MFCCs by applying the cepstral mean normalization (CMN) and appended the delta $\Delta\overline{\boldsymbol{c}}^{(t)}$ and delta-delta $\Delta^2\overline{\boldsymbol{c}}^{(t)}$ features to obtain the final inputs $\boldsymbol{x}^{(t)} \in \mathbb{R}^{39}$. Next, these inputs are employed in a multi-staged training procedure, resulting in the final TRI HMM-GMM model. For this, the transcript of the dataset $\mathcal{D}$ is sorted from the shortest to the longest utterances. The shortest 10 000 utterances with their corresponding MFCCs are utilized for training a standard MONO HMM-GMM model by MLE in the first stage. Then, MONO models carry out a forced alignment on the entire dataset $\mathcal{D}$ and serve as the initialization model of the subsequent TRI HMM-GMM model. Finally, the TRI HMM-GMM model is optimized by MLE on the entire dataset $\mathcal{D}$. Afterwards, the trained TRI HMM-GMM model is utilized to conduct another forced alignment on the whole dataset, providing the TRI states $s^{(t)} \in \mathbb{B}^{2024}$ or MONO states $s^{(t)} \in \mathbb{B}^{127}$ as target labels $\hat{y}^{(t)}$ for the corresponding inputs $\boldsymbol{x}^{(t)}$.

4.4.2 Architecture of DNNQs

In the second step, they built up the network architecture of their DNNQs, depicted in Figure 4.3. The DNNQs are defined as MLPs with four hidden layers. The input layer is composed of 39 input nodes set to the inputs $\boldsymbol{x}^{(t)} \in \mathbb{R}^{39}$. Then, four subsequent hidden layers $\boldsymbol{h}^{[1](t)}, \boldsymbol{h}^{[2](t)}, \boldsymbol{h}^{[3](t)} \in \mathbb{R}_{+}^{512}$ followed, whereby each hidden layer includes 512 neurons, and the activation functions are set to rectified linear units (ReLUs) (compare Equation (2.68)). Each hidden layer is concluded by a batch normalization (BN) layer [71] to normalize its outputs for the subsequent layers. Watzel *et al.* [13†] appended a dropout

[1] 400 samples correspond to 25 ms in a 16 kHz sampled signal.

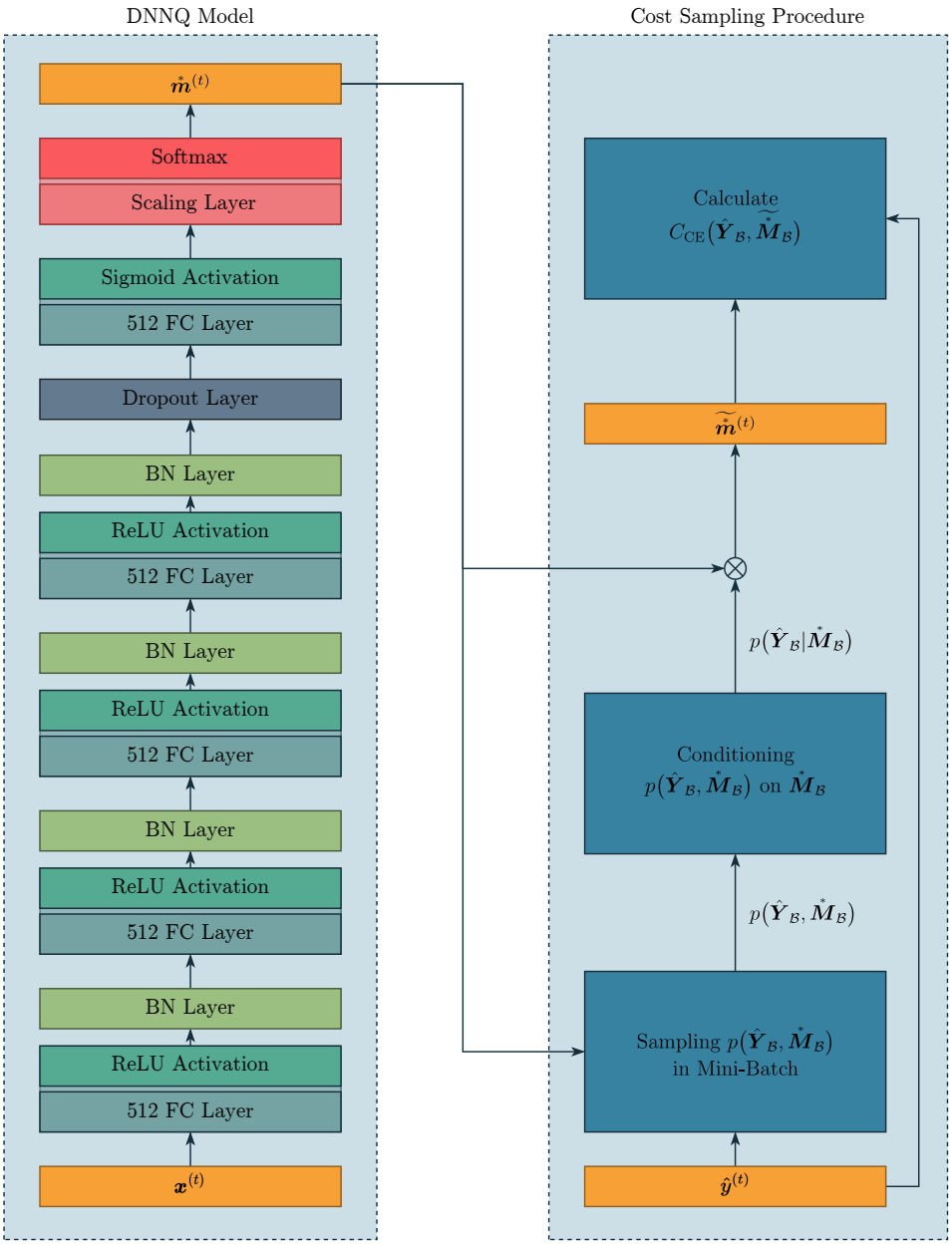


Figure 4.3: On the left the DNNQ architecture, and on the right, the novel sampling procedure is depicted. In order to transform the DNNQs outputs $\overset{*}{\boldsymbol{m}}^{(t)} \in \mathbb{P}^{\overset{*}{J}^{[L]}}$ into the label space $\mathbb{B}^{J^{[L]}}$, $p(\hat{\boldsymbol{Y}}_{\mathcal{B}}, \overset{*}{\boldsymbol{M}}_{\mathcal{B}})$ is sampled in the mini-batch $\mathcal{B}$, followed by conditioning on $\overset{*}{\boldsymbol{M}}_{\mathcal{B}}$ to obtain $p(\hat{\boldsymbol{Y}}_{\mathcal{B}}|\overset{*}{\boldsymbol{M}}_{\mathcal{B}})$. Then, $\overset{*}{\boldsymbol{m}}^{(t)}$ is transformed by $p(\hat{\boldsymbol{Y}}_{\mathcal{B}}|\overset{*}{\boldsymbol{M}}_{\mathcal{B}})$ into $\widetilde{\overset{*}{\boldsymbol{m}}}^{(t)}$, corresponding to a representation of $\overset{*}{\boldsymbol{m}}^{(t)}$ in the label space $\mathbb{B}^{J^{[L]}}$. Finally, the CE cost function $C_{\mathrm{CE}}(\hat{\boldsymbol{Y}}_{\mathcal{B}}, \overset{*}{\boldsymbol{M}}_{\mathcal{B}})$ can be calculated and backpropagated to optimize the DNNQs.

layer [66] with a keep probability $\widetilde{\rho} = 0.5$ after the fourth BN layer for regularization. A three-parted layer architecture finalizes the DNNQ models. First, MLPs generate the outputs $\boldsymbol{m}^{(t)} \in \mathbb{P}^{\overset{*}{J}^{[L]}}$ by applying sigmoid activation functions. Then, the outputs $\boldsymbol{m}^{(t)}$

are processed by scaling layers, where scaling factors of $\Psi = 15.0$ provide numerically stable training with satisfying model performance. In the end, softmax functions are employed on the scaled DNNQ outputs to obtain a quantization procedure comparable to standard VQs.

Besides regularizing the DNNQs by utilizing BN and dropout layers, the model weights are regularized by the weight decay term $\Pi = \sum_{l=1}^{L-1} \left\| \boldsymbol{W}^{[l]} \right\|$ (see also Equation (2.119)) scaled by $\gamma = 10^{-7}$.

4.4.3 Optimization of DNNQs

In the third step, the DNNQs are optimized by the Adam optimizer [76], whereby the frame-wise CE cost function $C_{\mathrm{CE}}\big(\hat{\boldsymbol{Y}}_{\mathcal{B}}, \overset{*}{\boldsymbol{M}}_{\mathcal{B}}\big)$ serves as the training criterion for the optimizer. The DNNQ outputs $\overset{*}{\boldsymbol{m}}^{(t)} \in \mathbb{P}^{\overset{*}{J}^{[L]}}$ are transformed into $\widetilde{\overset{*}{\boldsymbol{m}}}^{(t)} \in \mathbb{P}^{J^{[L]}}$ by the transformation matrix $\boldsymbol{P}$ (compare Equation (4.23)) in order to apply the CE cost function $C_{\mathrm{CE}}\big(\hat{\boldsymbol{Y}}_{\mathcal{B}}, \overset{*}{\boldsymbol{M}}_{\mathcal{B}}\big)$. The initial learning rate is set to $\eta = 0.01$. If Watzel *et al.* [13†] did not observe an MI improvement during validation for several subsequent epochs, they halved the current learning rate η. As $\boldsymbol{P}$ requires sufficient statistics to satisfy $\boldsymbol{P} \approx p\big(\hat{\boldsymbol{Y}} | \overset{*}{\boldsymbol{M}}\big)$, Watzel *et al.* [13†] employed mini-batches $\mathcal{B}$ of size $B = 15\,000$. Moreover, they smoothed the values in $\boldsymbol{P}$ by setting $\varepsilon_{\mathrm{prob}} = 0.01$, which remained unchanged in all subsequent experiments. Additionally, they suggested employing the label smoothing procedure (see also Equation (2.127)) by selecting the smoothing parameter $\iota = 0.1$, which led to a more steady and faster training.

4.4.4 Decoding Procedure for DNNQs

In the last step, the optimized DNNQs are evaluated on the dev and test set of the TED-LIUM-v2 dataset. The evaluation is split into several parts. The trained DNNQs are utilized in the first part, generating inferences about the train, dev, and test set. During inference, the scaled softmax function $\widetilde{\psi}$ is omitted, and the class label $\overset{*}{m}^{(t)}$ is obtained by the arg max function:

$$\overset{*}{m}^{(t)} = \underset{1 \leq j \leq \overset{*}{J}^{[L]}}{\arg\max}\, m_j^{(t)} \quad \forall t \in \{1, 2, \cdots, T\}. \tag{4.28}$$

Then, the DHMMs are optimized by utilizing the predicted $\overset{*}{m}^{(t)}$ and the given transcript of the train set in a Viterbi training (compare Section 4.3.4). Finally, the obtained DHMMs are decoded on the dev and test, where the provided 4-gram language model (LM) of the dataset is incorporated in the decoding process. The decoding procedure is executed as a beam search (see also Section 2.1.7) with a beam size $K_{\mathrm{best}} = 13$, returning the final WERs on the dev and test sets.

Table 4.1: The WERs (%) for different DNNQ output layer sizes $\overset{*}{J}^{[L]} \in \{400, 700, 1000, 1500\}$ trained on the subset $\widetilde{\mathcal{D}}$, which corresponds to the 20 000 shortest utterances in TED-LIUM-v2 [138].

Monophone States					
	DNNQ				GMM
$\overset{*}{J}^{[L]}$	400	700	**1000**	1500	-
dev	52.0	45.8	**44.5**	45.8	54.2
test	52.2	46.9	**45.1**	47.0	54.7

4.5 Evaluation

Watzel *et al.* [13†] conducted two ablation studies to determine the optimal DNNQ output layer size $\overset{*}{J}^{[L]}$ and the number N_{spl} of subsequent inputs $\boldsymbol{x}^{(t)}$, which were spliced together to increase the context. Based on the outcome of these studies, they retrained their DNNQs to derive the best models.

4.5.1 Ablation Study on DNNQ Output Layer Sizes

Their first ablation study aimed to determine the optimal output layer size of the DNNQs. Therefore, Watzel *et al.* [13†] trained the DNNQs on a subset $\widetilde{\mathcal{D}} \subseteq \mathcal{D}$ of the entire dataset $\mathcal{D}$, more precisely, on the 20 000 shortest utterances to speed up the time-consuming optimization process. The target labels for the subset $\widetilde{\mathcal{D}}$ are obtained following the multi-stage training procedure of TRI HMM-GMMs above, in which the obtained TRI states $\boldsymbol{s}^{(t)} \in \mathbb{B}^{2024}$ are mapped to the MONO states $\boldsymbol{s}^{(t)} \in \mathbb{B}^{127}$ and serve as one-hot encoded target labels $\hat{\boldsymbol{y}}^{(t)} \in \mathbb{B}^{127}$.

Then, Watzel *et al.* trained four DNNQ setups for 100 epochs with varying output layer sizes $\overset{*}{J}^{[L]} \in \{400, 700, 1000, 1500\}$. The learning rate η is halved if there is a performance stagnation on the dev set for six subsequent epochs. The DNNQs are decoded following the decoding procedure in Section 4.4.4.

Besides the DNNQs, a baseline model is established. Therefore, standard MONO HMM-GMMs are optimized by MLE on the subset $\widetilde{\mathcal{D}}$. The trained model is decoded by the standard decoding procedures of MONO HMM-GMMs [22], whereby the 4-gram LM provided by the dataset is included, and a beam size $K_{\text{best}} = 13$ is applied.

Even though the MONO DHMM-DNNQ models are restricted due to their quantization process, all four DNNQ setups from Watzel *et al.* [13†] outperformed the standard MONO HMM-GMM systems, whose results are summarized in Table 4.1. Since standard NNVQs [134] failed to surpass continuous models in the past, the authors suspected that DNNQs possessed a higher modeling capability. Compared to NNVQ, the DNNQs are built up by multiple hidden layers, increasing their generalization capabilities. Moreover, they assumed that the BN and dropout layers further improved generalization, as BN layers stabilize and accelerate the training process, and the dropout layers force the

Table 4.2: The reported WERs (%) for $N_{\text{clu}} = 1000$ and $N_{\text{spl}} \in \{0, 1, 2\}$, where the DNNQs are trained on the entire train set of TED-LIUM-v2 [138].

Monophone States			
	DNNQ		
N_{spl}	0	1	**2**
dev	43.7	37.5	**36.2**
test	45.1	38.9	**37.2**

DNNQs to make more robust predictions.

Furthermore, the results in Table 4.1 reveal that the optimal DNNQ output layer size is specified by $\overset{*}{J}^{[L]} = 1000$, which returns the overall best WERs of 44.5 % and 45.1 % on the dev and test set, respectively, and corresponds to a relative WER reduction of 17.9 % and 17.6 % compared to standard HMM-GMM systems. Although Watzel *et al.* [13†] achieved superior results by setting $\overset{*}{J}^{[L]} = 1000$, it remains unclear if more suitable values for $\overset{*}{J}^{[L]}$ exist, as the differences between $\overset{*}{J}^{[L]} = 700$ and $\overset{*}{J}^{[L]} = 1500$ are significant. Nevertheless, the first ablation study already demonstrates the potential of DHMM-DNNQ models.

4.5.2 Ablation Study on Splicing Adjacent Input Frames

In the second ablation study, Watzel *et al.* [13†] analyzed the splicing impact of varying numbers of subsequent inputs $\widetilde{\boldsymbol{x}}^{(t)}$, where they defined three splicing setups with $N_{\text{spl}} \in \{0, 1, 2\}$. The first splicing setup with $N_{\text{spl}} = 0$ corresponds to a standard single frame input $\widetilde{\boldsymbol{x}}^{(t)} = \boldsymbol{x}^{(t)}$ with $\widetilde{\boldsymbol{x}} \in \mathbb{R}^{39}$. In the second setup, the context of the current input $\boldsymbol{x}^{(t)}$ is extended by concatenating the $N_{\text{spl}} = 1$ adjacent frames $\widetilde{\boldsymbol{x}}^{(t)} = \boldsymbol{x}^{(t-1)} \oplus \boldsymbol{x}^{(t)} \oplus \boldsymbol{x}^{(t+1)}$ to obtain the new spliced input $\widetilde{\boldsymbol{x}}^{(t)} \in \mathbb{R}^{117}$. The context is further extended with $N_{\text{spl}} = 2$ in the last setup, resulting in the spliced input $\widetilde{\boldsymbol{x}}^{(t)} \in \mathbb{R}^{195}$. Watzel *et al.* [13†] optimized three DNNQs with spliced inputs $\widetilde{\boldsymbol{x}}^{(t)}$ in three training setups.

In contrast to the first study, the optimization is conducted on the entire dataset $\mathcal{D}$ and MONOs are chosen as target labels $\hat{\boldsymbol{y}}^{(t)}$. The dropout layers in the DNNQs are omitted, as dropout layers typically increase the training time, which the authors wanted to avoid. Overall, the DNNQs are trained for 50 epochs, and the learning rate η is halved if no improvement on the dev set is observable for ten subsequent epochs.

The results of the ablation study are summarized in Table 4.2 and demonstrate that the WERs of DNNQs can be further reduced if a larger context is chosen. The derived DNNQs heavily benefit from an increased context $N_{\text{spl}} = 2$, which leads to a WER of 36.2% and 37.2% on the dev and test set. This corresponds to a relative WER reduction of 17.2% on the dev and 15.5% on the test, respectively, compared to DNNQs trained on standard single-frame input $\boldsymbol{x}^{(t)}$.

Table 4.3: The final WERs (%) for $N_{\text{spl}} \in \{0, 1, 2\}$ and $N_{\text{clu}} = 1000$, where the entire train set of TED-LIUM-v2 is utilized. [138].

Triphone States						
	DNNQ			GMM		
N_{spl}	0	1	**2**	0	1	2
dev	30.5	**27.1**	**27.1**	28.2	35.2	45.9
test	31.7	28.1	**27.1**	27.7	36.3	48.0

4.5.3 Final Results of DNNQ Architectures

Based on conducted ablations studies, Watzel *et al.* [13†] expected superior DNNQ models for spliced inputs $\widetilde{\boldsymbol{x}}^{(t)} \in \mathbb{R}^{195}$ with $N_{\text{spl}} = 2$ and output size of $\overset{*}{J}^{[L]} = 1000$. Therefore, they trained three DNNQs with $N_{\text{spl}} \in \{0, 1, 2\}$ on the entire dataset $\mathcal{D}$ for 50 epochs, halved the η for performance stagnation of 10 subsequent epochs on the dev set, and selected TRIs states as target labels $\hat{\boldsymbol{y}}^{(t)} \in \mathbb{B}^{2024}$. The results of the TRI DHMM-DNNQ models are compared to standard TRI HMM-GMMs with spliced inputs $\widetilde{\boldsymbol{x}}^{(t)}$ with $N_{\text{spl}} \in \{0, 1, 2\}$ to ensure a fair comparison.

In Table 4.3, the results of all model structures are depicted. Interestingly, Watzel *et al.* [13†] reported that for $N_{\text{spl}} = 0$, the DHMM-DNNQ models could not surpass the standard HMM-GMMs. However, as soon the context is increased, *i.e.*, for $N_{\text{spl}} \in \{1, 2\}$, the DHMM-DNNQ models exceeded the HMM-GMMs, whose performance even deteriorates. Similar performance declines of HMM-GMMs applying higher dimensional inputs $\widetilde{\boldsymbol{x}}^{(t)}$ were already reported in another work from Rath *et al.* [131], who suspected that the massive increase of GMM parameters caused by $\widetilde{\boldsymbol{x}}^{(t)}$ led to the declining performance of HMM-GMMs.

The DHMM-DNNQ models achieve the overall best performance by setting $N_{\text{spl}} = 2$. Watzel *et al.* [13†] reported a WER of 27.1% on the dev and 27.1% on the test, corresponding to a relative WER decline of 3.9 % and 2.2 %, respectively. However, the authors did not observe similar performance improvements between $N_{\text{spl}} \in \{0, 1, 2\}$ in the ablation study from Section 4.5.3. They argued that the employed TRI target labels $\hat{\boldsymbol{y}}^{(t)}$, which were applied in the final evaluation, already included valuable context-dependent information, leading to minor improvements between $N_{\text{spl}} \in \{0, 1, 2\}$.

4.6 Conclusion

In their work, Watzel *et al.* demonstrated that discrete AMs have the potential to challenge continuous AMs. Their sampling approach provided an efficient alternative to train standard NNVQs with arbitrary output layer sizes based on given target labels. As a result, standard NNVQs could be deepened by increasing the number of hidden layers, leading to DNNQ architectures. Besides the standard weight decay for regularization, they employed SOTA regularizing layers, such as BN or dropout layers, to improve the

generalization of DNNQs models. Moreover, they carried out several ablation studies to obtain the optimal DNNQ output layer size or to examine the impact of a larger context where multiple adjacent input frames were spliced together. The final evaluation demonstrated how DHMM-DNNQ models surpass standard HMM-GMMs, even though their quantization procedure corresponded to losing valuable information.

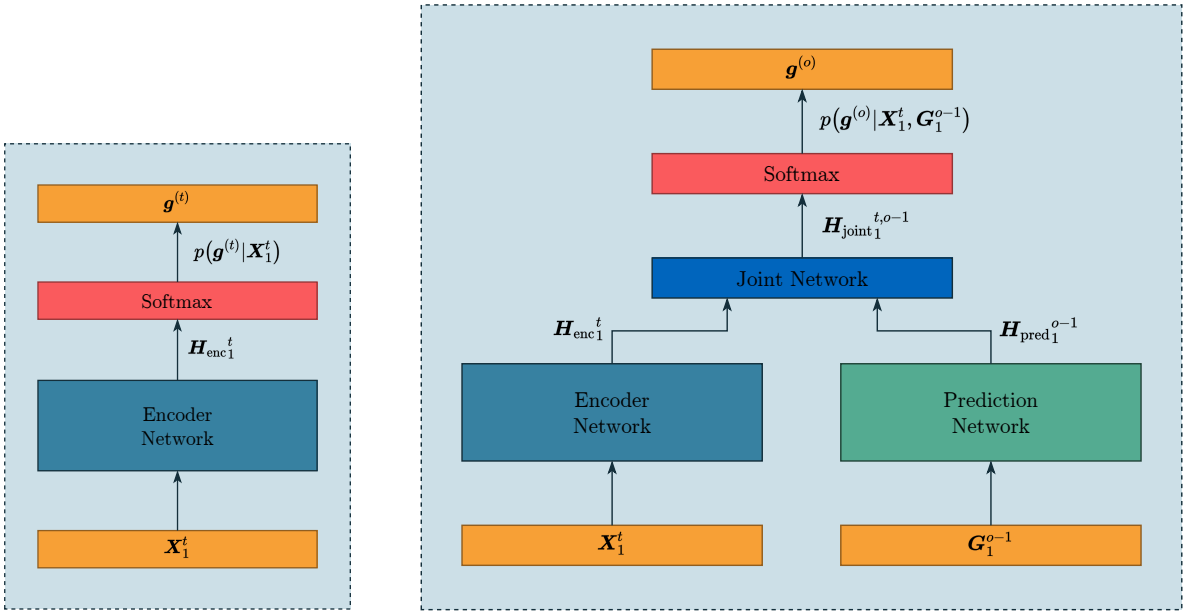
5

Forward-Backward Learning for Attentional Models

This chapter discusses the forward-backward learning strategies from Watzel *et al.* [10†, 12†], who examined the impact of time-reversed models on standard attention-based encoder-decoder (AED) and self-attention-based encoder-decoder (SAED) architectures in the optimization phases and how reversed structures could be utilized as auxiliary functions. In the first section, the most popular end-to-end (E2E) automatic speech recognition (ASR) architectures in the form of connectionist temporal classification (CTC), recurrent neural network transducer (RNNT), and AED models are introduced. In the second section, related approaches to the work of Watzel *et al.* [10†, 12†] are presented. Then, the subsequent section establishes the theory of AED and SAED architectures, followed by a theoretical integration of time-reversed model structures. The concept of efficient regularization procedures for even and odd lengths of target grapheme sequences is introduced, and how arbitrary distance metrics are applicable. In the fourth section, the experimental setups of AED and SAED models are described, including the pre-processing of the datasets, the structure of those models, the varying training procedures for examining the impact of reversed model structures on the standard architectures, and the decoding procedures of the regularized AED and SAED models. In the subsequent evaluation section, the outcome of the established training procedures is discussed. The chapter concludes by reviewing the established approaches from Watzel *et al.* [10†, 12†].

5.1 Introduction

Recently, hybrid ASR models have been replaced by E2E ASR models to simplify the modeling procedure of ASR architectures. In the past, building up standard hybrid ASR approaches required specific expert knowledge since hybrid approaches were composed of separate components, roughly dividable into acoustic model (AM), pronunciation model (PM), and language model (LM) modules. Typically, conditional independence assumptions are applied in order to optimize each module independently. However, these



(a) CTC **(b)** RNNT

Figure 5.1: On the left, the CTC approach with single encoder RNNs is presented, where each $\boldsymbol{g}^{(t)}$ is predicted without incorporating adjacent graphemes $\boldsymbol{g}^{(t-1)}$ or $\boldsymbol{g}^{(t+1)}$. On the right, RNNTs architecture with its encoder, prediction, and joint networks is depicted. In contrast to CTC, the grapheme prediction $\boldsymbol{g}^{(o)}$ is conditionally dependent on the previously predicted grapheme sequence $\boldsymbol{G}_1^{o-1}$.

assumptions heavily simplify the general theory of ASR models (see also Equation (3.1)) and primarily enable independent optimization of ASR components in practice. However, this leads to the drawback of restricting the flow of information between components in practical approaches, and valuable information is not transferred from one component to another.

In E2E models, all these modules are combined into a single deep neural network (DNN), and neither expert knowledge of specific ASR components nor a solution for the formerly mentioned problems of hybrid ASR models is required. Therefore, E2E ASR models have become very popular over the last few years [10, 26, 30, 33, 34, 44, 56, 57, 59, 95, 149, 157, 165] and surpassed established hybrid ASR approaches for sufficiently large training dataset sizes [30]. In general, E2E approaches can be divided into three categories [88]: models applying CTC [55, 56], RNNT models [53, 57], and models relying on either recurrent neural network (RNN)-based attentional [9, 10, 32–34, 54, 152] or self-attentional transformer [44, 59, 121, 149, 159] encoder-decoder structures.

In the past, standard RNN approaches were typically trained by objective functions, which treated each grapheme predictions $g^{(t)}$ based on an input sequence $\boldsymbol{X}$ as independent predictions [55]. Therefore, pre-segmentation procedures are mandatory to generate frame-wise samples $(\boldsymbol{x}^{(t)}, \hat{\boldsymbol{y}}^{(t)} = \hat{\boldsymbol{g}}^{(t)})$, commonly generated by a forced alignment procedure of pre-trained hidden Markov model (HMM)-Gaussian mixture models (GMMs). The final predicted grapheme sequence $\boldsymbol{G}$ is then obtained through a post-processing

procedure.

In 2006, Graves [55] introduced the alternative CTC objective function, where *encoder* RNNs learn the absolute probabilities $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$ (compare to Figure 5.1a). Instead of maximizing each relative prediction $p\big(\boldsymbol{g}^{(t)}|\boldsymbol{x}^{(t)}\big)$ independently, the correct predictions of the entire transcript are maximized. Therefore, a unique *blank* grapheme output unit is added to the existing output layer, allowing the encoder RNNs to generate null predictions $\varnothing$. Graves interpreted these extended encoder RNN outputs $\boldsymbol{u}^{(t)} \in \{\boldsymbol{g}^{(t)} \cup \varnothing\}$ as alignments [55] and compared them to standard HMMs transition probabilities. Here, the activation of a single grapheme output $g_j^{(t)}$ defines the state transition into another state, and activations for the blank grapheme $\varnothing$ specify the state staying condition [56]. The absolute probability $p\big(\boldsymbol{G}|\boldsymbol{U},\boldsymbol{X}\big)$ is then found by summing over all possible alignments $\boldsymbol{U}$ (compare Section 3.1.2), where the length of the alignments is limited to the input length T of $\boldsymbol{X}$. Although the summation seems infeasible at first sight, Graves solved this issue by assuming conditional independent graphemes $\boldsymbol{g}^{(t)}$, which enabled him to apply the forward-back algorithm (see also Section 2.1.3) for calculating $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$. Finally, the derivatives of $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$ w.r.t. to the encoder RNNs output layer $\boldsymbol{y}^{(t)}$ are determined and backpropagated through the network.

A drawback of encoder RNNs trained by the CTC criterion is the strong conditional independence assumption of each grapheme $\boldsymbol{g}^{(t)}$, which harms the performance of RNNs by predicting adjacent grapheme combinations in $\boldsymbol{G}$ without considering neighboring grapheme context $\boldsymbol{g}^{(t-1)}$ or $\boldsymbol{g}^{(t+1)}$. Such encoder RNNs require strong LMs to incorporate the grapheme context for correcting wrong grapheme predictions. Graves suggested an enhanced version of CTC [53], where he extended the standard encoder RNNs by additional *prediction* RNNs and *joint* multilayer perceptron (MLP) [55] (compare to Figure 5.1b). The encoder RNNs process the input sequence $\boldsymbol{X}_1^t$ up to the input time step t and the prediction RNNs are fed by the predicted grapheme sequence $\boldsymbol{G}_1^o = [\boldsymbol{g}^{(1)}, \cdots, \boldsymbol{g}^{(o)}]$ up to the current output time step o. The hidden representations of the encoder and prediction RNNs are then fused and further processed by the joint MLP. The normalized RNNT outputs, representing samples from the posterior distribution $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$, are obtained by applying the softmax function after the joint MLPs. As a result, the conditional independence assumption of $\boldsymbol{g}^{(o)}$ does not longer hold, as a conditional dependence of the prediction RNNs and the joint MLP exist. In addition, RNNT models are no longer limited to the output sequence length $O \leq T$ and can produce arbitrary-length O output sequences.

Alternative encoder-decoder E2E architectures were initially established in the domain of neural machine translation (NMT) [9, 32]. Therefore, Cho *et al.* [32] utilized *encoder* RNNs to encode the input sequence $\boldsymbol{X}$ into hidden representation sequences $\boldsymbol{H}_{\text{enc}}$. The *decoder* RNNs generate the posterior distribution $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$ based on the hidden representations $\boldsymbol{H}_{\text{enc}}$ of the encoder, its previous hidden representation $\boldsymbol{h}_{\text{dec}}^{(o-1)}$ and the predicted grapheme output $\boldsymbol{g}^{(o-1)}$. Later, Chorowski *et al.* [33] got inspired by a work from Graves [54] and introduced the AED models, where an attention mechanism weighted the relevance between the hidden encoder representations $\boldsymbol{H}_{\text{enc}}^{(t)}$ and the hidden decoder representations $\boldsymbol{H}_{\text{dec}}^{(o)}$. Moreover, they integrated penalty terms in the attention

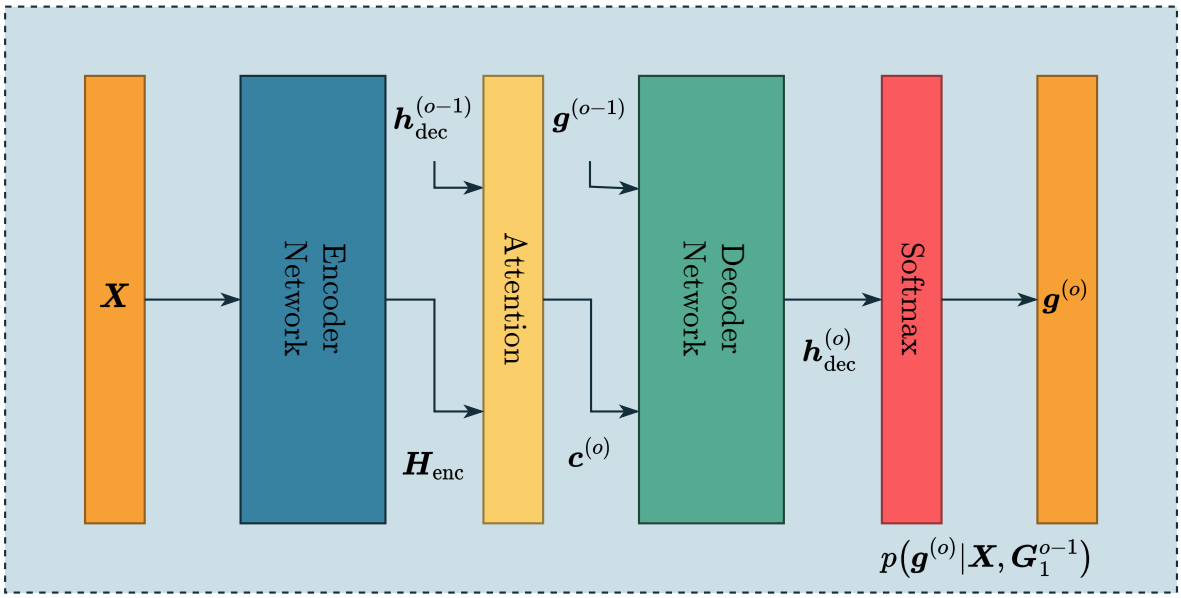


Figure 5.2: The AED architecture consists of encoder and decoder networks. The attention mechanism provides relevance of $\boldsymbol{H}_{\text{enc}}$ in the decoder networks. In comparison to RNNTs, the entire input sequence $\boldsymbol{X}$ has to be processed by the encoder before emitting grapheme predictions $\boldsymbol{g}^{(o)}$.

module forcing the model to prefer monotonic alignments. In a subsequent work by Chorowski *et al.* [34], the attention mechanism was further improved by extending the attentional functions with location-aware modules, where attention weights of previous time steps were considered.

The AED E2E models achieved state-of-the-art (SOTA) results in multiple large-vocabulary speech recognition (LVSR) setups [10, 26]. However, these architectures also have a major drawback: the application of RNNs. Although superior versions of RNNs exist, as the well-known long short-term memory networks (LSTMs) [67], they are limited in preserving information for long input sequence $\boldsymbol{X}$ [91]. Compared to standard MLPs, they require more resources for computation, and the training time is increased by the recurrent connections.

In 2017, Vaswani *et al.* [159] established a non-recurrent replacement for the LSTMs, built solely by standard MLPs, and global self-attention (SA) modules. In contrast to standard LSTMs, the SAED architectures are not limited to the length of input sequences since these networks can always attend to the entire input sequence [88]. In the following years, the SAED models, also known as transformer models, were successfully transferred into ASR [44, 149], further improved by deeper model versions [121], or were also forced to consider local context [59, 11†]. Recently, multiple studies also verified the superiority of SAED models to LSTMs [75, 90, 181].

Even though AED or SAED architectures achieve SOTA results on several datasets, Watzel *et al.* demonstrated in their approaches [10†, 12†] that the performance of these architectures could be enhanced if specific model extensions are deployed. For AED

models, Watzel *et al.* [12†] added a second decoder network in the training phase, which was optimized on time-reversed target labels. The decoder serves as a regularizer and is discarded in the subsequent decoding phase. Watzel *et al.* also established a similar approach for SAED models, where an entire SAED model is trained on time-reversed target labels for regularizing standard SAEDs models. Additionally, the impact of various distance functions and the utilization of byte pair encoding (BPE) and character (CHAR) target labels in the regularization procedure were examined. The contributions of both works [10†, 12†] can be summarized as follows:

- An efficient incorporation of the second decoder in AED and the second transformer network in SAED in the standard model setups.
- An examination of BPE and CHAR target labels on the model performance in AED and SAED models.
- The integration of the novel regularization procedures into standard AED and SAED models if BPE sequences of odd lengths occur in the training phase.
- An analysis of the Euclidean and Cosine metrics in regularizing SAED architectures.
- The advantage of integrating additional time-reversed network structures in AED and SAED models.

5.2 Related Work

The first analysis of bidirectional RNN architectures was conducted by Liu *et al.* [92, 93] for general E2E modeling in 2016. They examined the impact of incorporating additional right-to-left (R2L) RNNs into standard left-to-right (L2R) RNNs, where the R2L RNNs were trained on time-reversed target label sequences and the L2R RNNs on regular target labels. Their motivation was driven by the fact that RNNs suffered in maintaining correct predictions of large input sequences, which was discovered in a study in 2018 [91]. Liu *et al.* [92, 93] suggested independently optimizing additional R2L RNNs from the L2R RNNs and utilizing the L2R and R2L RNNs in a joint search approximation. Consequently, they merged the k-best hypothesis of the L2R and R2L RNNs into a joint search space and executed an exhaustively rescoring of all variations in the search space. Their novel approach returned good results compared with standard decoded L2R RNNs. However, both RNNs were optimized independently and could not share information.

A few years later, Mimura *et al.* [105] transferred the idea of R2L models into the domain of ASR and applied such a concept to AED models. Instead of training two entire L2R and R2L AED models, they proposed using single encoder networks whose hidden representations were shared between individual L2R and R2L decoder networks. Their training phase was defined as multitask learning, in which the shared encoder and the corresponding L2R and R2L decoders were jointly optimized by a weighted cost function. For the final decoding procedure, a three-pass decoding algorithm was established. The k-best hypotheses generated by the L2R and R2L decoders were obtained in the first

and second passes. In the final pass, these hypotheses were merged into a combined hypothesis by considering the attention weights of the L2R and R2L attention modules, followed by a rescoring procedure. In the subsequent evaluation, Mimura *et al.* [105] demonstrated that their novel AED model outperformed standard L2R AED models, even though they solely deployed a simple joint training procedure.

In the same year, Zhang *et al.* [185] addressed the joint training procedure of L2R and R2L models in NMT, as all former approaches mainly examined the effect of R2L models in the decoding phase. They established and proposed a novel training process in which each L2R and R2L model was initially pre-trained independently. Next, they divided the subsequent training iterations into two phases. In the first phase, L2R and R2L models generated predictions for the corresponding mini-batch and determined the Kullback-Leibler divergence (KLD) regularization terms for the L2R model. The regularization terms were added to the standard cost functions, the resulting error was then backpropagated through the L2R networks, and their weights were updated by stochastic gradient descent (SGD). The parameters of the R2L models were frozen, as they served as auxiliary networks. The training iteration was finalized in the second phase by repeating the same procedure for the R2L models, whereby the L2R models corresponded to the auxiliary networks. The iterative training procedure was repeated until convergence was observed. The regularized models indicated better performance in the evaluation. In contrast to prior works, Zhang *et al.* [185] did not apply shared model structures.

In the domain of text-to-speech, Zheng *et al.* [186] established a similar joint training approach to the former procedure [185]. However, they exclusively examined the mutual impact of regularizing bidirectional decoder RNNs. They also pre-trained the L2R and R2L decoders and utilized shared encoder RNNs [105]. Then, they individually optimized the L2R or R2L decoders in each training iteration [185] and applied the opposite decoder network as auxiliary models. In contrast to Zhang *et al.* [185], they further extended the loss by an additional regularizing term, which minimized the L2 distances between the hidden representations of the L2R or R2L decoders. Their evaluation revealed superior results compared with models relying on standard L2R decoders.

Recently, Chen *et al.* [28] proposed bidirectional decoders for SAED transformer models in ASR, and they followed the approach from Mimura *et al.* [105] by utilizing a single shared encoder. However, instead of integrating additional R2L decoders into the existing L2R decoders, they solely reversed the query, key, and value matrices of the SA modules in individual model branches and shared the weights between the L2R and R2L decoders. In the evaluation, the novel SAED model returned lower word error rates (WERs) compared with standard SAED models, and they were able to demonstrate that these models could effectively employ the time-reversed information without increasing the number of model parameters. Interestingly, Chen *et al.* did not integrate a separate regularizer term into the loss function. Therefore, it remained unclear whether the model could be further improved by setting alternative training objectives.

5.3 Proposed Method

The first part independently introduces the theory of AED and SAED architectures. Since Watzel *et al.* established methods for AED [12†] and SAED [10†] architectures, which share similarities, most of their methods are jointly described, and the corresponding model architectures are referenced. Afterward, the integration of R2L structures into existing L2R architectures is explained, whereby the implementation differences between AED and SAED models are discussed. Then, the challenge of regularizing equal and unequal sequence lengths is described. For SAED models, the analysis is extended by examining the impact of various distance functions.

5.3.1 Theory of AED Models

The theory of AED models is established by relying on the approach by Chorowski *et al.* [34], corresponding to the most popular AED model structure nowadays. Standard AED models consist of three major model components: the *encoder* networks $\mathcal{H}_{\text{enc}}(\,\cdot\,)$, the *attention* mechanisms $\mathcal{A}_{\text{aed}}(\,\cdot\,)$, and the *decoder* networks $\mathcal{H}_{\text{dec}}(\,\cdot\,)$ (see also Figure 5.2). Generally, the input sequences $\boldsymbol{X}$ are transformed by RNNs into the hidden encoder representations $\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}]}$:

$$\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}]} = \big[\boldsymbol{h}_{\text{enc}}^{[l_{\text{enc}}](1)}, \ldots, \boldsymbol{h}_{\text{enc}}^{[l_{\text{enc}}](t)}, \ldots, \boldsymbol{h}_{\text{enc}}^{[l_{\text{enc}}](T)}\big] \quad \forall l_{\text{enc}} \in \{1, 2, \cdots, L_{\text{enc}}\}, \tag{5.1}$$

which encode crucial aspects of the input sequences $\boldsymbol{X}$. The hidden representations $\boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]}$ of the last encoder layer L_{enc} conclude the encoder network $\mathcal{H}_{\text{enc}}$:

$$\boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]} = \mathcal{H}_{\text{enc}}\big(\boldsymbol{X}\big). \tag{5.2}$$

Afterward, the encoded sequence $\boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]}$ and $\boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o-1)}$ are processed by an attention module $\mathcal{A}_{\text{aed}}$, whereby $\boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o-1)} \in \mathbb{R}^{J^{[L_{\text{dec}}]}}$ corresponds to the hidden representations of the second last hidden layer L_{dec} of the RNN decoder at the previous time step $o-1$:

$$\boldsymbol{\alpha}_{\text{att}}^{(o)} = \mathcal{A}_{\text{aed}}\big(\boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]}, \boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o-1)}\big) \quad \forall o \in \{1, 2, \cdots, O\}, \tag{5.3}$$

with $\boldsymbol{\alpha}_{\text{att}}^{(o)} \in \mathbb{P}^T$ referring to attention weights for weighting the hidden representations $\boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]}$.

The subsequent attention modules $\mathcal{A}_{\text{aed}}(\,\cdot\,)$ are commonly defined by scoring networks $\mathcal{S}_{\text{aed}}(\cdot)$:

$$\boldsymbol{e}^{(o)} = \mathcal{S}_{\text{aed}}\big(\boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]}, \boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o-1)}\big) \quad \forall o \in \{1, 2, \cdots, O\}, \tag{5.4}$$

whose scoring values $\boldsymbol{e}^{(o)} \in \mathbb{R}^T$ are normalized along the input sequence axis T by the softmax function $\psi(\,\cdot\,)$ to generate the attention weights $\boldsymbol{\alpha}_{\text{att}}^{(o)}$:

$$\boldsymbol{\alpha}_{\text{att}}^{(o)} = \psi\big(\boldsymbol{e}^{(o)}\big) = \frac{\exp(\boldsymbol{e}^{(o)})}{\sum_{\tau=1}^{T} \exp(e_{\tau}^{(o)})} \quad \forall o \in \{1, 2, \cdots, O\}. \tag{5.5}$$

Notice that the attention weights $\boldsymbol{\alpha}_{\text{att}}^{(o)}$ can also be interpreted as alignments [9], as each hidden representation $\boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o-1)}$ of the decoder RNNs receives a score for each latent layer output $\boldsymbol{h}_{\text{enc}}^{[L_{\text{enc}}](t)}$ of the encoder.

The attention networks $\mathcal{A}_{\text{aed}}$ and their scoring functions $\mathcal{S}_{\text{aed}}$ can be arbitrarily defined, depending on which task has to be solved. For ASR models, Chorowski *et al.* [34] enhanced the standard content-based attention module in Equation (5.3) to $\mathcal{A}_{\text{aed}}\big(\boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]}, \boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o-1)}, \boldsymbol{\alpha}_{\text{att}}^{(o-1)}\big)$, which includes the attention weights of the previous output time step $o-1$. As a result, the scoring values $\boldsymbol{e}^{(o)}$ in Equation (5.4) are modified, leading to a content-based and location-aware module:

$$e_t^{(o)} = \mathcal{S}_{\text{aed}}\big(\boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]}, \boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o-1)}, \boldsymbol{\alpha}_{\text{att}}^{(o-1)}\big) \tag{5.6}$$

$$= \boldsymbol{w}_{\text{aed}}^{\top} \tanh(\boldsymbol{W}_{\text{aed}} \boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o-1)} + \boldsymbol{V}_{\text{aed}} \boldsymbol{h}_{\text{enc}}^{(t)} + \boldsymbol{U}_{\text{aed}} \boldsymbol{f}^{(o)} + \boldsymbol{b}_{\text{aed}}) \quad \forall t \in \{1, 2, \cdots, T\} \tag{5.7}$$

whereby $\boldsymbol{W}_{\text{aed}} \in \mathbb{R}^{J^{[L_{\text{dec}}]} \times J^{[L_{\text{dec}}]}}$, $\boldsymbol{U}_{\text{aed}} \in \mathbb{R}^{J^{[L_{\text{dec}}]} \times Z}$, and $\boldsymbol{V}_{\text{aed}} \in \mathbb{R}^{J^{[L_{\text{dec}}]} \times J^{[L_{\text{enc}}]}}$ are weight matrices, and $\boldsymbol{w}_{\text{aed}}, \boldsymbol{b}_{\text{aed}} \in \mathbb{R}^{J^{[L_{\text{dec}}]}}$ are weight vectors. The vectors $\boldsymbol{f}^{(o)} \in \mathbb{R}^{Z}$ are obtained by the location-aware module, represented by 1D convolutional neural networks (CNNs):

$$\boldsymbol{f}^{(o)} = \boldsymbol{F} \circledast \boldsymbol{\alpha}_{\text{att}}^{(o-1)}. \tag{5.8}$$

where the operator $\circledast$ defines convolution, $\boldsymbol{F} \in \mathbb{R}^{Z \times W}$ refers to the convolutional kernel of width W, and Z defines the number of output channels. Notice that the parameter H is dropped as $H = 1$ for 1D CNNs.

In the next step, the alignments $\boldsymbol{\alpha}_{\text{att}}^{(o)}$ are utilized for weighting the relevance of each hidden encoder representation $\boldsymbol{h}_{\text{enc}}^{[L_{\text{enc}}](t)}$ and aggregated into the content vector $\boldsymbol{c}^{(o)} \in \mathbb{R}^{J^{[L_{\text{enc}}]}}$ for the current output time step o:

$$\boldsymbol{c}^{(o)} = \sum_{t=1}^{T} \alpha_t^{(o)} \boldsymbol{h}_{\text{enc}}^{[L_{\text{enc}}](t)} \quad \forall o \in \{1, 2, \cdots, O\}. \tag{5.9}$$

Based on the content vector $\boldsymbol{c}^{(o)}$, the previous grapheme output $\boldsymbol{g}^{(o-1)}$, and hidden layer output $\boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o-1)}$ of the second last decoder layer, RNNs are generating the last latent layer output $\boldsymbol{H}_{\text{dec}}^{[L_{\text{dec}}]}$ of the decoder networks:

$$\boldsymbol{H}_{\text{dec}}^{[L_{\text{dec}}]} = \mathcal{H}_{\text{dec}}\big(\boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o-1)}, \boldsymbol{c}^{(o)}, \boldsymbol{g}^{(o-1)}\big) \quad \forall o \in \{1, 2, \cdots, O\}, \tag{5.10}$$

with:

$$\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}]} = \big[\boldsymbol{h}_{\text{dec}}^{[l_{\text{dec}}](1)}, \ldots, \boldsymbol{h}_{\text{dec}}^{[l_{\text{dec}}](o)}, \ldots, \boldsymbol{h}_{\text{dec}}^{[l_{\text{dec}}](O)}\big] \quad \forall l_{\text{dec}} \in \{1, 2, \cdots, L_{\text{dec}}\}. \tag{5.11}$$

The final output grapheme sequence $\boldsymbol{G}$ is obtained by applying a single-layer perceptron with linear activation functions (compare Section 2.2.1):

$$\boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}+1](o)} = \boldsymbol{W}^{[L_{\text{dec}}+1]}\boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}](o)} + \boldsymbol{b}^{[L_{\text{dec}}+1]} \quad \forall o \in \{1, 2, \cdots, O\}, \tag{5.12}$$

whereby $\boldsymbol{W}^{[L_{\text{dec}}+1]} \in \mathbb{R}^{J \times J^{[L_{\text{dec}}]}}$ corresponds to the weight matrix and $\boldsymbol{b}^{[L_{\text{dec}}+1]} \in \mathbb{R}^{J}$ to the bias of the J-dimensional output perceptron. In order to retrieve the final grapheme distribution $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$, the softmax function $\psi\big(\,\cdot\,\big)$ is applied:

$$p\big(\boldsymbol{g}^{(o)}|\boldsymbol{X}, \boldsymbol{G}_1^o\big) = \psi\big(\boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}+1](o)}\big) \quad \forall o \in \{1, 2, \cdots, O\}. \tag{5.13}$$

5.3.2 Theory of SAED Models

Initially, SAED models were introduced by Vaswani *et al.* [159] in 2017. A year later, Dong *et al.* [44] successfully transferred them to the domain of ASR. The subsequent theory is thus established following their approach.

The SAED models are related to AED models, as they also consist of three fundamental components: the encoder networks $\mathcal{H}_{\text{enc}}$, the attention modules $\mathcal{A}_{\text{saed}}$, and the decoder networks $\mathcal{H}_{\text{dec}}$. In contrast to AED models, SAED models are not integrating recurrent or convolutional network structures into their architecture and are solely constructed out of MLPs. As a result, SAED layers only depend on the outputs of the previous layers and avoid the computationally expensive calculation costs of RNNs or CNNs [159].

The first core module in SAED architectures, which is depicted in Figure 5.3a, refers to the self-attention (SA) module $\mathcal{A}_{\text{sa}}\big(\,\cdot\,\big)$:

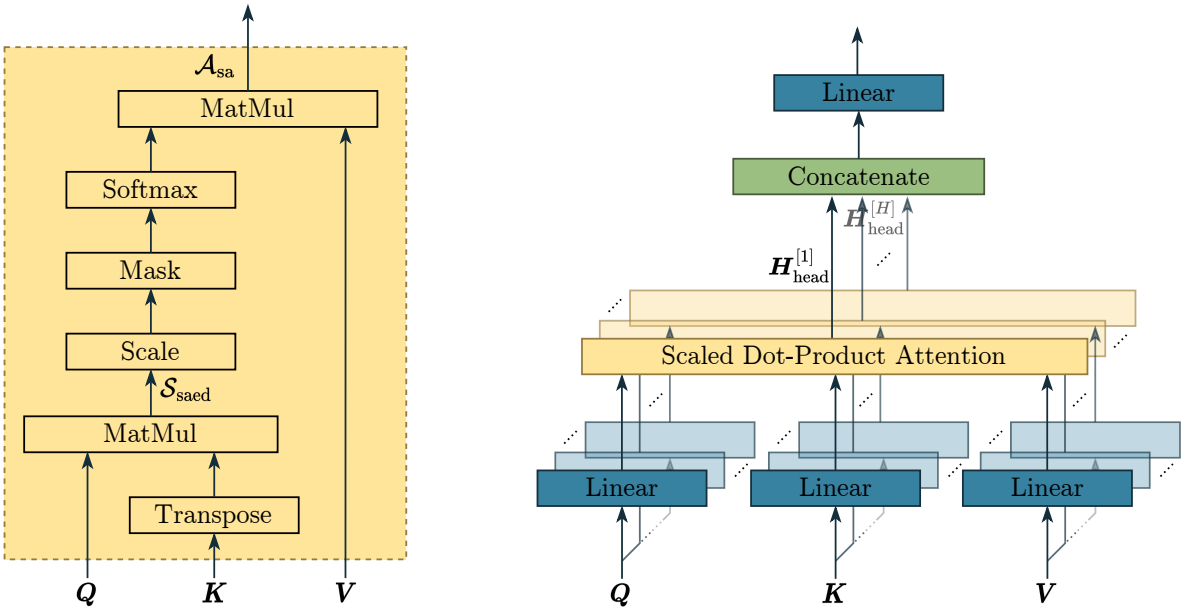
$$\mathcal{A}_{\text{sa}}\big(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\big) = \psi\big(\mathcal{S}_{\text{saed}}\big(\boldsymbol{Q}, \boldsymbol{K}\big)\big)\boldsymbol{V}, \tag{5.14}$$

for which a score function $\mathcal{S}_{\text{saed}}\big(\cdot\big)$ is introduced, which utilizes the matrix dot product [44]:

$$\mathcal{S}_{\text{saed}}\big(\boldsymbol{Q}, \boldsymbol{K}\big) = \frac{\boldsymbol{Q}\boldsymbol{K}^{\top}}{\sqrt{I_{\text{k}}}} \odot \boldsymbol{M}, \tag{5.15}$$

and includes the optional binary mask $\boldsymbol{M} \in \mathbb{B}^{T_{\text{q}} \times T_{\text{k}}}$. The score function $\mathcal{S}_{\text{saed}}$ is normalized by a softmax function ψ and scaled by the matrix $\boldsymbol{V}$. The scalar $1/\sqrt{I_{\text{k}}}$ prevents the subsequent softmax function from reaching regions where small gradients appear [159]. The matrices $\boldsymbol{Q} = \big[\boldsymbol{q}^{(1)}, \ldots, \boldsymbol{q}^{(t_{\text{q}})}, \ldots, \boldsymbol{q}^{(T_{\text{q}})}\big]$ of length T_{q}, $\boldsymbol{K} = \big[\boldsymbol{k}^{(1)}, \ldots, \boldsymbol{k}^{(t_{\text{k}})}, \ldots, \boldsymbol{k}^{(T_{\text{k}})}\big]$ of length T_{k}, and $\boldsymbol{V} = \big[\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(t_{\text{v}})}, \ldots, \boldsymbol{v}^{(T_{\text{v}})}\big]$ of length T_{v} refer to the query, key, and value matrix and gather the query $\boldsymbol{q}^{(t_{\text{q}})} \in \mathbb{R}^{1 \times I_{\text{q}}}$, key $\boldsymbol{k}^{(t_{\text{k}})} \in \mathbb{R}^{1 \times I_{\text{k}}}$, and value $\boldsymbol{v}^{(t_{\text{v}})} \in \mathbb{R}^{1 \times I_{\text{k}}}$ vectors, respectively. The sequence lengths are generally $T_{\text{k}} = T_{\text{v}}$, and the dimensions are $I_{\text{q}} = I_{\text{k}}$ [44].

A major drawback of single SA modules $\mathcal{A}_{\text{sa}}$ is their limitation on specific subspaces, which leads to SA modules focussing on similar parts of the sequences. In order to obtain more flexible yet parameter-efficient SAED models, Vaswani *et al.* [159] suggested employing multi-head attention (MHA) modules $\mathcal{A}_{\text{saed}}\big(\,\cdot\,\big)$ (compare to Figure 5.3b), defining an extension of the SA of SAED models:



(a) Scaled dot-product attention **(b)** Multi-head Attention

Figure 5.3: On the left, the SA module $\mathcal{A}_{\text{sa}}$ is depicted, determining a normalized score $\mathcal{S}_{\text{saed}}$ by including an optional mask $\boldsymbol{M}$ [44]. The MHA module is presented on the right, utilizing multiple heads $\boldsymbol{H}^{[h]}_{\text{head}}$ represented by individual SA modules [44]. Each head $\boldsymbol{H}^{[h]}_{\text{head}}$ is fed by the output of one-layer MLPs, followed by a concatenation of all heads and a linear projection onto the model dimension I_{model}.

$$\mathcal{A}_{\text{saed}}\big(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}\big) = \boldsymbol{W}_{\text{saed}}\big(\boldsymbol{H}^{[1]}_{\text{head}} \oplus \ldots \oplus \boldsymbol{H}^{[h]}_{\text{head}} \oplus \ldots \oplus \boldsymbol{H}^{[H]}_{\text{head}}\big) \quad \forall h \in \{1,2,\cdots,H\}, \tag{5.16}$$

where H heads $\boldsymbol{H}^{[h]}_{\text{head}}$ are concatenated and linearly transformed by the weight matrix $\boldsymbol{W}_{\text{saed}} \in \mathbb{R}^{I_{\text{model}} \times H I^{[h]}_{\text{v}}}$. Each head $\boldsymbol{H}^{[h]}_{\text{head}}$:

$$\boldsymbol{H}^{[h]}_{\text{head}} = \mathcal{A}_{\text{sa}}\left(\big(\boldsymbol{W}^{[h]}_{\text{q}}\boldsymbol{Q}^\top\big)^\top, \big(\boldsymbol{W}^{[h]}_{\text{k}}\boldsymbol{K}^\top\big)^\top, \big(\boldsymbol{W}^{[h]}_{\text{v}}\boldsymbol{V}^\top\big)^\top\right) \tag{5.17}$$

corresponds to an independent SA module $\mathcal{A}_{\text{sa}}$, as the matrices $\boldsymbol{Q} \in \mathbb{R}^{T_{\text{q}} \times I_{\text{model}}}$ and $\boldsymbol{K},\boldsymbol{V} \in \mathbb{R}^{T_{\text{k}} \times I_{\text{model}}}$ are linearly transformed into H distinct subspaces by individual projection matrices $\boldsymbol{W}^{[h]}_{\text{q}}, \boldsymbol{W}^{[h]}_{\text{k}} \in \mathbb{R}^{I^{[h]}_{\text{q}} \times I_{\text{model}}}$ and $\boldsymbol{W}^{[h]}_{\text{v}} \in \mathbb{R}^{I^{[h]}_{\text{v}} \times I_{\text{model}}}$. The internal model dimension is defined by I_{model}, and $I^{[h]}_{\text{q}}, I^{[h]}_{\text{k}}, I^{[h]}_{\text{v}}$ refers to the dimensions of the query, key, and value vectors of each head $\boldsymbol{H}_{\text{head}}$, which are typically set to $I^{[h]}_{\text{q}} = I^{[h]}_{\text{k}} = I^{[h]}_{\text{v}} = I_{\text{model}}/H$. As a result, the MHA module $\mathcal{A}_{\text{saed}}$ jointly attends multiple different sections of the sequences induced by the varying subspaces of the heads $\boldsymbol{H}^{[h]}_{\text{head}}$.

The last essential modules of SAED models are the two-layer position-wise MLPs, which finalize the MHA modules $\mathcal{A}_{\text{saed}}$:

$$\mathcal{H}_{\text{pw}}\big(\boldsymbol{X}\big) = \zeta\big(\boldsymbol{W}^{[1]}_{\text{pw}}\boldsymbol{X} + \boldsymbol{b}^{[1]}_{\text{pw}}\big)\boldsymbol{W}^{[2]}_{\text{pw}} + \boldsymbol{b}^{[2]}_{\text{pw}}, \tag{5.18}$$

where the rectified linear unit (ReLU) functions are assigned as activation functions ζ, $\boldsymbol{W}_{\text{pw}}^{[1]} \in \mathbb{R}^{I_{\text{model}} \times I_{\text{pw}}}$ and $\boldsymbol{W}_{\text{pw}}^{[2]} \in \mathbb{R}^{I_{\text{pw}} \times I_{\text{model}}}$ specify weight matrices, $\boldsymbol{b}_{\text{pw}}^{[1]} \in \mathbb{R}^{I_{\text{pw}}}$ and $\boldsymbol{b}_{\text{pw}}^{[2]} \in \mathbb{R}^{I_{\text{model}}}$ are bias vectors, and I_{pw} defines the dimension of the internal projection in $\mathcal{H}_{\text{pw}}$.

The encoder networks $\mathcal{H}_{\text{enc}}\big(\,\cdot\,\big)$ and decoder networks $\mathcal{H}_{\text{dec}}\big(\,\cdot\,\big)$ of SAED models can be defined by applying the preceding modules and previously introduced concepts from Chapter 2. Therefore, a standard SAED encoder layer $\boldsymbol{H}^{[l_{\text{enc}}]}$ is established by:

$$\begin{aligned}
\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}]} &= \mathcal{H}_{\text{pw}}^{[l_{\text{enc}}][4]}\big(\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}][3]}\big) + \boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}][2]} \\
\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}][3]} &= \text{LayerNorm}^{[l_{\text{enc}}][3]}\big(\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}][2]}\big) \\
\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}][2]} &= \mathcal{A}_{\text{saed}}^{[l_{\text{enc}}][2]}\big(\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}][1]}, \boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}][1]}, \boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}][1]}\big) + \boldsymbol{X}^{[l_{\text{enc}}]} \\
\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}][1]} &= \text{LayerNorm}^{[l_{\text{enc}}][1]}\big(\boldsymbol{X}^{[l_{\text{enc}}]}\big),
\end{aligned} \qquad \forall l_{\text{enc}} \in \{1, 2, \cdots, L_{\text{enc}}\}, \tag{5.19}$$

with $\big(\,\cdot\,\big)^{[l_{\text{enc}}][\widetilde{l}_{\text{enc}}]}$ referring to the current encoder layer l_{enc} with its sub-encoder-layers $\widetilde{l}_{\text{enc}}$ and the summation operator describing residual connections [65]. The inputs of the next layer are assigned to $\boldsymbol{X}^{[l_{\text{enc}}]} = \boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}-1]}$, except for the input to the first encoder layer $\boldsymbol{X}^{[l_{\text{enc}}=1]} = \boldsymbol{X}$. The output of the entire encoder network $\mathcal{H}_{\text{enc}}$ is then obtained by:

$$\boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]} = \mathcal{H}_{\text{enc}}\big(\boldsymbol{X}\big), \tag{5.20}$$

Similarly, the decoder networks $\mathcal{H}_{\text{dec}}$ can be defined as:

$$\begin{aligned}
\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}]} &= \mathcal{H}_{\text{pw}}^{[l_{\text{dec}}][6]}\big(\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][5]}\big) + \boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][4]} \\
\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][5]} &= \text{LayerNorm}^{[l_{\text{dec}}][4]}\big(\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][4]}\big) \\
\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][4]} &= \mathcal{A}_{\text{saed}}^{[l_{\text{dec}}][4]}\big(\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][3]}, \widetilde{\boldsymbol{H}}_{\text{enc}}^{[L_{\text{enc}}]}, \widetilde{\boldsymbol{H}}_{\text{enc}}^{[L_{\text{enc}}]}\big) + \boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][2]} \\
\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][3]} &= \text{LayerNorm}^{[l_{\text{dec}}][2]}\big(\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][2]}\big) \\
\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][2]} &= \mathcal{A}_{\text{saed}}^{[l_{\text{dec}}][2]}\big(\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][1]}, \boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][1]}, \boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][1]}\big) + \boldsymbol{G}_1^{o-1[l_{\text{dec}}]} \\
\boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}][1]} &= \text{LayerNorm}^{[l_{\text{dec}}][1]}\big(\boldsymbol{G}_1^{o-1[l_{\text{dec}}]}\big)
\end{aligned} \qquad \forall l_{\text{dec}} \in \{1, 2, \cdots, L_{\text{dec}}\}, \tag{5.21}$$

whereby $\widetilde{\boldsymbol{H}}_{\text{enc}}^{[L_{\text{enc}}]} = \text{LayerNorm}\big(\boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]}\big)$ (compare Section 2.2.11) and the optimal mask $\boldsymbol{M}$ are applied in $\mathcal{A}_{\text{saed}}^{[l_{\text{dec}}][2]}$ to prevent access to future predicted graphemes sequences $\boldsymbol{G}_{o+1}^{O}$, which are exclusively available in training yet not in the evaluation phase. Furthermore, $\boldsymbol{G}_1^{o-1[l_{\text{dec}}]} = \boldsymbol{H}_{\text{dec}}^{[l_{\text{dec}}-1]}$ is assigned to all decoder layers L_{dec}, besides the first decoder layer $\boldsymbol{G}_1^{o-1[l_{\text{dec}}=1]} = \boldsymbol{G}_1^{o-1}$ is applied. The output of the entire decoder networks $\mathcal{H}_{\text{dec}}$ is retrieved by:

$$\boldsymbol{H}_{\text{dec}}^{[L_{\text{dec}}]} = \mathcal{H}_{\text{dec}}\big(\boldsymbol{G}_1^{o-1}, \boldsymbol{H}_{\text{enc}}^{[L_{\text{enc}}]}\big) \quad \forall o \in \{1, 2, \cdots, O\}. \tag{5.22}$$

In order to obtain the final grapheme posterior distribution $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$, the decoder output is normalized:

$$\widetilde{\boldsymbol{H}}_{\text{dec}}^{[L_{\text{dec}}]} = \text{LayerNorm}\big(\boldsymbol{H}_{\text{dec}}^{[L_{\text{dec}}]}\big), \tag{5.23}$$

followed by a subsequent single-layer J-dimensional output MLP with linear activation functions (see also Section 2.2.1):

$$\boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}+1](o)} = \boldsymbol{W}^{[L_{\text{dec}}+1]}\widetilde{\boldsymbol{H}}_{\text{dec}}^{[L_{\text{dec}}]} + \boldsymbol{b}^{[L_{\text{dec}}+1]} \quad \forall o \in \{1, 2, \cdots, O\}, \tag{5.24}$$

with its weight matrix $\boldsymbol{W}^{[L_{\text{dec}}+1]} \in \mathbb{R}^{J \times J^{[L_{\text{dec}}]}}$ and its bias $\boldsymbol{b}^{[L_{\text{dec}}+1]} \in \mathbb{R}^{J}$. Finally, the posterior distribution $p\big(\boldsymbol{G}|\boldsymbol{X}\big)$ is derived by employing the softmax functions ψ:

$$p\big(\boldsymbol{g}^{(o)}|\boldsymbol{X}, \boldsymbol{G}_1^o\big) = \psi\big(\boldsymbol{h}_{\text{dec}}^{[L_{\text{dec}}+1](o)}\big) \quad \forall o \in \{1, 2, \cdots, O\}. \tag{5.25}$$

5.3.3 Model Extension by Additional Reversed Structures

Standard AED and SAED models generate the current grapheme output $\boldsymbol{g}^{(o)}$ based on the posterior distribution $p\big(\boldsymbol{g}^{(o)}|\boldsymbol{X}, \boldsymbol{G}_1^o\big)$, where the prediction of $\boldsymbol{g}^{(o)}$ relies on the entire input sequence $\boldsymbol{X}$ and the past grapheme prediction $\boldsymbol{G}_1^o$. Watzel *et al.* [10†, 12†] described these models as L2R architectures and denoted the grapheme distribution as $p\big(\overrightarrow{\boldsymbol{g}}^{(o)}|\boldsymbol{X}, \overrightarrow{\boldsymbol{G}}_1^o\big) = p\big(\boldsymbol{g}^{(o)}|\boldsymbol{X}, \boldsymbol{G}_1^o\big)$, as they predict future graphemes $\overrightarrow{\boldsymbol{g}}^{(o)}$ on the right by incorporating the left context of past graphemes $\overrightarrow{\boldsymbol{G}}_1^o$. Consequently, L2R models autoregressively predict future graphemes $\overrightarrow{\boldsymbol{g}}^{(o)}$, although they have access to the entire input sequence $\boldsymbol{X}$.

Watzel *et al.* extended the standard AED models with additional R2L decoders [12†] and the standard SAED models with entire R2L structures [10†] to access future context during optimization. Both additional structures are trained on time-reversed target grapheme sequences $\overleftarrow{\widetilde{\boldsymbol{G}}} = [\overleftarrow{\widetilde{\boldsymbol{g}}}^{(1)} = \overrightarrow{\widetilde{\boldsymbol{g}}}^{(O)}, \ldots, \overleftarrow{\widetilde{\boldsymbol{g}}}^{(\widetilde{o})} = \overrightarrow{\widetilde{\boldsymbol{g}}}^{(o)}, \ldots, \overleftarrow{\widetilde{\boldsymbol{g}}}^{(\widetilde{O})} = \overrightarrow{\widetilde{\boldsymbol{g}}}^{(1)}]$. For instance, the exemplary L2R grapheme sequence $\overrightarrow{\widetilde{\boldsymbol{G}}} = [\text{c}, \text{a}, \text{t}]$ would correspond to the time-reversed R2L grapheme sequence $\overleftarrow{\widetilde{\boldsymbol{G}}} = [\text{t}, \text{a}, \text{c}]$. Therefore, the training objective of their approaches [10†, 12†] was to determine, in addition to the standard grapheme distributions $p\big(\overrightarrow{\boldsymbol{g}}^{(o)}|\boldsymbol{X}, \overrightarrow{\boldsymbol{G}}_1^o\big)$, the time-reversed character distributions $p\big(\overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}|\boldsymbol{X}, \overleftarrow{\boldsymbol{G}}_1^{\widetilde{o}}\big)$. The information of future grapheme predictions $\overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}$ of R2L architectures is transferred to the regular L2R predictions $\overrightarrow{\boldsymbol{g}}^{(o)}$ by a novel regularization procedure.

In optimal scenarios, Watzel *et al.* [10†, 12†] expected L2R and R2L model structures, generating similar grapheme predictions for the current output time step $o = \widetilde{o}$:

$$p\big(\overrightarrow{\boldsymbol{g}}^{(o)}|\boldsymbol{X}, \overrightarrow{\boldsymbol{G}}_1^o\big) = p\big(\overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}|\boldsymbol{X}, \overleftarrow{\boldsymbol{G}}_1^{\widetilde{o}}\big), \tag{5.26}$$

since both structures obtain similar information, either in standard or time-reversed order.

5.3.4 Knowledge Transfer of AED and SAED Architectures

In order to enable a flow of information between the L2R and R2L model structures, Watzel *et al.* [10†, 12†] followed the approach of Zheng *et al.* [186], which introduced regularization terms $\Pi\big(\overrightarrow{\boldsymbol{G}}_{\mathcal{B}}, \overleftarrow{\boldsymbol{G}}_{\mathcal{B}}\big)$, utilizing the grapheme predictions $\overrightarrow{\boldsymbol{g}}_{\mathcal{B}}^{(o)}$, $\overleftarrow{\boldsymbol{g}}_{\mathcal{B}}^{(\widetilde{o})}$ of the L2R

and R2L structures in the joint cross entropy (CE) cost function $\widetilde{C}(\overrightarrow{\boldsymbol{G}}_{\mathcal{B}}, \overrightarrow{\tilde{\boldsymbol{G}}}_{\mathcal{B}}, \overleftarrow{\boldsymbol{G}}_{\mathcal{B}}, \overleftarrow{\hat{\boldsymbol{G}}}_{\mathcal{B}})$ for the current mini-batch $\mathcal{B}$. The enhanced cost function $\widetilde{C}_{\text{aed}}$ for AED models [12†] is defined as:

$$\begin{aligned}\widetilde{C}_{\text{aed}}(\overrightarrow{\boldsymbol{G}}_{\mathcal{B}}, \overrightarrow{\tilde{\boldsymbol{G}}}_{\mathcal{B}}, \overleftarrow{\boldsymbol{G}}_{\mathcal{B}}, \overleftarrow{\hat{\boldsymbol{G}}}_{\mathcal{B}}) = \alpha_{\text{aed}} C_{\text{CE}}(\overrightarrow{\boldsymbol{G}}_{\mathcal{B}}, \overrightarrow{\tilde{\boldsymbol{G}}}_{\mathcal{B}}) \\ + (1 - \alpha_{\text{aed}}) C_{\text{CE}}(\overleftarrow{\boldsymbol{G}}_{\mathcal{B}}, \overleftarrow{\hat{\boldsymbol{G}}}_{\mathcal{B}}) + \gamma_{\text{aed}} \Pi(\overrightarrow{\boldsymbol{G}}_{\mathcal{B}}, \overleftarrow{\boldsymbol{G}}_{\mathcal{B}}).\end{aligned} \tag{5.27}$$

The cost functions for SAED models are slightly modified [10†], as the KLD cost functions are combined with the CTC cost functions:

$$\begin{aligned}\widetilde{C}_{\text{saed}}(\overrightarrow{\boldsymbol{G}}_{\mathcal{B}}, \overrightarrow{\tilde{\boldsymbol{G}}}_{\mathcal{B}}, \overleftarrow{\boldsymbol{G}}_{\mathcal{B}}, \overleftarrow{\hat{\boldsymbol{G}}}_{\mathcal{B}}) = \alpha_{\text{saed}} \big[\beta_{\text{saed}} C_{\text{CTC}}(\overrightarrow{\boldsymbol{G}}_{\mathcal{B}}, \overrightarrow{\tilde{\boldsymbol{G}}}_{\mathcal{B}}) + (1 - \beta_{\text{saed}}) C_{\text{KLD}}(\overrightarrow{\boldsymbol{G}}_{\mathcal{B}}, \overrightarrow{\tilde{\boldsymbol{G}}}_{\mathcal{B}})\big] \\ + (1 - \alpha_{\text{saed}}) \big[\beta_{\text{saed}} C_{\text{CTC}}(\overleftarrow{\boldsymbol{G}}_{\mathcal{B}}, \overleftarrow{\hat{\boldsymbol{G}}}_{\mathcal{B}}) \\ + (1 - \beta_{\text{saed}}) C_{\text{KLD}}(\overleftarrow{\boldsymbol{G}}_{\mathcal{B}}, \overleftarrow{\hat{\boldsymbol{G}}}_{\mathcal{B}})\big] + \gamma_{\text{saed}} \Pi(\overrightarrow{\boldsymbol{G}}_{\mathcal{B}}, \overleftarrow{\boldsymbol{G}}_{\mathcal{B}})\end{aligned} \tag{5.28}$$

where $\alpha_{(\cdot)}$ refers to the parameter for weighting the corresponding L2R and R2L costs, $\beta_{(\cdot)}$ to the parameter for weighting the corresponding KLD with the additional CTC cost, and $\gamma_{(\cdot)}$ to a parameter for modifying the impact of the regularization term Π.

5.3.5 Regularization of Even Grapheme Sequence Lengths

In general, the transcript of a given dataset $\mathcal{D}$ can be arbitrarily pre-processed to obtain different types of target label sequences $\hat{\boldsymbol{G}}$ (compare Section 3.4). Watzel *et al.* [10†, 12†] employed CHARs and BPE units [144] as target sequences $\hat{\boldsymbol{G}}$, even though the application of BPE units seemed superior in recent approaches [30, 157].

For CHARs, the lengths of L2R grapheme sequences $\overrightarrow{\boldsymbol{G}}_{\text{char}}$ and R2L grapheme sequences $\overleftarrow{\hat{\boldsymbol{G}}}_{\text{char}}$ are even, as the time-reverse procedure has no impact on the length of the standard sequence $\overrightarrow{\boldsymbol{G}}_{\text{char}}$. Therefore, Watzel *et al.* [10†, 12†] defined the simple regularization term Π for even lengths of L2R and R2L grapheme sequences as:

$$\Pi(\overrightarrow{\boldsymbol{G}}_{\text{char},\mathcal{B}}, \overleftarrow{\boldsymbol{G}}_{\text{char},\mathcal{B}}) = \frac{1}{O} \sum_{o=1}^{O} d(\overrightarrow{\boldsymbol{g}}^{(o)}_{\text{char},\mathcal{B}}, \overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}_{\text{char},\mathcal{B}}), \tag{5.29}$$

with $d(\,\cdot\,,\cdot\,)$ representing a distance metric, and $o = \widetilde{o}$, since the lengths of the sequences $\overrightarrow{\boldsymbol{G}}_{\text{char},\mathcal{B}}$ and $\overleftarrow{\boldsymbol{G}}_{\text{char},\mathcal{B}}$ are even, *i.e.*, $O = \widetilde{O}$.

5.3.6 Regularization of Odd Grapheme Sequence Lengths

Watzel *et al.* [10†, 12†] faced a more challenging scenario in applying BPE units [144], as the generation of BPE sequences based on reversed transcripts leads to odd grapheme sequences ($O \neq \widetilde{O}$). For instance, although the number of BPE units is equal, *i.e.*, $\overrightarrow{\boldsymbol{g}}^{(o)}_{\text{bpe}}, \overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}_{\text{bpe}} \in \mathbb{R}^J$, the exemplary L2R grapheme $\overrightarrow{\boldsymbol{G}}_{\text{bpe}} = [\text{c}, \text{a}, \text{t\_}]$ would result in the

time-reversed R2L BPE sequence $\overleftarrow{\overleftarrow{\boldsymbol{G}}}_{\text{bpe}} = [\text{ta}, \text{c\_}]$. Therefore, the regularization term Π in Equation (5.29) cannot be applied, as it specifies scenarios with even sequence lengths.

The solution, which Watzel *et al.* established in their approaches [10†, 12†], was inspired by the concept of the soft-dynamic time warping (DTW). Initially, the standard DTW algorithm was proposed by Berndt *et al.* [15] for comparing temporal sequences. However, the algorithm cannot be utilized in neural network (NN) training, as it relies on the *min* operator, which is not differentiable. In 2017, Cuturi *et al.* [39] introduced the soft-DTW algorithm, which corresponds to a differentiable version of the standard DTW algorithm. Therefore, the soft-DTW algorithm redefines the min operators as softmin operators whose softness is modified by the heuristic parameter γ:

$$\min_{1 \leq i \leq I}{}^{\gamma} a_i = \begin{cases} \min\limits_{1 \leq i \leq I} a_i & \gamma = 0 \\ -\gamma \log \sum\limits_{i=1}^{I} \mathrm{e}^{-a_i/\gamma} & \gamma > 0, \end{cases} \tag{5.30}$$

with $\boldsymbol{a} \in \mathbb{R}^I$ defining an intermediate alignment cost vector. If $\gamma = 0$, the softmin operator recovers to the standard min operator, whereby $\gamma > 0$ leads to a smoothed version of the min operator [39]. Watzel *et al.* [10†, 12†] utilized the soft-DTW algorithm to obtain a regularization term Π for $\overrightarrow{\boldsymbol{G}}$ and $\overleftarrow{\boldsymbol{G}}$ with altering sequence lengths:

$$\Pi\big(\overrightarrow{\boldsymbol{G}}_{\text{bpe},\mathcal{B}}, \overleftarrow{\boldsymbol{G}}_{\text{bpe},\mathcal{B}}\big) = \min_{1 \leq i \leq I}{}^{\gamma}\big(\boldsymbol{D}\big(\overrightarrow{\boldsymbol{G}}_{\text{bpe},\mathcal{B}}, \overleftarrow{\boldsymbol{G}}_{\text{bpe},\mathcal{B}}\big)\boldsymbol{A}\big). \tag{5.31}$$

The matrix $\boldsymbol{D}\big(\,\cdot\,,\cdot\,\big) \in \mathbb{R}^{O \times \widetilde{O}}$ calculates the cost of the alignment procedure:

$$\boldsymbol{D}\big(\overrightarrow{\boldsymbol{G}}_{\text{bpe}}, \overleftarrow{\boldsymbol{G}}_{\text{bpe}}\big) = \sum_{o=1}^{O} \sum_{\widetilde{o}=1}^{\widetilde{O}} d\big(\overrightarrow{\boldsymbol{g}}^{(o)}_{\text{bpe}}, \overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}_{\text{bpe}}\big), \tag{5.32}$$

and $\boldsymbol{A} \in \mathbb{B}^{O \times \widetilde{O}}$ refers to a binary alignment matrix from a set $\mathcal{A} \subset \mathbb{B}^{O \times \widetilde{O}}$, gathering all allowed paths from $(1,1)$ to $(o, \widetilde{o})$ by solely performing the operations $\downarrow$, $\rightarrow$, and $\searrow$ for moving from the uppermost left to lower most right part in an alignment matrix $\boldsymbol{A}$.

By calculating the inner product between $\boldsymbol{D}\big(\overrightarrow{\boldsymbol{G}}_{\text{bpe}}, \overleftarrow{\boldsymbol{G}}_{\text{bpe}}\big)$ and $\boldsymbol{A}$ in Equation (5.31), Watzel *et al.* [10†, 12†] obtained the cost of all alignment variations of $\overrightarrow{\boldsymbol{G}}_{\text{bpe}}$ and $\overleftarrow{\boldsymbol{G}}_{\text{bpe}}$. As a result, the additional training objective of the regularization term Π constrains the overall model architectures for determining proper alignments between the predicted L2R and R2L grapheme sequences.

5.3.7 Distance Metrics

Generally, the definition of the distance metrics $d\big(\,\cdot\,,\cdot\,\big)$ is arbitrarily chosen. For AED models [12†], Watzel *et al.* selected the standard Euclidean metric $d_{\text{euc}}\big(\,\cdot\,,\cdot\,\big)$ to measure the distance between the L2R and R2L grapheme predictions:

$$d_{\text{euc}}\big(\overrightarrow{\boldsymbol{g}}^{(o)}, \overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}\big) = \|\overrightarrow{\boldsymbol{g}}^{(o)} - \overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}\|. \tag{5.33}$$

However, it remains unclear which impact a chosen distance metric has on the performance of SAED architectures. In 2018, Cer *et al.* [24] introduced the Cosine similarity in sentence encoding, in which the angle of two vectors served as a distance metric. Inspired by their work, Watzel *et al.* slightly modified the Cosine similarity metric and specified it in the following way:

$$d_{\cos}\big(\overrightarrow{\boldsymbol{g}}^{(o)}, \overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}\big) = \frac{1}{\pi} \arccos\left(\frac{\overrightarrow{\boldsymbol{g}}^{(o)} \overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}}{\|\overrightarrow{\boldsymbol{g}}^{(o)}\| \|\overleftarrow{\boldsymbol{g}}^{(\widetilde{o})}\|}\right). \tag{5.34}$$

5.4 Experimental Setup

The AED and SAED model structures were evaluated by Watzel *et al.* [10†, 12†] on the publicly available datasets TED-LIUM release 2 (TED-LIUM-v2) [138] and LibriSpeech [115]. They utilized the given train, test, and dev sets of the TED-LIUM-v2 dataset and followed a similar approach for the LibriSpeech dataset, where they employed the train-clean-100, train-clean-360, and train-other-500 as train sets, the test-clean and test-other as test sets, and the sets dev-clean and dev-other as dev sets. The AED and SAED architectures are implemented in the ESPnet framework [162], which utilizes the general machine learning toolkit PyTorch [118]. The ESPnet framework belongs to the most popular frameworks for E2E ASR, gathering all the required modules for pre-processing ASR datasets, optimizing SOTA DNN models, and providing efficient decoding procedures for evaluating the final models.

The experimental setup section is roughly dividable into four parts. In the first part, the pre-processing of the two datasets is introduced, which has been applied by Watzel *et al.* [10†, 12†]. Then, the subsequent section discusses the model structure of AED and SAED architectures. The third section establishes the training procedures for analyzing the impact of reversed model structures on standard AED and SAED models during training. The section is concluded with the decoding scheme for both model variants.

5.4.1 Pre-Processing of the TED-LIUM-v2 and LibriSpeech Datasets

First, Watzel *et al.* pre-processed the TED-LIUM-v2 and LibriSpeech datasets following the procedure described in Section 4.4.1. Their SAED approach [10†] augmented the TED-LIUM-v2 dataset by speed perturbation [77] with the speed factors 0.9, 1.0, and 1.1 before extracting the speech features (compare Section 3.3.1). In both approaches [10†, 12†], the 80-dimensional log Mel feature and three-dimensional pitch feature vector are extracted [103], leading to the final unnormalized 83-dimensional input vectors $\boldsymbol{x}^{(t)} \in \mathbb{R}^{83}$.

They chose standard CHARs and BPE units as target label types. The CHAR sequences are directly extracted from the given transcript of the datasets. Additionally, each CHAR sequence is framed by the <sos> and <eos> tokens, respectively, in order to specify the start and end of the grapheme sequences. The procedure is repeated

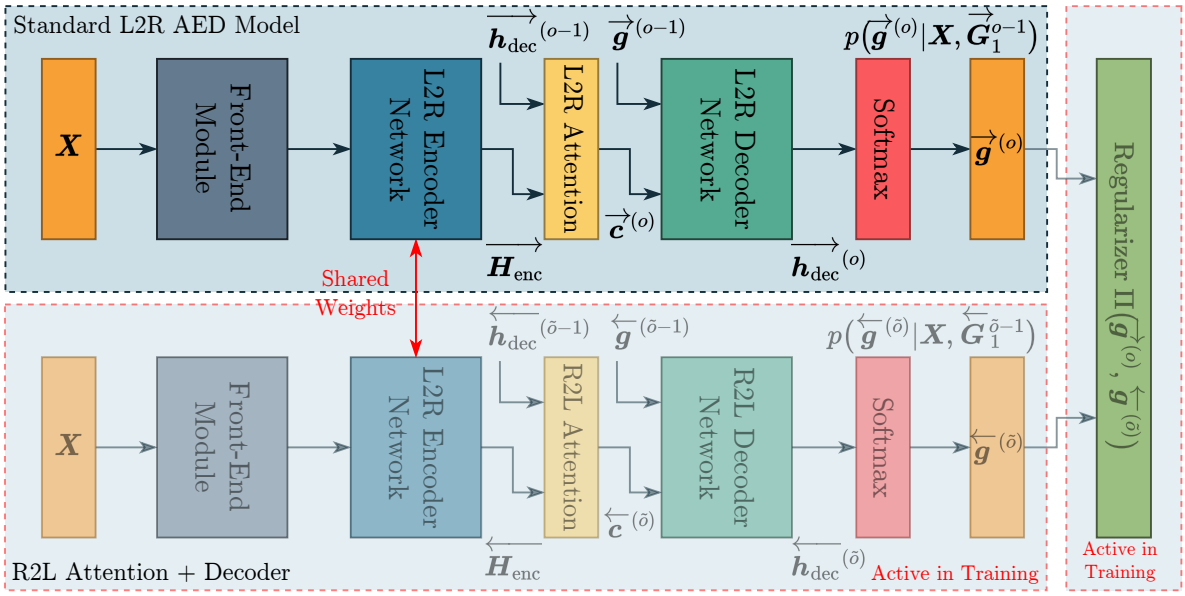


Figure 5.4: The dual AED architecture consists of shared encoder networks $\mathcal{H}_{\text{enc}} = \overleftarrow{\mathcal{H}_{\text{enc}}} = \overrightarrow{\mathcal{H}_{\text{enc}}}$ and separate attention modules and decoder networks for each L2R and R2L model. The R2L AED networks and the regularization terms Π are solely active during optimization, leading to a knowledge transfer between L2R and R2L AED architectures. The R2L structures and the regularization terms are excluded in the decoding phase to avoid increasing complexity.

for a time-reversed version of the given transcript to obtain the L2R target graphemes $\overrightarrow{\boldsymbol{g}}^{(o)}_{\text{char}} \in \mathbb{B}^{28}$ and the R2L target sequence $\overleftarrow{\boldsymbol{g}}^{(\tilde{o})}_{\text{char}} \in \mathbb{B}^{28}$.

The generating procedure of BPE units is similarly initialized by enclosing each standard and time-reversed grapheme sequence with <sos> and <eos> tokens in the transcript. Then, they utilized the tokenizer SentencePiece [79] to determine the 100 optimal BPE units [144] for representing the framed L2R and R2L sequences, leading to the final L2R BPE units $\overrightarrow{\boldsymbol{g}}^{(o)}_{\text{bpe}} \in \mathbb{B}^{100}$ and the R2L BPE units $\overleftarrow{\boldsymbol{g}}^{(\tilde{o})}_{\text{bpe}} \in \mathbb{B}^{100}$.

In summary, the pre-processing procedures result in two individual training datasets $\mathcal{D}_{\text{char}} = \left\{\left(\boldsymbol{X}^{(n)}, \overrightarrow{\boldsymbol{G}}^{(n)}_{\text{char}}, \overleftarrow{\boldsymbol{G}}^{(n)}_{\text{char}}\right)\right\}_{n=1}^{N_1}$ and $\mathcal{D}_{\text{bpe}} = \left\{\left(\boldsymbol{X}^{(n)}, \overrightarrow{\boldsymbol{G}}^{(n)}_{\text{bpe}}, \overleftarrow{\boldsymbol{G}}^{(n)}_{\text{bpe}}\right)\right\}_{n=1}^{N_2}$, for each TED-LIUMv2 and LibriSpeech dataset, respectively.

5.4.2 The Architecture of AED Models

For AED models [12†], the input sequences $\boldsymbol{X}$ are not directly fed into the encoder networks $\mathcal{H}_{\text{enc}}$ since these sequences are typically too long, leading to inferior performance of the subsequent RNN networks [91]. Instead, Watzel *et al.* [12†] pre-processed the input sequences $\boldsymbol{X}$ by front-end CNN networks, which extract local features, and reduce the length of the input sequences $\boldsymbol{X}$. Therefore, they applied a two-block CNN as a front-end structure [162], initially inspired by the visual geometry group (VGG) architecture [147]. The first block consists of two CNN layers with 64 output channels, respectively, whereby the input channels of the first layer are assigned to one and the input channels of the

second layer to 64. Furthermore, 3×3 filters with stride lengths of one are applied, leading to the two CNN layer kernels $\boldsymbol{K}^{[1]} \in \mathbb{R}^{1\times64\times3\times3}$ and $\boldsymbol{K}^{[2]} \in \mathbb{R}^{64\times64\times3\times3}$. Each layer applies ReLU activation functions, and the entire CNN block is finalized by 2D max-pooling layers with 2×2 kernels and stride lengths of two. The second CNN block structure corresponds to the one above. However, they distinguish by the number of output channels set to 128, leading to the two CNN layer kernels $\boldsymbol{K}^{[3]} \in \mathbb{R}^{64\times128\times3\times3}$ and $\boldsymbol{K}^{[4]} \in \mathbb{R}^{128\times128\times3\times3}$. The output of the second block is vectorized (compare Section 2.2.4) and defines the new input sequences $\boldsymbol{X} \in \mathbb{R}^{2656\times T/4}$ to the encoder networks $\mathcal{H}_{\text{enc}}$. The encoder consists of four bidirectional long short-term memory projected (BLSTMP) layers $\boldsymbol{H}_{\text{enc}}^{[l]} \in \mathbb{R}^{1024}$ with $J^{[l_{\text{enc}}]} = 1024$ units for each direction and a projection layer $\boldsymbol{P}^{[l_{\text{enc}}]} \in \mathbb{R}^{1024}$ of size $\widetilde{J}^{[l_{\text{enc}}]} = 1024$. Each L2R decoder $\overrightarrow{\mathcal{H}_{\text{dec}}}$ and R2L decoder $\overleftarrow{\mathcal{H}_{\text{dec}}}$ network is defined as one-layer LSTMs with 1024 hidden units, respectively, and concluded by individual MLPs with linear activation functions, followed by a softmax function to obtain the grapheme predictions $\overrightarrow{\boldsymbol{g}}^{(o)}$ and $\overleftarrow{\boldsymbol{g}}^{(\tilde{o})}$. Moreover, each decoder output is generated by applying the independent attention modules $\overrightarrow{\mathcal{A}_{\text{aed}}}$ and $\overleftarrow{\mathcal{A}_{\text{aed}}}$, whereby the dimensions of the convolutional kernels are assigned to $\overrightarrow{\boldsymbol{F}}, \overleftarrow{\boldsymbol{F}} \in \mathbb{R}^{10\times100}$, leading to the final model structure in Figure 5.4.

5.4.3 The Architecture of SAED Models

For SAED models [10†], Watzel *et al.* also did not directly feed the input sequences $\boldsymbol{X}$ into the encoder networks and instead pre-processed them by a convolutional front-end network to reduce the length of input sequences [44]. The front-end models consist of two-layer CNNs with the kernels $\boldsymbol{K}^{[1]} \in \mathbb{R}^{1\times83\times3\times3}$ and $\boldsymbol{K}^{[2]} \in \mathbb{R}^{83\times83\times3\times3}$, where the stride length is set to two, and each layer utilizes ReLU activation functions. The outputs of the second CNN layer are vectorized (compare Section 2.2.4) and linearly mapped to I_{model}-dimensional input vectors by MLPs. Since SAED models do not incorporate recurrent or convolutional structures, they have no information on the order of sequences. Therefore, Vaswani *et al.* [159] proposed a positional encoding function $\text{PE}(\,\cdot\,,\cdot\,)$, which inserts information about the relative input vector position in the overall input sequence:

$$\text{PE}(i,t) = \begin{cases} \sin\left(t/10\,000^{2i/I_{\text{model}}}\right) & 0 \leq i < I_{\text{model}}/2 \\ \cos\left(t/10\,000^{2i/I_{\text{model}}}\right) & I_{\text{model}}/2 \leq i < I_{\text{model}} \end{cases} \tag{5.35}$$

whereby i refers to the current indices of the I_{model}-dimensional input vectors and t to the current time-step of the output sequence with length $T/4$ generated by front-end models. The final input sequences $\boldsymbol{X} \in \mathbb{R}^{I_{\text{model}}\times T/4}$ for the encoder networks $\mathcal{H}_{\text{enc}}$ are obtained by summing the SAED front-end outputs and the positional encoding PE of the corresponding sequences. The sequences $\boldsymbol{X}$ are then fed to L2R encoder $\overleftarrow{\mathcal{H}_{\text{enc}}}$ and R2L encoder $\overrightarrow{\mathcal{H}_{\text{enc}}}$, where each encoder is composed of 12 encoder layers $\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}]}$ with 2048 units, respectively. In each attention module $\overrightarrow{\mathcal{A}_{\text{saed}}}^{[l_{\text{enc}}]}$ and $\overleftarrow{\mathcal{A}_{\text{saed}}}^{[l_{\text{enc}}]}$, Watzel *et al.* [10†] employed a total of $H = 4$ independent heads with the corresponding internal dimension $I_{\text{model}} = 256$. The optional mask $\boldsymbol{M}$ is omitted, as attention to all sections of the input sequence $\boldsymbol{X}$ is allowed. The dimension of the position-wise MLP was set to $I_{\text{pw}} = 2048$.

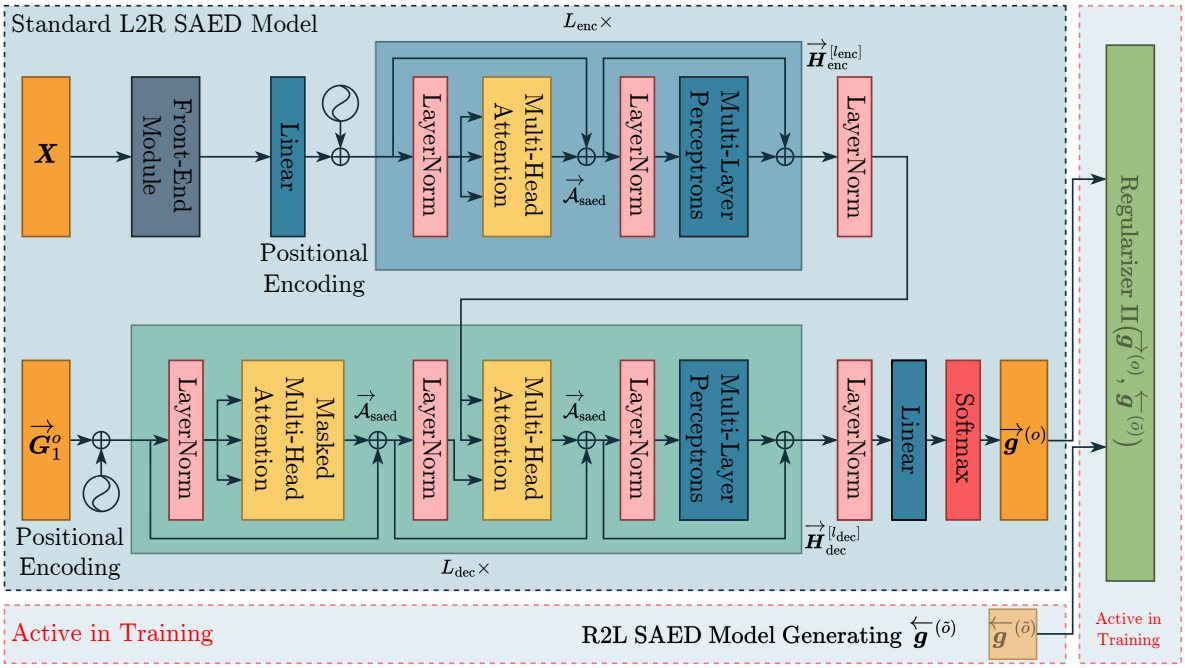


Figure 5.5: The dual SAED architecture consists of entire L2R and R2L SAED architectures. Similarly to the dual AED networks, the R2L SAED models and the regularization terms Π are only active in the training phase and are not utilized during decoding.

A similar parameter setup is conducted for the L2R decoder $\overleftarrow{\mathcal{H}}_{\text{dec}}$ and R2L decoder $\overrightarrow{\mathcal{H}}_{\text{dec}}$ with its independent $\overrightarrow{\mathcal{A}}_{\text{saed}}^{[l_{\text{dec}}]}$ and $\overleftarrow{\mathcal{A}}_{\text{saed}}^{[l_{\text{dec}}]}$ modules by applying solely six decoder layers. In contrast to the encoder networks $\mathcal{H}_{\text{enc}}$, they applied a lower triangular matrix of ones as the mask $\boldsymbol{M}$ in the SA modules $\overrightarrow{\mathcal{A}}_{\text{sa}}^{[l_{\text{dec}}][2]}$ and $\overleftarrow{\mathcal{A}}_{\text{sa}}^{[l_{\text{dec}}][2]}$ to avoid attention to future predicted graphemes in the sequences $\overrightarrow{\boldsymbol{G}}_{o+1}^{O}$ and $\overleftarrow{\boldsymbol{G}}_{\tilde{o}+1}^{\tilde{O}}$. The last L2R decoder layers $\overrightarrow{\boldsymbol{H}}_{\text{dec}}^{(L_{\text{dec}})}$ and R2L decoder layers $\overleftarrow{\boldsymbol{H}}_{\text{dec}}^{(L_{\text{dec}})}$ are concluded by separate LayerNorm layers, linear output layers, and softmax functions to obtain the grapheme predictions $\overrightarrow{\boldsymbol{g}}^{(o)}$ and $\overleftarrow{\boldsymbol{g}}^{(\tilde{o})}$ (compare to Figure 5.5). Notice that the grapheme sequences also require a positional encoding procedure to provide relative temporal information for the decoder networks. For regularization, a dropout layer [66] is applied after each $\mathcal{A}_{\text{sa}}$ and $\mathcal{A}_{\text{saed}}$ module, whereby the dropout probability is assigned to $\rho = 0.1$.

5.4.4 Optimization and Training Procedures of AED Models

Watzel *et al.* [12†] optimized several AED model structures in their approach and established a general optimization scheme. Typically, Adadelta optimizers [180] are applied for optimizing AED models whose learning rate is initialized by $\eta = 10^{-8}$. The optimizer minimizes the cost function $\widetilde{C}_{\text{aed}}$ utilizing mini-batches $\mathcal{B}$ of size $B = 30$. The learning rate η is decayed by 0.01, and the value of a patient counter is increased by one if no improvements in the dev set are observed. The procedure is terminated if the patient counter exceeds the value of three.

For AED models [12†], Watzel *et al.* conducted five different training procedures:

1. In the first *Forward* setup, standard AED models with L2R decoder networks $\overrightarrow{\mathcal{H}}_{\text{dec}}$ are optimized, and the weighting $\alpha_{\text{aed}} = 1.0$ and regularization parameter $\gamma_{\text{aed}} = 0.0$ are assigned, defining the baseline.

2. The second *Backward* training procedure is specified by setting the two parameters $\alpha_{\text{aed}}, \gamma_{\text{aed}}$ to 0.0, leading to AED models with R2L decoder networks $\overleftarrow{\mathcal{H}}_{\text{dec}}$.

3. The third *Backward Fixed* training setup corresponded to the *Backward* procedure, whereby the weights of the encoder network $\overleftarrow{\mathcal{H}}^{\text{fixed}}_{\text{enc}} = \overrightarrow{\mathcal{H}}_{\text{enc}}$ are frozen and initialized by the final encoder weights from the *Forward* stage.

4. In the fourth *Dual Decoder* stage, the encoder networks $\mathcal{H}_{\text{enc}}$ are shared between the L2R decoder networks $\overrightarrow{\mathcal{H}}_{\text{dec}}$ and the R2L decoder networks $\overleftarrow{\mathcal{H}}_{\text{dec}}$. The encoder $\mathcal{H}_{\text{enc}}$ and L2R decoder $\overrightarrow{\mathcal{H}}_{\text{dec}}$ are initialized by the final weights from the *Forward* setup, and the R2L decoder $\overleftarrow{\mathcal{H}}_{\text{dec}}$ by the weights of the final *Backward Fixed* model. To solely allow minor impacts of R2L models on standard L2R structures, large weighting values are set for $\alpha_{\text{aed}} = 0.9$, whereby the remaining parameters are not modified.

5. The last *Dual Decoder Euc* setup corresponds to the final setup, integrating the regularization term Π into the overall cost function $\widetilde{C}_{\text{aed}}$. The distance metric in Π is set to the Euclidean metric d_{euc}. Since the regularization terms differ between CHAR and BPE grapheme sequences, the parameter γ_{aed} is accordingly adjusted. For CHARs, the weighting of the regularization parameter is assigned to $\gamma_{\text{aed}} = 1.0$ and further reduced to $\gamma_{\text{aed}} = 10^{-5}$ for the BPE units. The additionally softening parameter of the soft-DTW algorithm is set to $\gamma = 1.0$.

5.4.5 Optimization and Training Procedures of SAED Models

Watzel *et al.* [10†] introduced a similar optimization with subsequent training procedures for SAED architectures. In general, standard SAED models [44] are trained by Adam optimizers [76]. The cost function $\widetilde{C}_{\text{saed}}$ defines the training objective, which is minimized for a mini-batch $\mathcal{B}$ of size $B = 32$. Additionally, the one-hot encoded target graphemes $\overrightarrow{\boldsymbol{g}}^{(o)}$ and $\overleftarrow{\boldsymbol{g}}^{(\tilde{o})}$ are pre-processed by the label smoothing procedure (compare Equation (2.127)) with a smoothing value of $\iota = 0.1$ before being further processed in the model cost function $\widetilde{C}_{\text{saed}}$. In all procedures below, CTC cost functions C_{CTC} are applied by weighting the costs with $\beta_{\text{saed}} = 0.3$. In contrast to AED models, SAED models typically require a warmup phase, where the learning rate η is linearly increased before transiting into the actual optimization phase, in which the learning rate η is gradually decreased. Watzel *et al.* [10†] applied a slightly modified learning rate schedule compared with the initially proposed version [159]:

$$\eta = I_{\text{model}}^{-0.5} \min\left(\tau^{-0.5}, \tau \cdot 25\,000^{-1.5}\right), \tag{5.36}$$

whereby τ refers to the current iteration step (see also Section 2.2.8) and I_{model} to the internal model dimension of SAED models. Notice that based on Equation (5.36), the warmup phase lasts for 25 000 iterations, in which the learning rate η is linearly increased.

The subsequent training procedures of SAED, which Watzel *et al.* established in their approach [10†], are comparable to the AED procedures defined above. In contrast to regularized AED models, which solely integrate additional decoder networks $\overleftarrow{\mathcal{H}_{\text{dec}}}$ and rely on shared encoder networks $\mathcal{H}_{\text{enc}}$, regularized SAED architectures utilize additional R2L architectures $\overleftarrow{\mathcal{H}_{\text{saed}}}$ in the standard model architectures $\mathcal{H}_{\text{saed}} = \overrightarrow{\mathcal{H}_{\text{saed}}}$. Apart from these major differences, four training procedures are employed:

1. The first training procedure is also specified as a *Forward* setup. Therefore, Watzel *et al.* [10†] trained a standard SAED model $\overrightarrow{\mathcal{H}_{\text{saed}}}$ and set the weighting parameter to $\alpha_{\text{saed}} = 1.0$ and the regularization parameter to $\gamma_{\text{saed}} = 0.0$.

2. In the second *Backward* procedure, the total cost function $\widetilde{C}_{\text{saed}}$ is modified by assigning $\alpha_{\text{saed}} = 0.0$ and $\gamma_{\text{saed}} = 0.0$ to obtain R2L SAED architectures $\overleftarrow{\mathcal{H}_{\text{saed}}}$.

3. The third *Dual SAED Euc* setup already defines a regularized procedure in which L2R SAEDs $\overrightarrow{\mathcal{H}_{\text{saed}}}$ and R2L SAEDs $\overleftarrow{\mathcal{H}_{\text{saed}}}$ models are utilized. The weights of the L2R model structure $\overrightarrow{\mathcal{H}_{\text{saed}}}$ are initialized with the final model weights of the *Forward* stage, whereas the initial weights of R2L models are taken from the optimized model in the *Backward* procedure. In order to achieve a large impact on L2R SAED architectures, Watzel *et al.* chose a large weighting parameter $\alpha_{\text{saed}} = 0.9$ in the total cost function $\widetilde{C}_{\text{saed}}$. The regularization term Π employs the Euclidean distance metrics d_{euc} and its impact was set to $\gamma_{\text{saed}} = 1.0$ for CHARs and $\gamma_{\text{saed}} = 10^{-3}$ for BPE units by Watzel *et al.* [10†]. In the case of the BPE units, the softening parameter of the soft-DTW algorithm is assigned to $\gamma = 1.0$.

4. The final *Dual SAED Cos* training procedure is comparable to the former setup, applying the Cosine distance metric $d_{\cos}$ in the regularization term Π. The impact of the regularization is altered to $\gamma_{\text{saed}} = 15.0$ for CHARs and further to $\gamma_{\text{saed}} = 10^{-1}$ for BPE units. The remaining parameters are not modified.

5.4.6 Decoding Procedure for AED and SAED Models

In both decoding procedures of AED and SAED models [10†, 12†], the integration of LMs is omitted. Watzel *et al.* concatenated solely on the AM structures and followed a similar decoding strategy in both regularized AED and SAED approaches [10†, 12†] in which the regularized structures $\overrightarrow{\mathcal{H}_{\text{dec}}}$ and $\overrightarrow{\mathcal{H}_{\text{saed}}}$ were decoded, and the auxiliary structures $\overleftarrow{\mathcal{H}_{\text{dec}}}$ and $\overleftarrow{\mathcal{H}_{\text{saed}}}$ were excluded from the decoding phase.

The AED architectures are decoded by a beam size of $K_{\text{best}} = 20.0$. For SAED models, the beam size is slightly reduced to $K_{\text{best}} = 10.0$. In contrast to AED architectures, where the overall best model is commonly utilized for the decoding phase, SAED models are decoded by first averaging their weights of the last ten epochs before decoding the averaged SAED model. Additionally, the CTC networks are incorporated with a factor of 0.3 into the decoding procedures.

Table 5.1: The results of the five AED training procedures evaluated on the TED-LIUM-v2 dataset [138] and reported in WER (%).

	TED-LIUM-v2 [138]			
	CHARs		BPE units	
Methods	dev	test	dev	test
Forward	16.77	17.32	17.83	18.00
Backward	18.12	18.47	18.57	17.99
Backward Fixed	23.34	23.77	25.55	25.01
Dual Decoder	16.47	17.12	17.70	18.08
Dual Decoder Euc	**15.68**	**15.94**	**16.75**	**17.42**

5.5 Evaluation

This section discusses the results of both approaches [10†, 12†] from Watzel *et al.* Related approaches to AED models [12†] are presented and examined in the first part. Then, the corresponding SAED model results are analyzed and further discussed.

5.5.1 Results of AED Architectures

Watzel *et al.* [12†] evaluated their AED architectures based on the five training procedures defined above. They also considered the impact of larger training data quantities by evaluating the smaller dataset TED-LIUM-v2 [138] and larger dataset LibriSpeech [115]. The results of both datasets are summarized in Table 5.1 and Table 5.2, respectively.

In the first two *Forward* and *Backward* training procedures, Watzel *et al.* expected an equal performance of the L2R decoders $\overrightarrow{\mathcal{H}_{\text{dec}}}$ and R2L $\overleftarrow{\mathcal{H}_{\text{dec}}}$ since both model structures obtained the same amount of information, leading to equally modeled grapheme distributions in Equation (5.26). For the larger LibriSpeech dataset, they verified their expectation, as only a minor performance difference between the setups was noticeable in Table 5.2. However, the results of the smaller dataset TED-LIUM-v2 in Table 5.1 reveal that the statement in Equation (5.26) is not confirmed since L2R decoders $\overrightarrow{\mathcal{H}_{\text{dec}}}$ returned superior results compared to R2L decoders $\overleftarrow{\mathcal{H}_{\text{dec}}}$. Watzel *et al.* [12†] identified the dataset size as the cause of the performance imbalance between both datasets. They argued that rather more unique variations of sentence endings exist than variations of sentence beginnings. Therefore, R2L architectures require more training samples for better generalization of R2L grapheme sequences.

The *Backward Fixed* setup demonstrates that the hidden representations $\boldsymbol{H}^{[L_{\text{enc}}]}$ of the encoder networks $\mathcal{H}_{\text{enc}}$ also encode temporal information of the L2R decoder structures and are therefore heavily influenced by the subsequent decoder networks $\mathcal{H}_{\text{dec}}$. Based on the results in Table 5.1 and Table 5.2, Watzel *et al.* confirmed these findings for both datasets. They claimed that the frozen encoder weights led to a restricted flow of information between the encoder and decoder structures. Compared to standard

Table 5.2: The results of the five AED training procedures evaluated on the LibriSpeech dataset [115] and reported in WER (%).

	LibriSpeech [115]							
	CHARs				BPE units			
	dev		test		dev		test	
Methods	clean	other	clean	other	clean	other	clean	other
Forward	7.69	20.67	7.72	21.63	7.59	20.98	7.67	21.92
Backward	7.60	20.78	7.54	21.83	7.53	20.94	7.60	21.71
Backward Fixed	11.39	28.36	11.75	28.53	12.07	28.63	12.39	29.06
Dual Decoder	7.29	20.99	7.60	22.00	7.46	21.29	7.70	22.01
Dual Decoder Euc	**7.24**	**19.96**	**7.02**	**20.95**	**7.17**	**20.01**	**7.33**	**20.63**

Backward training procedures, R2L models with fixed encoders experience a major performance decline.

In the third *Dual Decoder* training procedure, Watzel *et al.* were inspired by the approach of Mimura *et al.* [105], who applied shared encoder networks $\mathcal{H}_{\text{enc}}$ with $\overrightarrow{\mathcal{H}_{\text{dec}}}$ and $\overleftarrow{\mathcal{H}_{\text{dec}}}$ decoders in training. However, they did not evaluate the resulting AED model consisting of the shared encoder $\mathcal{H}_{\text{enc}}$ and the L2R decoder $\overrightarrow{\mathcal{H}_{\text{dec}}}$ in their approach, and therefore the impact of R2L model structures on standard L2R architectures remained unclear. The results in Table 5.1 and Table 5.2 indicate that the impact of a reverse model structure on the final model performance is negligible. Although enhanced L2R models return minor improved results in both datasets, a clear tendency for major improvements is not observable.

The last *Dual Decoder Euc* setup reveals the largest impact on the performance of standard AED models. Compared to the baseline models established in the *Forward* procedures, the regularized approach from Watzel *et al.* [12†] returned superior AED models. These models achieve the best performance without increasing the complexity of the decoding procedures. The results in Table 5.1 and Table 5.2 confirm consistent improvements in both datasets. On the smaller dataset TED-LIUM-v2, relative WER improvements of 7.2% for the CHAR and 4.4% for the BPE graphemes are observed. These improvements are reproducible on the larger LibriSpeech dataset, where the regularized L2R models experience relative WER improvements of 4.9% and 5.1% for CHAR and BPE graphemes, respectively.

5.5.2 Results of SAED Architectures

The SAED models are evaluated in four different setups, whereby Watzel *et al.* [10†] solely employed the smaller dataset TED-LIUM-v2 [138]. The results are presented in Table 5.3.

Based on the experience of the *Forward* and *Backward* training setups for AED models [12†], they assumed a similar performance divergence between the L2R SAED

Table 5.3: The results of the four SAED training procedures evaluated on the TED-LIUM-v2 dataset [138] and reported in WER (%).

TED-LIUM-v2 [138]				
	CHARs		BPE units	
Methods	dev	test	dev	test
Forward	18.28	16.27	16.08	14.13
Backward	23.12	17.75	24.33	19.39
Dual SAED Cos	17.85	15.31	15.14	13.62
Dual SAED Euc	**16.49**	**15.25**	**14.49**	**13.19**

models $\overrightarrow{\mathcal{H}_{\text{saed}}}$ and R2L SAED models $\overleftarrow{\mathcal{H}_{\text{saed}}}$. Their expectation was confirmed and summarized results in Table 5.3. If trained on time-reversed grapheme sequences, the R2L SAED models $\overleftarrow{\mathcal{H}_{\text{saed}}}$ experience a major decline compared to standard L2R SAED models. Even though SAED models possess global attention modules, which should be superior to the attention modules of AED models, they cannot deploy their advantages. Moreover, it remains unclear if SAED models would benefit from larger datasets and if the performance gap between $\overrightarrow{\mathcal{H}_{\text{saed}}}$ and $\overleftarrow{\mathcal{H}_{\text{saed}}}$ models would diminish.

Since the examination of the *Dual Decoder* training procedure solely returns minor improvements in AED models, Watzel *et al.* directly analyzed the impact of the regularization in the third *Dual SAED Cos* stage. In Table 5.3, the results of their regularization procedure are presented and reveal consistent improvements for L2R SAED models $\overrightarrow{\mathcal{H}_{\text{saed}}}$ if Cosine distance metrics $d_{\cos}$ are applied in the regularization terms Π.

Watzel *et al.* further improved the performance of the L2R SAED model $\overrightarrow{\mathcal{H}_{\text{saed}}}$ in the last *Dual SAED Euc* setup. The evaluation results are depicted in Table 5.3 and demonstrate that employing Euclidean distance metrics d_{euc} in the regularization term Π has the largest impact on L2R model structures. For CHAR graphemes, the superior models achieve relative WER improvements of 9.8% on the dev and 6.3% on the test sets. Similar relative WER reductions are confirmed for BPE graphemes, whereby Watzel *et al.* observed a decrease of 9.9% and 6.7% on the test and dev sets, respectively.

5.6 Conclusion

This chapter discussed the approaches from Watzel *et al.* [10†, 12†], which examined forward-backward learning strategies for AED and SAED architectures. Their exhaustive evaluation demonstrated that information from R2L models is transmittable to standard AED and SAED architectures if regularization terms are integrated into the total cost functions. The application of their regularization approach was straightforward for CHARs, as reversed CHAR sequences did not modify their sequence length, thus leading to standard and reversed CHAR sequences of even length. However, their regularization approach was not simply transferable for the popular BPE units, as the encoding procedure of time-reversed sequences did not necessarily lead to equally long

BPE sequences. Consequently, sequence mismatches for BPE units were noticeable and required special treatment. Therefore, Watzel *et al.* established methods for the odd BPE sequences by utilizing a differentiable version of the DTW algorithm. As a result, they could apply their regularization approach, despite appearing sequence mismatches in the training phase.

To evaluate the impact of R2L models on standard L2R architectures, Watzel *et al.* defined several training procedures on smaller and larger datasets. Their evaluation revealed that the influence of reversed model structures on L2R models was negligible if no regularization procedures were applied. However, the integration of regularization terms led to explicit improvements in the final decoding phase of the regularized L2R architectures. The results of the regularized AED and SAED models indicated that reversed structures were generally effective auxiliary functions, whereas utilizing Euclidean metrics returned slightly better results.

6

Localness and Fusion Strategies in SAED Models

In this chapter, the standard self-attention (SA) mechanism of self-attention-based encoder-decoder (SAED) architectures is extended by a novel localness and fusion strategy, which has been established by Watzel *et al.* [11†]. The first section introduces the standard scoring function in SAED, discusses its drawbacks, and outlines the contributions of their approach [11†]. Related approaches in neural machine translation (NMT) and automatic speech recognition (ASR) are explained in section two, and their different concepts are described. In the third section, the score function of SAED models is revised, and Gaussian masks for inducing local context are introduced. Finally, the section analyzes several fusion strategies, which allow efficient procedures for fusing the standard global scores with the novel local scores. The fourth section elaborates on the pre-processing of the dataset, the architecture of the enhanced SAED model, its optimization, and the decoding procedure to retrieve the final results. In the fifth section, two ablation studies are conducted to obtain the optimal fusion strategy and to determine the most effective location of localness in the encoder layers of SAED architectures. The chapter concludes by summarizing the contributions of Watzel *et al.* [11†].

6.1 Introduction

Nowadays, the SAED models based on self-attention (SA) mechanisms [159] are slowly but steadily replacing standard long short-term memory (LSTM) architectures. Over the years, multiple SAED and related approaches have been established [44, 59, 121, 149, 159]. The SA module prevents computationally expensive calculations of recurrent model structures and allows global attention on the entire sequence, leading to efficiently connected information located far apart.

Even though these SAED approaches returned remarkable results on multiple datasets, the SA core module is prone to suppress local context. Typically, the score matrix $\mathcal{S}_{\text{saed}}$ is calculated by (compare Section 5.3.2):

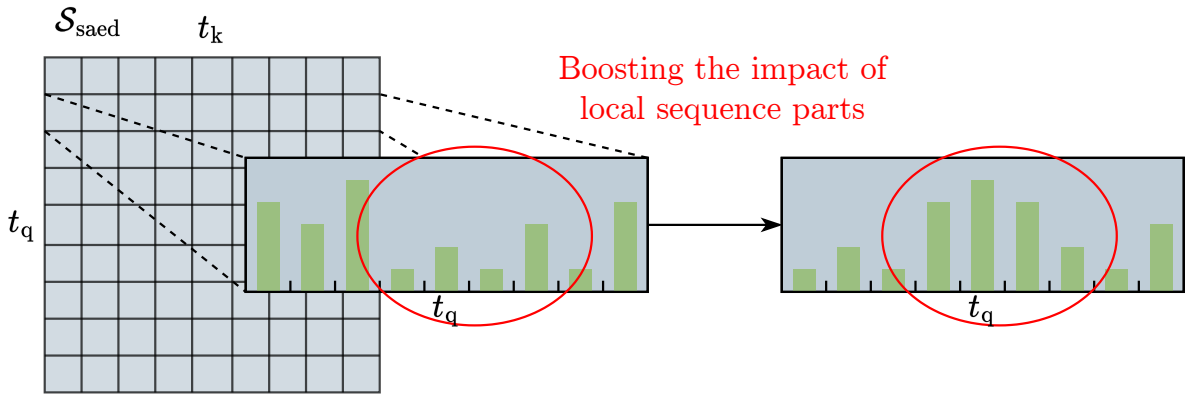


Figure 6.1: The concept of boosting the local context in the scoring function $\mathcal{S}_{\text{saed}}$, inspired by [174]. For an exemplary row in $\mathcal{S}_{\text{saed}}$, a local score matrix has been determined, which is then applied to modifying specific entries of $\mathcal{S}_{\text{saed}}$.

$$\mathcal{S}_{\text{saed}}\left(\boldsymbol{Q}, \boldsymbol{K}\right) = \frac{\boldsymbol{Q}\boldsymbol{K}^{\top}}{\sqrt{I_{\text{k}}}}, \tag{6.1}$$

where the optional mask $\boldsymbol{M}$ is excluded from the query matrix $\boldsymbol{Q} = \left[\boldsymbol{q}^{(1)}, \ldots, \boldsymbol{q}^{(t_{\text{q}})}, \ldots, \boldsymbol{q}^{(T_{\text{q}})}\right]$ with $\boldsymbol{Q} \in \mathbb{R}^{T_{\text{q}} \times I_{\text{model}}}$ consisting of the query vectors $\boldsymbol{q}^{(t_{\text{q}})} \in \mathbb{R}^{1 \times I_{\text{model}}}$ and the key matrix $\boldsymbol{K} = \left[\boldsymbol{k}^{(1)}, \ldots, \boldsymbol{k}^{(t_{\text{k}})}, \ldots, \boldsymbol{k}^{(T_{\text{k}})}\right]$ with $\boldsymbol{K} \in \mathbb{R}^{T_{\text{k}} \times I_{\text{model}}}$ containing the key vectors $\boldsymbol{k}^{(t_{\text{k}})} \in \mathbb{R}^{1 \times I_{\text{model}}}$. The result of the scoring operation $\mathcal{S}_{\text{saed}}$ can be considered as values, representing the relevance between $\boldsymbol{q}^{(t_{\text{q}})}$ and $\boldsymbol{k}^{(t_{\text{k}})}$ at the time steps $(t_{\text{q}}, t_{\text{k}})$. The softmax operation (see Equation (2.97)) normalizes the resulting score matrix $\mathcal{S}_{\text{saed}}$ along the entire sequence length T_{k}, leading to normalized relevance values with strong global dependency. The global scoring modules $\mathcal{S}_{\text{saed}}$ in the SA mechanisms seem superior to standard recurrent approaches, which are restricted to specific parts of the sequence (compare Section 5.2). However, they may also be suboptimal since the module does not consider local dependencies and equally attends to all input sequence positions [174]. In standard ASR approaches, the input feature sequences $\boldsymbol{X}$ and the output sequences $\boldsymbol{G}$ commonly contain strong local dependencies, which need to be considered to retrieve state-of-the-art (SOTA) ASR architectures.

Generally, the score functions $\mathcal{S}_{\text{aed}}$ can be modified into two distinct parts: Choosing proper local score functions with their corresponding positions and window sizes in which the impact of localness should be increased or selecting methods that efficiently combine the result of the original score function $\mathcal{S}_{\text{aed}}$ and the novel integrated local score function before further processed by the softmax operator ψ. Figure 6.1 depicts the outcome of these combined operations. In the red circle, a section of column vector in $\mathcal{S}_{\text{saed}}$ is chosen and modified by the values of the local score matrix in a window at a specific position, typically learned by a multilayer perceptron (MLP).

In the past, multiple approaches [97, 111, 145, 149, 174] have examined procedures to increase the impact of local dependencies. However, most of these approaches did not examine the effect of localness in ASR applications or chose simple methods to combine

the global and local dependencies. Watzel *et al.* [11†] transferred the concept of induced localness into SAED models and examined effective fusion strategies to combine the global and local score matrices. Their contributions in [11†] can be summarized into four parts:

- The transfer of the concept of local SA into ASR.
- The analysis of multiple fusion strategies for combining global and local score functions.
- The in-depth examination of the most effective location for applying localness in SAED architectures.
- Achieving SOTA results on the TED-LIUM-v2 dataset [138] by combining the most effective location of localness with the optimal fusion strategy.

6.2 Related Work

The concept of localness or local attention was introduced by Luong *et al.* [97] for NMT in 2015. They observed that obtaining each attention weight by the standard global attention operation in attention-based encoder-decoder (AED) models always required attending to all hidden representations, leading to computationally expensive approaches. Therefore, they established a local attention method, where they restricted the attention in each time step on small sections of the representations. The selected parts were chosen by centering a fixed-size Gaussian window around a specific time step, which is learned by MLPs. Even though the novel approach of localness returned promising results, the flexibility of the Gaussian windows was limited, as the window size was fixed. Moreover, it is questionable if multiplicative Gaussian masks define the optimal strategy, as their application leads to dominant local scores, which suppress valuable information about the global scores.

The first implementation of localness in SAED models was done in 2018 by Sperger *et al.* [149]. In contrast to the initial concept of Luong *et al.* [97], they added local score matrices to the standard global score to induce local context. Initially, they applied a hard masking procedure, where the local score matrices were constructed as inversely banded matrices. They also established a more flexible approach by utilizing soft Gaussian masks, where the location of the masks was deterministically set, and the window size was learned during optimization. The local Gaussian masks and the global attention scores were then fused by a simple summation; alternative fusion strategies were not further investigated.

In the same year, Yang *et al.* [174] proposed an entirely flexible Gaussian mask for NMT, where the positions and the sizes of the masks were learned by MLPs. The central positions of the Gaussian windows were predicted by MLPs based on the query vectors of the query matrix. For determining optimal window sizes, Yang *et al.* examined several strategies. The most straightforward strategy was based on the original concept

of localness, where the window size was fixed to heuristic values. An extension of the former strategy was introduced by layer-specific window sizes, which were learned by the mean of all key vectors in the current layer. The last strategy defined a highly flexible query-specific approach, where the flexible window sizes were independently learned for each query vector. All strategies were also examined in the multi-head attention (MHA) layers and in the context of finding the optimal location of localness in SAED architectures. Although Yang *et al.* verified that the performance of SAED models improved, they did not further analyze strategies for fusing the global and local attention scores and relied on the simple addition operation.

A more recent study for optimal fusion strategies for local and global scores in SAED models was carried out by Nguyen *et al.* [111] in NMT. Even though they did not apply Gaussian masks, they proposed two novel fusion procedures. The first method replaces the summation of the local masks and the global attention scores with multiplicative operators. The second method entirely separated the calculation of local and global scores, where the standard score function in each SA module was divided into local and global branches with their individual weights. The masks were then solely multiplied to the local score matrices and additively fused with the global scores. The outcome was employed in the SA module and finalized by the softmax function. Even though their novel fusion approach returned superior results, the regular addition equally weighted the output of the local and global model branches and did not apply a weighted addition procedure.

6.3 Proposed Method

Before introducing the novel localness and fusion strategy of Watzel *et al.* [11†], the standard score functions in SAED architectures are revised. Then, the popular Gaussian masks for inducing localness are introduced. The parameters of these masks are learned by standard MLPs, whose predictions are limited to the length of the query matrices. In the last part, several fusion strategies are defined, corresponding to procedures of related approaches or extensions proposed by Watzel *et al.* [11†].

6.3.1 Standard Score Functions in SAED Models

One of the crucial elements in the SA modules $\mathcal{A}_{\text{sa}}$ of SAED models refers to the scoring functions introduced in Section 5.3.2 in detail. Generally, multiple separated SA modules, also known as heads, are implemented as an MHA module. As the following approaches are seamlessly transferable to MHA applications, the revised SA $\mathcal{A}_{\text{sa}}$ modules are rewritten without specifying the current head to improve readability:

$$\mathcal{A}_{\text{sa}}\big(\boldsymbol{Q}\boldsymbol{W}_{\text{q}}, \boldsymbol{K}\boldsymbol{W}_{\text{k}}, \boldsymbol{V}\boldsymbol{W}_{\text{v}}\big) = \psi\big(\mathcal{S}_{\text{saed}}\big(\boldsymbol{Q}\boldsymbol{W}_{\text{q}}, \boldsymbol{K}\boldsymbol{W}_{\text{k}}\big)\big)\boldsymbol{V}\boldsymbol{W}_{\text{v}}, \tag{6.2}$$

where $\boldsymbol{W}_{\text{q}} \in \mathbb{R}^{I_{\text{model}} \times I_{\text{q}}}$, $\boldsymbol{W}_{\text{k}} \in \mathbb{R}^{I_{\text{model}} \times I_{\text{k}}}$, and $\boldsymbol{W}_{\text{v}} \in \mathbb{R}^{I_{\text{model}} \times I_{\text{v}}}$ refer to weight matrices. Similar to the standard scoring functions $\mathcal{S}_{\text{saed}}$ in the MHA, the corresponding weight

matrices are considered in the score function of each head:

$$\mathcal{S}_{\text{saed}}\big(\boldsymbol{Q}\boldsymbol{W}_{\text{q}}, \boldsymbol{K}\boldsymbol{W}_{\text{k}}\big) = \frac{\big(\boldsymbol{Q}\boldsymbol{W}_{\text{q}}\big)\big(\boldsymbol{K}\boldsymbol{W}_{\text{k}}\big)^{\top}}{\sqrt{I_{\text{k}}}}, \tag{6.3}$$

leading to the standard score output $\mathcal{S}_{\text{saed}} \in \mathbb{R}^{T_{\text{q}} \times T_{\text{k}}}$ after calculating the dot-product of the query $\boldsymbol{Q} \in \mathbb{R}^{T_{\text{q}} \times I_{\text{model}}}$ and key $\boldsymbol{K} \in \mathbb{R}^{T_{\text{k}} \times I_{\text{model}}}$ matrices. In the standard configuration, the scoring function $\mathcal{S}_{\text{saed}}$ treats all queries $\boldsymbol{q}^{(t_{\text{q}})}$ and keys $\boldsymbol{k}^{(t_{\text{k}})}$ as equally important and does not consider any local dependencies between neighboring queries or keys. Even though these global assumptions allow an exchange of information between far-away elements of sequences, it can be problematic for ASR models. For instance, sequence sections that contain silence with lower information are similarly treated as standard sections, which include regular speech.

6.3.2 Gaussian Masks for Inducing Localness

In order to achieve localness in SAED architectures, Watzel *et al.* [11†] followed the approach of Yang *et al.* [174], who established flexible Gaussian masks $\boldsymbol{G}$:

$$g_{t_{\text{q}},t_{\text{k}}} = -\frac{\big(t_{\text{k}} - p^{(t_{\text{q}})}\big)^2}{2{\sigma^{(t_{\text{q}})}}^2} \quad \forall t_{\text{q}} \in \{1, 2, \cdots, T_{\text{q}}\}, \quad \forall t_{\text{k}} \in \{1, 2, \cdots, T_{\text{k}}\}, \tag{6.4}$$

where $\boldsymbol{G} \in \mathbb{R}_{-}^{T_{\text{q}} \times T_{\text{k}}}$, and $\sigma^{(t_{\text{q}})} \in \mathbb{R}$ is implemented as $\sigma^{(t_{\text{q}})} = \frac{d^{(t_{\text{q}})}}{2}$. The parameters $p^{(t_{\text{q}})} \in \mathbb{R}$ for the central position and $d^{(t_{\text{q}})} \in \mathbb{R}$ for the size of the window allow a modification of the Gaussian mask $\boldsymbol{G}$. The parameters $p^{(t_{\text{q}})}$ and $d^{(t_{\text{q}})}$ are commonly adjusted by the trainable parameters $\widetilde{p}^{(t_{\text{q}})} \in \mathbb{R}$ and $\widetilde{d}^{(t_{\text{q}})} \in \mathbb{R}$, which are learned by MLPs in training:

$$\begin{pmatrix} p^{(t_{\text{q}})} \\ d^{(t_{\text{q}})} \end{pmatrix} = T_{\text{q}} \operatorname{sigmoid}\begin{pmatrix} \widetilde{p}^{(t_{\text{q}})} \\ \widetilde{d}^{(t_{\text{q}})} \end{pmatrix} \quad \forall t_{\text{q}} \in \{1, 2, \cdots, T_{\text{q}}\}. \tag{6.5}$$

The combination of the parameter T_{q} and the sigmoid activation function ensures that the predicted parameters $\widetilde{p}^{(t_{\text{q}})}$, for the central position, and $\widetilde{d}^{(t_{\text{q}})}$, for the window size, are generated by Gaussian masks $\boldsymbol{G}$ located inside the dimensions of the global attention score matrix $\mathcal{A}_{\text{saed}}$.

Central Position Prediction for Gaussian Masks Yang *et al.* [174] predicted the central positions $\widetilde{p}^{(t_{\text{q}})}$ by applying two-layer MLPs, whereby their predictions relied on the transposed query vectors $\boldsymbol{q}^{(t_{\text{q}})}$ in $\boldsymbol{Q}$:

$$\widetilde{p}^{(t_{\text{q}})} = \boldsymbol{W}_{\text{p}}^{[2]} \tanh\big(\boldsymbol{W}_{\text{p}}^{[1]} \boldsymbol{q}^{(t_{\text{q}})\top}\big) \quad \forall t_{\text{q}} \in \{1, 2, \cdots, T_{\text{q}}\}, \tag{6.6}$$

where $\boldsymbol{W}_{\text{p}}^{[1]} \in \mathbb{R}^{I_{\text{model}} \times I_{\text{model}}}$ and $\boldsymbol{W}_{\text{p}}^{[2]} \in \mathbb{R}^{1 \times I_{\text{model}}}$ represent weight matrices. Notice that it is impractical to utilize key vectors $\boldsymbol{k}^{(t_{\text{k}})}$ for predicting $\widetilde{p}^{(t_{\text{q}})}$, as the predictions in the cross SA module (compare the unmasked MHA in the decoder of the SAED model in Figure 5.5) would lead to Gaussian masks $\boldsymbol{G}$ located outside of $\mathcal{A}_{\text{saed}}$ as $T_{\text{k}} > T_{\text{q}}$.

Window Size Prediction for Gaussian Masks Watzel *et al.* [11†] examined multiple input procedures in order to predict the optimal window size $\widetilde{d}^{(t_\mathrm{q})}$. Inspired by Yang *et al.* [174], they also analyzed the impact of fixed window sizes $\widetilde{d}^{(t_\mathrm{q})}$, either heuristically defined or determined by the mean key vector $\overline{\boldsymbol{k}}$ over the entire key matrix $\boldsymbol{K}$ for a layer-specific approach. Since these procedures did not lead to satisfying results and seemed to limit the flexibility of Gaussian masks $\boldsymbol{G}$, Watzel *et al.* [11†] decided to employ a query-specific approach similar to [174]:

$$\widetilde{d}^{(t_\mathrm{q})} = \boldsymbol{W}_\mathrm{d}^{[2]} \tanh\left(\boldsymbol{W}_\mathrm{p}^{[1]} \boldsymbol{q}^{(t_\mathrm{q})\top}\right) \quad \forall t_\mathrm{q} \in \{1, 2, \cdots, T_\mathrm{q}\}, \tag{6.7}$$

where $\boldsymbol{W}_\mathrm{p}^{[1]} \in \mathbb{R}^{I_\mathrm{model} \times I_\mathrm{model}}$ and $\boldsymbol{W}_\mathrm{d}^{[2]} \in \mathbb{R}^{1 \times I_\mathrm{model}}$ specify weight matrices. Notice that the resulting hidden representations of $\boldsymbol{W}_\mathrm{p}^{[1]} \boldsymbol{q}^{(t_\mathrm{q})\top}$ are reused from Equation (6.6) above, as these contain prior knowledge about the location of the window, leading to more precise window size predictions $\widetilde{d}^{(t_\mathrm{q})}$. Besides, the computational costs are reduced, resulting in more efficient architectures.

6.3.3 Fusion Strategies for Global and Local Attention Scores

The optimal fusion strategy of localness, either in the simplest form of Gaussian masks $\boldsymbol{G}$ or in entirely separated model structures, is essential to being effective in the SA modules of SAED architectures. Therefore, Watzel *et al.* [11†] revised several fusion strategies of related approaches and proposed an extension of the existing ones.

Biased Fusion Strategy The most straightforward fusion strategy can be considered if the global score functions $\mathcal{A}_\mathrm{saed}$ represent weight matrices in linear transformations, whose bias terms are represented by local Gaussian masks $\boldsymbol{G}$:

$$\mathcal{S}_\mathrm{saed}\big(\boldsymbol{Q}\boldsymbol{W}_\mathrm{q}, \boldsymbol{K}\boldsymbol{W}_\mathrm{k}\big) = \frac{\big(\boldsymbol{Q}\boldsymbol{W}_\mathrm{q}\big)\big(\boldsymbol{K}\boldsymbol{W}_\mathrm{k}\big)^\top}{\sqrt{I_\mathrm{k}}} + \boldsymbol{G}. \tag{6.8}$$

This procedure was already successfully applied by Sperger *et al.* [149], Yang *et al.* [174], and Lohrenz *et al.* [94].

Global and Local Score Fusion Strategy The next fusion strategy is motivated by the novel fusion approach from Nguyen *et al.* [111] in NMT. Nguyen *et al.* introduced separated global scores:

$$\mathcal{S}_\mathrm{glob} = \big(\boldsymbol{Q}\boldsymbol{W}_\mathrm{q,glob}\big)\big(\boldsymbol{K}\boldsymbol{W}_\mathrm{k,glob}\big)^\top, \tag{6.9}$$

and local scores:

$$\mathcal{S}_\mathrm{loc} = \big(\boldsymbol{Q}\boldsymbol{W}_\mathrm{q,loc}\big)\big(\boldsymbol{K}\boldsymbol{W}_\mathrm{k,loc}\big)^\top \odot \boldsymbol{G}. \tag{6.10}$$

Separate model branches generate both scores with individual weight matrices $\boldsymbol{W}_\mathrm{q,glob} = \boldsymbol{W}_\mathrm{q}$ and $\boldsymbol{W}_\mathrm{k,glob} = \boldsymbol{W}_\mathrm{k}$ for the standard global score $\mathcal{S}_\mathrm{saed}$, and individual weight matrices

$\boldsymbol{W}_{\mathrm{q,loc}} \in \mathbb{R}^{I_{\mathrm{model}} \times I_{\mathrm{q}}}$, $\boldsymbol{W}_{\mathrm{k,loc}} \in \mathbb{R}^{I_{\mathrm{model}} \times I_{\mathrm{k}}}$ for the local scores $\mathcal{S}_{\mathrm{loc}}$. Notice that Watzel *et al.* replaced the initially proposed differentiable masks in [111] with Gaussian masks $\boldsymbol{G}$. The global and local scores were then fused by simple addition, leading to the final scoring functions $\mathcal{S}_{\mathrm{saed}}$:

$$\mathcal{S}_{\mathrm{saed}}\big(\boldsymbol{Q}, \boldsymbol{K}\big) = \frac{\mathcal{S}_{\mathrm{glob}} + \mathcal{S}_{\mathrm{loc}}}{\sqrt{I_{\mathrm{k}}}}, \tag{6.11}$$

and enable the corresponding models to incorporate more global or local context into their final score function $\mathcal{S}_{\mathrm{saed}}$, achieved by individual weight matrices for each score.

Adjustable Global and Local Score Fusion Strategy Although the previous strategy led to a more flexible model, Watzel *et al.* [11†] proposed an extended method. They argued that the global score $\mathcal{S}_{\mathrm{glob}}$ and the local score $\mathcal{S}_{\mathrm{loc}}$ are still equally weighted in the summation and introduced a trainable parameter α, for weighting the relevance of local and global scores:

$$\mathcal{S}_{\mathrm{saed}}\big(\boldsymbol{Q}, \boldsymbol{K}\big) = \frac{\alpha\mathcal{S}_{\mathrm{glob}} + \big(1-\alpha\big)\mathcal{S}_{\mathrm{loc}}}{\sqrt{I_{\mathrm{k}}}}. \tag{6.12}$$

The weighting parameter α is predicted by two-layer MLPs, which are fed by the mean of the key vectors $\overline{\boldsymbol{k}} \in \mathbb{R}^{I_{\mathrm{model}}}$ [174]:

$$\alpha = \mathrm{sigmoid}\big(\boldsymbol{W}_{\alpha}^{[2]} \tanh\left(\boldsymbol{W}_{\alpha}^{[1]} \overline{\boldsymbol{k}}^{\top}\right)\big) \tag{6.13}$$

where $\boldsymbol{W}_{\alpha}^{[1]} \in \mathbb{R}^{I_{\mathrm{model}} \times I_{\mathrm{model}}}$ and $\boldsymbol{W}_{\alpha}^{[2]} \in \mathbb{R}^{1 \times I_{\mathrm{model}}}$ are weight matrices, and the activation functions are set to sigmoid functions.

6.4 Experimental Setup

The following section is divided into four parts which partly overlap with the sections of the last chapters. In the first section, the pre-processing procedure of the TED-LIUM release 2 (TED-LIUM-v2) dataset is revised and slightly modified. Then, the enhanced SAED models of Watzel *et al.* [11†] are defined, which are utilized for analyzing their localness and fusion strategies. The third part discusses the optimization of the SAED architecture. The section is concluded by specifying the parameter setup for the decoding phase.

6.4.1 Pre-Processing of the TED-LIUM-v2 Dataset

Watzel *et al.* [11†] followed the pre-processing procedures described in Section 5.4.1. In the first step, the speed perturbation augmentation technique [77] on the TED-LIUM-v2 dataset is applied, where the speed factors of the recorded speech signal are modified to 0.9, 1.0, and 1.1 before generating the input features. Based on these three augmented

speech signals, 80-dimensional log Mel features and three-dimensional pitch features [103] are extracted, which are utilized as the input features $\boldsymbol{x}^{(t)} \in \mathbb{R}^{83}$.

The corresponding target grapheme sequences $\boldsymbol{G}$ are generated from the given transcript of the TED-LIUM-v2 dataset. First, each target sequence is enclosed by the auxiliary tokens <sos> and <eos> to specify the start and the end of the sequence. Then, the tokenizer SentencePiece [79] is employed to generate the 500 optimal byte pair encoding (BPE) [144] units, resulting in sequences with target vectors $g_{\text{bpe}}^{(o)} \in \mathbb{B}^{500}$.

6.4.2 The Standard Architecture of SAED Models

The SAED model utilized by Watzel *et al.* [11†] corresponds to similar architectures established in Section 5.4.3 with minor modifications. The feature sequences $\boldsymbol{X}$ are pre-processed with convolutional front-end modules in order to lower their overall sequence length. The module is designed as two-layer convolutional neural networks (CNNs) with a fixed stride length of two and whose kernels are set to $\boldsymbol{K}^{[1]} \in \mathbb{R}^{1\times 83\times 3\times 3}$ and $\boldsymbol{K}^{[2]} \in \mathbb{R}^{83\times 83\times 3\times 3}$. Each CNN layer is finalized by rectified linear unit (ReLU) activation functions, whereby the output activations of the last CNN layer are vectorized following the procedure in Section 2.2.4. The resulting high-dimensional vectors are then fed to MLPs, which linearly transform these into the I_{model}-dimensional input vectors. Since SAED models are not utilizing recurrent neural networks (RNNs) or CNNs, the positional encoding function proposed in [159] is added, which includes relative positional information of each input vector (compare Section 5.4.3).

The resulting input vectors are then processed in the encoder networks $\mathcal{H}_{\text{enc}}$, consisting of $L_{\text{enc}} = 12$ layers $\boldsymbol{H}_{\text{enc}}^{[l_{\text{enc}}]}$, whereby 2048 units are utilized in each layer. Watzel *et al.* [11†] set the number of heads H to four in the entire attention network $\mathcal{A}_{\text{saed}}^{[l_{\text{enc}}]}$ without applying the optional mask $\boldsymbol{M}$ and specified an internal dimension of $I_{\text{model}} = 256$ for the overall SAED architecture. The hidden representations $\boldsymbol{H}_{\text{enc}}^{[12]}$ of the last encoder layer are further processed in the decoder networks $\mathcal{H}_{\text{dec}}$. Additionally, $\boldsymbol{H}_{\text{enc}}^{[12]}$ is fed into MLPs, which linearly map the representations into the grapheme dimension $\mathbb{R}^{83}$. Then, the output is transformed into the connectionist temporal classification (CTC) grapheme distribution $\boldsymbol{g}_{\text{CTC}}^{(o)}$, required in the overall cost function $\widetilde{C}_{\text{saed}}$.

The decoder networks $\mathcal{H}_{\text{dec}}$ are composed of similar parameter setups. However, the number of decoder layers is set to $L_{\text{dec}} = 6$, and optimal $\boldsymbol{M}$ in the form of lower triangular matrices prevents the attention modules $\mathcal{A}_{\text{sa}}^{[l_{\text{dec}}][2]}$ from accessing future predicted grapheme sequences $\boldsymbol{G}_{o+1}^{O}$. The final decoder layer $\boldsymbol{H}_{\text{dec}}^{(L_{\text{dec}})}$ is finalized by a layer normalization (LN) and a single-layer linear MLP, whose output is transformed into the grapheme prediction distribution $\boldsymbol{g}^{(o)}$ by softmax functions. Before the predictions $\boldsymbol{g}^{(o)}$ are fed back to the decoder networks $\mathcal{H}_{\text{dec}}$, the positional encoding needs to be appended to integrate relative temporal information. In order to prevent overfitting, the SAED architecture is regularized by dropout layers [66] with $\rho = 0.1$ following each $\mathcal{A}_{\text{sa}}$ and $\mathcal{A}_{\text{saed}}$ module.

6.4.3 Optimization of SAED Models

In the optimization phase, Watzel *et al.* [11†] augmented the input features $\boldsymbol{X}$ by the SpecAugment augmentation method [116] to increase the number of training samples and reduce the impact of overfitting. The uniform distributions for the time masking deformation are bounded by $T_\Delta = 40$, limiting the maximum length of the time masks, and by $N_{\text{tm}} = 2$, specifying the maximum repetitions of the time masking procedure. Similarly, the uniform distribution for the length of the frequency mask is limited to $T_\Delta = 30$, and the frequency masking procedure is repeated for a maximum of $N_{\text{fm}} = 2$ times. Additionally, the target graphemes $\boldsymbol{g}^{(o)}$ are smoothed with the label smoothing method established in Equation (2.127), whereby the smoothing values are set to $\iota = 0.1$.

The SAED models are trained based on the scheme introduced in Section 5.4.5. The Adam optimizer [76] applies a learning rate schedule with a warmup phase of 25 000 iteration steps, where the learning rate η is slowly and linearly increased before passing over into the training phase with gradually decreasing learning rates η. Furthermore, the standard Kullback-Leibler divergence (KLD) cost function is utilized as a training objective with a mini-batch size of $B = 50$ and further extended the cost by the CTC cost functions C_{CTC} with a weighting parameter $\beta_{\text{saed}} = 0.3$, leading to the overall cost function $\widetilde{C}_{\text{saed}}$:

$$\widetilde{C}_{\text{saed}}\big(\boldsymbol{G}_{\mathcal{B}}, \boldsymbol{G}_{\text{CTC},\mathcal{B}}, \hat{\boldsymbol{G}}_{\mathcal{B}},\big) = \big(1-\beta_{\text{saed}}\big) C_{\text{KLD}}\big(\boldsymbol{G}_{\mathcal{B}}, \hat{\boldsymbol{G}}_{\mathcal{B}}\big) + \beta_{\text{saed}} C_{\text{CTC}}\big(\boldsymbol{G}_{\text{CTC},\mathcal{B}}, \hat{\boldsymbol{G}}_{\mathcal{B}}\big). \quad (6.14)$$

6.4.4 Decoding Procedure for Standard SAED Models

In the decoding phase, the standard grapheme prediction $\boldsymbol{g}^{(o)}$ and CTC grapheme prediction $\boldsymbol{g}^{(o)}_{\text{CTC}}$ are combined by the weighting factors of $\beta_{\text{saed}} = 0.3$, language models (LMs) are omitted, and the beam size is assigned to $K_{\text{best}} = 20.0$.

6.5 Evaluation

Yang *et al.* [174] and Nguyen *et al.* [111] reported improvements in applying and fusing local attention scores in NMT. Therefore, Watzel *et al.* [11†] defined two ablation studies to examine the impact of different fusion strategies and to determine the most effective location of localness in the SAED architectures. Based on the results of these studies, the enhanced SAED architectures are retrained and compared to current SOTA architectures. Finally, qualitative results are analyzed in detail to observe the modification of standard global scores by the localness procedure.

6.5.1 Ablation Study on Optimal Fusion Strategies

In Table 6.1, the results of the first ablation study are summarized in which the impact of the predefined fusion strategies is analyzed. The local scores are solely integrated into

Table 6.1: Ablation study on optimal fusion strategies in SAED models. The results are reported in word error rates (WERs) (%) and evaluated on TED-LIUM-v2 dataset[138].

TED-LIUM-v2		
Model	dev	test
Baseline without Localness	10.0	9.2
+ Bias Fusion Strategy [174]	10.1	9.0
+ Global and Local Score Fusing Strategy	10.3	**8.8**
+ Adjustable Global and Local Score Fusing Strategy	**9.8**	9.1

the encoder networks $\mathcal{H}_{\text{enc}}$, which returned superior results for SAED models in recent studies [111, 174].

The *Baseline without Localness* setup specifies the baseline of the ablation study, where standard SAED architectures are optimized for 50 epochs following the training scheme above. The decoding procedure yields 10.0% and 9.2% WERs for the dev and test set, respectively. These results are slightly worse than the SOTA results reported in the ESPnet framework, as the SAED models are solely optimized for 50 epochs and decoded with a reduced beam size of $K_{\text{best}} = 20.0$ caused by time constraints.

The more flexible *Bias Fusion Strategy* setup incorporates the Gaussian mask $\boldsymbol{G}$, initially established in [174]. Although Yang *et al.* reported improvements in integrating such flexible masks into their approach, Watzel *et al.* [11†] did not observe consistent reductions in the WERs. On the test set, a WER reduction from 9.2% to 9.0% is observable. However, similar declines are not returned for the dev set, where the WER is slightly increased from 10.0% to 10.1%. Watzel *et al.* [11†] assumed that including localness with Gaussian masks could be problematic, as these masks are always active even if localness is not intended.

The *Global and Local Score Fusion Strategy* defines an extension of former strategies as the generation process of the local scores $\mathcal{S}_{\text{loc}}$ and global scores $\mathcal{S}_{\text{glob}}$ is achieved by separate network branches [11†]. Therefore, the SAED architectures can focus either solely on the local, global, or both branches equally. Besides, the Gaussian mask $\boldsymbol{G}$ is solely multiplied by the local scores $\mathcal{S}_{\text{loc}}$, hence leaving the standard global attention $\mathcal{A}_{\text{saed}}$ unaltered. Although applying such a fusion strategy reduces the WER from 9.2% to 8.8% on the test set, the overall strategy remains inconsistent, as similar improvements are not noticeable on the dev set, where the WER rises from 10.0% to 10.3%. Watzel *et al.* [11†] suspected the equally weighted summation of $\mathcal{S}_{\text{loc}}$ and $\mathcal{S}_{\text{glob}}$ for leading to these inconsistent results since neither of these scores could be independently active to represent the final score of the SA module $\mathcal{A}_{\text{saed}}$.

Watzel *et al.* [11†] suggested a solution for the previous issue in the *Adjustable Global and Local Score Fusion Strategy* setup. The SAED architectures are enhanced by also predicting a relevance parameter α based on the mean key $\overline{\boldsymbol{k}}$ of the current layer for allowing the model to decide if either the local score $\mathcal{S}_{\text{loc}}$ or the global $\mathcal{S}_{\text{glob}}$ is more relevant in the SA module $\mathcal{A}_{\text{saed}}$. As a result, consistent improvements on the dev and

Table 6.2: Ablation study on the optimal location of localness in the encoder layers of SAED models. The results are listed in WERs (%) and evaluated on the TED-LIUM-v2 dataset [138].

TED-LIUM-v2		
Model	dev	test
Baseline without localness	10.0	9.2
+ Adjustable Global and Local Score Fusing Strategy in Layer 1-3	10.1	**8.8**
+ Adjustable Global and Local Score Fusing Strategy in Layer 1-6	10.2	8.9
+ Adjustable Global and Local Score Fusing Strategy in Layer 1-9	10.0	**8.8**
+ Adjustable Global and Local Score Fusing Strategy in Layer 1-12	**9.8**	9.1

test sets are observed, and the WERs on the dev, and on the test set, are reduced from 10.0% to 9.8% and 9.2% to 9.1%, respectively.

6.5.2 Ablation Study on the Optimal Location of Localness

In the second ablation study, Watzel *et al.* [11†] examined the most effective location of the *Adjustable Global and Local Score Fusion Strategy* in the encoder networks $\mathcal{H}_{\text{enc}}$ to improve the performance of SAED models. They expected that inducing localness in the lowest layers should lead to significant improvements, as local information is commonly present in lower layers of deep neural networks (DNNs). Therefore, four setups are defined to analyze the impact of integrating localness and subsequent fusion strategy into the SA $\mathcal{A}_{\text{saed}}$ modules: Applying the strategy in the first three, six, nine, and all encoder layers, respectively.

The results are depicted in Table 6.2 and demonstrate that three out of four experiments lead to inconsistent improvements in SAED models. If localness is employed in the first three layers, a WER decrease from 9.2% to 8.8% on the test set and a minor increase from 10.0% to 10.1% on the dev set is observed. Similar inconsistent results are noticed by utilizing the novel fusion strategy in the first six layers. Here, the performance on the test set improves from 9.2% to 8.9% but deteriorates from 10.0% to 10.2% on the dev set. A more consistent improvement is recognizable for deploying localness in the first nine encoder layers. The WER decreases from 9.2% to 8.8% on the test set and remains stable with a 10.0% WER on the dev set. The most consistent improvements are returned for utilizing the fusion strategy in all encoder layers, where a WER decline from 9.2% to 9.1% on the test set and a minor decline from 10.0% to 9.8% on the dev set is observable.

Interestingly, Watzel *et al.* [11†] could not verify the consistent improvements reported in NMT [111, 174]. They assumed that despite further pre-processing of the input sequence $\boldsymbol{X}$ by front-end modules, the sequence length remained multiple times the length of the target grapheme sequence $\hat{\boldsymbol{G}}$ and made it challenging to efficiently integrate localness even in lower layers of the encoder networks $\mathcal{H}_{\text{enc}}$.

Table 6.3: The final WER (%) results of the current SOTA model and the enhanced SAED model with optimal parameters. The evaluation is executed on TEDLIUMv2 [138], and both models are decoded with a beam size of 40.

TED-LIUM-v2			
Model	#Param	dev	test
Baseline ESPnet	28 M	10.1	8.9
+ Adjustable Global and Local Score Fusing Strategy	29 M	10.0	8.7

6.5.3 Final Results of Enhanced SAED Models

In order to compare the novel *Adjustable Global and Local Score Fusion Strategy* to recent SOTA results listed in the ESPnet framework [162] (commit c881192), Watzel *et al.* [11†] modified the training parameters of their enhanced SAED architecture to correspond to the parameter setup of the SOTA model, leading to optimization for 100 epochs, a decoding procedure with a beam size of $K_{\text{best}} = 40.0$, and a minor increase of the model parameters from 28 M to 29 M.

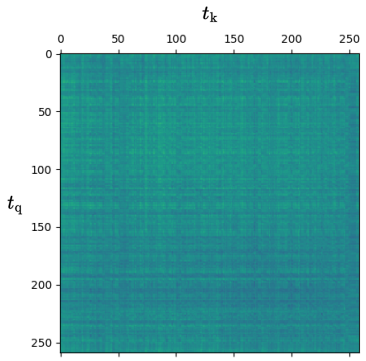
The results in Table 6.3 reveal that the enhanced model structure surpasses current SOTA models by reducing the WER from 8.9% to 8.7% on the test and from 10.1% to 10.0% on the dev set. The novel approach from Watzel *et al.* [11†] demonstrates the advantage of utilizing localness in SAED models and why it should be considered in future ASR models.

6.5.4 Qualitative Results of Local and Global Scores

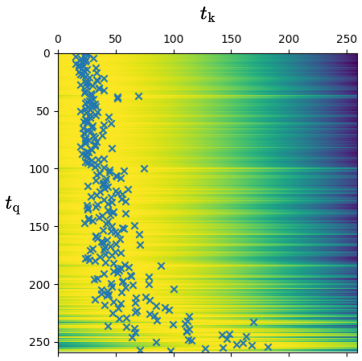
Besides the quantitative results, the qualitative results are depicted in Figure 6.2. Brighter colors define higher value scores, whereby darker colors specify lower value scores. The first row presents the generation of the local scores $\mathcal{S}_{\text{loc}}$. In the left Figure 6.2a, the local scores are generated in the local attention branch, relying on their individual weight matrices, and do not include the multiplicative Gaussian masks $\boldsymbol{G}$. In the middle Figure 6.2b, the Gaussians for each $\boldsymbol{q}^{(t_{\text{q}})\top}$ are plotted, leading to the Gaussian mask $\boldsymbol{G}$.

Additionally, the predicted positions $p^{(t_{\text{q}})}$ marked by blue crosses and window sizes $d^{(t_{\text{q}})}$ colored in yellow are included. The Gaussian mask $\boldsymbol{G}$ is then multiplied by the local scores $\mathcal{S}_{\text{loc}}$, resulting in heavily smoothed local scores $\mathcal{S}_{\text{loc}}$, which are presented in Figure 6.2c.

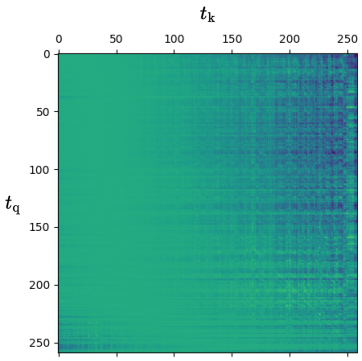
In the second row, the global score $\mathcal{S}_{\text{glob}}$ generated by the global attention branch is presented in Figure 6.2d. This score corresponds to the standard score of SAED models without integrating localness or applying fusion strategies. The Figure 6.2e reveals the final score $\mathcal{S}_{\text{saed}}$ of the fusion procedure established by Watzel *et al.* [11†]. Compared to the global score $\mathcal{S}_{\text{glob}}$, some sections in $\mathcal{S}_{\text{saed}}$ are boosted by the induced localness, which are normally suppressed.



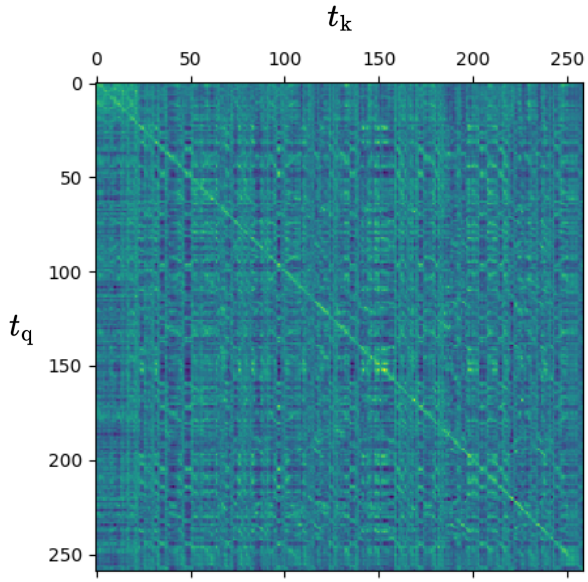
(a) $\mathcal{S}_{\text{loc}}$ without the Gaussian mask $\boldsymbol{G}$.



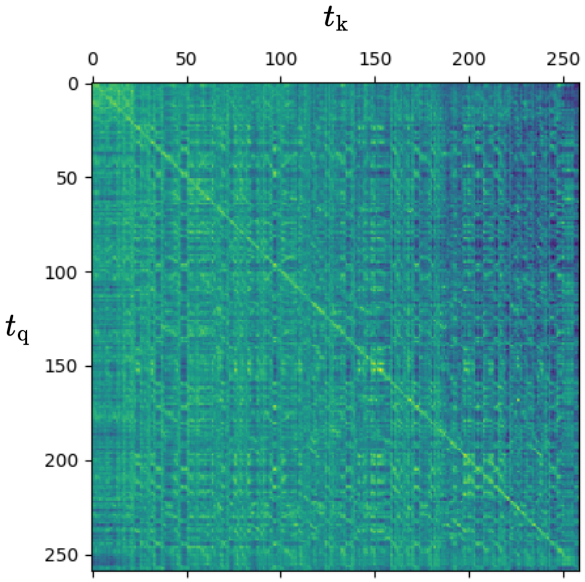
(b) The obtained Gaussian mask $\boldsymbol{G}$ with its marked $p^{(t_q)}$ and $d^{(t_q)}$.



(c) The local score $\mathcal{S}_{\text{loc}}$ with multiplied $\boldsymbol{G}$.



(d) The global score $\mathcal{S}_{\text{glob}}$ without fusion.



(e) The final attention score $\mathcal{S}_{\text{saed}}$ represents the fusion of the scores $\mathcal{S}_{\text{glob}}$ and $\mathcal{S}_{\text{loc}}$ with $\alpha = 0.423$.

Figure 6.2: The *Adjustable Global and Local Score Fusion Strategy* with the final attention score of $\mathcal{A}_{\text{saed}}$ adapted from [11†]. The global score $\mathcal{S}_{\text{glob}}$ and the local $\mathcal{S}_{\text{loc}}$ are determined and weighted by the parameter α. Relevant sections of the final attention score in Figure 6.2e are boosted by $\mathcal{S}_{\text{loc}}$.

6.6 Conclusion

This chapter introduced a novel approach from Watzel *et al.* [11†] for inducing and fusing localness into standard SAED architectures. Based on the popular Gaussian masks, the novel fusion strategy successfully transferred the concept of localness to SAED models in ASR, leading to a flexible fusion procedure.

The superiority of the novel fusion strategy was examined in an ablation study, which evaluated the novel strategy against current fusion methods. In contrast to localness in NMT applications, which heavily improved models if applied in the lowest encoder

layers, Watzel *et al.* [11†] could not verify such findings in ASR models.

In their evaluation of the novel fusion strategy, they could also achieve current SOTA results on the TED-LIUM-v2, which required a minor parameter increment.

7

Conclusion

This work presents three model enhancements for popular model architectures in automatic speech recognition (ASR), beginning with classic hybrid approaches in Chapter 4, followed by attention-based encoder-decoder (AED) models in Chapter 5, and concluding with self-attention-based encoder-decoder (SAED) architectures in Chapter 6. The model enhancements were initially proposed by Watzel *et al.* and are defined at different locations in the training setups, such as in standard cost functions by introducing novel cost sampling procedures, in given model architectures by adding additional network components, or in the network layers by modifying low-level functions. The objectives from Section 1.2 are briefly revised before the conclusions w.r.t. each objective are drawn:

1. Discrete acoustic models (AMs) have the potential to surpass continuous AMs in hybrid approaches.

2. Additional model components in AED and SAED models, which are trained on time-reversed target labels, improve the model performance.

3. Inducing localness with effective fusion strategies between local and global contexts leads to a performance gain in SAED architectures.

In the first model enhancement from Chapter 4, discrete AMs are defined as deep neural network quantizers (DNNQs). They emit discrete class labels in quantization procedures, where the mathematical max operators are utilized on the outputs of DNNQs. As these operators are non-differentiable, such models cannot be optimized by the backpropagation algorithm. Therefore, the max operators are replaced with scaled softmax operators to allow the standard backpropagation approach. In contrast to the initially established neural network vector quantizer (NNVQ) architectures, DNNQ models are implemented in state-of-the-art (SOTA) training frameworks and employ recent regularization techniques such as batch normalization (BN) and dropout layers for better generalization.

Furthermore, DNNQ models are required to emit a certain number of unique class labels to compete with continuous AMs, represented as Gaussian mixture models (GMMs). The number of unique labels is generally limited to the number of unique classes in a given dataset. The novel cost sampling procedure defines a mapping approach for setting

arbitrary output sizes in DNNQs, leading to discrete AMs competitive with continuous AMs. The optimal DNNQ parameters are determined by two ablation studies. The first study examines the optimal output size of DNNQ models, and the second study obtains the ideal number of adjacent input features, which should be spliced together into a single feature input. In a final evaluation, these optimal parameters are applied and lead to discrete DNNQ models surpassing continuous GMMs despite the major disadvantage of losing information in the quantization process of DNNQs.

The second model enhancement in Chapter 5 utilizes the popular AED architectures, consisting of the encoder, decoder networks, and attention modules, where the attention modules learn to weigh the impact of hidden representations of the encoder networks and decoder networks. The AED models are modified in five different setups and evaluated on a smaller and larger dataset, where the standard character (CHAR) and byte pair encoding (BPE) units are employed as target label types. The first setup defines a standard AED model setup, where the architecture is trained on both datasets, and their results specify the baseline of subsequent experiments. In the second setup, the AED models are optimized on time-reverse target labels, which leads to a minor performance decrease. The third setup, in which a pre-trained encoder network is applied, further decreases the model performance, as the encoder network is initially trained on standard target labels. In the fourth setup, the AED architecture is extended by additional decoder networks, which are solely active in training. The obtained AED models are decoded by excluding the additional model structures, leading to minor performance improvements. The best results are reported in the fifth setup, where information exchange between both decoder networks is enforced by introducing regularization terms for minimizing the distances between the decoder sequence outputs. As the regular and time-reversed BPE target sequences do not match their length, a novel regularization term based on the soft-dynamic time warping (DTW) algorithm is established, which enables the minimization of sequences with uneven lengths.

Additionally, the superiority of the model extension is also verified for SAED models. In contrast to additional decoder structures in AED models, the SAED model approach is extended by another entire SAED model, where the regular SAED model is trained on standard target labels and the additional SAED model on time-reversed target labels. Furthermore, the impact of Euclidean and Cosine distance metrics in the regularization procedures are examined and revealed that the Euclidean distance metric returns the best results for the two target label types CHAR and BPE units.

The third model enhancement described in Chapter 6 investigates the impact of local context and the corresponding fusion strategies in SAED architectures. The SAED models belong to the most popular architectures in ASR, as they enable the modeling of AMs without any recurrent or convolutional model components. Their self-attention (SA) modules are commonly extended into multi-head attention (MHA) modules to learn global dependencies between features or hidden representations whose locations in the sequences are far apart. However, local context is often suppressed by SA modules despite playing a major role in the recognition process of ASR models. Therefore, the standard scores of SAED architectures are modified by adding local Gaussian masks, whose parameters are learned by multilayer perceptrons (MLPs). The optimal fusion

strategy for the local and global scores and their most effective location in the SAED architecture is determined in two ablation studies. The first ablation study examines the impact of three fusion strategies and compares the result to standard SAED models. The first fusion strategy is a simple summation between the standard global score and local Gaussian masks, leading to minor and inconsistent performance improvements. The second fusion strategy divides the generation of the final SA score into separate local and global score generation procedures, which are determined by two individual model branches. However, in such a fusion strategy, consistent improvements are not observable.

The last strategy established a novel fusion method by extending the former strategy with a learnable relevance weighting parameter for the local and global scores, resulting in the best and most consistent performance improvements in enhanced SAED models. The second ablation investigates the most effective location of the former best fusion strategy. It reveals that the biggest impact is achieved by applying the strategy in all SA modules of the encoder layers. In the final evaluation, an enhanced SAED model is constructed and evaluated against recent SOTA models. The results demonstrate the superiority of localness in the novel fusion approach, as the enhanced SAED architectures surpass current SOTA models.

The general conclusion of this work is that despite utilizing different model architectures in ASR, there is still room for improvement, as most of the popular ASR models possess non-optimal training schemes, model components, or layer functions. However, this work also shows that the improvements of novel approaches are reaching saturation, and it is questionable if consistent performance gains of clean speech ASR approaches are expectable in the future. Nevertheless, multiple research directions of the established approaches exist:

1. The potential of discrete AMs in the form of DNNQs can be further enhanced if these models are directly trained on triphone (TRI) instead of classic monophone (MONO) labels. Moreover, even though DNNQs cannot surpass standard continuous deep neural network (DNN) architectures, they could be beneficial in tandem approaches, which combine discrete DNNQs with continuous DNNs.

2. The additional decoders in AED models, which are solely active in training, demonstrate their importance in standard AED architectures. These decoder predictions also contain valuable information and should be integrated into the decoding phase.

3. The induced localness with the optimal fusion strategy is solely integrated into the encoder layers of SAED models, and further investigation of its impact in the decoder layers would be reasonable.

A

Appendix: Notation

This appendix briefly introduces the notation used throughout this work.

References

References are divided into three groups to allow for better identification:

1. Self-citations are highlighted by a †, for example, [11†].
2. Citations of manuscripts written by supervised students are highlighted by a $^{+}$, for example, [19^{+}].
3. Any other citations are not highlighted, for example, [22].

Multi-references are separated by a comma in alphanumerical order, not in the order of appearance in the text, for example: The works of Ronecker [17^{+}] and Schnell [19^{+}] were both completed in 2018 in a seminar [17^{+}, 19^{+}]. To avoid clutter from frequently recurring references, each reference is usually repeated *only once per paragraph* unless an attribution is ambiguous.

Mathematics

The mathematical notation follows the ISO 80000-2 standard where possible [70]. The most important aspects are summarized below:

- Scalars are written in lower-case letters, for example, a.
- Constant scalars are written in capital letters, for example, C.
- Vectors are written in bold letters, for example, $\boldsymbol{v}$.
- Matrices and tensors are written in bold capital letters, for example, $\boldsymbol{X}$.
- Distributions and sets are written in calligraphic capital letters, for example, $\mathcal{N}$.

In addition to ISO 80000-2, the following rules apply:

- It is implicitly assumed that $p(\boldsymbol{x}^{(1)}|\boldsymbol{x}^{(0)}) = p(\boldsymbol{x}^{(1)})$ as $\boldsymbol{x}^{(0)}$ is not defined.
- Whenever partial and total derivative are identical, for example, because the derived function is separable in its variables, partial notation is preferred. In other words, $\frac{\partial x}{\partial t}$ is preferred over $\frac{\mathrm{d}x}{\mathrm{d}t}$ unless the meaning changes.
- All mathematical operations are subject to automatic *singleton expansion*, also called *broadcasting*. That is, if two tensors are included in a calculation that requires identical size, their mismatching (singleton) dimensions are expanded via repetition until their dimensions match. For example, consider an $A{\times}B$ matrix $\boldsymbol{X} = (\boldsymbol{x}_1, .., \boldsymbol{x}_B)$ and a $1{\times}B$ vector $\boldsymbol{v} = (v_1, ..., v_B)$, then $\boldsymbol{X} + \boldsymbol{v} = (\boldsymbol{x}_1 + v_1, .., \boldsymbol{x}_B + v_B)$, and analogously, for an $A{\times}1$ vector $\boldsymbol{u} = (u_1, ..., u_A)^T$, $\boldsymbol{X} + \boldsymbol{u} = (\boldsymbol{x}_1 + \boldsymbol{u}, .., \boldsymbol{x}_B + \boldsymbol{u})$.
- Index dereferences always happen from right to left, the indexed dimension(s) being collapsed. For example, an $A{\times}B{\times}C{\times}D$ tensor $\boldsymbol{X}$ indexed as $\boldsymbol{X}_{ij}$ has dimension $A{\times}B$, where j is an index into D and i is an index into C.

Acronyms

Acronyms are introduced (at least) once when they first appear on a per-chapter basis. For article usage it is assumed that capitalized acronyms are pronounced as letter sequences and others are read as if they were common words. For example, it is *an* hidden Markov model (HMM) and *a* convolutional neural network (CNN), but *a* rectified linear unit (ReLU). To avoid confusion with plural usage, acronyms are suffixed by a plural "s", for example, CNNs is the plural of CNN.

References

[1] VoxForge. `http://www.voxforge.org`, 2006. [Online; accessed 07.06.2022].

[2] CMUdict. `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`, 2014. [Online; accessed 07.06.2022].

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 265–283, 2016.

[4] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete Cosine Transform. *IEEE Transactions on Computers*, 100(1):90–93. IEEE, 1974.

[5] C. Anderson. TED. `https://www.ted.com/`, 2019. [Online; accessed 03.06.2022].

[6] L. J. Ba, J. R. Kiros, and G. E. Hinton. Layer Normalization. *arXiv*, abs/1607.06450, 2016.

[7] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli. Wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[8] P. Bahar, A. Zeyer, R. Schlüter, and H. Ney. On Using SpecAugment for End-to-End Speech Translation. In *Proceedings of the 16th International Conference on Spoken Language Translation, IWSLT 2019, Hong Kong, November 2-3, 2019*, 2019.

[9] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[10] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio. End-to-End Attention-Based Large Vocabulary Speech Recognition. In *2016 IEEE International*

Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016, pages 4945–4949, 2016.

[11] L. E. Baum and T. Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563. JSTOR, 1966.

[12] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171. JSTOR, 1970.

[13] R. Bellman. Dynamic Programming. *Science*, 153(3731):34–37. American Association for the Advancement of Science, 1966.

[14] Y. Bengio, P. Frasconi, and P. Y. Simard. The Problem of Learning Long-Term Dependencies in Recurrent Networks. In *Proceedings of International Conference on Neural Networks (ICNN'88), San Francisco, CA, USA, March 28 - April 1, 1993*, pages 1183–1188, 1993.

[15] D. J. Berndt and J. Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In U. M. Fayyad and R. Uthurusamy, editors, *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, USA, July 1994. Technical Report WS-94-03*, pages 359–370. AAAI Press, 1994.

[16] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic press, 2014.

[17] C. M. Bishop. Training With Noise Is Equivalent to Tikhonov Regularization. *Neural Computation*, 7(1):108–116, 1995.

[18] C. M. Bishop. *Pattern Recognition and Machine Learning, 5th Edition*. Information science and statistics. Springer, 2007.

[19] J. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger. Understanding Batch Normalization. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7705–7716, 2018.

[20] A. Blum, N. Haghtalab, and A. D. Procaccia. Variational Dropout and the Local Reparameterization Trick. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2575–2583, 2015.

[21] B. P. Bogert. The Quefrency Alanysis of Time Series for Echoes; Cepstrum, Pseudo-Autocovariance, Cross-Cepstrum and Saphe Cracking. *Time Series Analysis*, pages 209–243. John Wiley & Sons, Inc., 1963.

[22] H. A. Bourlard and N. Morgan. *Connectionist Speech Recognition: A Hybrid Approach*, volume 247. Springer Science & Business Media, 2012.

[23] J. Carletta, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, W. Kraaij, M. Kronenthal, G. Lathoud, M. Lincoln, A. Lisowska,

I. McCowan, W. Post, D. Reidsma, and P. Wellner. The AMI Meeting Corpus: A Pre-Announcement. In *Machine Learning for Multimodal Interaction, Second International Workshop, MLMI 2005, Edinburgh, UK, July 11-13, 2005, Revised Selected Papers*, pages 28–39, 2005.

[24] D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, B. Strope, and R. Kurzweil. Universal Sentence Encoder for English. In E. Blanco and W. Lu, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 169–174. Association for Computational Linguistics, 2018.

[25] P. L. Cerf, W. Ma, and D. V. Compernolle. Multilayer Perceptrons as Labelers for Hidden Markov Models. *IEEE Transactions on Speech and Audio Processing*, 2(1):185–193, 1994.

[26] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals. Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 4960–4964, 2016.

[27] G. Chen, S. Chai, G. Wang, J. Du, W. Zhang, C. Weng, D. Su, D. Povey, J. Trmal, J. Zhang, M. Jin, S. Khudanpur, S. Watanabe, S. Zhao, W. Zou, X. Li, X. Yao, Y. Wang, Z. You, and Z. Yan. GigaSpeech: An Evolving, Multi-Domain ASR Corpus With 10, 000 Hours of Transcribed Audio. In *Interspeech 2021, 22nd Annual Conference of the International Speech Communication Association, Brno, Czechia, 30 August - 3 September 2021*, pages 3670–3674, 2021.

[28] X. Chen, S. Zhang, D. Song, P. Ouyang, and S. Yin. Transformer With Bidirectional Decoder for Speech Recognition. In *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pages 1773–1777, 2020.

[29] G. Cheng, V. Peddinti, D. Povey, V. Manohar, S. Khudanpur, and Y. Yan. An Exploration of Dropout With LSTMs. In *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, pages 1586–1590, 2017.

[30] C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani. State-of-the-Art Speech Recognition With Sequence-to-Sequence Models. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*, pages 4774–4778, 2018.

[31] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *Proceedings of SSSTEMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111, 2014.

[32] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734, 2014.

[33] J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio. End-to-End Continuous Speech Recognition Using Attention-Based Recurrent NN: First Results. *arXiv*, abs/1412.1602, 2014.

[34] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-Based Models for Speech Recognition. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 577–585, 2015.

[35] G. G. Chowdhury. Natural Language Processing. *Annual Review of Information Science and Technology*, 37(1):51–89, 2003.

[36] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv*, abs/1412.3555, 2014.

[37] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[38] C. Coupé, Y. M. Oh, D. Dediu, and F. Pellegrino. Different Languages, Similar Encoding Efficiency: Comparable Information Rates Across the Human Communicative Niche. *Science advances*, 5(9):eaaw2594. American Association for the Advancement of Science, 2019.

[39] M. Cuturi and M. Blondel. Soft-Dtw: A Differentiable Loss Function for Time-Series. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 894–903. PMLR, 2017.

[40] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable Convolutional Networks. *arXiv*, abs/1703.06211, 2017.

[41] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language Modeling With Gated Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 933–941, 2017.

[42] S. Davis and P. Mermelstein. Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366. IEEE, 1980.

[43] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood From

Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22. Wiley Online Library, 1977.

[44] L. Dong, S. Xu, and B. Xu. Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*, pages 5884–5888. IEEE, 2018.

[45] J. C. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[46] D. M. Eberhard, G. F. Simons, and C. D. Fennig. Ethnologue: Languages of the World. *Online version: http://www.ethnologue.com*, 24(24). SIL international, 2021.

[47] J. L. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990.

[48] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and R. Reddy. The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *ACM Computing Surveys*, 12(2):213–253, 1980.

[49] S. Furui. Speaker-Independent Isolated Word Recognition Using Dynamic Features of Speech Spectrum. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(1):52–59, 1986.

[50] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber. Learning Precise Timing With LSTM Recurrent Networks. *Journal of Machine Learning Research*, 3:115–143, 2002.

[51] M. Gibson and T. Hain. Hypothesis Spaces for Minimum Bayes Risk Training in Large Vocabulary Speech Recognition. In *Interspeech 2006 - ICSLP, Ninth International Conference on Spoken Language Processing, Pittsburgh, PA, USA, September 17-21, 2006*, 2006.

[52] X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 315–323, 2011.

[53] A. Graves. Sequence Transduction With Recurrent Neural Networks. *arXiv*, abs/1211.3711, 2012.

[54] A. Graves. Generating Sequences With Recurrent Neural Networks. *arXiv*, abs/1308.0850, 2013.

[55] A. Graves, S. Fernández, F. J. Gomez, and J. Schmidhuber. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data With Recurrent Neural Networks. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 369–376, 2006.

[56] A. Graves and N. Jaitly. Towards End-to-End Speech Recognition With Recurrent Neural Networks. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1764–1772, 2014.

[57] A. Graves, A. Mohamed, and G. E. Hinton. Speech Recognition With Deep Recurrent Neural Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649, 2013.

[58] A. Graves and J. Schmidhuber. Framewise Phoneme Classification With Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[59] A. Gulati, J. Qin, C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang. Conformer: Convolution-Augmented Transformer for Speech Recognition. In *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pages 5036–5040, 2020.

[60] P. Guo, F. Boyer, X. Chang, T. Hayashi, Y. Higuchi, H. Inaguma, N. Kamo, C. Li, D. Garcia-Romero, J. Shi, J. Shi, S. Watanabe, K. Wei, W. Zhang, and Y. Zhang. Recent Developments on Espnet Toolkit Boosted by Conformer. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2021, Toronto, ON, Canada, June 6-11, 2021*, pages 5874–5878, 2021.

[61] K. J. Han, A. Chandrashekaran, J. Kim, and I. R. Lane. The CAPIO 2017 Conversational Speech Recognition System. *arXiv*, abs/1801.00059, 2018.

[62] K. J. Han, J. Huang, Y. Tang, X. He, and B. Zhou. Multi-Stride Self-Attention for Speech Recognition. In *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, pages 2788–2792, 2019.

[63] M. Hart. Project Gutenberg. `https://www.gutenberg.org/`, 1971. [Online; accessed 06.06.2022].

[64] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition.* Springer Series in Statistics. Springer, 2009.

[65] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.

[66] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. *arXiv*, abs/1207.0580, 2012.

[67] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[68] A. L. Hodgkin and A. F. Huxley. A Quantitative Description of Membrane Current and Its Application to Conduction and Excitation in Nerve. *The Journal of Physiology*, 117(4):500. Wiley-Blackwell, 1952.

[69] K. Hornik, M. B. Stinchcombe, and H. White. Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks*, 2(5):359–366, 1989.

[70] International Organization for Standardization. ISO 80000-2: Quantities and Units: Part 2: Mathematical Signs and Symbols to Be Used in the Natural Sciences and Technology. ISO, 2009.

[71] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

[72] N. Jaitly and G. E. Hinton. Vocal Tract Length Perturbation (VTLP) Improves Speech Recognition. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*. JMLR.org, 2013.

[73] M. I. Jordan. Serial Order: A Parallel Distributed Processing Approach. In *Advances in Psychology*, volume 121, pages 471–495. Elsevier, 1997.

[74] N. Kanda, R. Takeda, and Y. Obuchi. Elastic Spectral Distortion for Low Resource Speech Recognition With Deep Neural Networks. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, Olomouc, Czech Republic, December 8-12, 2013*, pages 309–314, 2013.

[75] S. Karita, X. Wang, S. Watanabe, T. Yoshimura, W. Zhang, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. E. Y. Soplin, and R. Yamamoto. A Comparative Study on Transformer vs RNN in Speech Applications. In *IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2019, Singapore, December 14-18, 2019*, pages 449–456, 2019.

[76] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[77] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur. Audio Augmentation for Speech Recognition. In *Interspeech 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 3586–3589, 2015.

[78] A. Krogh, B. Larsson, G. Von Heijne, and E. L. Sonnhammer. Predicting Transmembrane Protein Topology With a Hidden Markov Model: Application to Complete Genomes. *Journal of Molecular Biology*, 305(3):567–580. Elsevier, 2001.

[79] T. Kudo and J. Richardson. SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing,*

EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018, pages 66–71, 2018.

[80] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86. JSTOR, 1951.

[81] L. Kürzinger, N. Lindae, P. Klewitz, and G. Rigoll. Lightweight End-to-End Speech Recognition From Raw Audio Data Using Sinc-Convolutions. In *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pages 1659–1663, 2020.

[82] Y. LeCun, Y. Bengio, and G. E. Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015.

[83] Y. LeCun et al. Generalization and Network Design Strategies. *Connectionism in Perspective*, 19(143-155):18. North Holland, 1989.

[84] C.-H. Lee, L. R. Rabiner, R. Pieraccini, and J. G. Wilpon. Acoustic Modeling for Large Vocabulary Speech Recognition. *Computer Speech & Language*, 4(2):127–165. Elsevier, 1990.

[85] K. Lee. Context-Dependent Phonetic Hidden Markov Models for Speaker-Independent Continuous Speech Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(4):599–609, 1990.

[86] K.-F. Lee. *Automatic Speech Recognition: The Development of the SPHINX System*, volume 62. Springer Science & Business Media, 1988.

[87] V. I. Levenshtein et al. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710. Soviet Union, 1966.

[88] J. Li. Recent Advances in End-to-End Automatic Speech Recognition. *arXiv*, abs/2111.01690, 2021.

[89] J. Li, L. Deng, Y. Gong, and R. Haeb-Umbach. An Overview of Noise-Robust Automatic Speech Recognition. *IEEE ACM Transactions on Audio, Speech and Language Processing*, 22(4):745–777, 2014.

[90] J. Li, Y. Wu, Y. Gaur, C. Wang, R. Zhao, and S. Liu. On the Comparison of Popular End-to-End Models for Large Scale Speech Recognition. In H. Meng, B. Xu, and T. F. Zheng, editors, *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pages 1–5. ISCA, 2020.

[91] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao. Independently Recurrent Neural Network (IndRNN): Building a Longer and Deeper RNN. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 5457–5466. Computer Vision Foundation / IEEE Computer Society, 2018.

[92] L. Liu, A. M. Finch, M. Utiyama, and E. Sumita. Agreement on Target-Bidirectional

LSTMs for Sequence-to-Sequence Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2630–2637, 2016.

[93] L. Liu, M. Utiyama, A. M. Finch, and E. Sumita. Agreement on Target-Bidirectional Neural Machine Translation. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 411–416, 2016.

[94] T. Lohrenz, P. Schwarz, Z. Li, and T. Fingscheidt. Relaxed Attention: A Simple Method to Boost Performance of End-to-End Automatic Speech Recognition. In *IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2021, Cartagena, Colombia, December 13-17, 2021*, pages 177–184. IEEE, 2021.

[95] L. Lu, X. Zhang, and S. Renals. On Training the Recurrent Neural Network Encoder-Decoder for Large Vocabulary End-to-End Speech Recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 5060–5064, 2016.

[96] P. Luo, X. Wang, W. Shao, and Z. Peng. Towards Understanding Regularization in Batch Normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[97] T. Luong, H. Pham, and C. D. Manning. Effective Approaches to Attention-Based Neural Machine Translation. In L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, and Y. Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1412–1421. The Association for Computational Linguistics, 2015.

[98] C. Lüscher, E. Beck, K. Irie, M. Kitza, W. Michel, A. Zeyer, R. Schlüter, and H. Ney. RWTH ASR Systems for LibriSpeech: Hybrid vs Attention - w/o Data Augmentation. *arXiv*, abs/1905.03072, 2019.

[99] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*. JMLR.org, 2013.

[100] J. MacQueen. Classification and Analysis of Multivariate Observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297, 1967.

[101] J. Makhoul, S. E. Roucos, and H. Gish. Vector Quantization in Speech Coding. *Proceedings of the IEEE*, 73(11):1551–1588, 1985.

[102] H. McGuire. LibriVox. `https://librivox.org/`, 2005. [Online; accessed 06.06.2022].

[103] Y. Miao, M. Gowayyed, X. Na, T. Ko, F. Metze, and A. Waibel. An Empirical Exploration of CTC Acoustic Models. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March*

20-25, 2016, pages 2623–2627. IEEE, 2016.

[104] L. M. Milne-Thomson. *The Calculus of Finite Differences*. American Mathematical Soc., 2000.

[105] M. Mimura, S. Sakai, and T. Kawahara. Forward-Backward Attention Decoder. In *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, pages 2232–2236, 2018.

[106] Y. Mittal, P. Toshniwal, S. Sharma, D. Singhal, R. Gupta, and V. K. Mittal. A Voice-Controlled Multi-Functional Smart Home Automation System. In *2015 Annual IEEE India Conference (INDICON)*, pages 1–6. IEEE, 2015.

[107] L. Muda, M. Begam, and I. Elamvazuthi. Voice Recognition Algorithms Using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques. *arXiv*, abs/1003.4083, 2010.

[108] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010.

[109] C. Neukirchen. *Integration neuronaler Vektorquantisierer in ein Hidden-Markov-Modell-Basiertes System zur automatischen Spracherkennung*. PhD thesis, University of Duisburg-Essen, Germany, 1999.

[110] C. Neukirchen and G. Rigoll. Advanced Training Methods and New Network Topologies for Hybrid MMI-Connectionist/HMM Speech Recognition Systems. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '97, Munich, Germany, April 21-24, 1997*, pages 3257–3260, 1997.

[111] T. Nguyen, X. Nguyen, S. R. Joty, and X. Li. Differentiable Window for Dynamic Local Attention. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6589–6599, 2020.

[112] M. A. Nielsen. *Neural Networks and Deep Learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[113] A. Nuttall. Some Windows With Very Good Sidelobe Behavior. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(1):84–91. IEEE, 1981.

[114] J. Pan, J. Shapiro, J. Wohlwend, K. J. Han, T. Lei, and T. Ma. ASAPP-ASR: Multistream CNN and Self-Attentive SRU for SOTA Speech Recognition. In *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pages 16–20, 2020.

[115] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. LibriSpeech: An ASR Corpus Based on Public Domain Audio Books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South*

Brisbane, Queensland, Australia, April 19-24, 2015, pages 5206–5210, 2015.

[116] D. S. Park, W. Chan, Y. Zhang, C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, pages 2613–2617, 2019.

[117] R. Pascanu, T. Mikolov, and Y. Bengio. On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1310–1318, 2013.

[118] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic Differentiation in Pytorch. 2017.

[119] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019.

[120] D. B. Paul and J. M. Baker. The Design for the Wall Street Journal-Based CSR Corpus. In *The Second International Conference on Spoken Language Processing, ICSLP 1992, Banff, Alberta, Canada, October 13-16, 1992*, 1992.

[121] N. Pham, T. Nguyen, J. Niehues, M. Müller, and A. Waibel. Very Deep Self-Attention Networks for End-to-End Speech Recognition. In G. Kubin and Z. Kacic, editors, *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, pages 66–70. ISCA, 2019.

[122] J. W. Picone. Signal Modeling Techniques in Speech Recognition. *Proceedings of the IEEE*, 81(9):1215–1247, 1993.

[123] D. Povey. *Discriminative Training for Large Vocabulary Speech Recognition*. PhD thesis, University of Cambridge, 2005.

[124] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, et al. The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, number EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

[125] D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur. Purely Sequence-Trained Neural Networks for ASR Based on Lattice-Free MMI. In *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, pages 2751–2755, 2016.

[126] P. Price, W. M. Fisher, J. Bernstein, and D. S. Pallett. The DARPA 1000-Word Resource Management Database for Continuous Speech Recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '88, New York, New York, USA, April 11-14, 1988*, pages 651–654, 1988.

[127] N. Qian. On the Momentum Term in Gradient Descent Learning Algorithms. *Neural Networks*, 12(1):145–151, 1999.

[128] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[129] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for Activation Functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018.

[130] K. S. Rao and K. Manjunath. *Speech Recognition Using Articulatory and Excitation Source Features*. Springer, 2017.

[131] S. P. Rath, D. Povey, K. Veselý, and J. Cernocký. Improved Feature Processing for Deep Neural Networks. In *Interspeech 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013*, pages 109–113, 2013.

[132] G. Rigoll. Unsupervised Information Theory-Based Training Algorithms for Multilayer Neural Networks. In *1992 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '92, San Francisco, California, USA, March 23-26, 1992*, pages 393–396, 1992.

[133] G. Rigoll. Maximum Mutual Information Neural Networks for Hybrid Connectionist-HMM Speech Recognition Systems. *IEEE Transactions on Speech and Audio Processing*, 2(1):175–184, 1994.

[134] G. Rigoll and C. Neukirchen. A New Approach to Hybrid HMM/ANN Speech Recognition Using Mutual Information Neural Networks. In *Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996*, pages 772–778, 1996.

[135] G. Rigoll, C. Neukirchen, and J. Rottland. A New Hybrid System Based on MMI-Neural Networks for the RM Speech Recognition Task. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings, ICASSP '96, Atlanta, Georgia, USA, May 7-10, 1996*, pages 865–868, 1996.

[136] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386. American Psychological Association, 1958.

[137] J. Rottland, C. Neukirchen, D. Willett, and G. Rigoll. Large Vocabulary Speech Recognition With Context Dependent MMI-Connectionist /HMM Systems Using the WSJ Database. In *Fifth European Conference on Speech Communication and Technology, EUROSPEECH 1997, Rhodes, Greece, September 22-25, 1997*, 1997.

[138] A. Rousseau, P. Deléglise, and Y. Estève. Enhancing the TED-LIUM Corpus With Selected Data for Language Modeling and More TED Talks. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014, Reykjavik, Iceland, May 26-31, 2014*, pages 3935–3939, 2014.

[139] H. Sak, A. W. Senior, and F. Beaufays. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *arXiv*, abs/1402.1128, 2014.

[140] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How Does Batch Normalization Help Optimization? In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2488–2498, 2018.

[141] B. W. Schuller, G. Rigoll, and M. K. Lang. Speech Emotion Recognition Combining Acoustic Features and Linguistic Information in a Hybrid Support Vector Machine-Belief Network Architecture. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2004, Montreal, Quebec, Canada, May 17-21, 2004*, pages 577–580. IEEE, 2004.

[142] M. Schuster and K. K. Paliwal. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[143] R. M. Schwartz, Y. Chow, S. E. Roucos, M. A. Krasner, and J. I. Makhoul. Improved Hidden Markov Modeling of Phonemes for Continuous Speech Recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '84, San Diego, California, USA, March 19-21, 1984*, pages 21–24, 1984.

[144] R. Sennrich, B. Haddow, and A. Birch. Neural Machine Translation of Rare Words With Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[145] P. Shaw, J. Uszkoreit, and A. Vaswani. Self-Attention With Relative Position Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 464–468, 2018.

[146] J. Sietsma and R. J. F. Dow. Creating Artificial Neural Networks That Generalize. *Neural Networks*, 4(1):67–79, 1991.

[147] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[148] T. F. Smith, M. S. Waterman, et al. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147(1):195–197. Elsevier Science, 1981.

[149] M. Sperber, J. Niehues, G. Neubig, S. Stüker, and A. Waibel. Self-Attentional

Acoustic Models. In *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, pages 3723–3727, 2018.

[150] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks From Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[151] S. S. Stevens, J. Volkmann, and E. B. Newman. A Scale for the Measurement of the Psychological Magnitude Pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190. Acoustical Society of America, 1937.

[152] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to Sequence Learning With Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.

[153] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 4278–4284, 2017.

[154] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper With Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9, 2015.

[155] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826, 2016.

[156] V. Tiwari. MFCC and Its Applications in Speaker Recognition. *International Journal on Emerging Technologies*, 1(1):19–22. Citeseer, 2010.

[157] Z. Tüske, K. Audhkhasi, and G. Saon. Advancing Sequence-to-Sequence Based Speech Recognition. In *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, pages 3780–3784, 2019.

[158] W. F. Twaddell. On Defining the Phoneme. *Language*, 11(1):5–62. JSTOR, 1935.

[159] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[160] K. Veselý, A. Ghoshal, L. Burget, and D. Povey. Sequence-Discriminative Training of Deep Neural Networks. In *Interspeech 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013*, pages 2345–2349, 2013.

[161] A. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano, and K. J. Lang. Phoneme Recognition Using Time-Delay Neural Networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, 1989.

[162] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. E. Y. Soplin, J. Heymann, M. Wiesner, N. Chen, A. Renduchintala, and T. Ochiai. ESPnet: End-to-End Speech Processing Toolkit. In B. Yegnanarayana, editor, *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, pages 2207–2211. ISCA, 2018.

[163] S. Watanabe, T. Hori, S. Kim, J. R. Hershey, and T. Hayashi. Hybrid CTC/Attention Architecture for End-to-End Speech Recognition. *IEEE Journal of Selected Topics in Signal Processing*, 11(8):1240–1253, 2017.

[164] S. Watanabe, M. I. Mandel, J. Barker, and E. Vincent. CHiME-6 Challenge: Tackling Multispeaker Speech Recognition for Unsegmented Recordings. *arXiv*, abs/2004.09249, 2020.

[165] C. Weng, J. Cui, G. Wang, J. Wang, C. Yu, D. Su, and D. Yu. Improving Attention Based Sequence-to-Sequence Models for End-to-End English Conversational Speech Recognition. In *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, pages 761–765, 2018.

[166] P. Werbos. New Tools for Prediction and Analysis in the Behavioral Sciences. *Ph. D. Dissertation, Harvard University*, 1974.

[167] P. J. Werbos. Applications of Advances in Nonlinear Sensitivity Analysis. In *System Modeling and Optimization*, pages 762–770. Springer Berlin Heidelberg, 1982.

[168] P. J. Werbos. Generalization of Backpropagation With Application to a Recurrent Gas Market Model. *Neural Networks*, 1(4):339–356, 1988.

[169] M. Westphal. The Use of Cepstral Means in Conversational Speech Recognition. In *Fifth European Conference on Speech Communication and Technology, EUROSPEECH 1997, Rhodes, Greece, September 22-25, 1997*, 1997.

[170] I. Williams, A. Kannan, P. S. Aleksic, D. Rybach, and T. N. Sainath. Contextual Speech Recognition in End-to-End Neural Network Systems Using Beam Search. In *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, pages 2227–2231, 2018.

[171] J. Wu. Introduction to Convolutional Neural Networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495, 2017.

[172] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig. Achieving Human Parity in Conversational Speech Recognition. *arXiv*, abs/1610.05256, 2016.

[173] J. Yamato, J. Ohya, and K. Ishii. Recognizing Human Action in Time-Sequential Images Using Hidden Markov Model. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 1992, Proceedings, 15-18 June, 1992, Champaign, Illinois, USA*, pages 379–385, 1992.

[174] B. Yang, Z. Tu, D. F. Wong, F. Meng, L. S. Chao, and T. Zhang. Modeling Localness for Self-Attention Networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4449–4458, 2018.

[175] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, et al. The HTK Book. *Cambridge University Engineering Department*, 3(175):12, 2002.

[176] S. J. Young, J. J. Odell, and P. C. Woodland. Tree-Based State Tying for High Accuracy Modelling. In *Human Language Technology, Proceedings of a Workshop held at Plainsboro, New Jerey, USA, March 8-11, 1994*, 1994.

[177] F. Yu and V. Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[178] G. Yu, W. Russell, R. M. Schwartz, and J. Makhoul. Discriminant Analysis and Supervised Vector Quantization for Continuous Speech Recognition. In *1990 International Conference on Acoustics, Speech, and Signal Processing, ICASSP '90, Albuquerque, New Mexico, USA, April 3-6, 1990*, pages 685–688, 1990.

[179] N. Zeghidour, Q. Xu, V. Liptchinsky, N. Usunier, G. Synnaeve, and R. Collobert. Fully Convolutional Speech Recognition. *arXiv*, abs/1812.06864, 2018.

[180] M. D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv*, abs/1212.5701, 2012.

[181] A. Zeyer, P. Bahar, K. Irie, R. Schlüter, and H. Ney. A Comparison of Transformer and LSTM Encoder Decoder Models for ASR. In *IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2019, Singapore, December 14-18, 2019*, pages 8–15, 2019.

[182] A. Zeyer, K. Irie, R. Schlüter, and H. Ney. Improved Training of End-to-End Attention Models for Speech Recognition. In *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, pages 7–11, 2018.

[183] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. Dive Into Deep Learning. *arXiv*, abs/2106.11342, 2021.

[184] Y. Zhang, N. Suda, L. Lai, and V. Chandra. Hello Edge: Keyword Spotting on Microcontrollers. *arXiv*, abs/1711.07128, 2017.

[185] Z. Zhang, S. Wu, S. Liu, M. Li, M. Zhou, and T. Xu. Regularizing Neural Machine Translation by Target-Bidirectional Agreement. In *The Thirty-Third*

AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pages 443–450, 2019.

[186] Y. Zheng, J. Tao, Z. Wen, and J. Yi. Forward-Backward Decoding Sequence for Regularizing End-to-End TTS. *IEEE ACM Transactions on Audio, Speech, and Language Processing*, 27(12):2067–2079, 2019.

[187] W. Zhou, W. Michel, K. Irie, M. Kitza, R. Schlüter, and H. Ney. The RWTH ASR System for TED-LIUM Release 2: Improving Hybrid HMM With SpecAugment. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pages 7839–7843, 2020.

Publications

[1†] L. Kürzinger, E. R. C. Rosas, L. Li, T. Watzel, and G. Rigoll. Audio Adversarial Examples for Robust Hybrid CTC/Attention Speech Recognition. In A. Karpov and R. Potapova, editors, *Speech and Computer - 22nd International Conference, SPECOM 2020, St. Petersburg, Russia, October 7-9, 2020, Proceedings*, volume 12335 of *Lecture Notes in Computer Science*, pages 255–266. Springer, 2020.

[2†] L. Kürzinger, T. Watzel, L. Li, R. Baumgartner, and G. Rigoll. Exploring Hybrid CTC/Attention End-to-End Speech Recognition with Gaussian Processes. In A. A. Salah, A. Karpov, and R. Potapova, editors, *Speech and Computer - 21st International Conference, SPECOM 2019, Istanbul, Turkey, August 20-25, 2019, Proceedings*, volume 11658 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2019.

[3†] L. Kürzinger, D. Winkelbauer, L. Li, T. Watzel, and G. Rigoll. CTC-Segmentation of Large Corpora for German End-to-End Speech Recognition. In A. Karpov and R. Potapova, editors, *Speech and Computer - 22nd International Conference, SPECOM 2020, St. Petersburg, Russia, October 7-9, 2020, Proceedings*, volume 12335 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 2020.

[4†] L. Li, Y. Kang, Y. Shi, L. Kürzinger, T. Watzel, and G. Rigoll. Adversarial Joint Training With Self-Attention Mechanism for Robust End-to-End Speech Recognition. *EURASIP J. Audio Speech Music. Process.*, 2021.

[5†] L. Li, L. Kürzinger, T. Watzel, and G. Rigoll. A Global Discriminant Joint Training Framework for Robust Speech Recognition. In *33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021*, pages 544–551. IEEE, 2021.

[6†] L. Li, L. Kürzinger, T. Watzel, G. Rigoll, et al. Lightweight End-to-End Speech Enhancement Generative Adversarial Network Using Sinc Convolutions. *Applied Sciences*. Multidisciplinary Digital Publishing Institute, 2021.

[7†] L. Li, Z. Lu, T. Watzel, L. Kürzinger, and G. Rigoll. Light-Weight Self-Attention Augmented Generative Adversarial Networks for Speech Enhancement. *Electronics*. Multidisciplinary Digital Publishing Institute, 2021.

[8†] L. Li, T. Watzel, L. Kürzinger, and G. Rigoll. Towards Constructing HMM Structure for Speech Recognition With Deep Neural Fenonic Baseform Growing. *IEEE Access*, 2021.

[9†] L. Li, X. Zhou, Z. Song, T. Watzel, L. Kürzinger, and G. Rigoll. Deep Neural Fenonic Baseform Growing: A Novel Approach to Construct HMM Topologies for Speech Recognition. In *2020 International Conference on High Performance Computing Simulation (HPCS)*, 2021.

[10†] T. Watzel, L. Kürzinger, L. Li, and G. Rigoll. Synchronized Forward-Backward Transformer for End-to-End Speech Recognition. In A. Karpov and R. Potapova, editors, *Speech and Computer - 22nd International Conference, SPECOM 2020, St. Petersburg, Russia, October 7-9, 2020, Proceedings*, volume 12335 of *Lecture Notes in Computer Science*, pages 646–656. Springer, 2020.

[11†] T. Watzel, L. Kürzinger, L. Li, and G. Rigoll. Induced Local Attention for Transformer Models in Speech Recognition. In A. Karpov and R. Potapova, editors, *Speech and Computer - 23rd International Conference, SPECOM 2021, St. Petersburg, Russia, September 27-30, 2021, Proceedings*, volume 12997 of *Lecture Notes in Computer Science*, pages 795–806. Springer, 2021.

[12†] T. Watzel, L. Kürzinger, L. Li, and G. Rigoll. Regularized Forward-Backward Decoder for Attention Models. In A. Karpov and R. Potapova, editors, *Speech and Computer - 23rd International Conference, SPECOM 2021, St. Petersburg, Russia, September 27-30, 2021, Proceedings*, volume 12997 of *Lecture Notes in Computer Science*, pages 786–794. Springer, 2021.

[13†] T. Watzel, L. Li, L. Kürzinger, and G. Rigoll. Deep Neural Network Quantizers Outperforming Continuous Speech Recognition Systems. In A. A. Salah, A. Karpov, and R. Potapova, editors, *Speech and Computer - 21st International Conference, SPECOM 2019, Istanbul, Turkey, August 20-25, 2019, Proceedings*, volume 11658 of *Lecture Notes in Computer Science*, pages 530–539. Springer, 2019.

[14†] T. Watzel and G. Rigoll. Performance Comparison of Deep Neural Network Quantizers to Continuous ASR Systems. In *Fortschritte der Akustik–DAGA'19*, pages pp–947, 2019.

Supervised Student Theses, Seminars, and Internships

[1+] Y. K. Akan. A Comparative Study of Pre-trained Language Models for Contextual Document Representation. *Bachelor's Thesis*. Technical University of Munich, 2019.

[2+] W. Cao. Modeling Localness for Transformer Models in Speech Recognition. *Bachelor's Thesis*. Technical University of Munich, 2021.

[3+] Z. Chen. Regularizing Self-Attention in Synchronized Forward-Backward Transformer. *Research Internship*. Technical University of Munich, 2021.

[4+] F. Fraaz. Detection of Defects in Sensor Wiring. *Research Internship*. Technical University of Munich, 2020.

[5+] B. Hofmann. Sprachsystem für Spiel "Professor Pünschge". *IDP*. Technical University of Munich, 2018.

[6+] J. Hua. Improved Training of End-to-End Attention Models for Speech Recognition. *Scientific Seminar*. Technical University of Munich, 2018.

[7+] Z. Huang. Wake-Up Word Recognition with DNNs. *Research Internship*. Technical University of Munich, 2019.

[8+] P. Joppich. Sensorfusion und Zustandsschätzung auf Grundlage rauschbehafteter Messdaten. *Research Internship*. Technical University of Munich, 2019.

[9+] S. Li. Attacks on Neural Networks for Speech Recognition. *Scientific Seminar*. Technical University of Munich, 2018.

[10+] Y. Liu. Framework Comparison of Tensorflow and Kaldi Using TDNN and LSTM for Hybrid Speech Recognition. *Master's Thesis*. Technical University of Munich, 2019.

[11+] L. Lohmer. A Comparison of Techniques for Language Model Integration in Encoder-Decoder Speech Recognition. *Scientific Seminar*. Technical University of Munich, 2019.

[12+] Y. Mao. Test of Mask-CTC on Transformer and Conformer. *Research Internship*. Technical University of Munich, 2021.

[13+] S. B. Mesfin. Feature Comparison in Unsupervised Speech Recognition. *Bachelor's Thesis*. Technical University of Munich, 2022.

[14+] X. Niu. Application of Gaussian Mixture VAE in Natural Language Processing. *Research Internship*. Technical University of Munich, 2019.

[15+] Y. Pan. LSTM Neural Network Quantizers. *Research Internship*. Technical University of Munich, 2020.

[16+] R. Ragunathan. Deep k-means Clustering for Discrete Speech Recognition. *Bachelor's Thesis*. Technical University of Munich, 2021.

[17+] M. Ronecker. Development of a Perception System for Autonomous Driving. *Research Internship*. Technical University of Munich, 2018.

[18+] M. Ronecker. Deep Reinforcement Learning for Decision Making in Autonomous Driving. *Master's Thesis*. Technical University of Munich, 2019.

[19+] B. Schnell. Improving ASR Systems with E-Vectors. *Scientific Seminar*. Technical University of Munich, 2018.

[20+] B. Shen. Application of MFCC Features in Speech Enhancement Generative Adversarial Network. *Research Internship*. Technical University of Munich, 2018.

[21+] Z. Yan. A Specific Scenario of Named Entity Recognition for Cyber Threat Intelligence. *Master's Thesis*. Technical University of Munich, 2022.