BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

PERSONAL VOICE ACTIVITY DETECTION

ŘEČNÍKEM PODMÍNĚNÁ DETEKCE HLASU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **ŠIMON SEDLÁČEK**
AUTOR PRÁCE

SUPERVISOR **Ing. JÁN ŠVEC**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2020/2021

Bachelor's Thesis Specification

Student: **Sedláček Šimon**
Programme: Information Technology
Title: **Personal Voice Activity Detection**
Category: Speech and Natural Language Processing

Assignment:

1. Get acquainted with automatic speech activity detection.
2. Get acquainted with recurrent neural networks.
3. Implement the method using an appropriate toolkit (e.g PyTorch).
4. Train and evaluate on a standard dataset, compare with published results.
5. Evaluate the results and suggest ways to further improve them.

Recommended literature:

- Ding, S., Wang, Q., Chang, S., Wan, L., & Moreno, I. L. (2019). *Personal VAD: Speaker-Conditioned Voice Activity Detection*. http://arxiv.org/abs/1908.04284
- as recommended by the supervisor

Requirements for the first semester:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Švec Ján, Ing.**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2020
Submission deadline: May 12, 2021
Approval date: October 30, 2020

Abstract

This work aims to implement, test, and evaluate a speaker-conditioned Voice Activity Detection (VAD) method called "Personal VAD". The method builds upon an LSTM-based approach to VAD and its purpose is to introduce a system that can reliably detect speech of a target speaker, while retaining the typical characteristics of a VAD system, mainly in terms of small model size, low latency, and low necessary computational resources. The system is trained to distinguish between three classes: non-speech, target speaker speech, and non-target speaker speech. For this purpose, the method utilizes speaker embeddings as a part of the input feature vector to represent the target speaker. Some of the more heavyweight personal VAD variants also make use of speaker verification scores issued to each frame based on the target embedding, resulting in a more robust system. In addition to the one scoring method presented in the original article, two other scoring approaches are introduced, both outperforming the baseline method and improving the performance even for acoustically challenging conditions.

Abstrakt

Cílem této práce je implementovat, otestovat a vyhodnotit řečníkem podmíněnou metodu pro detekci hlasu (*Voice Activity Detection*, VAD) nazvanou "Personal VAD". Pro detekci využívá tato metoda LSTM neuronových sítí a jejím účelem je vytvoření systému schopného spolehlivě detekovat řečové signály cílového řečníka při zachování vlastností typického VAD systému co se velikosti modelu, odezvy a nízkých nároků na zdroje týče. Systém je trénován pro klasifikaci řečových rámců do tří tříd: neřeč, řeč necílového a řeč cílového řečníka. Za tímto účelem využívá metoda speaker embedding vektory pro reprezentaci cílového řečníka jako součást vstupních příznaků. Některé z náročnějších variant systému využívají skórování rámců systémem pro verifikaci řečníka, což vede ke zvýšení spolehlivosti klasifikace. Vedle základní metody skórování představené v originálním článku byly navrženy dvě modifikace, jež základní metodu překonaly a zlepšily spolehlivost výsledného systému i v akusticky náročných prostředích.

Keywords

voice activity detection, speech detection, recurrent neural networks, long short-term memory, LSTM, speaker recognition, speaker embeddings, d-vector

Klíčová slova

detekce hlasové aktivity, detekce řeči, rekurentní neuronové sítě, long short-term memory, LSTM, rozpoznání mluvčího, speaker embeddings, d-vector

Reference

SEDLÁČEK, Šimon. *Personal Voice Activity Detection*. Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ján Švec

Rozšířený abstrakt

Tato práce se zabývá implementací a evaluací řečníkem podmíněné metody detekce hlasu (*Voice Activity Detection*, VAD), původně nazvané "Personal VAD" (PVAD). Tato metoda byla poprvé představena v [13] a jejím cílem je vytvoření VAD systému, který je schopen rozlišit řečové rámce jednoho cílového řečníka od řečových rámců ostatních mluvčích.

Původní motivací pro vytvoření takového systému bylo jeho potenciální využití pro *on-device* rozpoznávání řeči u osobních a mobilních zařízení. Pro tento účel by bylo výhodné, aby si výsledný systém zachoval základní charakteristiky typického VAD systému, primárně co se nároků na zdroje a odezvy týče. z tohoto důvodu je tedy snaha o adaptaci klasického VAD systému na daného cílového řečníka potenciálně výhodnější než například přímé využití diarizačních systémů nebo systémů pro verifikaci řečníka pro stejný účel.

Výsledný systém je založen na architektuře LSTM neuronových sítích, které jsou jednou z populárních architektur neuronových sítí pro tvorbu VAD systémů [14], a které umožňují elegantní modelování temporálního kontextu ve vstupních datech a poskytují možnost proudového zpracování. Tato architektura je pro všechny PVAD systémy společná a výsledný model má pak něco málo přes 130 tisíc parametrů. Pro adaptaci na cílového mluvčího jsou pak využity dva typy vstupních příznaků specifických pro daného řečníka. Jednotlivé varianty PVAD systémů se pak liší právě tím, jakou kombinaci těchto vstupních příznaků využijí.

Prvním z těchto příznaků je řečníkova *d-vector embedding* [48] reprezentace. Ta má v prvé řadě sloužit jako jakýsi abstraktní vzor pro výsledný PVAD systém, na základě kterého má cílového řečníka v akustických příznacích identifikovat. Tento embedding vektor je využíván primárně architekturou ET, která představuje ideální řešení problému PVAD, protože nemá žádné speciální nároky na zdroje, na rozdíl od ostatních PVAD variant. Dále je tato embedding reprezentace využívána jako součást příznakového vektoru architektury SET.

Náročnější varianty PVAD (co se nároků na zdroje týče) pak mohou tuto d-vector reprezentaci možné využít i pro skórování jednotlivých akustických rámců. Pro každý jednotlivý rámec je extrahován sekundární d-vector, který je kosinovou podobností porovnán s embedding vektorem cílového řečníka. Takto dostaneme skalární skóre, které lze využít jako další příznak na vstupu systému. Toto skóre je využíváno architekturami SC, ST a SET. Architektura SET tedy využívá jak embedding vektor cílového řečníka, tak zmíněná skóre, a je tedy očekáváno, že bude mít nejlepší klasifikační výsledky.

Pro natrénování jednotlivých PVAD architektur je také třeba najít vhodnou datovou sadu. Ideální datová sada by obsahovala střídavé promluvy řečníků, ale také jejich samostatné promluvy v dostatečné kvalitě, které by bylo možné využít pro extrakci výchozích d-vector reprezentací každého řečníka. Pro trénování PVAD architektur, které využívají právě pouze tuto výchozí reprezentaci, je pak také třeba, aby daná datová sada obsahovala dostatečnou variabilitu řečníků, tedy jejich dostatečný počet.

Pro tento účel byl využit standardní LibriSpeech [32] korpus, s pomocí kterého byly střídavé promluvy simulovány konkatenací několika promluv několika náhodně zvolených řečníků. Výsledné systémy pak byly trénovány na celém téměř tisíci-hodinovém rozsahu LibriSpeech korpusu, přičemž byla na vygenerovanou trénovací datovou sadu aplikována augmentace, která dále zvětšila její rozsah na čtyřnásobek.

Výsledky základních experimentů se všemi čtyřmi architekturami přinesly jeden důležitý poznatek: hodnoty skóre pro verifikaci řečníka přiřazená každému rámci u architektur SC, ST a SET nebyly co se týče rozlišení cílových (*target speaker speech*, `tss`) a necílových (*non-target speaker speech*, `ntss`) řečových rámců dostatečně diskriminativní. Architektura

ST, která pro detekci `tss` rámců vedle akustických příznaků využívá právě pouze toto skóre, dosáhla pro čistou validační sadu přesnosti klasifikace pouze 84.29% (tedy podíl korektně klasifikovaných rámců). Oproti tomu architektura ET, využívající pouze d-vector embedding reprezentaci cílového řečníka dosáhla pro stejný úkol dosáhla přesnosti 88.02%.

Z tohoto důvodu byly navrženy dvě modifikace základní metody skórování, které v průběhu zpracování vstupní nahrávky extrahují embedding vektory s využitím posuvného kontextuálního okna. Tím je za cenu drobného zvýšení nároků na zdroje dosaženo vyšší kvality extrahovaných sekundárních embedding vektorů a tím i vyšší diskriminativity výsledných hodnot skóre. Architektury ST a SET natrénovány s takto získanými skóre zaznamenaly výrazné zlepšení ve schopnosti rozlišování `tss` a `ntss` rámců (a to i v akusticky náročných situacích) a překonaly tak výsledky nejlepšího SET systému prezentovaného v [13]. Nejlepší SET systém využívající těchto modifikací skórování dosáhl pro čistou řeč přesnosti 92.23%.

U architektury ET bylo také experimentováno s jinými typy vektorů pro reprezentaci cílového řečníka, konkrétně *i-vector* [11] a *x-vector* [43]. Přestože oba typy reprezentací dosáhly obstojných výsledků, v případě x-vectorů bylo dosaženo výsledků téměř srovnatelných s d-vectory, nebyla zaznamenána žádná zlepšení oproti ET systému využívajícího d-vector reprezentace.

Pro další zlepšení dosažených výsledků a zvýšení robustnosti všech PVAD systémů by do budoucna bylo vhodné využít pro jejich trénování kromě simulovaných také reálná data. Toto by umožnilo řádné vyhodnocení schopnosti adaptace jednotlivých systémů například na překrývající se řeč v klasifikovaných nahrávkách.

Personal Voice Activity Detection

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Ján Švec. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Šimon Sedláček
May 11, 2021

Acknowledgements

I would like to sincerely thank my supervisor Ing. Ján Švec for his support, advice, and valuable remarks while working on this thesis. Additionally, i would like to thank Ing. Jan Brukner for his feedback with regard to the experiments conducted in this work.

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Contents

1 Introduction **3**

2 Artificial Neural Networks **5**
2.1 Feed-forward neural networks 5
2.2 Training the network 7
2.3 The problem of sequential modeling 7
2.4 Recurrent neural networks 8
2.4.1 Modeling long temporal dependencies 8
2.4.2 Long short-term memory 9

3 Speaker representations **12**
3.1 i-vectors 12
3.2 x-vectors 13
3.3 d-vectors 13
3.4 Embedding similarity metrics 14

4 Speech detection **16**
4.1 Voice activity detection 16
4.1.1 Voice activity detection methods 17
4.2 Speaker-conditioned voice activity detection 18
4.2.1 Personal voice activity detection approach overview 20
4.2.2 Personal voice activity detection system architecture 20
4.2.3 Loss functions 24

5 Data **25**
5.1 LibriSpeech 25
5.2 Generating the dataset 25
5.3 Data augmentation 27

6 Implementation **28**
6.1 Data preparation and feature extraction 28
6.1.1 Speaker embedding extractors 30
6.2 Model implementation and training 30
6.2.1 Dealing with utterances of variable lengths 31

7 Experiments **33**
7.1 Evaluation metrics 33
7.2 Training configuration and conditions 34

7.2.1 Last hidden layer activation functions 35
7.3 Comparing the architectures 35
7.3.1 Weighted pairwise loss 37
7.4 Comparison of different target speaker embedding types 39
7.5 Altering the frame scoring method 41
7.5.1 Comparing frame-level and window-level d-vector discriminativity . 41
7.5.2 Proposed scoring methods 43
7.5.3 Scoring alteration performance results 44
7.6 Summary and possible improvements 45

8 Conclusions **47**

Bibliography **49**

A Contents of the enclosed storage unit **54**

Chapter 1

Introduction

Voice Activity Detection (VAD) is the problem of distinguishing speech signals in audio recordings from silence and background noise. It is typically used as a part of most speech processing systems, taking a role of a pre-processing component, responsible for filtering out irrelevant information from the processed signal. This is beneficial for a multitude of reasons.

One, the downstream system might be sensitive to noise, and filtering out such signals might improve the system's performance.

Two, the likes of speaker verification and automatic speech recognition systems are typically quite expensive to run in terms of the required computational resources, which is especially crucial when dealing with mobile personal devices. Voice activity detection can therefore somewhat limit the resource and energy consumption by triggering the downstream systems only when necessary.

For some scenarios, it might be useful to extend the basic VAD problem by introducing a speaker constraint – detecting speech frames of one *target* speaker and distinguishing them from other *non-target* speakers.

This could generally be addressed by utilizing a speaker verification or a speaker diarization system, however, it can be argued that a dedicated solution might offer some advantages.

One such dedicated solution was recently proposed in [13], presenting a method of adapting a classical LSTM-based VAD system to the target speaker by utilizing their d-vector [48] speaker embedding representation. One of the primary goals of this method (originally called "Personal VAD") is for the resulting system to maintain the characteristics of a typical VAD system. That is mainly in terms of latency and low resource requirements so that such a system could be used even for on-device speech recognition scenarios. However, some more heavyweight approaches to the personal VAD problem are also proposed, as in some situations, the additional robustness might prove useful even despite the additional resource requirements.

This thesis aims to implement, evaluate and expand upon this speaker-conditioned VAD method, as there is currently no working implementation of this method (or training/evaluation data for that matter) available to the public.

The rest of this thesis is organized as follows: Chapter 2 provides a brief introduction to the topic of artificial neural networks, focusing on their use for temporal context modeling. Special attention is given to recurrent neural networks, as they are at the core of the VAD systems implemented in this work. Chapter 3 introduces the concept of speaker representation vectors, also referred to as *speaker embeddings*. In Chapter 4, common

approaches to voice activity detection are discussed, as well as the core speaker-conditioned VAD approach explored in this work. Chapter 5 describes the data used for training the implemented systems. Chapter 6 explains some of the key details regarding the feature extraction, and personal VAD system implementation and training processes. Lastly, in Chapter 7, the systems are evaluated and some potential ways of expanding the system capabilities and improving the obtained results are introduced and discussed.

Chapter 2

Artificial Neural Networks

This chapter gives a brief introduction to the topic of *Artificial Neural Networks* (ANN) – predominantly their types, architectures, use cases, the process of their design, and training. Special attention is given to recurrent neural networks as they are the core network type used in this work. The contents of this chapter (both for sections about feed-forward and recurrent neural networks) were mostly derived from [16] and [3].

2.1 Feed-forward neural networks

Deep feed-forward networks (also referred to as *feed-forward neural networks*) are a machine learning model type, which allowed the creation and evolution of the *deep learning* discipline. a feed-forward network essentially has one task, which is to approximate some arbitrary function f^*. This function represents a mapping of a system's inputs to their corresponding outputs – for example, for a classification problem, the target mapping could be defined as assigning a category label to the classified input. The neural network then defines a mapping[1]:

$$\mathbf{y} = f(\mathbf{x}; \theta),$$

where $\mathbf{x}$ is the input feature vector, $\mathbf{y}$ is the network's output vector, $f(\cdot)$ is the mapping function and θ represents the network's learned parameters. The network is supposed to learn θ in such a way that results in f becoming the best possible approximation of the target function f^*.

Deep feed-forward networks are called *deep* because they consist of several layers: an input layer, an output layer, and a number of the so-called *hidden layers*. These layers are stacked one behind the other, each taking the output of the previous layer as its input, thus creating a network.

Each layer consists of a number of *artificial neurons*, which are called *hidden units* for the hidden layers or *output units* for the output layer. Every artificial neuron has a vector of input weights $\mathbf{w}$ and a bias b; parameters, which the neuron uses to transform its input $\mathbf{x}$:

$$a = \mathbf{w}^T\mathbf{x} + b \tag{2.1}$$

[1] Regarding vector notation: in this section, any variable that is a vector or matrix is depicted as bold to differentiate them from scalar values. However, in later sections, some equations only contain vectors and matrices and since it is unnecessary to distinguish them from scalars, they are written in a regular font only to improve readability.

This single neuron case can be generalized for the whole layer, giving us the following affine transformation:

$$\mathbf{a}^i = \mathbf{W}^i \mathbf{z}^{i-1} + \mathbf{b}^i, \tag{2.2}$$

where $\mathbf{z}^{i-1}$ is the output vector of the previous hidden layer serving as the input of the current i-th layer, and $\mathbf{W}^i$ is a matrix of *weights* assigned to the inputs. The dimensionality of the weight matrix corresponds to the number of *hidden units* in the current layer and the dimensionality of $\mathbf{z}^{i-1}$. Finally, $\mathbf{b}^i$ is a vector of *biases* assigned to the hidden units, and $\mathbf{a}^i$ is a vector of output *activations*.

These activations are then transformed using a non-linear, differentiable (with some exceptions) *activation function* $h(\cdot)$:

$$\mathbf{z}^i = h(\mathbf{a}^i), \tag{2.3}$$

giving us $\mathbf{z}^i$, the final output of the layer.

Which activation function should be used depends on several factors. Usually, the activation function used by the hidden units will be different from the one used in the output layer. The output layers typically use either the *logistic sigmoid* or the *softmax* for classification problems. For regression problems, no activation function is used. For hidden units, the *rectified linear unit* (and its variants) has become very popular in recent years, although there are many other options – such as the *hyperbolic tangent* or the previously mentioned and in the past widely used logistic sigmoid, both of which are often used inside recurrent neural network cells. All of these activation functions are shown in Figure 2.1.

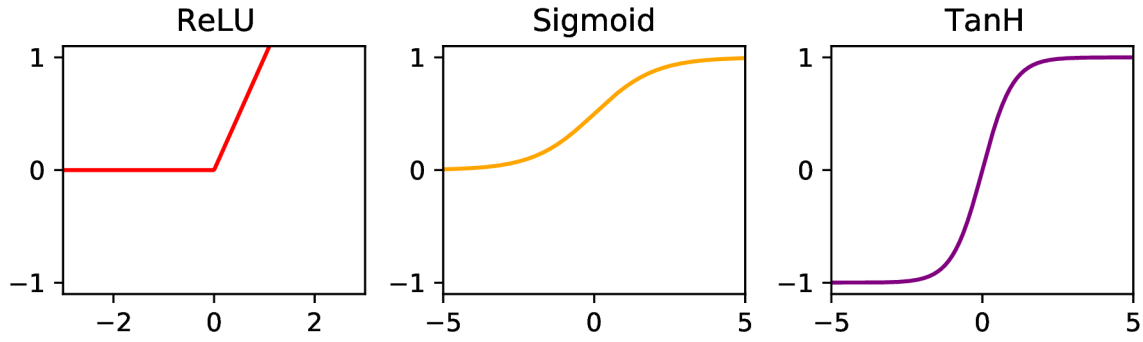

Figure 2.1: Visualization of the rectified linear unit (ReLU), logistic sigmoid (Sigmoid) and the hyperbolic tangent (TanH) activation functions.

Above all, it is crucial for the activation function to be non-linear. The non-linearity allows the network to perform complex input transformations, not just affine transformations (rotation, translation, shear). These complex transformations allow the network to *fold* and *warp* the input feature space[2] so that, for example in the case of a complex classification problem, boundaries between classes can be modeled more easily.

It is apparent that a deep neural network simply represents a series of non-trivial functional transformations. It should be stated that according to the *universal approximation theorem* [18], a neural network with only one hidden layer with a non-linear activation function can approximate any Borel-measurable function to any desired degree of accuracy. This assumes that the hidden layer is given enough hidden units. In practice, a solution

[2] A nice visualization of these effects can be found on Christopher Olah's blog: https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

of this nature is however often unscalable as the width of the hidden layer would become impractically large for even very simple problems. Moreover, there is virtually no guarantee that the network would be able to learn the desired approximation in real-life scenarios. It is therefore common – and intuitively makes much sense – to deepen the network by adding more hidden layers. This allows the network to better approximate more complex functions using a series of simpler transformations, rather than only one much very complex transformation.

2.2 Training the network

As it was established in Section 2.1, the goal for the network is to approximate some target function f^*. Additionally, it was established that the network represents a series of differentiable functional transformations, each having its own set of parameters.

The process of training the network is therefore nothing more than an *optimization problem* – one of finding a network parameter set θ that allows the network to approximate the target function f^* in the best possible way.

To evaluate the quality of this approximation, we choose a differentiable *objective function* (often also referred to as *loss* or *error function*), which allows us to quantify the accuracy of the network's predictions. Which particular loss function should be used is determined by the nature of the problem that the network is meant to solve – for regression problems, one can use the *mean squared error* loss, for classification purposes, the *cross-entropy* loss is a common choice.

The process of minimizing the value of the chosen loss function then corresponds to optimizing the network parameters to achieve better performance. We say that the network *learns* the parameter values by minimizing the loss. Because the loss function is always differentiable with respect to any weight or bias in the network, a *gradient-based* numerical approach to the parameter optimization can be used.

To update the network parameters, first, it is necessary to compute the gradients of the loss function with respect to every network parameter. This is done using the *backpropagation algorithm*. The computed gradients indicate the direction of the steepest loss function value increase. Therefore, subtracting the gradient (multiplied by a small constant called *learning rate*) from the corresponding parameter values is directly equal to updating the network parameters in such a way that the loss function value decreases.

This whole process of obtaining the network prediction for the training set, computing the loss function value, computing the gradients, and then updating the network parameters accordingly is referred to as the *gradient descent* algorithm. This algorithm also has many variants, which further optimize and increase the efficiency of the whole learning process, typically based on some stochastic assumptions, e.g. the *stochastic gradient descent* variant.

2.3 The problem of sequential modeling

The traditional feed-forward neural network topology, although powerful, is not a universal solution for all deep learning problems. One of the limitations of this architecture is its implicit ability to only process each input data point as a singular piece of information with no temporal context. For each input vector, we typically obtain one corresponding output regardless of what other inputs the network has seen up to this point – in other words, feed-forward networks have no sense of memory.

For some deep learning problems and tasks, this temporal context is absolutely crucial, if we are to find an effective solution. Domains like speech recognition, speaker verification, and natural language processing (or in the case of this thesis – voice activity detection) present us with challenges requiring us to be able to process sequential data.

Some neural network topologies are able to address such problems without introducing significant changes to the basic model structure, for example, *time-delay neural networks* [47, 35], which have been found quite successful in solving many speech processing-related problems such as speaker identification [43]. Other approaches simply combine the input features at several neighboring time steps into one feature vector, bringing in some sense of the past and future temporal contexts, and use it as an input of a regular deep feed-forward network [45].

However, there is one neural network topology, whose most basic purpose is to be able to model these temporal dependencies implicitly – *recurrent neural networks*.

2.4 Recurrent neural networks

Recurrent Neural Networks (RNN) are a class of artificial neural networks designed primarily for sequence modeling. Rather than regular artificial neurons, RNNs consist of units referred to as *recurrent cells*. These cells have two special properties the regular neurons lack:

- a *hidden state* (sometimes referred to as a *cell state*),
- a *recurrent connection* to the hidden state from the previous time step.

The hidden state is the cell's abstract representation of previous time steps, essentially acting as a memory unit. This allows the RNN cell to condition its output not only on its learned parameters and the current input but also on the accumulated value of this hidden state. Therefore the network can learn to account for temporal dependencies in the processed data, thus becoming suitable for sequential modeling.

The basic, simple recurrent neural network layer can be formalized using the following formulas:

$$\begin{aligned} a_t &= b + Wh_{t-1} + Ux_t, \\ h_t &= \tanh(a_t), \\ o_t &= c + Vh_t, \end{aligned} \tag{2.4}$$

where U, V, W are weight matrices, b, c denote the bias vectors, a_t is the vector of hidden state activations for time step t, h_t is the hidden state vector transformed using the *hyperbolic tangent* function, h_{t-1} is the hidden state vector from the previous time step, and o_t denotes the output activation of the recurrent layer for time step t. The output activation vector then can be further transformed using another non-linear activation function.

Recurrent networks can be visualized in the form of a computational graph as shown in Fig. 2.2. This graph can also be depicted as *unfolded*, meaning we visualize the dependencies between the individual time steps, emphasizing the recurrent connections.

2.4.1 Modeling long temporal dependencies

Recurrent neural networks could – in theory – be used for many types of tasks, that are sequential in nature. However, it turns out that the basic simple RNN architecture car-

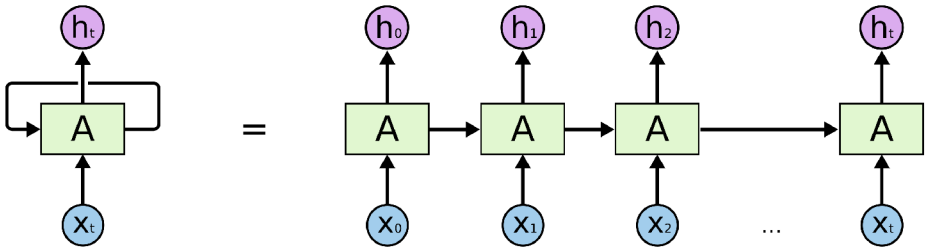

Figure 2.2: Visualization of a recurrent neural network, depicted both in the regular and unfolded variants. The diagram was taken from [31].

ries some crucial limitations, which prevent it from being practically applicable to most problems.

The basic problem is that it is rather difficult to reliably train RNNs, as they tend to suffer from the problems of *vanishing* and *exploding* gradients when using traditional simple gradient-based optimization methods [1, 2, 33].

Moreover, even if the network is stable during training (the gradients do not explode), and can store memories, it is generally unable to reliably learn and represent long-term dependencies in the input data. This is because the weights assigned to long-term interactions become exponentially smaller compared to the short-term ones.

Long temporal dependencies are however crucial for some types of problems. Taking an anecdotal example: long sentences often contain words at the beginning, which are crucial for a correct understanding of the sentence – they provide long-term context, which sometimes prevails across multiple sentences.

These limitations eventually led to the introduction of *Gated Recurrent Neural Networks* (gated RNNs). Gated RNNs are a special category of recurrent neural networks which expand upon the basic simple RNN architecture by introducing several internal gating mechanisms. These gating mechanisms have two primary roles:

1. They introduce recurrent pathways, where gradients can flow more freely, as these pathways are not statically controlled by any weight matrices, but rather dynamically using these gates, whose parameters are learned instead.

2. They allow the network to form its hidden state in a more refined manner, preserving important contextual information or, on the contrary, intentionally ignoring or forgetting other pieces of it.

The two main representatives of this class of RNNs are the *Long Short-term Memory* (LSTM) and the *Gated Recurrent Unit* (GRU). The LSTM – the most widely used gated RNN variant nowadays – is discussed in the next Section 2.4.2.

2.4.2 Long short-term memory

The *Long Short-term Memory* (LSTM) is a gated RNN variant first introduced in [17]. Its purpose is to address some of the problems described in Section 2.4.1, most importantly the problem of learning and exploiting long temporal contexts.

The LSTM splits the original RNN state into two properties, both of which are recurrent. The first property is the LSTM *cell state* c_t, which serves as an information accumulator and is controlled by the LSTM gating mechanisms. The second property, now called the *hidden state* h_t, becomes the output of the LSTM cell at each time step while also being

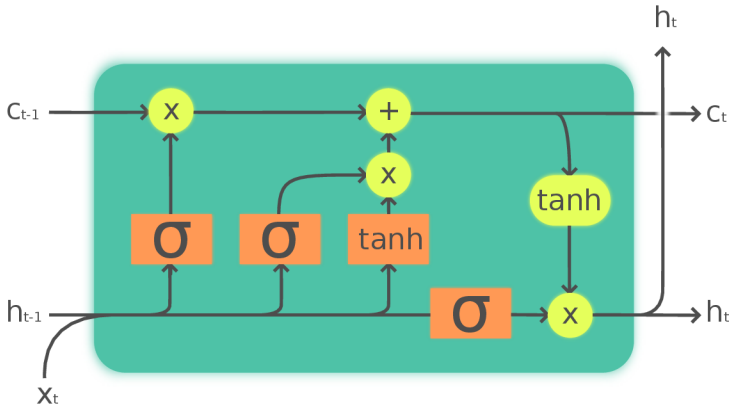

Figure 2.3: Diagram of an LSTM cell. The σ and *tanh* symbols denote the logistic sigmoid and hyperbolic tangent activation functions, respectively. c_t, and c_{t-1} denote the cell state values, h_t, and h_{t-1} denote the hidden state values, and x_t denotes the input of the LSTM cell. The diagram was obtained from [10] and is licensed under the CC-BY License, by Guillaume Chevalier. The legend part of the original image was removed.

recurrent. For each gating mechanism, this hidden state value is used as a secondary input along with the LSTM input x_t (see Figure 2.3).

An LSTM layer (and every cell for that matter) has three separate gating mechanisms [31, 51], the first one being the so-called *forget gate*:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{2.5}$$

where W_f, U_f are weight matrices, b_f is the bias vector, f_t denotes the forget gate value vector at time step t, $\sigma(\cdot)$ is the logistic sigmoid, and h_{t-1} is the hidden state vector from the previous time step. The forget gate can restrict the information retained in the cell state.

The second gating mechanism is the *input gate*, which controls the accumulation of the LSTM cell input to the internal *cell state*:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i). \tag{2.6}$$

The last gating mechanism is the *output gate*, which controls the output of the LSTM cell:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o). \tag{2.7}$$

Before updating the LSTM state, a vector of *candidate values* $\tilde{C}_t$ is computed. These values could potentially be accumulated to the cell state, that is if the input gate allows it. This candidate vector is then combined with the previous cell state vector C_{t-1}, controlled by the forget and input gates, resulting in a new value of the cell state C_t:

$$\begin{aligned} \tilde{C}_t &= \tanh(W_C x_t + U_C h_{t-1} + b_C) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \end{aligned} \tag{2.8}$$

where $\odot$ denotes element-wise multiplication (also known as the *Hadamard product*).

The resulting new cell state is then further transformed using the hyperbolic tangent function and the current value of the output gate, to produce the new hidden state vector h_t:

$$h_t = o_t \odot \tanh(C_t). \tag{2.9}$$

Contrary to the fixed-value weight assigned to the recurrent connection of a regular RNN hidden unit, the gates allow the LSTM cell to manipulate the internal cell state dynamically – information accumulated at one time step can be retained for many time steps, or suddenly discarded or added to if a more crucial piece of information emerges in the input. Of course, the parameters – the weights and biases – corresponding to these gates are learned during training.

The LSTM is nowadays by far the most widely and commonly used RNN variant, achieving state-of-the-art performance for many sequential modeling problems.

Chapter 3

Speaker representations

This chapter briefly introduces the concept of *speaker embeddings*[1] – low-dimensional vectors of fixed length, used as a means for representing a particular speaker.

These speaker representations are a crucial concept in the speaker identification/verification domain since they allow to compactly store information about an utterance's acoustic characteristics. When used for speech, these vectors contain information about a particular speaker's voice characteristics, essentially creating a *voice print* of the encoded speaker.

The encoded information can be extracted in a number of ways, depending on the embedding type. Some types of speaker representations are based on a statistical approach to speaker modeling (i.e. i-vectors [11]), other types are extracted using deep neural networks that are generally trained for the task of speaker identification/verification (x-vectors [43], d-vectors [45, 48]). Each speaker embedding type, therefore, has different properties, different interpretations, and can be suitable for different applications.

Apart from speaker recognition-related tasks, speaker embeddings can also be used to perform speaker adaptation, if one is to build a system that requires such techniques for optimal performance – this is further discussed in Chapter 4.

The following Sections 3.1, 3.2, and 3.3 introduce three of the nowadays most widely used types of speaker representations. Special attention is given to the d-vector, as this embedding type is integral to the target speaker voice activity detection method explored in this work.

3.1 i-vectors

The first speaker vector type to discuss is the i-vector [11]. I-vectors (also referred as *intermediate* or *identity vectors*) are speaker representation vectors based on a statistical, unsupervised approach to speaker modeling. The method was first introduced as an evolution of the *Joint Factor Analysis* (JFA) [22] approach to speaker representation.

The JFA approach was based on the notion of modeling the channel and speaker variability as independent subspaces. However, later experiments showed that the resulting estimated channel factors also contain information about the speakers, despite being supposed to only model channel effects. As a consequence, these two subspaces are for the purpose of i-vector estimation considered as one *total variability space*.

[1] Even though the term *embedding* is generally used for speaker representations obtained from a deep neural network, to avoid confusion, please note that in this thesis this term is sometimes used even for i-vectors, as they in a sense fit the definition of an embedding vector too.

I-vector extraction is the process of mapping a sequence of feature vectors (typically Mel-frequency cepstral coefficients), to a fixed-length vector. First, a k-component Gaussian Mixture Model (GMM) referred to as the *Universal Background Model* (UBM) has to be computed using high amounts of training data. The encoded speaker utterance is then represented by a speaker and channel-dependent supervector M, which is obtained by appending together first-order Baum-Welch statistics extracted from the utterance using the UBM. That is for each GMM component of the UBM. The obtained supervector is then assumed to obey a factor analysis model:

$$M = m + Tw \tag{3.1}$$

where m is a speaker and channel-independent mean supervector, obtained from the UBM, T is the so-called total variability matrix, rectangular and of low-rank. This matrix contains the eigenvectors with the largest eigenvalues of the total variability covariance matrix, essentially modeling the directions of the largest variability in the training data. Finally, w is a latent vector with a standard-normal prior, which represents the total variability factors.

Given an utterance u, the i-vector is then obtained as a MAP point estimate of w for this utterance [15].

3.2 x-vectors

I-vectors were for a long time the industry standard for both text-dependent and text-independent speaker recognition tasks for many years. However, given the success of deep neural networks in virtually any other machine learning domain, a lot of research had been dedicated to deep neural network-based speaker modeling, resulting in the introduction of new embedding types such as x-vectors and d-vectors.

X-vectors [42, 43] are a speaker embedding type extracted using a time-delay neural network [47, 35], which is trained to identify the speakers from the training set based on the supplied utterance (see Figure 3.1).

The network processes the utterance frame by frame as a whole, passing the output to a *statistics pooling* layer. This layer aggregates over the input segment and computes its mean and standard deviation. These statistics are then concatenated and passed to an additional hidden layer, from which the resulting embedding vectors can be extracted.

X-vectors are nowadays a widely used speaker embedding type and are a popular speaker embedding choice e.g. for speaker diarization purposes [25].

3.3 d-vectors

The third and in the context of this work the most important speaker embedding type is the d-vector. The d-vector is a term that can generally be used for a speaker embedding, which is extracted from a deep neural network trained for the speaker verification/identification task (one of the key differences from x-vectors being the absence of the statistics pooling layer in d-vector systems).

The d-vector concept was first introduced as a deep neural network speaker embedding in [45]. This approach was designed for text-dependent speaker verification tasks, though it was suggested to be extendable to text-independent problems. The method utilized a deep feed-forward network architecture, to process the input filterbank features on frame-level.

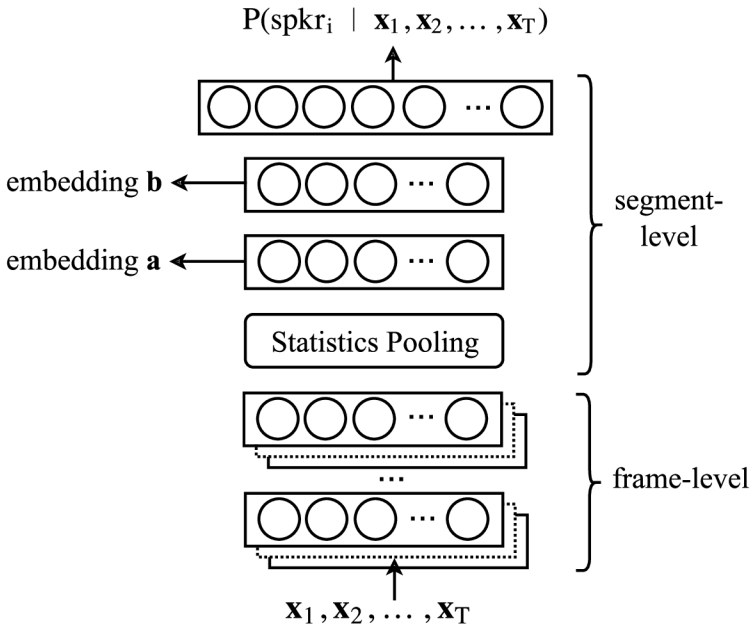

Figure 3.1: Simplified visualization of the DNN used to extract x-vectors. The *frame-level* segment depicts the time-delay neural network part of the whole system, passing its output to the *segment-level* part of the network, which ultimately produces probabilities for each speaker in the training set. The x-vectors are then extracted at the embedding layer **a**. The diagram was obtained from [42].

The filterbank features from several neighboring frames are combined into one feature vector, bringing in some sense of the present and future temporal contexts, and passed through the network. At each time step, the activations from the last hidden layer are extracted, L2-normalized, and averaged over the whole utterance, resulting in an utterance-level embedding vector, the d-vector.

Consequently, a lot of effort has been dedicated to researching DNN-based speaker verification systems, which rely on an end-to-end training approach [9, 26, 52].

One of the more prominent end-to-end approaches was recently introduced in [48]. In this approach, an LSTM-based speaker verification system was trained using a custom loss function, referred to as the *generalized end-to-end loss*, designed to always maximize the discriminativity between the most similar speaker pairs. The d-vector embeddings are extracted in a sliding-window manner, L2 normalized, and averaged over the whole utterance (as shown in Figure 3.2).

This system achieved state-of-the-art performance for both text-dependent and text-independent speaker verification tasks, being successfully used (among other areas) for speaker diarization [49], source separation [50], or target-speaker voice activity detection [13].

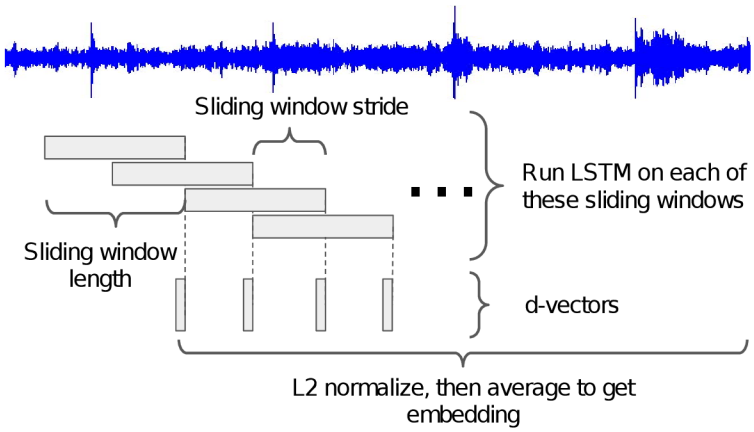

Figure 3.2: Visualization of the sliding window d-vector embedding inference from [48]. The diagram was obtained from the same source.

3.4 Embedding similarity metrics

For some scenarios – typically for the purpose of speaker verification – it is useful to be able to quantify the similarity of two embedding vectors, with the result ideally in the form of a scalar similarity score. The method used for computing the speaker verification score for the two vectors is then usually dependent on the type of the embedding vectors, assumptions about their spatial distribution, etc.

Speaker verification systems based on i-vectors often use a Probabilistic Linear Discriminant Analysis (PLDA) [19] backend to compare the speaker representations and enable the speaker verification decisions. However, PLDA-based classifiers are not limited to i-vectors only, as they are often used for other embedding types, such as x-vectors [43].

Another (rather simpler) similarity measure that can be used is the *cosine similarity*. The cosine similarity score of two vectors is computed using the following formula:

$$similarity = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|}, \tag{3.2}$$

where $\mathbf{A}$ and $\mathbf{B}$ are the two embedding vectors respectively.

The value of the score is directly dependent on the angle between the vectors – vectors that form a smaller angle will also have a higher cosine similarity score. Intuitively, this angle, especially in the case of the high-dimensional, sparse embedding space, can be a good indication as to whether the two embedding vectors are similar or not.

The speaker verification decision can then be made by simply choosing a score threshold – if the cosine similarity value is higher than the chosen threshold, it can be ruled that the embeddings come from the same speaker.

Chapter 4

Speech detection

In this chapter, the core topic explored in this work is covered – target speaker speech detection.

Section 4.1 starts with presenting the general problem of voice activity detection, discussing some of the typically used techniques to implement voice activity detection systems.

Section 4.2 then expands upon the basic problem by focusing on a particular target speaker when trying to detect speech signals. Section 4.2.1 then covers the speaker-conditioned voice activity detection method proposed in [13], which is the core target speaker speech detection approach investigated in this thesis.

4.1 Voice activity detection

Voice Activity Detection (VAD) represents the problem of detecting speech in audio signals. In its purest form, it can be thought of as a binary classification problem. Every frame of the source audio is evaluated against two hypotheses [37]:

$$\begin{aligned} H_0: &\quad \mathbf{x}_t = n \\ H_1: &\quad \mathbf{x}_t = n + s, \end{aligned} \tag{4.1}$$

where the first hypothesis H_0 indicates that the classified frame only consists of non-speech signals n such as noise, and the second hypothesis H_1 expresses that the current frame consists of a speech signal s and potential background noise signals.

To classify the frame, one can simply choose the hypothesis with the higher posterior conditional probability of the two with respect to the current frame $\mathbf{x}_t$, effectively enforcing the maximum a posteriori classification approach:

$$\text{VAD}(\mathbf{x}_t) = \begin{cases} \text{non-speech} & P(H_0|\mathbf{x}_t) > P(H_1|\mathbf{x}_t) \\ \text{speech} & \text{else.} \end{cases} \tag{4.2}$$

Voice activity detection is typically used as a pre-processing component of larger speech processing systems [37]. This is because the presence of irrelevant information and noise in the processed speech signal can hinder the performance of systems such as Automatic Speech Recognition systems (ASR) or Speaker Verification systems (SV). Moreover, ASR and SV systems are typically quite demanding in terms of computational resources, especially when compared to a typically very small and lightweight VAD model. Therefore it might be desirable to simply discard all source audio frames that do not contain any speech

information, as doing so can both improve the downstream system performance and save some computational power.

On the other hand, it is undesirable for any VAD system to false-reject any speech frames as then important information might be lost. This is especially a challenge if the VAD is to operate in acoustically challenging conditions, including environments with high levels of background noise or reverb. It is therefore both useful and necessary to account for these conditions when designing the system, for example by applying augmentation strategies to the training data.

Voice activity detection usually consists of three main stages:

1. Feature extraction,
2. VAD decision,
3. and VAD decision smoothing.

The feature extraction stage is highly dependent on the approach taken to VAD modeling, as different methods might require different features. However, since most modern machine learning-based VAD methods primarily use acoustic features only, it can be useful to build the VAD system around the same acoustic feature type used by the downstream components. This can lead to further resource savings, as the acoustic features are in this case computed only once.

The task of the VAD model itself is then to classify the input features as either speech or non-speech. The possible approaches to implementing such a model are discussed in Section 4.1.1.

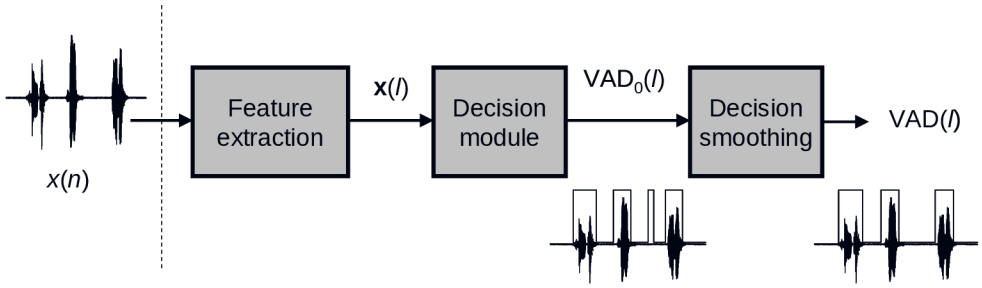

Figure 4.1: Example of a typical VAD system architecture. Diagram taken from [37].

Lastly, the decision smoothing stage is used to combat VAD decision errors, most often caused due to high levels of background noise. In such conditions, the VAD decision might become "jittery" and unreliable, increasing the amount of false negative/positive decisions.

The heuristics used for VAD decision post-processing are generally quite simple, with one of the most widely used ones being *hangover* [4].

4.1.1 Voice activity detection methods

Depending on the use case, available resources, or system accuracy/performance requirements, one can choose one of many methods to implement voice activity detection. Generally, there are two main ways to categorize the different approaches:

- Feature and heuristics-driven methods, which make use of different features and statistics extracted from the source audio.

- Machine learning-based methods, which rely on statistical or neural network-based approaches to infer the decision policies from the training data.

Feature and heuristics-driven methods typically extract a number of different features from the processed signal and try to make an informed decision, often with the help of a heuristic or a specialized algorithm [44]. The extracted features often include energy information, spectral information, zero-crossing rate, long-term spectral divergence [38], etc.

The heuristics-driven methods were, however, slowly driven out by machine learning-based VAD approaches, which nowadays offer state-of-the-art performance, accuracy, and noise robustness. On the other hand, some applications might still find benefit even in naive VAD methods such as simple spectral energy thresholding – used for example in [42] – if excellent noise robustness is not required.

Some of the newer VAD methods rely on a statistical approach to the problem [6]. However, even the statistical approaches are being pushed aside by VAD methods that utilize artificial neural networks, typically operating on acoustic features only. a multitude of neural network architectures have been tried and evaluated for this task [53, 8], with one of the more popular being the LSTM [14, 7], showing state-of-the-art performance. Additionally, LSTM-based approaches are especially interesting in the context of this particular work, because LSTM networks can be used as *streaming* models, making them easily adaptable for online inference scenarios.

4.2 Speaker-conditioned voice activity detection

Speaker-conditioned Voice Activity Detection (SCVAD) is essentially an evolution of the standard VAD task, as now the system is also required to distinguish between speech frames coming from one particular target speaker and everyone else. We can therefore modify the the hypotheses defined in Equation 4.1 to accommodate for the new classification classes:

$$\begin{aligned} H_0 &: \quad \mathbf{x}_t = n \\ H_1 &: \quad \mathbf{x}_t = n + s_n \\ H_2 &: \quad \mathbf{x}_t = n + s_t, \end{aligned} \tag{4.3}$$

where s_t and s_n denote target speaker and non-target speaker speech signals, respectively.

Similarly to the previous binary VAD classification case, we can again choose the hypothesis with the highest posterior probability:

$$\text{SCVAD}(\mathbf{x}_t) = \arg\max_w P(H_w|\mathbf{x}_t), \;\; w \in \{0, 1, 2\}. \tag{4.4}$$

The first thing that comes to mind when designing a SCVAD system is that there are already two important speech processing disciplines, which focus primarily on being able to distinguish between different speakers: speaker recognition and speaker diarization.

Therefore naturally, the first possible way to approach implementing a SCVAD system would be to combine a speaker recognition system with a classical VAD system. The speaker recognition system would then simply classify the audio frames that the VAD had labeled as speech. The problem with such an approach is that speaker verification systems are generally implemented using models that are quite big, typically in terms of millions of parameters. Using such a system therefore inherently leads to higher resource consumption,

limiting its usefulness in scenarios, where the resources are expensive. This is especially apparent in contrast to the generally very lightweight basic VAD models.

Additionally, one of the challenges this VAD and SV system combination would have to overcome is the final decision granularity. VAD systems are generally capable of frame-level streaming inference, producing a speech/non-speech probability for every individual frame. On the other hand, SV systems often operate in a window-level or a segment-level manner, which could result in higher overall decision latency. What is more, adapting a speaker verification system to frame-level inference can pose quite a challenge in terms of retaining the speaker verification decision quality.

The second option would be to directly use a speaker diarization system. Speaker diarization is the problem of establishing boundaries between individual speakers in a recording – which brings us to the first drawback.

A conventional diarization system [25, 49] is designed to establish boundaries between *all* present speakers. Therefore, a lot of effort has to go towards determining the number of speakers in the recording. That includes extracting embedding representations for the whole utterance in a sliding window manner, clustering, and then segmenting the original recording based on the calculated boundaries (see Figure 4.2).

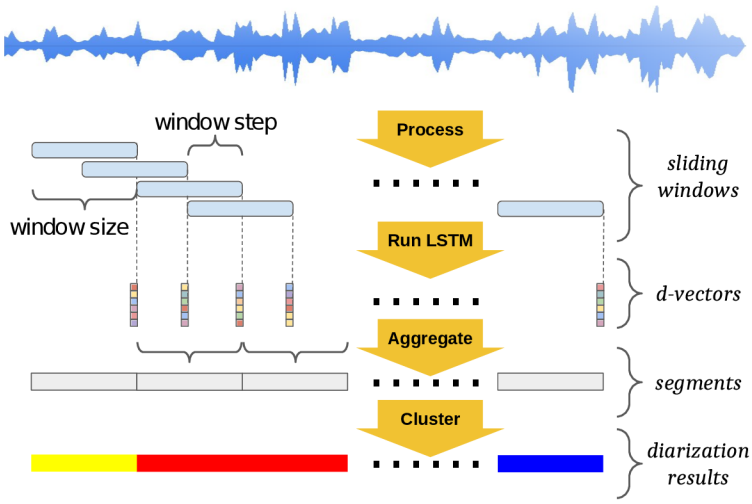

Figure 4.2: Example of a clustering-based speaker diarization method utilizing d-vector embeddings. Flowchart obtained from [49].

For one target speaker, all this is unnecessary, as it is only the target speaker that the system is required to find reliably. Therefore, again, such a solution is needlessly expensive in terms of computational resources, though it would most likely provide the desired results accuracy-wise.

The third option for implementing a SCVAD system would be to adopt a conventional VAD system for target speaker speech detection. This could potentially result in a SCVAD system, that retains the typical VAD system characteristics in terms of model size, resource demand, and latency, while still being able to detect a target speaker's speech signals.

To perform such adaptation, the system could be trained to *draw its attention* towards the target speaker by providing their abstract representation along with the acoustic features, most often in the form of a speaker embedding (i-vectors, d-vectors) obtained previously during an enrollment phase. These representations are often obtained from systems much more complex than a typical VAD, so effectively, the aim would be to teach

the VAD to "distill" some knowledge from these abstract representations and identify the target speaker based on their acoustic "profile".

Similar speaker adaptation approaches have previously been adopted also in the domains of speech recognition [21], speech extraction [50, 12], and quite successfully also in diarization [30]. Some of these approaches use secondary auxiliary systems trained to extract the speaker representation in a way that is specific for that particular system, other approaches use speaker embedding vectors directly.

4.2.1 Personal voice activity detection approach overview

One method, which utilizes embedding-based speaker adaptation, is the rather novel approach to speaker-conditioned VAD recently introduced in [13].

This method, originally called *Personal VAD* (PVAD), aims to address the SCVAD problem by expanding the classification capabilities of an LSTM-based classical VAD model. The whole system is trained to distinguish not only between speech and non-speech audio frames but also to detect and identify speech frames belonging to a particular target speaker.

The original motivation for this SCVAD method was its potential use for on-device speech recognition scenarios. That is, the goal was to create a system capable of detecting speech signals of a target speaker in real-time, ideally while also retaining some of the characteristics of a typical VAD system. The end result would ideally be:

- A small, lightweight model with minimal latency and minimal computational resource requirements.
- A model that is able to operate accurately in acoustically challenging environments, including noisy and reverberant conditions.

As we are dealing with an online classification scenario (and to minimize latency), it would be best if the resulting system could operate as a *streaming* model. This is why using a VAD architecture based on an LSTM network might be desirable. LSTM-based VAD architectures have become increasingly popular for sequential modeling of the VAD task, all that while showing state-of-the-art performance even in acoustically challenging conditions [14].

Now, not all methods presented in this work do actually meet the lightweight criteria for the system. Some of the personal VAD architecture variants presented in section 4.2.2 require a speaker verification system at runtime, essentially creating a fusion of SV, diarization, and VAD systems. That being said, the more heavy-weight solutions generally offer better performance in terms of prediction accuracy. Therefore, it can be argued that it is still worth exploring those particular approaches, as they might prove useful in situations, where resource limits are not a concern.

The following Section 4.2.2 describes the four main personal VAD architectures as presented in [13].

4.2.2 Personal voice activity detection system architecture

A personal VAD system consists of two main components.

Speaker verification system The first component is a speaker verification system used to extract speaker embeddings from the processed audio. For this purpose, the text-

independent d-vector system introduced in [48] was used. This system has two primary uses:

1. To extract an *enrollment speaker embedding* for the target speaker. The embedding will be used to either provide the system with a representation of the target speaker's voice characteristics or to obtain speaker verification scores for each individual frame.

2. To extract frame-level embedding vectors over the whole processed utterance. These secondary embedding vectors are used to issue *speaker verification scores* to each individual frame.

To satisfy the latter of these two requirements for the SV system, it was necessary to modify the actually used implementation of the system.

The d-vector system used for embedding extraction operates in a sliding window manner, always returning one 256-dimensional embedding vector for a window of 160 frames. However, as the system's architecture is LSTM-based, the system can be modified to operate in a streaming manner, returning an embedding vector for every single input frame. Each d-vector is then compared with the target speaker enrollment embedding using cosine similarity, giving us the speaker verification scores for each frame.

Obviously, such modification can raise questions about the quality of the extracted d-vectors, as the system is forced to process sequences of arbitrary lengths, without resetting the LSTM state. These concerns are addressed in Section 7.5, where this baseline scoring method is evaluated against two other methods that I propose as potential alternatives.

VAD system The second and primary component is the actual VAD system, which will be trained for the personal VAD task. This system was proposed to consist of a 2-layer LSTM network of 64 cells, followed by one additional fully connected layer of 64 neurons. This network architecture is the same for all personal VAD system variants described in the following sections.

The inputs of the personal VAD are then a combination of the following:

- Acoustic features $\mathbf{x}_t$,
- the speaker verification scores s_t issued to each individual frame,
- the target speaker embedding $\mathbf{e}_{\texttt{target}}$ obtained during the enrollment process.

The acoustic features used in this work were 40-dimensional log Mel-filterbank energies with 25 ms width and 10 ms overlap. The same acoustic features are used by the d-vector extractor system, therefore they can be extracted only once and used for both the SV and the VAD systems. This is especially helpful when performing frame scoring, as every frame has to be processed by the SV system to obtain a speaker verification score, and afterward, both the obtained score and the audio frame are passed to the PVAD system.

The resulting combined feature vector $\hat{\mathbf{x}}_t$ is then used as the input of the personal VAD system, which produces class probabilities $\mathbf{z}_t$ for target speaker speech (`tss`), non-target speaker speech (`ntss`) and non-speech (`ns`):

$$\text{PVAD}(\hat{\mathbf{x}}_t) = \mathbf{z}_t = [z_t^{\texttt{ns}}, z_t^{\texttt{ntss}}, z_t^{\texttt{tss}}]. \tag{4.5}$$

The following sections further describe the four personal VAD architecture variants, as they were introduced in [13]. The main differences between the systems stemming from the

input feature combination used by each particular architecture. Three of the architectures – SC, ST, and SET – make use of a speaker verification system to support the personal VAD decision. Diagrams for architectures are depicted in Figure 4.3.

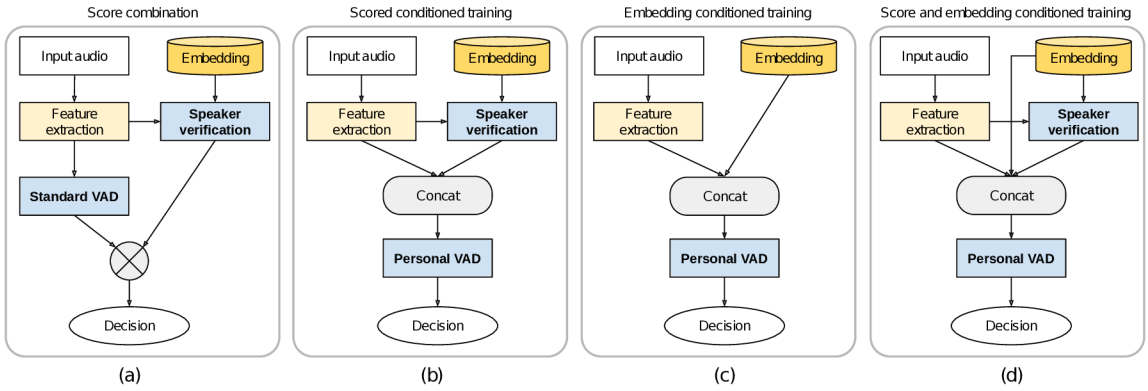

Figure 4.3: Personal VAD architecture diagrams. (a) depicts the baseline SC architecture, (b) depicts the ST architecture, which utilizes a combination of acoustic features and frame scoring. (c) depicts the embedding conditioned ET architecture, and (d) is the SET system, which combines the acoustic features, the scores and the target speaker embedding vector. The diagrams were taken from [13]

System 1: Score combination (SC)

The baseline architecture for the personal VAD task consists of a classical VAD model, which takes the acoustic features features as input and produces speech probability p_t^s for each incoming frame $\mathbf{x}_t$ at each time step t:

$$p_t^s = \mathrm{VAD}(\mathbf{x}_t).$$

Each individual frame is also passed through the speaker verification model[1] and a d-vector embedding $\mathbf{e}_t$ is obtained for that frame. This d-vector is then compared with the target speaker embedding $\mathbf{e}_{\texttt{target}}$ using cosine similarity, thus obtaining a speaker verification score for each frame:

$$s_t = \cos(\mathbf{e}_t, \mathbf{e}_{\texttt{target}}).$$

The obtained score is then combined with the speech probability p_t^s to produce unnormalized personal VAD probability value z_t^k for every class k, using the following formula:

$$\begin{aligned} z_t^{\texttt{ns}} &= 1 - p_t^s \\ z_t^{\texttt{ntss}} &= (1 - s_t) \cdot p_t^s \\ z_t^{\texttt{tss}} &= s_t \cdot p_t^s \end{aligned} \tag{4.6}$$

It is obvious, that this baseline system represents quite a naive approach to the personal VAD task. The biggest issue with this approach is that there is no explicit threshold selected for the speaker verification scores – it would at least be sensible to statistically derive this threshold using the training score values. This was tried in the final experiments in Section 7.3, however, with not much success. The next architecture addresses this problem by treating the score value as an additional feature.

[1] Which is, as previously mentioned, modified to support frame-level streaming d-vector inference.

System 2: Score conditioned training (ST)

The ST architecture expands on the baseline by combining the acoustic features $\mathbf{x}_t$ with the speaker verification score s_t into one 41-dimensional feature vector:

$$\hat{\mathbf{x}}_t = [\mathbf{x}_t, s_t].$$

The system is then trained using these features to directly produce Personal VAD class probabilities – non-speech, non-target speaker speech and target-speaker speech:

$$\mathbf{z}_t = [z_t^{\texttt{ns}}, z_t^{\texttt{ntss}}, z_t^{\texttt{tss}}].$$

This system is expected to perform better than the baseline, as it learns to infer the output probabilities from the input acoustic features and scores directly, rather than using a set-in-stone transformation function as the SC architecture.

However, the main drawback of this architecture – which is shared between the ST and the SC systems – still prevails. It is the fact that the quality and accuracy of the system's final decision are directly dependent on the utilized speaker verification score values. In order for this system to perform well, the discriminativity of the embedding vectors used for frame scoring has to be as high as possible. Otherwise, the system's output will contain more false positives and false negatives for both target and non-target speaker speech. This problem is further addressed and discussed in Section 7.5.

System 3: Embedding conditioned training (ET)

The ET architecture represents the *ideal desired solution* to the personal voice activity detection problem, as it does not require a speaker verification system at runtime for frame scoring, making it a very lightweight solution.

This architecture combines the enrollment embedding $\mathbf{e}_{\texttt{target}}$ for the target speaker with the acoustic features, resulting in a 296-dimensional feature vector:

$$\hat{\mathbf{x}}_t = [\mathbf{x}_t, \mathbf{e}_{\texttt{target}}].$$

This system is expected to learn to infer the relationship between the input features and the target embedding, distilling this knowledge for classification purposes, and adapting to the target speaker. However, as the d-vector embedding space can potentially be quite sparse due to the dimensionality of the embeddings, it is expected that this system will only perform and generalize well when trained on a dataset with a large number of speakers.

System 4: Score and embedding conditioned training (SET)

The last personal VAD architecture combines the characteristics of the previous two systems. The system input consists of the acoustic features, the target speaker embedding, as well as the speaker verification score for the current frame. This gives us a 297-dimensional input feature vector:

$$\hat{\mathbf{x}}_t = [\mathbf{x}_t, s_t, \mathbf{e}_{\texttt{target}}].$$

Even though it is expected that this architecture will provide the best results of the four, it still requires a running speaker verification model at runtime, so that frame scoring can be performed.

4.2.3 Loss functions

Because personal VAD represents a multiclass classification problem, it is possible to train the model by minimizing the categorical cross-entropy loss (also known as the *softmax loss*):

$$L_{\text{CE}}(y, \mathbf{z}) = -\log \frac{\exp(z^y)}{\sum_k \exp(z^k)}, \tag{4.7}$$

where $\mathbf{z}$ is the vector of pre-softmax network outputs for each class, y denotes the target class label, z^y denotes the system's output for the target class and z^k denotes the system's output for the k-th class.

Additionally, [13] also proposes the use of a new loss function, the *Weighted Pairwise Loss* (WPL), which allows to issue different weights to each class pair:

$$L_{\text{WPL}}(y, \mathbf{z}) = -\mathbb{E}_{k \neq y} \left[w_{<k,y>} \cdot \log \frac{\exp(z^y)}{\exp(z^y) + \exp(z^k)} \right], \tag{4.8}$$

where $w_{<k,y>}$ is the weight between the classes k and y. In doing so, confusion errors between certain classes can have lesser impact on the system's performance. By setting the weight of `(ns,ntss)` to a smaller value than `(tss,ntss)` or `(ns,tss)`, the system should focus more on distinguishing the target speaker's speech from the other two classes, more so than preoccupying itself with `(ns,ntss)` confusion errors.

Chapter 5

Data

To train the proposed systems, it is necessary to find a suitable speech dataset, ideally one with the following properties:

- The dataset should contain speaker turns so that the system can learn to distinguish the target speaker from the other speakers present in the recording.
- For each individual speaker, enrollment utterances should be present so that it is possible to extract their embedding representations.

For this purpose, the openly available LibriSpeech [32] corpus was used to generate a dataset that would match the criteria above. The process of generating the resulting dataset is further described in Section 5.2.

5.1 LibriSpeech

The LibriSpeech [32] corpus is a standard, freely available[1] dataset of read English speech, totaling at almost 1000 hours of speech data.

LibriSpeech consists of seven separate subsets, each having one of two suffixes: `clean` or `other`. These suffixes were assigned to the particular speakers in the sets based on word error rate scores achieved in one of the initial LibriSpeech evaluations. There are four smaller subsets: two `dev` sets, and two `test` sets, primarily meant for development and testing, respectively. Then, the three primary LibriSpeech subsets are the 100-hour, 360-hour, and 500-hour `train` sets. The parameters of the individual subsets are shown in Table 5.1.

The whole LibriSpeech corpus was sampled at 16 kHz and the audio is stored in the `flac` format. Additionally, word transcripts are provided for all utterances.

Lastly, to create VAD ground truth annotations for the data, it was necessary to get hold of transcript alignments for each individual utterance in the LibriSpeech dataset. The alignments used in this work were obtained from [27] and originally generated using the Montreal Forced Aligner [29].

5.2 Generating the dataset

To simulate speaker turns in the training data, I adopted the approach presented in [13]. The approach suggests to always retrieve n randomly chosen utterances from the original

[1] https://www.openslr.org/12

subset	hours	per-speaker minutes	female speakers	male speakers	total speakers
dev-clean	5.4	8	20	20	40
test-clean	5.4	8	20	20	40
dev-other	5.3	10	16	17	33
test-other	5.1	10	17	16	33
train-clean-100	100.6	25	125	126	251
train-clean-360	363.6	25	439	482	921
train-other-500	496.7	30	564	602	1166

Table 5.1: LibriSpeech corpus subsets and their respective parameters. Table taken from [32].

dataset, each coming from a different speaker. The parameter n coming from a uniform distribution:

$$n \sim \text{Uniform}(a, b),$$

where $a = 1$ and $b = 3$. These utterances are then concatenated, as are their ground truth annotations. After the selected utterances are used, they are erased from the pool of available utterances, so that no utterance is used more than once.

Additionally, at the time of feature extraction, a speaker present in the resulting utterance is chosen randomly as the target speaker and the ground truth labels are also altered accordingly. It should also be noted that since augmentation is performed on the generated data, the chosen target speaker may differ across the augmented variants of the original utterance, providing a little more variety.

A method that one could call *speaker dropout*, inspired by [30], was also experimented with. This approach alters the way a target speaker is chosen for utterances consisting of only one speaker's speech, randomly selecting a *different* speaker as the target with a probability of 0.3. This approach was, however, not used in the final generated datasets, as no improvements in the trained system performances were observed.

Using this approach, two separate datasets were generated for model training and evaluation, respectively. Because the system is expected to separate the target speakers from the non-targets, it was – for the purposes of cross-validation – important to make the validation set to be completely separate from the training set. That way, all the speakers in the validation set would represent novelty encounters for the system and the obtained results would better indicate the system's ability to generalize.

Training set The training set was generated using the three main LibriSpeech subsets. The three `train` sets together – the 100-hour, the 360-hour, and the 500-hour – end up totaling at around 960 hours of recorded speech and 2338 different speakers. Using the method described in Section 5.2, approximately 140 thousand unique concatenated utterances were generated, with no source utterance being used more than once. a histogram of the resulting concatenated utterance lengths is shown in Figure 5.1.

Validation set The validation set was generated using the remaining LibriSpeech subsets, concretely the two `dev`[2] and two `test` subsets. These four subsets are completely separate

[2]The `dev` subsets were used in addition to the `test` sets to provide some additional speaker variability to the validation set.

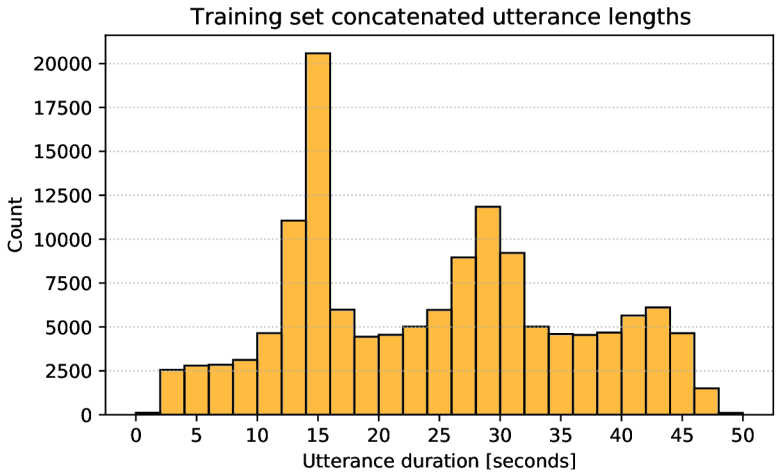

Figure 5.1: Histogram of utterance length distribution in the generated training set of 140 thousand concatenated utterances.

from the three main ones and though they are much smaller – totaling at around 20 hours of speech when combined only – they contain recordings of 146 more unique speakers that are not present in the three main partitions. The resulting generated validation set consists of approximately 5500 concatenated utterances.

5.3 Data augmentation

A very important aspect of creating a robust and accurate VAD system is to ensure its ability to perform well even in acoustically challenging conditions. These conditions can include the effects of reverberant rooms and spaces, and often also high levels of background noise. It is therefore desirable to augment the training data to match these potential conditions so that the model can learn to account for them.

To augment the training and testing data, the MUSAN [41] corpus was used in conjunction with a set of room both real and simulated room impulse responses from [24]. The resulting augmentation strategy used in this work is similar to [43]. Each concatenated utterances is augmented and thus replicated three times using:

1. **Reverb** – an impulse response is randomly chosen from the `RIRS_NOISES` corpus and applied to the clean utterance via convolution.

2. **Noise** – randomly chosen background noises are added to the clean track at one-second intervals, at levels ranging from 0 to 15 dB SNR.

3. **Music** – an instrumental music piece is randomly chosen from MUSAN and added to the clean utterance at levels ranging from 5 to 15 dB SNR.

The same augmentation strategy was applied to both the training and validation sets, resulting in both sets becoming four times their original size after augmentation.

Chapter 6

Implementation

This chapter discusses some of the interesting implementation aspects of creating the data preparation, feature extraction, and model training pipelines.

6.1 Data preparation and feature extraction

The implementation of the data preparation and feature extraction pipelines was one of the more challenging aspects of this work and has gone through multiple iterations. The pipelines were implemented using the Python 3 language in combination with some occasional shell scripting. Shell scripts were primarily used for manipulating the generated dataset and features and also to interface with the Kaldi Speech Recognition Toolkit [36], which was used for data augmentation.

The initial notion for the pipeline was to first generate the concatenated utterances and then use the Kaldi toolkit for augmentation and feature extraction. However, this turned out to be unscalable due to the incompatibility between the Kaldi filterbank implementation and the features required by the speaker verification system used to extract frame-level d-vectors. It was necessary to only extract the acoustic features once and use them both as PVAD input features as well as input to the d-vector extractor, otherwise, too much computation time would be consumed. Therefore, this approach was abandoned and Kaldi was used for quick and efficient augmentation only.

That being said, what turned out to be quite useful, was Kaldi's system for describing and storing data and features.

Kaldi utilizes pairs of `.scp` and `.ark` files to efficiently store and describe data. The `.scp` files usually hold information about how to obtain a particular resource, which is identified by a key (for example the utterance id). Each key is then associated with a *recipe*, which describes how the resource can be obtained. This could for example be the path to the source file, a shell command describing the augmentation process of an utterance, or an address referring to a specific position in an `.ark` file.

The `.ark` files are essentially archive files designed for efficient data storing, typically used to store extracted features. Initially, NumPy's serialization interface was used for feature storing, however, using the `.scp`/`.ark` framework proved to be a much more sensible approach, both in terms of accessing the resources and especially in terms of disc space savings.

The `kaldiio`[1] Python library was used for interfacing the Kaldi file formats, both in terms of reading and writing resources from and to the `.scp`/`.ark` files.

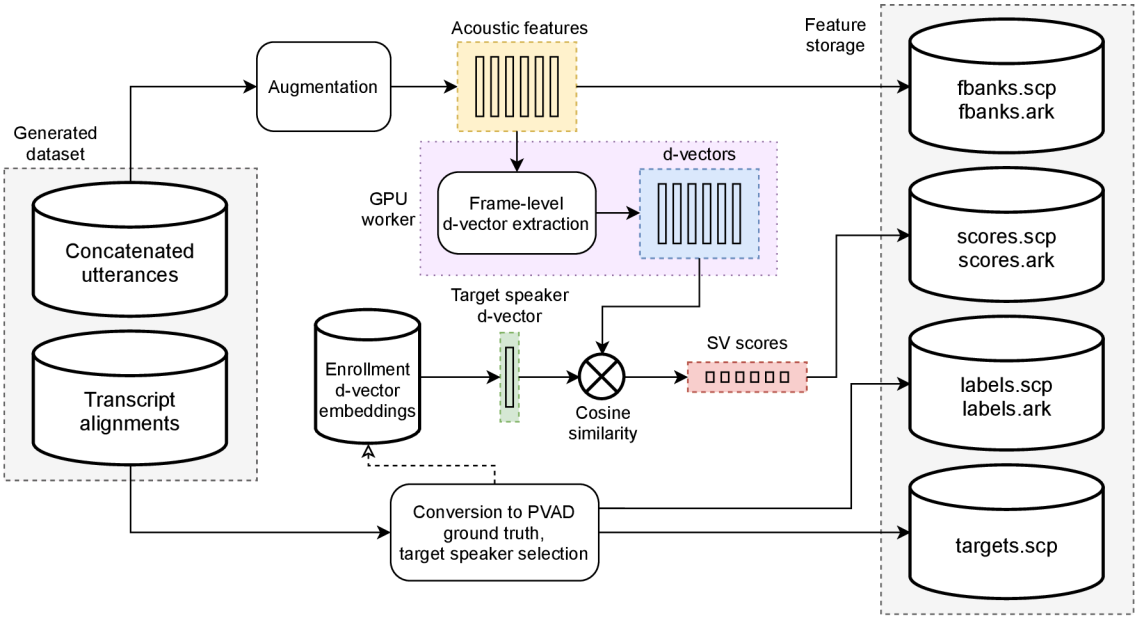

Figure 6.1: Illustration of the implemented feature extraction pipeline. The gray areas on the left and on the right denote the source dataset, and the generated training features and labels, respectively. The purple area denotes the GPU worker, responsible for frame-level d-vector extraction.

The final iteration of the process of preparing the training data and extracting the features can be summarized into the following stages:

1. **Utterance concatenation.** Generate the concatenated utterances, combine their respective transcripts. Describe the generated dataset using Kaldi-specific description files: `wav.scp`, `utt2spk`, `spk2utt`. Extract enrollment d-vector[2] embeddings for each speaker in the dataset.

2. **Augmentation.** Perform data augmentation via Kaldi. Combine the augmented `wav.scp` files into one that describes the whole dataset.

3. **Feature extraction.** For each utterance in the augmented `wav.scp`, load the waveform, extract the acoustic features, choose the target speaker and generate ground truth labels for the whole utterance. Then, perform frame scoring using the target speaker's enrollment embedding. Save the extracted features and ground truth labels into separate `.scp` and `.ark` files.

The feature extraction stage was particularly heavy on both resources and time required for processing the whole dataset. To at least somewhat mitigate this, multiprocessing was used. An arbitrary number of CPU worker processes load the augmented waveform and extract acoustic features. Then these features are passed to a secondary GPU worker

[1] https://github.com/nttcslab-sp/kaldiio

[2] For later experiments i-vectors and x-vectors were also extracted, however, they are not necessary for the feature extraction stage.

process, which is responsible for extracting the d-vector embedding for each frame in the utterance. The d-vectors are then returned to the original CPU worker so that SV scores can be computed.

6.1.1 Speaker embedding extractors

Aside from feature extraction, multiple types of speaker embedding vectors were also used for training. Following are the systems, which were used for extracting these speaker representations.

d-vectors The d-vector extractor implementation used in this work is called *Resemblyzer*[3]. It is a freely available community implementation of the text-independent speaker verification method proposed in [48]. For the purpose of this work, the actual model class had to be modified to support frame-level embedding extraction in addition to the default method, which extracts one d-vector for a sliding window of 160 frames. The extracted d-vectors have a dimensionality of 256.

x-vectors The x-vector implementation used in the conducted experiments is available via the SpeechBrain [39] toolkit. This system was trained on the VoxCeleb[4] dataset and is based on the original x-vector approach proposed in [43]. The extracted x-vectors have a dimensionality of 512.

i-vectors Lastly, the i-vector system used in the experiments was kindly provided by the Speech@FIT research group via my supervisor, Ing. Ján Švec. The extracted i-vectors have a dimensionality of 400.

When extracting the enrollment embedding vectors, three utterances were randomly selected for each speaker and concatenated to provide the systems with enough information about the speaker's voice characteristics.

6.2 Model implementation and training

Similar to the feature extraction pipeline, the training pipeline was also implemented using the Python 3 language.

Specifically, the popular PyTorch [34] deep learning toolkit was used for implementing the models described in Section 4.2.2, and also for training. The crucial part being the ability to use GPU acceleration to speed up the training process.

For training, each model has a dedicated dataset class, which is used for loading features and labels from `.scp` and `.ark` files and building the final feature vector from the acoustic features, scores, and target speaker embedding vector, depending on the PVAD architecture.

The loading itself is managed by a data loader class, which is additionally responsible for batching the loaded data. The data loader class also utilizes multiprocessing to avoid CPU/GPU data transfer bottlenecks. There are two instances of this class for each training session: one used for the training data, and one used for the validation data.

[3] https://github.com/resemble-ai/Resemblyzer

[4] https://www.robots.ox.ac.uk/~vgg/data/voxceleb/

6.2.1 Dealing with utterances of variable lengths

One of the challenges of training RNN models in PyTorch (and generally) for speech processing tasks is that the training data often consists of utterances of variable lengths. When used for training, the extracted features then form mini-batches of variable sizes. This prevents us from converting the mini-batches into PyTorch tensors and stacking them together to form the training batch tensor, as the tensors would have to be of the same dimensions.

There are two main ways to generally address this problem. One is to simply split the training utterances into partial utterances, which would all have the same length. The resulting training batch is then created from these partial utterances. The partial utterances may or may not vary in lengths across batches. An example of this technique can be found in [48].

However, there are two problems with this approach in the case of this work. The dataset that was generated (see Section 5.2) for personal VAD training varies quite heavily in terms of utterance lengths and it is undesirable to split the longer utterances into multiple shorter ones. This is because personal VAD is supposed to operate as a streaming model with frame-level inference. Thus it is crucial not to constrain its training to limited context windows. The longest utterances in the dataset also contain three different speakers. In order to reliably detect the speech frames of the target speaker, the model has to learn to adapt to the speaker context changes present in these longer utterances.

The second method of addressing the problem of variable-length utterances is utilizing padding [46], for which PyTorch has dedicated functions. All feature vectors in the batch are padded to the length of the longest sequence in the batch. The lengths of the original sequences before applying padding are stored and will be used later when calculating the loss. The padded batch is then passed to the model to perform the forward pass.

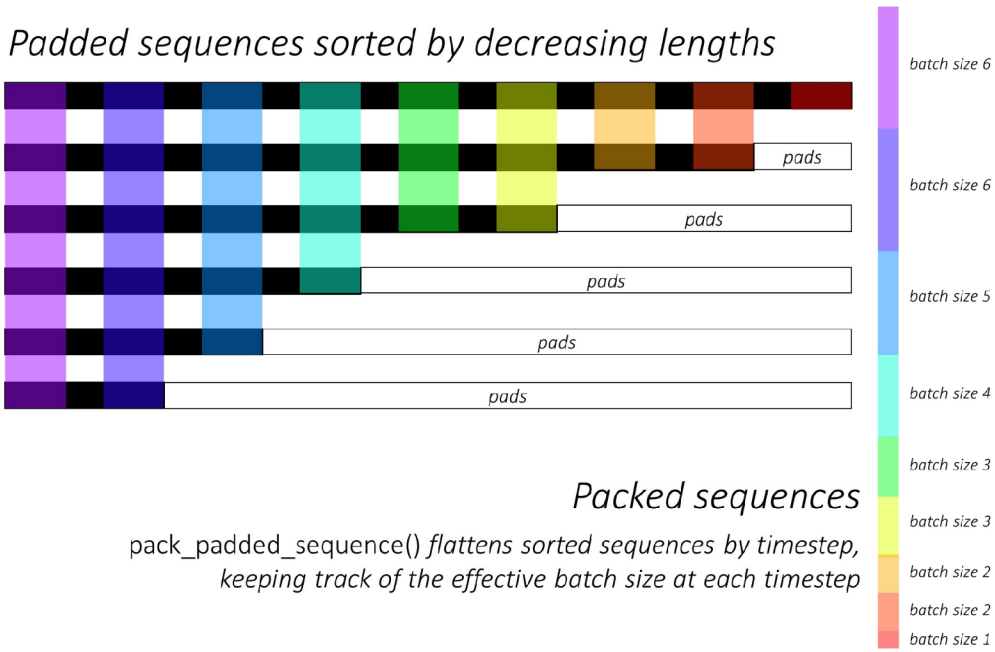

Figure 6.2: Visualization of a batch of six padded sequences and its packed sequence counterpart. Diagram taken from [46].

Before processing the batch via the RNN, the padded sequences are flattened via the `pack_padded_sequence` function, the resulting *packed sequence* format being suitable for RNN processing (see Figure 6.2 for a visualized example of a packed sequence structure). After the RNN pass, the sequences are padded once again, using the `pad_packed_sequence` function. Regular hidden layers can process the padded sequences without any modification.

When calculating the loss, the original sequence lengths are used to mask out the padding, so that it does not affect gradient calculations.

Chapter 7

Experiments

In this chapter, the systems presented in Chapter 4 are evaluated. Section 7.1 introduces the main metrics used for performance evaluation,

Section 7.3 then evaluates the baseline versions of the four personal VAD systems as presented in Section 4.2.2.

Section 7.4 then explores the possibilities of using i-vectors and x-vectors as the target speaker embedding vectors with the ET architecture.

Lastly, in Section 7.5 I also investigate the performance of the baseline streaming frame-level scoring method introduced in Section 4.2.2, address some concerns about its performance, and propose two alterations to this frame scoring method.

For all experiments, evaluation is performed twice. First, each system is evaluated using the clean utterances from the validation set only to establish the system's baseline performance level for clean speech. Then the whole augmented scope of the validation set is used to determine the system performance for clean and noisy speech combined, simulating the ever-changing real-life acoustic conditions.

7.1 Evaluation metrics

The information presented in this section was derived from [40].

The main metrics used for model evaluation were the *Average Precision* (AP) and *Mean Average Precision* (mAP). In order to properly understand these two metrics, it is first necessary to define *precision* an *recall*.

Intuitively, precision is a measure classifier's ability not to label negative samples as positive for a particular decision threshold (often referred to as the *operating point* of the classifier). It is defined using the following formula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{7.1}$$

where TP denotes the number of true positives and FP denotes the number of false positives.

Recall on the other hand represents the ability of the classifier to find all the positive samples in the set, again for a particular decision threshold:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{7.2}$$

where TP denotes the number of true positives and FN denotes the number of false negatives.

Average precision then summarizes the relationship between precision and recall across different classifier operating points:

$$\text{AP} = \sum_n (R_n - R_{n-1}) \cdot P_n, \tag{7.3}$$

where P_n and R_n denote precision and recall at an operating point n, respectively, with the difference in the recall at thresholds n and $n-1$ being used as a weight to the precision value at n. This effectively corresponds to computing the area under the precision-recall curve constructed for the different decision thresholds. It should however be noted, that this is the `sklearn`'s[1] uninterpolated AP implementation, as AP computed from the linearly interpolated precision-recall curve might at times be too optimistic.

In the conducted experiments, AP was always computed for every class to quantify how precise the model is with respect to that particular class.

To quantify the overall model performance, mean average precision was then computed across all classes, adopting the *micro-mean* approach, which calculates the AP metric across all predicted samples.

In addition to mAP, raw classification *accuracy* (as in the percentage of correctly classified samples in the validation set) for each model is also reported to provide an easily interpretable, general indicator of the model performance:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{7.4}$$

Lastly, *confusion matrices* were used to better understand the model performance in terms of distinguishing between different class pairs. On one axis, the confusion matrix is indexed by the true class of the sample, on the other axis by the actually predicted class. Therefore, given two indexes, i and j, the confusion matrix entry at these indices is the count of how many times class i was predicted given that the true class was j[2].

7.2 Training configuration and conditions

All systems were trained and evaluated using the same training and validation datasets. The final augmented training set consisted of approximately 562 thousand utterances and the augmented validation set of approximately 22 thousand utterances. More on the process of creating the training and validation sets can be found in Sections 5.2, and 6.1.

The final model consisted of a 2-layer LSTM network of 64 cells each, followed by one hidden layer of 64 neurons. The activation function assigned to this hidden layer depends on the performed experiment, more about this is explained in Section 7.2.1. In this configuration, the model only has 130 thousand parameters.

During training, the Adam optimizer [23] was used with a variable learning rate set to 1×10^{-3} for the first epoch, progressing down to 1×10^{-5} using learning rate scheduling. The models, which utilized the target embedding vectors as one of the features, were trained for 10 epochs maximum to avoid overfitting. The models without the target embedding among the input features were trained for 8 epochs maximum.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html

[2] This, of course, depends on the orientation of the confusion matrix, as different implementations might have the axes swapped.

The computational resources required for feature extraction and training were kindly provided by MetaCentrum[3].

7.2.1 Last hidden layer activation functions

An important aspect of the conducted experiments was to determine which activation function works the best for the one hidden layer after the LSTM. The compared activation functions were the hyperbolic tangent (*tanh*), the leaky rectified linear unit (*leaky relu*). On top of that, the activation function was removed altogether, leaving us with just a *linear* activation.

As the fully connected layer is only really supposed to further transform the LSTM output, it is expected that the potential differences in the results should not be too dramatic. The 2-layer LSTM should be powerful enough to suffice for the personal VAD task by itself and the fully-connected layer should therefore play the role of a "stabilizer", refining the LSTM output.

However, it was quickly determined that using the *leaky relu* is not ideal for any of the model variants as the obtained results were albeit marginally but consistently worse than any of the results obtained with the *tanh* and *linear* activations.

It seems that the ability of these two latter activation functions to better preserve negative values has a positive effect on the accuracy of the system, as the *leaky relu* discards most of the negative value information. Therefore, in the further comparisons, only the results for the *tanh* and *linear* activations are reported[4], as the *leaky relu* model variants provided no useful results.

7.3 Comparing the architectures

In this set of experiments, all four personal VAD architectures as presented in Section 4.2.2 were trained and compared. This is to establish a performance baseline for other experiments and also to compare the implemented system performances against the results published in [13]. The experiment results are shown in Table 7.2.

For the SC system, only the *linear* baseline VAD system variant was used, since it showed slightly better performance as shown in Table 7.1. For all other systems, results for both the *linear* and *tanh* system variants are reported.

System	Clean				Augmented			
	`ns`	`s`	mAP	acc [%]	`ns`	`s`	mAP	acc [%]
VAD (*linear*)	0.949	0.998	0.995	96.48	0.915	0.996	0.991	94.93
VAD (*tanh*)	0.947	0.998	0.995	96.34	0.913	0.996	0.990	94.85

Table 7.1: Baseline pure VAD system evaluation results. The `ns` and `s` labels denote non-speech and speech AP scores, respectively.

As expected, the SC baseline system performed rather poorly in this comparison, reaching AP scores of only 0.846 and 0.864 for clean `tss` and `ntss`, respectively.

The ST architectures ended up providing much better results than the baseline SC, with the *tanh* ST variant reaching AP score of 0.920/0.864 for clean/augmented `tss`. However,

[3] https://metavo.metacentrum.cz

[4] To differentiate between the used activations in the text, the particular model is always referred to as a "variant", for example, the *tanh* ET system variant.

in comparison to the ST architecture from [13], this result is still quite poor. The reason for this is most likely stems from the d-vector extractor system used in this work and its poor adaptation to streaming frame-level d-vector inference. Therefore, it seems that the frame-level speaker verification scores used by the SC, ST, and SET architectures are not discriminative enough for the systems to perform optimally.

The ET architectures ended up reaching quality results, even surpassing the original work's results for clean `tss` and `ntss` AP scores. However, that is for the systems trained using cross-entropy. The best ET model presented in the original work was trained using the WPL and scored an AP score for clean `tss` of 0.955. This is further discussed in Section 7.3.1.

Even though the ET architecture results seem decent, it looks like this architecture is still quite sensitive to noise. As shown in Fig. 7.1, it is apparent that the presence of noise affects the system's ability to distinguish `tss` frames from `ntss`. While this architecture offers decent performance for clean speech, for it to be effective and reliable in acoustically challenging environments, the amount of `tss` frames misclassified as `ntss` would have to be reduced. Of course, this is not an easy problem to solve, mostly due to the self-imposed resource demand limitations for this architecture. The SET architecture represents a partial solution to this problem, especially when paired with a modified version of the baseline scoring method. This is further discussed in Section 7.5.

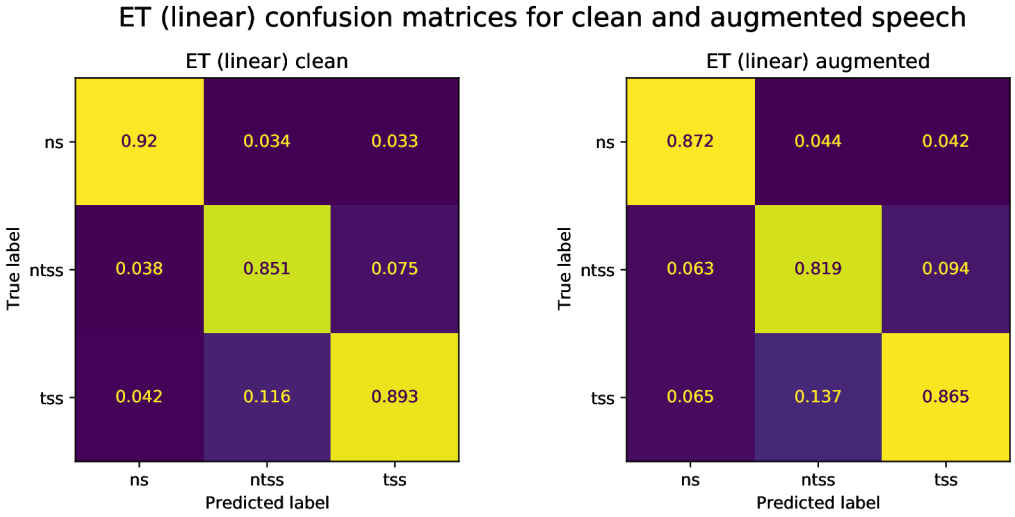

Figure 7.1: Comparison of confusion matrices for the ET (linear) system, obtained for the clean (left) and the augmented (right) validation sets.

The SET systems were the best performing models in this comparison, both for clean and augmented speech. However, due to the apparent poor quality of the used speaker verification scores, the SET systems were unable to surpass the original paper's results. In Section 7.5, this problem is addressed by introducing two alterations to the baseline scoring method, resulting in ST and SET systems capable of outperforming the original results.

Lastly, to address the rather naive approach of the SC system to the personal VAD task, a modified version of the SC architecture was also experimented with. The modification involved treating the SC architecture as a twofold classification problem. First, the VAD would determine, whether the current frame is a speech frame. Then, the speaker verifica-

System	Clean					Augmented				
	ns	ntss	tss	mAP	acc [%]	ns	ntss	tss	mAP	acc [%]
SC (*linear*)	0.948	0.846	0.864	0.825	73.44	0.915	0.775	0.811	0.796	72.09
ST (*linear*)	0.932	0.914	0.918	0.916	84.19	0.893	0.860	0.864	0.863	78.20
ST (*tanh*)	0.933	0.919	0.920	0.921	84.29	0.893	0.865	0.867	0.868	78.25
ET (*linear*)	0.936	0.951	0.945	0.948	88.02	0.897	0.931	0.924	0.924	84.73
ET (*tanh*)	0.936	0.946	0.938	0.942	87.10	0.897	0.925	0.916	0.918	83.84
SET (*linear*)	0.933	0.962	**0.962**	0.958	**89.14**	0.894	0.938	**0.937**	0.930	**85.31**
SET (*tanh*)	0.929	0.961	0.960	0.957	89.10	0.889	0.937	0.934	0.929	85.26
SC (orig. paper)	0.970	0.872	0.886	0.900	-	-	-	-	-	-
ST (orig. paper)	0.968	0.956	0.956	0.957	-	-	-	-	-	-
ET (orig. paper)	0.962	0.946	0.932	0.946	-	-	-	-	-	-
SET (orig. paper)	0.969	0.972	0.970	0.969	-	-	-	-	-	-

Table 7.2: Average precision score comparison of different personal VAD architectures for clean and augmented speech. Class labels: `ns` for non-speech, `ntss` for non-target speaker speech and `tss` for target speaker speech. The bottom part of the table shows the architecture results for clean speech obtained in [13]. The augmented results are not shown here, as different augmentation strategies were used.

tion score would be thresholded using its `tss`/`ntss` classification EER threshold value (see Section 7.5 for more information).

This way, the probabilistic nature of the classifier is lost, however, the raw classification accuracy was improved to 77.57% (clean) and 74.46% (augmented). The improvement in classification accuracy is rather noticeable, however, the ST architectures still outperform this modified SC system by a significant margin. This might indicate the LSTM's ability to not only learn the optimal decision threshold from the score values but also its ability to dynamically adjust this threshold based on the current temporal context in the processed score value stream. For this reason, the SC approach was not pursued anymore in the following experiments.

7.3.1 Weighted pairwise loss

To evaluate the effects of the weighted pairwise loss (WPL) (for further details see Section 4.2.3), the ET architecture was retrained several times, always using different `<ns, ntss>` values.

In [13], it is suggested that $w_{\texttt{<ns, ntss>}}$ values between 0.1 and 0.5 should help increase the AP score for `tss`. Weight values above and below these should generally lead to performance degradation as either not enough or needlessly much attention is given to `<ns, ntss>` confusion errors.

Therefore, both the *tanh* and *linear* ET variants were evaluated using the following set of $w_{\texttt{<ns, ntss>}}$ values: $\{0.1, 0.3, 0.5, 0.7\}$. The best results are compared against the baseline systems trained with cross-entropy (see Table 7.3).

The *tanh* ET variant did, in fact, benefit from using the WPL. However, it turns out that even though the value of $w_{\texttt{<ns, ntss>}} = 0.1$ was supposed to give the best performance, that was actually not the case for this experiment. For the *tanh* variant, the value of 0.1 lead to `tss` AP degradation, performing worse than the cross-entropy baseline.

For values of 0.3 and 0.5 however, the performance of the system improved over the baseline. For the value of 0.5, it reached the best AP score of 0.946/0.924 for clean/augmented `tss`, outperforming even the best *linear* ET variant from the previous experiment

in this aspect. Unfortunately, not even this system was able to outperform the best WPL-trained ET system from [13]. The effects of different `<ns, ntss>` weight values on `tss` AP score for the *tanh* ET system are shown in Fig. 7.2.

System	Loss	w	Clean					Augmented				
			ns	ntss	tss	mAP	acc [%]	ns	ntss	tss	mAP	acc [%]
ET (*linear*)	CE	-	0.936	0.951	0.945	0.948	**88.02**	0.897	**0.931**	**0.924**	0.924	**84.73**
ET (*tanh*)		-	0.936	0.946	0.938	0.942	87.10	0.897	0.925	0.916	0.918	83.84
ET (*linear*)	WPL	0.5	0.932	0.950	0.943	0.946	87.74	0.893	0.928	0.921	0.921	84.38
ET (*tanh*)		0.5	0.931	**0.952**	**0.946**	0.947	87.89	0.893	**0.931**	**0.924**	0.923	84.59
ET (orig. paper)	WPL	0.1	0.965	0.961	0.955	0.959	-	-	-	-	-	-

Table 7.3: Comparison of the two best ET systems trained using the cross-entropy loss (CE), and their best weighted pairwise loss (WPL) counterparts. The w column denotes the `<ns, ntss>` weight value used for WPL training. Both WPL models gave best performance when trained using $w_{\texttt{<ns, ntss>}} = 0.5$. The last row shows the best WPL-trained ET system from [13].

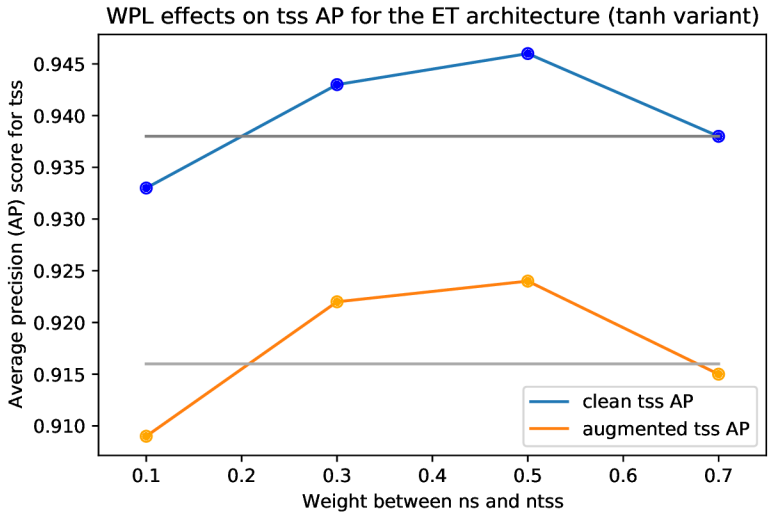

Figure 7.2: Effects of the weighted pairwise loss on `tss` AP score for the *tanh* ET system. The gray lines denote the `tss` AP scores of the baseline ET *tanh* system trained using cross-entropy for easy comparison with the WPL results.

For the *linear* ET variant, unfortunately, no performance improvements were observed, though the weight value effect on `tss` was similar to that of the *tanh* variant. The weight value of 0.5 still gave the best results (as shown in Table 7.3) but was unable to outperform the baseline *linear* ET system trained using cross-entropy. This experiment was conducted multiple times to rule out the possibility of the WPL model not converging properly, however, with no success.

The reason for this apparent ineffectiveness of the WPL might stem from multiple aspects of the training process. It is possible that the ground truth labels generated from the used LibriSpeech alignments did not allow the WPL to fully exploit the importance of `<ns, ntss>` errors, as generally, the trained models have a worse `ns` AP scores than in [13]. It is also possible that the best *linear* ET system variant from Section 7.3 was simply lucky enough to converge better than any other system during training and because of that,

the WPL was unable to surpass this result. It is, therefore, possible that further tuning and optimization of the model and training hyperparameters would in the end bring the desired improvements even for the *linear* ET variant, potentially even surpassing the results published in the original paper.

7.4 Comparison of different target speaker embedding types

Since in [13], only d-vectors are used as target speaker embeddings, I experiment with using two other speaker embedding types as targets in addition to the d-vector – the x-vector and the i-vector. I-vectors were successfully used in [30] for a task very similar to that explored in this work. On the other hand, the same paper was unable to successfully use x-vectors for the same task, suggesting that the system may have had an overfitting issue due to the sparse nature of the x-vector embedding space. The experiments were once again done using the ET architecture, both for the *linear* and *tanh* variants.

When training the first x-vector system, the same issue as in [30] was encountered – the system was unable to learn anything at all, reaching only 52.4% in raw accuracy. That was, however, not the case for the i-vector system, which was able to learn and perform reasonably well (see Table 7.4 for results), though definitely not on par with the baseline d-vector system.

The reason for the x-vector system performing so poorly was most likely that the value distributions for the individual x-vector dimensions generally had too large of a variance and it was impossible for the network to make use of the values.

On the other hand, i-vectors are by definition subjects of a standard normal distribution. Therefore the i-vector values are nicely centered around zero with a variance of one.

Additionally, the d-vectors computed using the method from [48] are, as it was previously established, L2-normalized after extraction, restricting them to the surface of a unit hypersphere. That is, most likely, why the system was able to generalize for i-vectors and d-vectors but not for x-vectors.

It is a common practice to apply length normalization to speaker vectors before backend modeling, restricting them to a smaller area in the embedding space. This can help limit the intra-speaker variability of the embedding vectors, while still retaining the inter-speaker variability [43, 5, 20]. Therefore, to address the sparsity problem of the x-vector and i-vector spaces, L2 normalization was applied to both i-vectors and x-vectors.

In the case that the L2 normalization was to cause (though very unlikely) a significant degradation to the inter-speaker variability of the speaker vector space, even a simple visual analysis such as t-SNE [28] could potentially show the loss of some or all discriminative properties of the embedding vectors.

To test this, 40 speakers were chosen from the LibriSpeech dataset. For each speaker, 40 i-vectors and 40 x-vectors were computed, each speaker vector from a different enrollment utterance. This way, 1600 i-vectors, and 1600 x-vectors were obtained in total. Both of these speaker vector sets were then L2-normalized and compared against the original ones using t-SNE. The results are shown in Fig. 7.3.

The results indicate the expected: there is indeed no visible significant degradation in the quality of the speaker vectors after applying L2 normalization. On the contrary, it seems that the L2 normalization could have potentially even benefited the i-vectors. This could be because the L2 normalization also has the effect of essentially spreading out vectors that lie close to the origin, pushing them away to a unit distance from it.

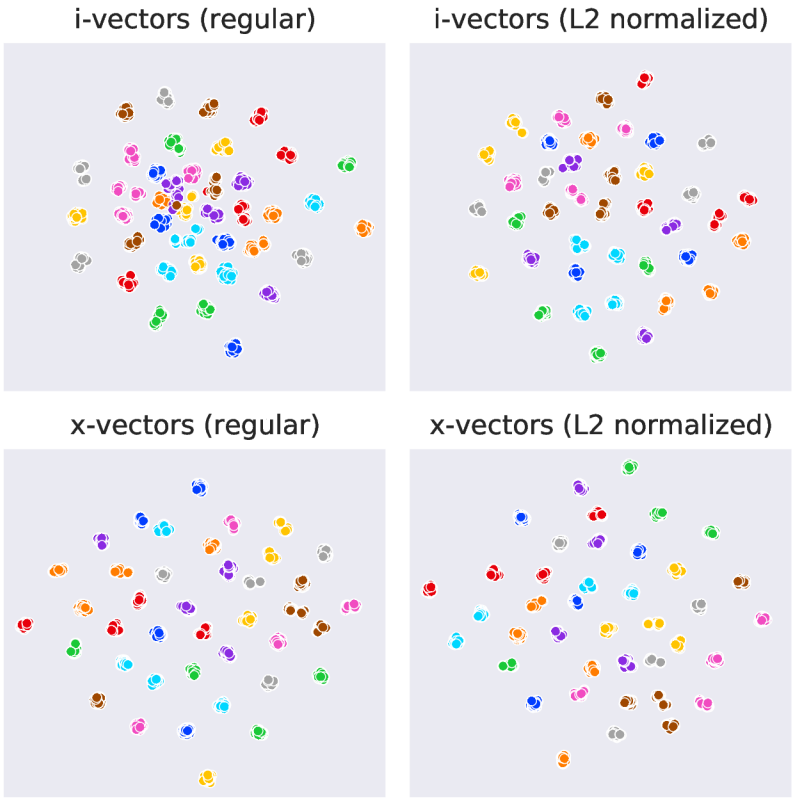

Figure 7.3: t-SNE visualization of both raw and L2 normalized i-vectors (top) and x-vectors (bottom). Each plot contains 1600 embedding vectors (40 from 40 different speakers). The speakers are the same in all four plots and each is represented by a single color.

Activation	Embedding type	Clean					Augmented				
		`ns`	`ntss`	`tss`	mAP	acc [%]	`ns`	`ntss`	`tss`	mAP	acc [%]
linear	d-vector	0.936	0.951	0.945	0.948	**88.02**	0.897	0.931	**0.924**	0.924	**84.73**
tanh[5]		0.931	0.952	**0.946**	0.947	87.89	0.893	0.931	**0.924**	0.923	84.59
linear	i-vector	0.930	0.860	0.854	0.830	78.16	0.892	0.834	0.831	0.810	75.97
tanh		0.931	0.863	0.854	0.833	78.43	0.892	0.839	0.833	0.816	76.37
linear	i-vector (L2)	0.939	0.927	0.918	0.918	85.00	0.901	0.904	0.895	0.895	82.18
tanh		0.940	0.926	**0.923**	0.921	**85.39**	0.904	0.905	**0.900**	0.898	**82.62**
linear	x-vector (L2)	0.938	0.939	0.928	0.935	86.30	0.901	0.920	0.907	0.912	83.21
tanh		0.940	0.945	**0.936**	0.942	**87.12**	0.904	0.925	**0.914**	0.919	**83.93**

Table 7.4: ET architecture performance comparison for different speaker embedding types. The best `tss` AP and raw accuracy scores for each embedding type are highlighted in bold.

After training the system with L2-normalized i-vectors and x-vectors, it is apparent that applying L2 normalization to the speaker vectors – restricting them to the surface of a unit hypersphere – does indeed help the model to generalize (see Table 7.4 for results). So much so that the *tanh* model variant trained with x-vectors performs almost on par with the models trained with d-vectors.

The i-vector system accuracy was greatly improved in comparison to the pre-L2 i-vectors, though not reaching the same performance level as x-vectors or d-vectors. This

[5]This is the *tanh* ET system that was trained using the WPL and scored the best result among the ET *tanh* model variants.

is possibly due to some residual channel information encoded in the i-vectors, which the model was unable to account for.

This experiment shows that for the personal VAD task it is possible to use also other speaker embedding types in addition to the d-vector. The only restriction being that for the network to generalize properly, it is important to length normalize the speaker vectors, restricting them to constrained space. This can be achieved for example via L2 normalization.

7.5 Altering the frame scoring method

So far, the speaker verification frame scoring method used in the experiments in Section 7.3 has been the one proposed by [13], described more in detail in Section 4.2.2. However, given the rather non-optimal performance of both the baseline SC and the ST architectures, in this section, I further analyze this scoring method and propose two modified alternatives to the original one.

In [13], some concern was expressed regarding the performance of the speaker verification system used to extract frame-level d-vectors. This is because the system's architecture is based on an LSTM network, and though LSTM networks can generally process sequences of variable lengths, the system proposed in [48] was trained on context windows of 140 – 180 frames (as was the Resemblyzer d-vector system implementation used in this work). Therefore, intuitively, optimal performance is guaranteed for these limited context windows only. In other words, the system may suffer from performance degradation when having to deal with long temporal contexts, as would be the case in a streaming frame-level inference scenario, which is used in the context of the baseline scoring method.

Additionally, due to an implementation decision specific for the Resemblyzer speaker encoder, the resulting d-vectors always have only positive values in all dimensions. Though this may not at first glance affect the discriminative properties of the d-vectors, it certainly limits the cosine similarity score value domain, concretely to the interval of $< 0, 1 >$.

For these reasons, I propose two alterations to the original scoring method, taking inspiration from conventional speaker diarization approaches [49]. Both new methods refrain from the streaming frame-level embedding extraction approach and instead utilize the d-vector extractor in the manner it was trained to – window-level d-vector inference. a diagram depicting the window-level d-vector inference can be found in Section 3.3.

Both scoring method alterations process the input utterance in a sliding window manner, extracting one d-vector for a window of 160 frames, with a 40 frame step in between the individual windows. These d-vectors are obtained at each time step t:

$$t = 160 + k \cdot 40; \;\; k = 0, 1, 2, \ldots, \tag{7.5}$$

each representing the past 40 frames, with the first d-vector representing the first 160 frames of the utterance. Then, the d-vectors at these time steps are compared against the target speaker embedding, once again using cosine similarity. This way, speaker verification scores are obtained for each time step t.

7.5.1 Comparing frame-level and window-level d-vector discriminativity

Before proceeding, it has to be determined, whether these window-level d-vectors have better discriminative properties than the ones extracted at frame-level using the original approach. What is especially crucial, is how discriminative the d-vectors are for utterances

containing speaker turns, as it is uncertain if the streaming frame-level d-vector extraction approach can properly "react" to these speaker context changes. For this, the following simple test was conducted.

For each utterance in the whole training set, the original frame-level scores were sub-sampled at the same time steps described in Equation 7.5. That is to obtain frame-level score values at time steps corresponding to the window-level scores.

For both speech classes (considering `tss` and `ntss` only, assuming that the VAD correctly discards `ns` frames) and d-vector extraction methods, the resulting cosine similarity score values were plotted in a histogram, which can is shown in Figure 7.4.

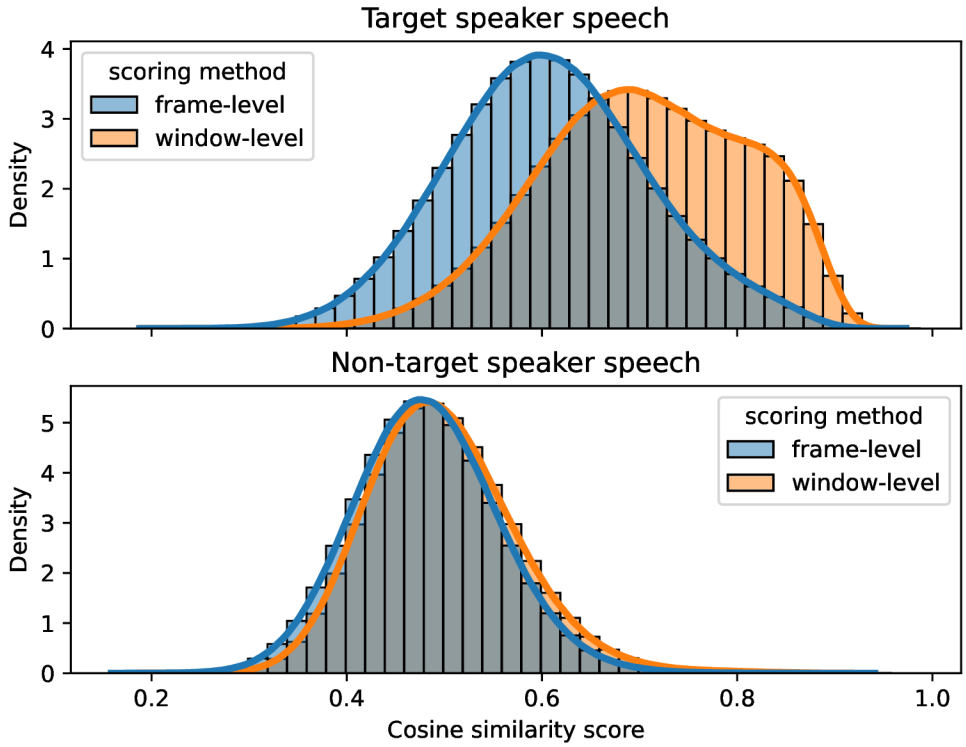

Figure 7.4: Comparison of the speaker verification score value distributions for the two d-vector extraction methods. Comparison for `tss` is in the upper part of the figure, `ntss` comparison is situated in the lower part.

The histogram clearly shows that while both d-vector extraction methods perform similarly well for `ntss`, the window-level d-vectors generally produce a higher cosine similarity score for `tss`. This intuitively corresponds to higher window-level d-vector discriminativity, as the system is much more confident in the resulting embedding representation, clearly benefiting from the limited context windows. The reason, why the frame-level approach falls behind is apparently due to the long temporal contexts the system has to process while at the same time dealing with speaker turns.

Additionally, using the same set of sub-sampled scores which was used for plotting the histograms, Receiver Operating Characteristic (ROC) and Equal Error Rate (EER) were computed for both methods, concretely for the task of classifying the corresponding speech frames as either `tss` and `ntss` based solely on the score values. For the baseline frame-level

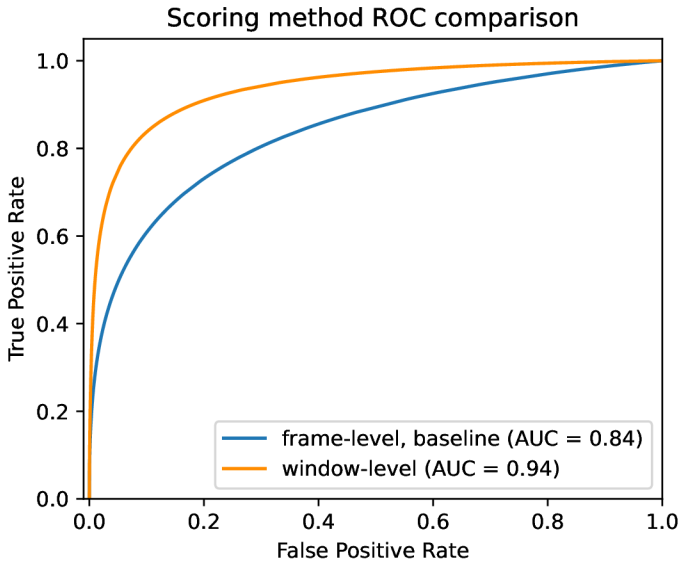

Figure 7.5: Receiver Operating Characteristic (ROC) comparison for the two embedding extraction methods, evaluated for `tss`/`ntss` classification based on the obtained speaker verification scores.

method, the EER was 0.238, whereas the window-level method scored an EER score of 0.132, clearly outperforming the baseline. The ROC comparison is shown in Fig. 7.5.

7.5.2 Proposed scoring methods

Now that it was established that the window-level d-vectors provide superior discriminativity and classification accuracy, it is necessary to distribute the speaker verification scores across the 40 frames they are supposed to represent (the 40 frames representing the step between the adjacent score values). This is where the two scoring alterations differ from each other.

The Partially Constant (PC) method The PC method simply assigns the same score value for the whole 40 frame segment. In other words, the speaker verification score value is now constant for each partial segment of 40 frames represented by the original d-vector.

The Linearly Interpolated (LI) method Rather than assigning the same score value for the whole 40 frame segment, the LI method linearly interpolates every two adjacent score values, resulting in a linear score change within the 40 frame segment. This method aims to simulate the more gradual score value change of the baseline frame-level scoring method.

A visualized example of all three scoring methods (the baseline frame-level, PC, LI) can be seen in Fig. 7.6, with the cosine similarity scores being plotted against the ground truth. The figure also shows the tendency of the baseline method scores to decline over time, whereas the altered methods roughly maintain the cosine similarity score values around a constant value for each ground truth segment.

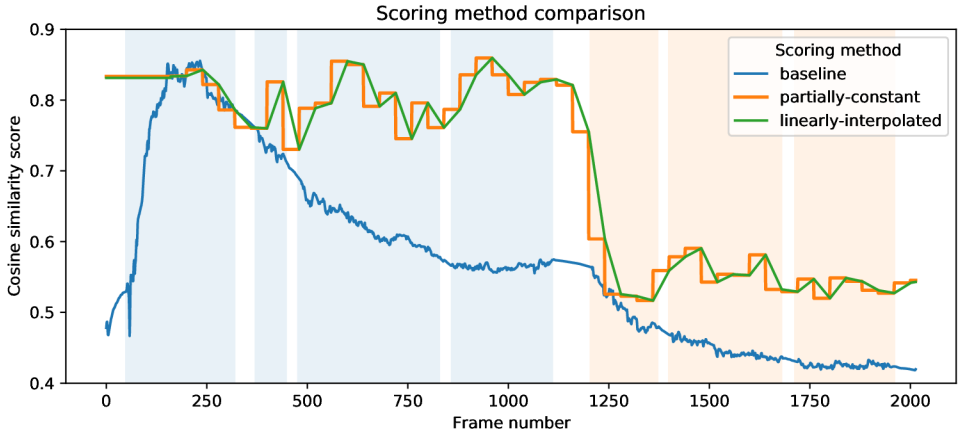

Figure 7.6: Exemplary comparison of the three frame scoring methods. Blue and orange areas denote `tss` and `ntss` segments, respectively.

7.5.3 Scoring alteration performance results

To evaluate the proposed scoring alterations, the ST and SET (and both the *tanh* and the *linear* activation variants) architectures were retrained using the newly obtained score values.

System	Scoring method	Clean					Augmented				
		`ns`	`ntss`	`tss`	mAP	acc [%]	`ns`	`ntss`	`tss`	mAP	acc [%]
ST (*tanh*)	baseline	0.933	0.919	0.920	0.921	84.29	0.893	0.865	0.867	0.868	78.25
SET (*linear*)		0.933	0.962	0.962	0.958	89.14	0.894	0.938	0.937	0.930	85.31
ST (*linear*)	PC	0.933	0.978	0.979	0.973	91.42	0.894	0.952	0.954	0.944	86.85
ST (*tanh*)		0.936	**0.980**	**0.981**	0.975	**91.95**	0.896	**0.955**	**0.957**	0.948	**87.33**
SET (*linear*)		0.933	0.980	0.982	0.976	92.10	0.894	0.961	0.963	0.954	88.35
SET (*tanh*)		0.934	**0.981**	**0.983**	0.976	**92.23**	0.895	**0.962**	**0.964**	0.954	**88.46**
ST (*linear*)	LI	0.927	0.974	0.972	0.967	90.89	0.887	0.946	0.944	0.937	86.25
ST (*tanh*)		0.935	0.974	0.976	0.970	90.86	0.895	0.947	0.950	0.940	86.39
SET (*linear*)		0.935	0.978	0.980	0.974	91.69	0.897	0.959	0.960	0.952	88.08
SET (*tanh*)		0.932	0.977	0.979	0.972	91.47	0.893	0.957	0.958	0.949	87.64
ST (orig. paper)	baseline	0.968	0.956	0.956	0.957	-	-	-	-	-	-
SET (orig. paper)		0.969	0.972	0.970	0.972	-	-	-	-	-	-

Table 7.5: Performance comparison of different frame scoring methods. The newly trained systems are compared against the best performing systems utilizing the baseline frame-level scoring method. The best `tss` AP, `ntss` AP, and accuracy results for both ST and SET are highlighted in bold. The last two rows show the performance of the ST and SET systems from [13], for which only the clean speech results are shown since the used data augmentation strategy is different in this work.

The results, which are shown in Table 7.5, are clear. Both the PC and the LI scoring methods outperform the baseline by a significant margin with the PC method performing the best for both ST and SET. For the PC method, the *tanh* system variants seem to have a slight edge on the *linear* in terms of overall model accuracy and `tss` AP scores in this comparison. For the LI method, the opposite is true.

Overall, the PC scoring improved the raw model accuracy by more than 7.23/8.56% absolute (clean/augmented) for the ST architectures and at least 2.96/3.04% absolute

(clean/augmented) for the SET architectures. This experiment, therefore, resulted in the best performing system created in this work, as the SET *tanh* variant utilizing the PC scoring method achieved 92.23% and 88.46% raw accuracy for clean and augmented evaluation sets, respectively, additionally scoring a `tss` AP score of 0.983/0.964 (clean/augmented). Moreover, both the PC and LI outperform even the best SET system presented in [13] in terms of `tss` and `ntss` AP scores.

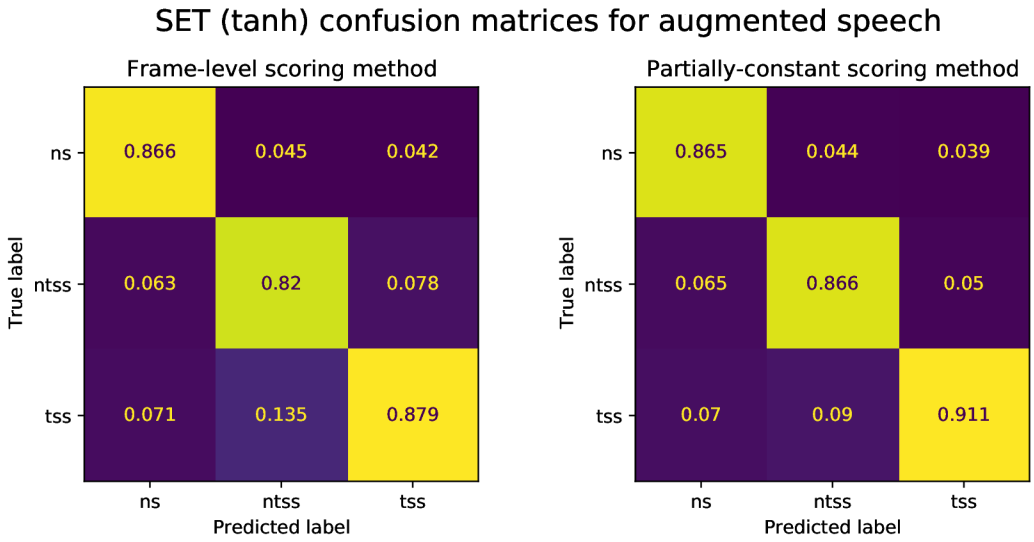

Figure 7.7: Confusion matrix comparison for the SET *tanh* systems trained using the baseline scoring method (left) and the PC scoring method (right). The model performance was evaluated for the augmented validation set.

Generally, the scoring modifications also improve the system's robustness against background noise. An example of this is shown in Figure 7.7, where the `<tss, ntss>` confusion error rates of the SET *tanh* system for augmented speech were visibly reduced by utilizing the PC scoring method.

It should be noted that even though the proposed scoring methods both bring significant classification improvements, they also require more computational power due to the window-level d-vector extraction. With the sliding window step set to 40 frames, most frames will be processed up to four times. The size of the sliding window step can, however, be experimented with and possibly lowered to save resources. Overall, it can be argued that these scoring method alterations might prove useful in scenarios, where the resource limits are not a concern, as they both provide results that are superior to all other presented systems.

7.6 Summary and possible improvements

The target ET architecture (the *linear* variant presented in Section 7.3) ended up providing solid results for clean speech classification. However, for augmented speech, the performance for `tss`/`ntss` classification seems to decline. This is most likely caused by the background noise essentially masking out the target speaker's voice characteristics, resulting in the system classifying those particular frames as `ntss`.

In order for the ET architecture to perform optimally under most conditions, it would probably be necessary to implement some model decision post-processing techniques to both smoothen out the system decision and perhaps to favor the `tss` class if the model becomes unsure. This could be done since the primary goal for the model is not to perform robust speaker verification but rather to filter out most of the irrelevant information. Therefore, occasionally classifying some `ntss` frames as `tss` would not harm the system's purpose in a significant way.

Obviously, if the situation and use-case allow it, the best overall solution would be to utilize one of the SET architectures along with the modified scoring methods. These architectures provided the best results for both clean and augmented speech and if one was to apply post-processing to the model decision, the systems provide good, usable results.

On the other hand, it can be ruled out that the baseline SC and the ST architectures are in the end not too relevant, especially in comparison to the SET architecture. The SET architecture offers a slight edge on the performance of any ST system simply due to the utilization of the target speaker embedding vector. In most situations, there would be little reason not to use this embedding representation in one can already perform frame scoring.

When contemplating how to further improve the obtained results, a few general points immediately come to mind.

First, while the generated concatenated utterance dataset might serve well as a baseline, a dataset consisting of real-life conversations would be incredibly beneficial for personal VAD development. That is mainly to fully test the capabilities and limits of the implemented systems, while also better preparing them for real-life scenarios. Ideally, such a real-life dataset would also contain overlapping speech, as it is unclear, how well would the current personal VAD implementations handle such situations.

Additionally, apart from real-life conversation data, training the systems using actual voice command recordings from actual mobile device users (essentially training the systems on in-domain data) may potentially bring some improvements, as this would allow for the models to be trained for the actual end goal use case scenario.

Second, the architecture of the model itself could be experimented with. Apart from experimenting with neural network topologies such as bidirectional LSTM networks, some basic speech enhancement methods could potentially increase the system's robustness against noise and thus reduce the number of `tss` frames classified as `ntss`. Implementing such architectural changes is, however, not completely straightforward with the original frame-level streaming model inference in mind and would therefore have to be considered with regard to a specific use case.

Lastly, in order to obtain the best results, it would be beneficial to further optimize the training process in terms of batch sizes and model parameter regularization, as most models show some improvement potential even at the very end of the training. This could also lead to the WPL providing better results, which would allow the models to further improve their precision regarding `tss` and `ntss` frame detection.

Chapter 8

Conclusions

This work aimed to implement, evaluate and expand upon a speaker-conditioned voice activity detection method proposed by [13], referred to as personal VAD. This method was based on adapting a conventional LSTM-based VAD model to the speech signals of a particular speaker. To do this, the method utilizes speaker embedding representations (namely d-vectors [48]) of the target speaker, either as a part of the input feature vector or to issue speaker verification scores to each individual acoustic feature frame.

To train the models, suitable training and evaluation datasets had to be created first. For this purpose, the standard LibriSpeech dataset was used to simulate speaker turns in recordings by concatenating multiple utterances from different speakers into one. The resulting training dataset utilized the full 1000-hour scale of the LibriSpeech dataset, consisting of approximately 140 thousand concatenated utterances. The generated datasets were also augmented using noise, music, and reverb to account for acoustically challenging conditions.

Four different personal VAD architectures were implemented and trained, each utilizing a different set of input features to identify the target speaker's speech frames. One of the more interesting architectures of the four was the ET system, mainly due to its lightweight properties, as this system utilizes the target speaker embedding only and does not require a speaker verification system at runtime. The best ET architecture reached an accuracy score of 88.02/84.73% for clean and augmented speech, respectively.

In addition to the d-vector, also i-vectors and x-vectors were experimented with as the target speaker embeddings in conjunction with the ET architecture, showing decent results with the x-vectors reaching a performance level similar to that of the d-vector-trained systems.

Architectures, which utilize also the frame-level speaker verification scores as a part of the input feature vector, did not bring great results at first. This was due to how the d-vector system was altered to operate in a streaming manner, producing one d-vector for each individual acoustic frame, as the original inference model of the d-vector extractor was sliding-window based. The obtained score values, therefore, did not have good enough discriminative properties for distinguishing `tss` frames from `ntss`.

To address the poor performance of the baseline frame scoring method, I propose two alterations to this method, which utilize sliding window d-vector inference. These alterations significantly improved the discriminativity of the obtained score values, resulting in the best performing SET system presented in this work. This system reached an accuracy score of 92.23/88.46% for clean and augmented speech, respectively, while also outperform-

ing the best SET system presented in the original work in terms of both `tss` and `ntss` average precision scores.

The next step in the development of personal VAD systems would be to train and evaluate them using real-life conversation data, as it is unclear, how well can the current models handle factors such as overlapping speech. Training on real data could also better prepare the systems for real-life use-case scenarios and increase the robustness of the models.

Bibliography

[1] Bengio, Y., Frasconi, P. and Simard, P. The problem of learning long-term dependencies in recurrent networks. In: *IEEE International Conference on Neural Networks*. February 1993, vol. 3, p. 1183–1188. DOI: 10.1109/ICNN.1993.298725.

[2] Bengio, Y., Simard, P. and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*. 1994, vol. 5, no. 2, p. 157–166. DOI: 10.1109/72.279181.

[3] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

[4] Bäckström, T. *Voice activity detection (VAD)* [online], 17. May 2019. Revised 2020-8-12 [cit. 2021-4-22]. Available at: https://wiki.aalto.fi/pages/viewpage.action?pageId=151500905.

[5] Cai, W., Chen, J. and Li, M. Analysis of Length Normalization in End-to-End Speaker Verification System. In: *Proc. Interspeech 2018*. 2018, p. 3618–3622. DOI: 10.21437/Interspeech.2018-92.

[6] Chang, J.-H., Kim, N. S. and Mitra, S. Voice activity detection based on multiple statistical models. *IEEE Transactions on Signal Processing*. 2006, vol. 54, no. 6, p. 1965–1976. DOI: 10.1109/TSP.2006.874403.

[7] Chang, S. yiin, Li, B., Sainath, T., Simko, G. and Parada, C. Endpoint detection using grid long short-term memory networks for streaming speech recognition. In: *Proc. Interspeech 2017*. 2017.

[8] Chang, S.-Y., Li, B., Simko, G., Sainath, T. N., Tripathi, A. et al. Temporal Modeling Using Dilated Convolution and Gating for Voice-Activity-Detection. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 5549–5553. DOI: 10.1109/ICASSP.2018.8461921.

[9] Chen, Y., Lopez-Moreno, I., Sainath, T. N., Visontai, M., Alvarez, R. et al. Locally-connected and convolutional neural networks for small footprint speaker recognition. In: *Proc. Interspeech 2015*. ISCA, 2015, p. 1136–1140.

[10] Chevalier, G. LARNN: Linear Attention Recurrent Neural Network. *CoRR*. 2018, abs/1808.05578.

[11] Dehak, N., Kenny, P. J., Dehak, R., Dumouchel, P. and Ouellet, P. Front-End Factor Analysis for Speaker Verification. *IEEE Transactions on Audio, Speech, and Language Processing*. 2011, vol. 19, no. 4, p. 788–798. DOI: 10.1109/TASL.2010.2064307.

[12] Delcroix, M., Zmolikova, K., Kinoshita, K., Ogawa, A. and Nakatani, T. Single Channel Target Speaker Extraction and Recognition with Speaker Beam. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 5554–5558. DOI: 10.1109/ICASSP.2018.8462661.

[13] Ding, S., Wang, Q., Chang, S.-Y., Wan, L. and Lopez Moreno, I. Personal VAD: Speaker-Conditioned Voice Activity Detection. In: *Proc. Odyssey 2020 The Speaker and Language Recognition Workshop*. 2020, p. 433–439. DOI: 10.21437/Odyssey.2020-62.

[14] Eyben, F., Weninger, F., Squartini, S. and Schuller, B. Real-life voice activity detection with LSTM Recurrent Neural Networks and an application to Hollywood movies. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2013, p. 483–487. DOI: 10.1109/ICASSP.2013.6637694.

[15] Garcia Romero, D. and Espy Wilson, C. Analysis of i-vector Length Normalization in Speaker Recognition Systems. In: *Proc. Interspeech 2011*. 2011, p. 249–252.

[16] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. MIT Press, 2016. Available at: http://www.deeplearningbook.org.

[17] Hochreiter, S. and Schmidhuber, J. Long Short-term Memory. *Neural computation*. MIT Press. December 1997, vol. 9, no. 8, p. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[18] Hornik, K., Stinchcombe, M. and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*. 1989, vol. 2, no. 5, p. 359–366. DOI: 10.1016/0893-6080(89)90020-8. ISSN 0893-6080.

[19] Ioffe, S. Probabilistic Linear Discriminant Analysis. In: Leonardis, A., Bischof, H. and Pinz, A., ed. *Computer Vision – ECCV 2006*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, p. 531–542. ISBN 978-3-540-33839-0.

[20] Ji, X., Yu, M., Zhang, C., Su, D., Yu, T. et al. Speaker-Aware Target Speaker Enhancement by Jointly Learning with Speaker Embedding Extraction. In: *ICASSP 2020–2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2020, p. 7294–7298. DOI: 10.1109/ICASSP40776.2020.9054311.

[21] Kanda, N., Horiguchi, S., Takashima, R., Fujita, Y., Nagamatsu, K. et al. Auxiliary Interference Speaker Loss for Target-Speaker Speech Recognition. *CoRR*. 2019, abs/1906.10876.

[22] Kenny, P., Boulianne, G., Ouellet, P. and Dumouchel, P. Speaker and Session Variability in GMM-Based Speaker Verification. *IEEE Transactions on Audio, Speech, and Language Processing*. 2007, vol. 15, no. 4, p. 1448–1460. DOI: 10.1109/TASL.2007.894527.

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

[24] Ko, T., Peddinti, V., Povey, D., Seltzer, M. L. and Khudanpur, S. A study on data augmentation of reverberant speech for robust speech recognition. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, p. 5220–5224. DOI: 10.1109/ICASSP.2017.7953152.

[25] Landini, F., Wang, S., Diez, M., Burget, L., Matějka, P. et al. But System for the Second Dihard Speech Diarization Challenge. In: *ICASSP 2020–2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, p. 6529–6533. DOI: 10.1109/ICASSP40776.2020.9054251.

[26] Li, C., Ma, X., Jiang, B., Li, X., Zhang, X. et al. Deep Speaker: an End-to-End Neural Speaker Embedding System. *CoRR*. 2017, abs/1705.02304.

[27] Lugosch, L., Ravanelli, M., Ignoto, P., Tomar, V. S. and Bengio, Y. Speech Model Pre-Training for End-to-End Spoken Language Understanding. In: *Proc. Interspeech 2019*. 2019, p. 814–818. DOI: 10.21437/Interspeech.2019-2396.

[28] Maaten, L. Van der and Hinton, G. Visualizing data using t-SNE. *Journal of machine learning research*. 2008, vol. 9, no. 11.

[29] McAuliffe, M., Socolof, M., Mihuc, S., Wagner, M. and Sonderegger, M. Montreal Forced Aligner: Trainable Text-Speech Alignment Using Kaldi. In: *Proc. Interspeech 2017*. 2017, p. 498–502. DOI: 10.21437/Interspeech.2017-1386.

[30] Medennikov, I., Korenevsky, M., Prisyach, T., Khokhlov, Y. Y., Korenevskaya, M. et al. Target-Speaker Voice Activity Detection: A Novel Approach for Multi-Speaker Diarization in a Dinner Party Scenario. In: Meng, H., Xu, B. and Zheng, T. F., ed. *Proc. Interspeech 2020*. ISCA, 2020, p. 274–278. DOI: 10.21437/Interspeech.2020-1602.

[31] Olah, C. *Understanding LSTM Networks* [online]. August 2015 [cit. 2021-4-21]. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs.

[32] Panayotov, V., Chen, G., Povey, D. and Khudanpur, S. Librispeech: An ASR corpus based on public domain audio books. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. April 2015, p. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.

[33] Pascanu, R., Mikolov, T. and Bengio, Y. On the difficulty of training recurrent neural networks. In: Dasgupta, S. and McAllester, D., ed. *Proceedings of the 30th International Conference on Machine Learning*. Atlanta, Georgia, USA: PMLR, 17–19 June 2013, vol. 28, no. 3, p. 1310–1318. Proceedings of Machine Learning Research.

[34] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché Buc, F. d', Fox, E. et al., ed. *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, p. 8024–8035.

[35] Peddinti, V., Povey, D. and Khudanpur, S. A time delay neural network architecture for efficient modeling of long temporal contexts. In: *Proc. Interspeech 2015*. ISCA, 2015, p. 3214–3218.

[36] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O. et al. The Kaldi Speech Recognition Toolkit. In: *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.

[37] Ramirez, J., Górriz, J. M. and Segura, J. C. Voice activity detection. fundamentals and speech recognition system robustness. *Robust speech recognition and understanding*. 2007, vol. 6, no. 9, p. 1–22.

[38] Ramirez, J., Segura, J. C., Benitez, C., De La Torre, A. and Rubio, A. Efficient voice activity detection algorithms using long-term speech information. *Speech communication*. Elsevier. 2004, vol. 42, 3-4, p. 271–287.

[39] Ravanelli, M., Parcollet, T., Rouhe, A., Plantinga, P., Rastorgueva, E. et al. *SpeechBrain* [online]. GitHub, 2021 [cit. 2021-5-1]. Available at: https://github.com/speechbrain/speechbrain.

[40] scikit-learn developers. *Metrics and scoring: quantifying the quality of predictions* [online]. 2020 [cit. 2021-5-3]. Available at: https://scikit-learn.org/stable/modules/model_evaluation.html.

[41] Snyder, D., Chen, G. and Povey, D. MUSAN: A Music, Speech, and Noise Corpus. *CoRR*. 2015, abs/1510.08484.

[42] Snyder, D., Garcia-Romero, D., Povey, D. and Khudanpur, S. Deep Neural Network Embeddings for Text-Independent Speaker Verification. In: Lacerda, F., ed. *Proc. Interspeech 2017*. ISCA, 2017, p. 999–1003.

[43] Snyder, D., Garcia-Romero, D., Sell, G., Povey, D. and Khudanpur, S. X-Vectors: Robust DNN Embeddings for Speaker Recognition. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, p. 5329–5333. DOI: 10.1109/ICASSP.2018.8461375.

[44] Tanyer, S. G. and Ozer, H. Voice activity detection in nonstationary noise. *IEEE Transactions on Speech and Audio Processing*. 2000, vol. 8, no. 4, p. 478–482. DOI: 10.1109/89.848229.

[45] Variani, E., Lei, X., McDermott, E., Moreno, I. L. and Gonzalez Dominguez, J. Deep neural networks for small footprint text-dependent speaker verification. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014, p. 4052–4056. DOI: 10.1109/ICASSP.2014.6854363.

[46] Vinodababu, S. *A PyTorch Tutorial to Sequence Labeling* [online]. 2018 [cit. 2021-4-25]. Available at: https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Sequence-Labeling.

[47] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. and Lang, K. J. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. 1989, vol. 37, no. 3, p. 328–339. DOI: 10.1109/29.21701.

[48] Wan, L., Wang, Q., Papir, A. and Moreno, I. L. Generalized end-to-end loss for speaker verification. In: IEEE. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 4879–4883. DOI: 10.1109/ICASSP.2018.8462665.

[49] Wang, Q., Downey, C., Wan, L., Mansfield, P. A. and Moreno, I. L. Speaker diarization with lstm. In: IEEE. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 5239–5243.

[50] Wang, Q., Muckenhirn, H., Wilson, K., Sridhar, P., Wu, Z. et al. VoiceFilter: Targeted Voice Separation by Speaker-Conditioned Spectrogram Masking. In: *Proc. Interspeech 2019*. 2019, p. 2728–2732. DOI: 10.21437/Interspeech.2019-1101.

[51] Wikipedia contributors. *Long short-term memory — Wikipedia, The Free Encyclopedia* [online]. 2021 [cit. 2021-5-1]. Available at: https://en.wikipedia.org/w/index.php?title=Long_short-term_memory&oldid=1014085982.

[52] Zhang, S., Chen, Z., Zhao, Y., Li, J. and Gong, Y. End-to-End attention based text-dependent speaker verification. In: *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2016, p. 171–178. DOI: 10.1109/SLT.2016.7846261.

[53] Zhang, X.-L. and Wu, J. Deep Belief Networks Based Voice Activity Detection. *IEEE Transactions on Audio, Speech, and Language Processing*. 2013, vol. 21, no. 4, p. 697–710. DOI: 10.1109/TASL.2012.2229986.

Appendix A

Contents of the enclosed storage unit

- `xsedla1h_thesis.pdf` – The final `.pdf` file version of this thesis.
- `xsedla1h_thesis.zip` – A `.zip` file containing the LaTeX source code files for this thesis.
- `src/` – A folder containing the source code files.
- `data/` – A folder containing a sample evaluation dataset, trained model files, and a LibriSpeech sample for data preparation and feature extraction demonstration.
- `README.md` – A file documenting the rest of the codebase and containing further instructions for using the software.