Deep learning for speaker recognition in noisy environments

Søren Trads Steen

Kongens Lyngby 2017

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

Speaker recognition is an emerging biometric field for areas in IT-security, such as online banking, access control, call center user validation, and consumer electronics. Speaker recognition is interesting due to the universal presence of microphones in mobile devices, and for use cases including the use of a smartphones, speaker recognition is an obvious tool for verification purposes. Mobile devices are limited in both computing power and storage space however, and for these devices, efficient methods are required. Sound waves propagate through the air from all sound sources, which induce noise from the environment in real world applications. Voice recordings of short durations are challenging as well, yet is good performance of these conditions desirable. The challenge exists as short duration voice samples do not contain a satisfactory amount of information to model each individual speaker, yet the desirability of good performance on short voice segments is high, as the capture time of biometric characteristics is desired short for practical applications, such as phone conversations, or quick access control.
The goal of this thesis is to perform research in increasing performance of speaker recognition systems in regards to sample completeness and signal degradation.

An increase in performance when subjected to sample incompleteness and signal degradation is in this work approached by using quality information about the type of external degradation source. The quality information is used in the decision making stage, after biometric comparisons have been computed, by adjusting the threshold adaptively according to the quality information. The conventional cohort based methods have disadvantages in having to store cohort data, inducing privacy and storage related concerns in the process by storing biometric data, as well as requiring many comparisons. In this work, a novel

approach is proposed using neural networks to perform quality informed comparison score normalization, by using machine learning to find patterns in the effect of environmental interactions on a cohort dataset. The privacy, storage, and computational concerns of traditional cohort based methods are avoided by using neural networks, where only network weights required stored, which do not contain biometric information.

In this thesis, the viability of the proposed method is researched by examining the performance of different configurations of neural networks. The baseline neural network has an improvement of 5.68% over the raw comparison score averaged over all the examined signal degradation conditions. By tuning the network size, a performance gain in equal error rate of 13.3% is reached over the whole dataset, compared to the raw comparison scores. The performance gain exists for every quality degradation pattern examined, yet the gain is higher in difficult situations, which promises performance gains in every situation. The configuration of the empirically tuned network is confirmed by performing Bayesian optimization. The sensitivity of the method towards unknown degradation patterns is also studied, where the model proves robust to unknown noise types, and less robust to unknown patterns of lower duration or higher noise level.

Performing quality-informed comparison score normalization by using neural networks results in tangible performance improvements and is shown in this work to have great potential for flexible integration, as well as allowing computation on mobile devices.

Summary (Danish)

Talergenkendelse, dvs. persongenkendelse gennem stemmen, er et område i vækst indenfor telefonisk bankvirksomhed, adgangskontrol, brugerbekræftelse i call-centre og forbrugerelektronik med mere. Talergenkendelse er brugbart grundet allestedsnærværende mikrofoner i bærbare enheder, herunder mobiltelefoner. Bærbare enheder er typisk begrænsede af både beregningskraft og lagringsplads til data, hvorfor effektive algoritmer er nødvendige. I praktiske brugssammenhænge er det svært at undgå støjgener fra omgivelserne, da lydbølger propagerer gennem luften. Samtidigt er korte lydsekvenser udfordrende, da de indeholder utilstrækkelig information til at modellere hver taler i nødvendigt omfang. Brugen af korte lydsekvenser er dog attraktiv i den virkelige verden, da optagelseslængden til talergenkendelsessystemer ønskes kort, både i tilfælde af telefonsamtaler og i scenarier med adgangskontrol.

Målet med dette speciale er at forske i øgning af ydeevnen for talergenkendelsessystemer i tilfælde af nedsættelse af kvaliteten af talesekvenser både i form af kortere længde af lydsekvens og støjforhold.

Denne øgning i ydeevne, når systemet er udsat for degradering af signalet, tilgås i dette speciale ved brug af kvalitetsinformation om typen af kvalitetsnedsættelsen. Kvalitetsinformationen bliver udnyttet i beslutningsfasen af genkendelsessystemet ved adaptivt at justere beslutningstærsklen ud fra kvalitetsinformationen efter beregningen af biometriske sammenligningsscores er foretaget. Konventionelle kohortbaserede metoder har ulemper ved for det første at skulle lagre data, der indeholder biometrisk information, og dermed medfører problemstillinger i forhold til værn af privatliv. For det andet kræver de konventionelle metoder meget lagringsplads. For det tredje kræver de konventionelle kohortba-

serede metoder megen beregningskraft, da det er nødvendigt at udføre mange sammenligninger.

I dette speciale foreslås en ny fremgangsmåde, som omhandler brugen af neurale netværk til udførelse af scorenormalisering ved hjælp af kvalitetsinformation. Metoden bruger maskinlæring til at finde mønstre i omgivelsernes indflydelse på de kohorte data. Problemstillinger indenfor privatlivssikkerhed, lagringsplads og beregningskraft tilhørende de traditionelle fremgangsmåder undgås ved kun at lagre netværkets parametre, som ikke indeholder biometrisk data.

I dette værk er den foreslåede metode undersøgt ved et studie af ydeevnen af forskellige konfigurationer af neurale netværk. Det neurale netværk brugt som grundlinje opnår en forbedring på 5.68% over den rå sammenligningsscore i gennemsnit over alle støjforhold, som er undersøgt. Ved at udføre en empirisk indstilling af størrelsen på netværket, opnås en forbedring af equal error rate på 13.3% over hele datasættet. Forbedringen opnås for alle støjsituationer, men særligt i de udfordrende, hvor problemer i ydeevne typisk ligger. Ved hjælp af bayesisk optimering er den fundne konfiguration bekræftet indenfor 5% relativt. Metodens følsomhed overfor ukendte støjforhold er ydermere undersøgt, og metoden findes robust over for nye støjtyper, mens robustheden er lavere mod ukendte, højere støjforhold eller lavere længde af lydsekvenser.

Normalisering af sammenligningsscores ved hjælp af kvalitetsinformation og neurale netværk giver en håndgribelig forøgelse i ydeevnen af talergenkendelsessystemer. Metoden har stort potentiale for fleksibel integration i talergenkendelsessystemer og muliggør beregningerne på mobile enheder.

Preface

This thesis was prepared at DTU Compute and Center of Research in Security and Privacy at the da_sec institute at Hochschule Darmstadt, in fulfilment of the requirements for acquiring an M.Sc. in Engineering. The thesis was supervised by Professor Ole Winther at DTU, and Professor Christoph Busch and Doctoral researcher Andreas Nautsch at Hochschule Darmstadt.

The thesis deals with improving biometric performance for speaker recognition in noisy conditions by using neural networks and quality information.

Lyngby, 31-January-2017

Søren Trads Steen

Acknowledgements

I would like to thank Professor Christoph Busch and Hochschule Darmstadt for inviting me to do the thesis with them at the da_sec institute in Darmstadt.

Furthermore I would like to thank my supervisors Professor Christoph Busch, Professor Ole Winther, and Doctoral researcher Andreas Nautsch for their supervision and guidance through the thesis.

I thank Hochschule Darmstadt for funding my stay in Darmstadt, where the work on the thesis was done.

A thanks goes to the I4U consortium as well for supplying the i-vector dataset used for this thesis.

Contents

Chapter 1

Introduction

Biometrics is an active field of research with many important use cases in phone banking, access control, and call center user validation. Other use cases include avoiding malicious or impractical access to voice control units, such as the Amazon Echo or Google home, or when using voice control of smartphones using Siri or Google Voice, by limiting the device to specific users. Furthermore, speaker recognition can be used in forensics for instance as evidence against criminals, or to identify victims of natural disasters.
In an increasingly digitalized and globalized world, information has a large value, and the security of digital data has an ever increasing importance. Using biometrics to secure access to data or areas has multiple advantages over using a password or PIN, including containing more information, as well as being much more difficult to forget, as biometric characteristics are inherently attached to the body of the subject. Therefore, Biometrics grant value both from a usability and security viewpoint. One example hereof is the inclusion of biometric fingerprint scanners into smartphones, adding usability and security. In smartphones, biometrics can be used for unlocking the smartphone, authorizing a mobile payment or changing user-specific settings, as microphones are present in all mobile phones. Speaker recognition is using the voice as biometric modality.
Fingerprints, which are used in most newer smartphones, currently have greater biometric performance than voice, yet speaker verification has further use cases such as verifying a subject calling an insurance company or a bank, to avoid fraudulent or illegal situations. Other use cases include suspect tracking and

cases against blackmailers or other criminals, where a voice recording is present. Furthermore, speaker verification does not rely on specialised hardware, but can use the microphone already present in the smartphone, as well as not needing close physical proximity. However, an issue in speaker recognition is that different microphones perform differently, which makes algorithms performing well with one microphone not necessarily perform well with another.

1.1 Motivation

Enrolment of speakers into the biometric database can most times be assumed to be performed under good conditions, yet the use cases of smartphones includes mobility and therefore interaction with the environment, as well as sample incompleteness, affecting the probe. The induced sample incompleteness and signal degradation from the external environment has an influence on the verification result – systems perform worse in very noisy conditions compared to clean, controlled environments.

In order to attempt to counteract the influence of the external environment, the conventional method is to use cohort information to normalize the comparison score from the system, which is a score denoting the likelihood of the subject being who they claim to be. Cohort information is information belonging to a group of individuals related to the individual being subjected to biometric comparison.
Using cohort information directly introduces privacy concerns, as the cohort data needs to be present and stored to perform the score normalization. Furthermore, storing the cohort data introduces some data storage concerns as well, as the amount of data stored is unsuitable for use in mobile devices. The computational power needed to perform the many comparisons to the cohort dataset is also infeasible for mobile devices.
Storage and computation issues are not as big problems however for non-mobile and offline problems, and the method of cohort selection is therefore applicable in fields like forensics, as well as in situations where a server-client communication structure is possible, such that the privacy concerns can be mitigated as well by harsh privacy procedures server side.

A possible alternative for mobile devices, which do not have the privacy issue of storing biometric data, is to use neural networks to perform the score normalization. Neural networks can be trained offline, while the feed-forward evaluation of new samples requires relatively little computational power, which makes it possible to perform the score normalization online in mobile devices, while sustaining the information provided by cohort-based score normalization.

Storage issues are also alleviated, as the only data required to be stored are the parameters/weights of the neural network, which however can add up for very large networks.

In this thesis, an approach is proposed by using neural networks for performing the comparison score normalization, using quality information of the samples to counteract unconstrained environmental factors and changing sample durations. As the method relates to noisy conditions, the robustness towards new, unseen or unknown conditions is also investigated, as unknown noise conditions appear in real world use cases.

1.2 Research questions

The scope of this thesis is investigating the feasibility of using neural networks for normalizing comparison scores to achieve better performance in regards to sample completeness and signal degradation.

- **How well can comparison scores be normalized using neural networks?**
  Using the non-normalized comparison scores as baseline, questions investigated are:
  1. Given i-vectors, quality information and comparison scores, can a simple neural network of 2 layers perform normalization of comparison scores with a better performance than baseline?
  2. Does using a deeper network architecture provide a relative gain of 5% or more over 1 layer models?
  3. Will using Bayesian optimization for tuning hyperparameters of the network confirm the empirically found hyperparameters to within a relative change of $\pm 5\%$?
- **How robust is the neural network approach to unknown conditions?**
  Robustness towards unknown noise and duration conditions can be simulated by excluding training data. Robustness is examined regarding noise type and noise level.
  4. Does the network perform to within 20% relative to its performance on a different noise type, when the noise type is not included in the training dataset?
  5. Does the network loose less than 20% relative to its performance on worse conditions, when these are not included in the training dataset?

1.3 Organisation of work

The fundamentals and theory behind the neural networks used in this thesis are explained in chapter 2, which will introduce biometrics and neural networks. In chapter 3, the proposed method of using neural networks for quality-informed score normalization is introduced and explained. In chapter 4, the experiments conducted to answer the research questions and the results of the same are reported and discussed. In chapter 5, a conclusion is made on the subjects of this thesis, as well as reflection on possible future work.

Chapter 2

Fundamentals

In this thesis, the focus will be on speaker recognition, and the use of neural networks for normalizing the comparison scores used in speaker recognition systems. To introduce fundamental concepts in these areas of research, a brief introduction will first be given to biometrics, followed by a section expanding on concepts specifically relevant for speaker recognition, such as extraction of the feature i-vectors and the processing of the i-vectors. The concepts of neural networks will be explained, including the structure of neural networks as well as details relating to the training of the same.

2.1 Biometrics

Biometrics is the measurement of biological traits. More specifically the measurement of biological traits to identify or verify an individual. The formal definition of biometrics is [ISO11]:

> **Biometrics:** automated recognition of individuals based on their behavioural and biological characteristics

The recognition of individuals can be sorted into two main problems – verification and identification. Identification is the problem of identifying an individual from a database. This is a one to many comparison, as the individual needs to be matched to the correct subject in the database. Verification on the other hand is a one to one comparison to determine whether or not the individual is the subject in the database they are claiming to be.

In biometrics, an individual is first enrolled into the system, when an individual is first captured by the system. The extracted enrolment template is referred to as the reference, which is later used for comparisons with a probe. A verification system operates with the individual either being a genuine comparison, if the probe is sufficiently alike to the reference stored, or being an imposter. Imposters are false matches that should not be verified by the system. Imposters can both be intentional, such as an individual attempting to spoof or cheat the system to gain access, or unintentional, such as an identical twin.

As the definition of biometrics instigates, there are different behavioural and biological characteristics to examine. Each of these characteristics is called a modality. Examples of biological modalities are fingerprint scans, iris scans, facial scans, and DNA profiling. These modalities are all something that are physical aspects of our bodies, which, if unique, can be read or scanned and therefore determine the identity of a subject. Behavioural modalities on the other hand denotes modalities such as gait, keystroke pattern, and signatures. These modalities are not something that is inherently built into the physical body, yet are inherently personal [JFR07].

2.1.1 Biometric systems

There is a distinction between the hard biometric modalities and soft biometric modalities. Soft biometric modalities are characteristics that are discriminative of the subject, but is assumed to be shared between many individuals, such as height, weight, eye colour, gender, and race. Soft modalities cannot be used to identify or verify a subject, however they can help reducing the space of individuals that needs to be searched – The search in the database can be restricted to within, for instance, tall women with blue eyes and dark hair. Soft biometrics cannot therefore identify or verify a subject alone, but can improve the performance of multimodal systems [JFR07]. Multimodal systems utilize several biometric modalities in order to increase the performance of the system over a system using only one modality.

Many modalities exist, yet some are softer than others. What attributes or factors are then important for a biometric modality to possess? In [JFR07] seven

attributes are mentioned, which describe the powerfulness of the modality. The purpose of these attractive attributes that a biometric modality possess is to compare different modalities by core elements. The seven attributes are:

- **Universality:** Every individual should possess the modality measured – if only certain people possess the modality, the system cannot perform universally on the general population.
- **Uniqueness:** The uniqueness of the modality to each individual is important, otherwise identification or verification does not make sense, if there is no difference in the characteristic across many individuals. For instance using height – many people are of the same height, and as such identifying people by this modality is not very precise.
- **Permanence:** The modality should persist over time – if the measurement of the characteristic is not invariable to the passing of time, discerning between individuals over time becomes rather impossible.
- **Measurability:** It should be possible to measure the modality consistently in a convenient matter.
- **Performance:** The characteristic should be discriminative enough for the given application and the verification or identification should be possible to perform without excessive computation power for the purpose as well.
- **Acceptability:** The modality should be something that individuals are prepared to show to the measurement device.
- **Circumvention:** It should be difficult to circumvent the system by using fake modalities. Detecting fake modalities is referred to as presentation attack detection. Two types of circumvention exist; to gain access as another user or to disguise as not to be recognized correctly or at all.

An ideal modality performs well in all seven criteria for all applications, however there does not exist an ideal biometric modality [JFR07]. However, biometrics can still be useful.
Using biometrics for verification has both advantages and disadvantages compared to using a traditional password or PIN code based systems.

An advantage is that it is impossible to forget a biometric characteristic compared to a password, as all that is required for verification is the presentation of a physical aspect of the body, such as a finger or face. Using biometrics to gain access to for instance a laptop or a smartphone is usually also as quick as entering a password – yet without having to remember the password. A fallback password is however usually needed in case of scanner malfunction or

sudden loss of biometric characteristic. Currently, fingerprint scanners are very widespread on smartphones, where the verification of the user can be performed in the same way as the normal use of the device, for instance touching the phone for fingerprint recognition, or looking at it for face recognition. Another advantage of biometrics is that it requires more effort to steal or synthesize the biometric characteristics compared to stealing or guessing a password or token, in a token based system. Biometric systems are of course not completely secure against people with malicious intent, however it requires more physical effort for instance to construct a fake finger or cheat a facial scanner. An ongoing research area in biometrics is the countermeasures towards attacks on biometric systems, such as detection whether the subject is alive or not, whether the image is printed or a real face, or if the phrase uttered for a speaker recognition system is a replayed recording. Another advantage of biometrics over for instance PIN codes is entropy of the feature space in term of the amount of information available. There is much more information available in a fingerprint, facial scan, or speech segment [NRSB15] than in a short number sequence.

A disadvantage to using biometrics is that if a password is compromised in some way, it can just be reset – which is much more difficult with biometric traits. Biometrics therefore are not cancelable. Research into revocable biometric templates [JFR07] is a countermeasure to this problem. A biometric template is the stored biometric features. Having the template revocable means that the biometric template stored in the database can be cancelled if compromised.

Another challenge in biometrics is privacy. Many conditions and diseases that should be held private can be inferred from biometric data. Furthermore it is possible to track individuals across services if biometric verification is used, as the biometric template will function as an easily trackable user ID. Privacy adds another challenge to biometric recognition in the way that biometrics are not secret. In the case of a password or token based system, it is reasonable to assume that only the correct person is in possession of the password – unless it has been compromised. Biometric modalities are on the other hand almost impossible to keep secret [JFR07], as you leave fingerprints everywhere you go, and seldom cover our faces in public. As such biometrics such as face recognition make surveillance systems available, where the user can be tracked from their identity without their consent.

A biometrics system conventionally consists of multiple subsystems, as seen in fig. 1 from [ISO11]. The subsystems are:

- **Data capture subsystem:** Collects sample of the biometric characteristics of the individual through a measurement device or sensor.

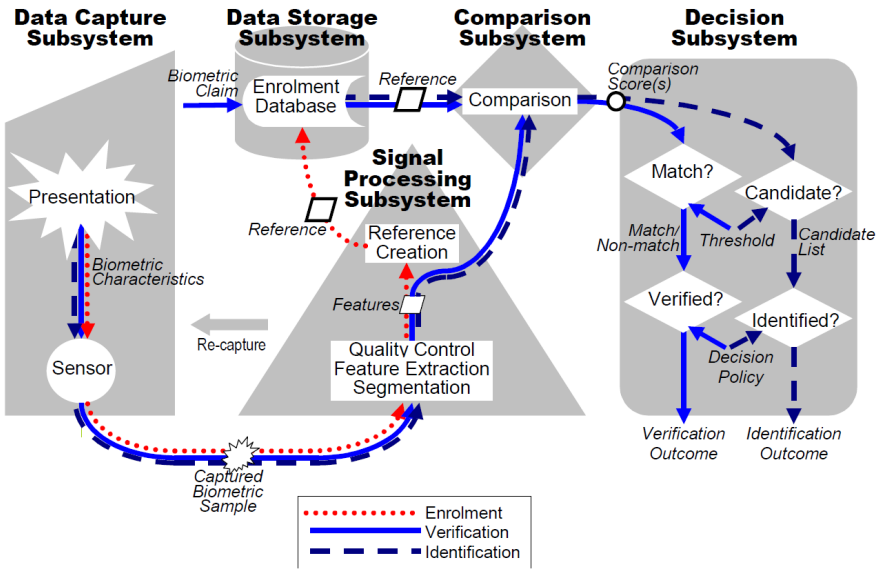


Figure 1: The subsystems typically included in a biometric system, see [ISO11].

- **Signal processing subsystem:** Performs different subtasks with the purpose of constructing features from the biometric characteristics extracted in the data capture subsystem. Subtasks performed include among other segmentation of the area of interest of the captured signal, performing quality enhancement and pre-processing of the signal, and extracting features.
- **Data storage subsystem:** Stores captured and computed templates from the previous enrolment of subjects.
- **Comparison subsystem:** Compares one or more references from the data storage to a probe captured from an individual. Computes a comparison score denoting the likeliness of the reference and probe belonging to the same individual.
- **Decision subsystem:** From the comparison score a decision is made from a threshold, as well as incorporating a decision policy, for instance requiring multiple correct attempts before choosing a decision.

When designing a system it is relevant to mind the purpose of the system. Different purposes may have different performance requirements. To measure the performance of a biometric system, different metrics are used.

2.1.2 Metrics

To be able to measure and compare the performance of a biometric system, a definition is first needed to define what performance is. The measurements of performance are computed in different ways, and are called metrics.

In a biometric comparison system, there are just four possible outcomes:

- **True match (TM)**: When the classifier correctly verifies a genuine subject.
- **False match (FM)**: When the classifier incorrectly verifies the genuine subject
- **True non-match (TNM)**: When the classifier correctly rejects an imposter.
- **False non-match (FNM)**: When the classifier incorrectly classifies the subject as an imposter.

The false match rate (**FMR**) denotes the proportion of imposter attempts that are accepted, and the false non-match rate (**FNMR**) denotes the proportion of genuine verification attempts that are rejected by the classifier.

The false match rate 100 (**FMR100**) metric is defined as the FNMR when FMR = 1%. Similarly, the equal error rate (**EER**) is the point, where the FNMR and FMR are equal, and therefore the point where the system rejects genuine comparisons at the same rate as accepting imposter comparisons.

Classifiers used in biometrics compute and make decisions on comparison scores, which can be similarity or dissimilarity scores. In binary decision problems, the comparison scores can be converted to, and thus treated as, likelihood ratios. The likelihood ratio (**LR**) is given as the ratio of the probability of the observation given match and the probability of the observation given non-match

$$LR = \frac{p(\mathcal{D}|H_0)}{p(\mathcal{D}|H_A)}, \tag{2.1}$$

where $\mathcal{D}$ is the observation, H_0 is the null hypothesis (genuine) and H_A is the alternative hypothesis (imposter). For several practical reasons, the likelihood ratio is transformed into the log-likelihood ratio (**LLR**), which is the logarithm of the LR. From the LLR a classification decision can be performed by a decision strategy, for instance by thresholding.

The receiver operating characteristic (**ROC**) curve is the true match rate, TMR, plotted against the FMR for different values of the the threshold used for determining the class of comparison scores. The EER can be read off the ROC curve at the point where FMR $= 1 -$ TMR. In practise however, there will rarely be a point on the empirical ROC curve, where this is equivalence is exactly satisfied [Brü10]. A simple way to compute the EER is to choose the point where $|FMR - FNMR|$ is minimized [KEY+16]. A more advanced solution is to compute the continuous convex hull of the ROC [Brü10, SNP98, Faw06], the **ROCCH**. Opposed to discrete or steppy ROC curves, the EER can be read directly from the ROCCH, which is referred to as the ROCCH-EER.

The ROCCH is computed using the pool-adjacent-violators (**PAV**) algorithm, see [Brü10, BdP06] for details. An example of both ROC and ROCCH curves can be seen in fig. 2a.

In order to visualize the difference between multiple systems, the detection error tradeoff, **DET**, can be plotted. This plot is constructed from the ROC curve, by the probit transformation [Brü10] of both axes, which is given as

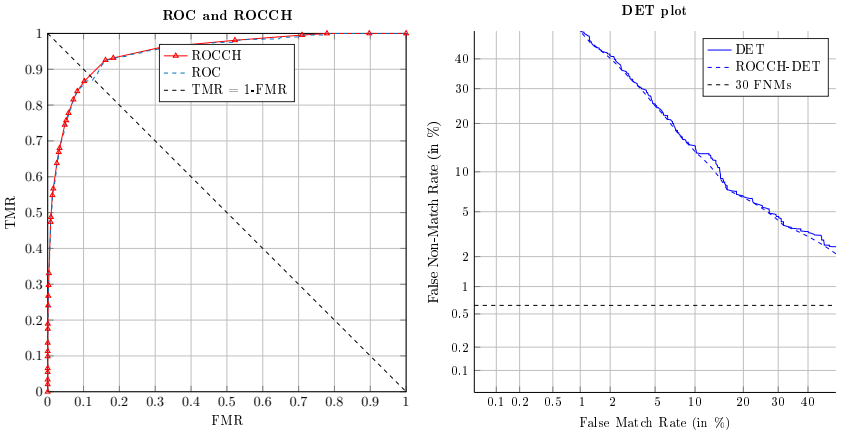
$$\text{probit}(p) = \sqrt{2}\text{erf}^{-1}(2p-1), \tag{2.2}$$

where the erf^{-1} is the inverse error function $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,\mathrm{d}t$. On the DET curve it is easier to distinguish relative differences between systems, even in areas of small error rates. The DET curve can also be constructed from the ROCCH curve, giving rise to the ROCCH-DET curve. An example of a DET curve and its corresponding ROCCH-DET curve can be seen in fig. 2b.

The cost of LLR, $\mathbf{C}_{\text{llr}}^{\text{min}}$, is a metric which describes the goodness of the LLR fit in well-calibrated systems [NSRB15, BdP06, Brü10], and is calculated from genuine and imposter LLR scores S_G, S_I by

$$\text{C}_{\text{llr}} = \frac{\sum_{g \in S_G} \log_2(1 + 1/\text{e}^g)}{2|S_G|} + \frac{\sum_{i \in S_I} \log_2(1 + \text{e}^i)}{2|S_I|}. \tag{2.3}$$

The minimum C_{llr}, $\mathbf{C}_{\text{llr}}^{\text{min}}$, denotes the C_{llr} of the system assuming perfect calibration. This metric can be computed using the PAV algorithm [BdP06] to find an optimum non-decreasing posterior mapping of every score, and thereby calculating the best threshold score between classes given the data.



(a) An example of a steppy ROC and a ROCCH curve.

(b) An example of a steppy DET and ROCCH-DET curve.

Figure 2: Examples of ROC and DET curves.

2.2 Speaker recognition

Voice is an important communication tool for humans, and as such includes a lot of information. Speech recognition is the problem of discerning which words are being said and what intention lie behind, whereas speaker recognition is the problem of determining not what is being said, but by whom it is being said. Here the focus is entirely on the biometrics problem of speaker recognition.

Speaker recognition systems can be both phrase dependent and independent – whether or not a certain phrase must be uttered for the system to work, or if the system works independently of the words being said. Text independent systems have a higher resistance to fraud and imposter than phrase dependent [JFR07].

Unlike biometric modalities such as fingerprints or irises, speech requires an active participation of the subject to produce the voice. By having to actively perform an action (speaking) to collect the biometric features, some variability sources are induced into the speech signal. These include [HH15]:

- **Situational stress**: The speaker is concentrating on another task while speaking, such as driving.
- **Vocal effort**: The speaker has altered speech through whispering, shouting, or singing.

- **Emotion**: The speaker is subjected to an emotion, which is communicated into the speech, such as anger or elation.
- **Physiological**: The speaker's voice is influenced by a physiological factor, such as illness or intoxication.
- **Disguise**: The speaker is intentionally altering their voice.
- **Situation**: Speech is different depending on the situation. Factors include the language or dialect spoken, whether the speech is prepared or spontaneous, or whether it is monologue or dialogue. Another factor is whether the speech is directed towards another human being, or towards a machine.
- **External factors**: Using different microphones with different characteristics, and using different sampling rates, durations and compression methods all induce variability to the speech signal. Furthermore, the external environment also has a big influence by having background noise, as well as room acoustics having an influence.

By having variability in the speech signal, a large challenge in automatic speaker recognition is to model the speaker mathematically while having a large robustness to different variability factors.

While humans communicate by speech and also have a good ability trained from childhood of discerning persons by their voice, there are several reasons why automatic speaker recognition is useful. Humans suffer from contextual bias (whether or not something is known about the subject already), as well as being limited by memory and fatigue [HH15]. Additionally is the question of repetition – in a forensic situation it may be feasible to use a human expert for speaker recognition, but for a mass-market applications, such as unlocking of phones or other security applications, it quickly becomes infeasible and too slow to employ speaker recognition experts to identify each person separately, due to e.g the amount of people involved and the speed required.

An automatic speaker recognition system conventionally can be segmented into two main parts, the frontend, which processes the raw speech signal and computes biometric features, which are fed into the backend. In the backend, the features are fed through an algorithm, typically machine learning, and computes a decision whether or not the is genuine or an imposter. The workflow of a speaker recognition system can be seen in fig. 3.

The focus of this thesis is on one of the last steps in the processing chain in the backend process. After computing a likelihood ratio that the subject is a genuine match, it is possible to normalize these scores to attempt to account for

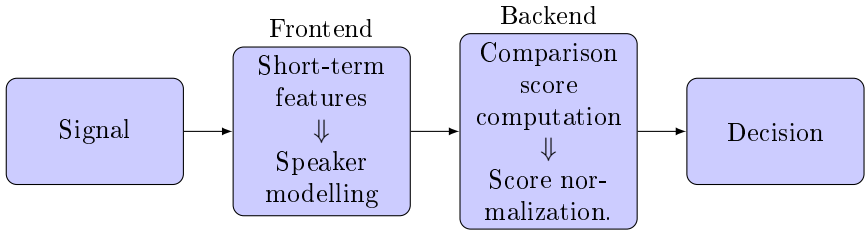


Figure 3: A typical workflow for speaker recognition.

noisy conditions and variability in the speech signal. A section will introduce the basics of the current state of the art speaker recognition systems in regards to frontend and backend processes by introducing short term features, modelling of the speaker space by using i-vectors as features, and normalizing the comparison scores. A section is included describing the fundamentals of neural networks, which is here used to perform the score normalization.

2.2.1 Short term features

Short term features are used in the frontend process by extracting features from very short time segments of the speech signal. One of the most popular [HH15] short term features is the Mel-frequency cepstral coefficients (MFCC). The short term features are usually not extracted on the whole signal, as speech has many pauses naturally included. Therefore the short term features are extracted only on the regions of interest in the signal – where there is actual speech. This area is extracted using various methods of voice activity detection (VAD) [HH15].

For the purpose of extracting the short term features (MFCCs) from the audio signal, the following steps are taken, which are sketched in fig. 4:

- Divide the signal into short overlapping intervals, typically of 10 milliseconds [HH15]. Weight the signal by a window function, as the sections are overlapping, for instance by multiplying by a hamming window function.
- Compute the Fourier power spectrum of the windowed signal, and compute the logarithm of the spectrum.
- Use a Mel-space filter bank on the logarithm of the power spectrum, resulting in the filter-bank energy coefficients [HH15]. These coefficients describe the different frequency bands investigated in the Mel-filter bank.

- Perform a discrete cosine transform (DCT) of the filter-bank energy coefficients, as if they were a signal. The Mel frequency cepstral coefficients are now the coefficients of the DCT.

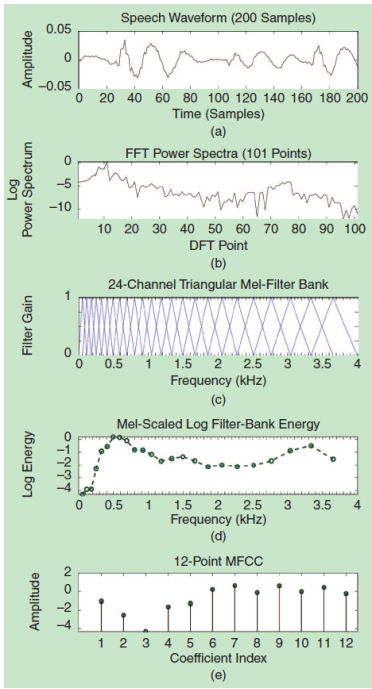


Figure 4: The process for extracting the MFCCs, see [HH15]

Short term features however only describe the speech signal in section of very short duration. To be able to discriminate between speakers, the need to be able to model and compare each speaker arises.

2.2.2 Modelling of speaker space

Using short term features supplies a description of the sound signal, but do not model the speaker, To be able to discern between speakers, a model of the speaker space extracted from the short term features is required.

The current state of the art methods are based on a Gaussian Mixture models (GMMs) approach [HH15].

In the original GMM method, each speaker is modeled by a multivariate probability density function (PDF) provided by a GMM clustering. By evaluating the PDF at different collected data points, a probability score of the data point belonging to the speaker modelled by the GMM is achieved. The GMM method works for identification of subjects by comparing the features collected to each GMM model enrolled in the system. The PDF given by a GMM of M mixtures, for an input vector $\mathbf{x_n}$ is given by a weighted sum of Gaussian distributions with mean μ_g and covariance matrices Σ_g, where $g = 1, 2, \ldots, M$.

$$p(\mathbf{x_n}|\lambda) = \sum_{g=1}^{M} \pi_g \mathcal{N}(\mathbf{x_n}|\boldsymbol{\mu_g}, \boldsymbol{\Sigma_g}), \tag{2.4}$$

with π_g denoting the weight of mixture component g. The GMM model can be denoted by its weight, mean, and covariance parameters by $\boldsymbol{\lambda}$. For a sequence of feature vectors, the probability of the observations are multiplied together to get the probability of observing all the feature vectors given the GMM.

2.2.3 GMM-UBM approach

For speaker verification, an alternate speaker model is needed to denote all speakers not the target. The verification can then be rejected if the measured data are more alike to the alternate speaker than to the target. The conventional choice of alternate model [HH15] is the universal background model (UBM). The UBM is a well trained GMM on a large dataset of enrolled speakers, therefore representing all speakers in general.

The modelling of the speaker subspaces and estimating all necessary GMM parameters, especially the covariances, is not possible given the comparatively small enrolment sets. The speaker dependent GMMs are therefore conventionally adapted from the UBM, which is done by a maximum a posteriori adaptation (MAP) of the UBM.
The MAP adaptation is a follows [HH15]. Given the collected feature vectors $\mathcal{X} = \{\mathbf{x_n}|n \in 1, \ldots, T\}$ the probabilistic alignment of a UBM GMM component g to the feature vectors is calculated

$$p(g, \mathbf{x_n}, \lambda_0) = \frac{\pi_g p(\mathbf{x_n}|g, \boldsymbol{\lambda}_0)}{\sum_{g=1}^{M} \pi_g p(\mathbf{x_n}|g, \boldsymbol{\lambda}_0)} = \gamma_n(g). \tag{2.5}$$

The zero, first-, and second-order Baum-Welch statistics are calculated as

$$N_s(g) = \sum_{n=1}^{T} \gamma_n(g), \tag{2.6}$$

$$\mathbf{F}_s(g) = \sum_{n=1}^{T} \gamma_n(g)\mathbf{x_n}, \tag{2.7}$$

$$S_s(g) = \sum_{n=1}^{T} \gamma_n(g)\mathbf{x_n}\mathbf{x_n^T}. \tag{2.8}$$

Extraction of the Baum-Welch statistics has also been attempted using deep neural networks. Even though the Baum-Welch statistics obtained this way perform worse than the normal procedure, there is an information gain to be had by fusing the methods [KGS+14].
The posterior mean and covariance matrix of the feature vectors given the dataset can be computed as

$$E_g[\mathbf{x_n}|\mathcal{X}] = \frac{\mathbf{F_s(g)}}{N_s(g)}, \tag{2.9}$$

$$E_g[\mathbf{x_n}\mathbf{x_n^T}|\mathcal{X}] = \frac{S_s(g)}{N_s(g)}. \tag{2.10}$$

By defining the variable α_g as

$$\alpha_g = \frac{N_s(g)}{N_s(g) + r},$$

where r denotes the relevance factor of the new feature vectors compared to the UBM model, the UBM model can be adapted to a speaker by updating weight, mean, and covariance matrix:

$$\widehat{\pi_g} = (\alpha_g N_s(g)/T + (1 - \alpha_g)\pi_g)\beta, \tag{2.11}$$

$$\widehat{\mu_g} = \alpha_g E_g[\mathbf{x_n}|\mathcal{X}] + (1 - \alpha_g)\mu_g, \tag{2.12}$$

$$\widehat{\Sigma_g} = \alpha_g E_g[\mathbf{x_n}\mathbf{x_n^T}|\mathcal{X}] + (1 - \alpha_g)(\Sigma_g + \mu_g\mu_g^T) - \hat{\mu_g}\widehat{\mu_g^T}. \tag{2.13}$$

Here β is a scaling factor computed so that the sum of all weights of the mixtures sum to 1. The reason for adapting the speaker model from the UBM model is that these models are more robust and reliable [HH15] than individually trained models for each speaker.

The parameters of the GMM model can be concatenated into a GMM supervector, which will have a fixed size from the amount of mixtures used to cluster the short term features. Generally, only the mean vectors are used in the GMM supervector [HH15]. The GMM-UBM modelling approach is seen in fig. 5.

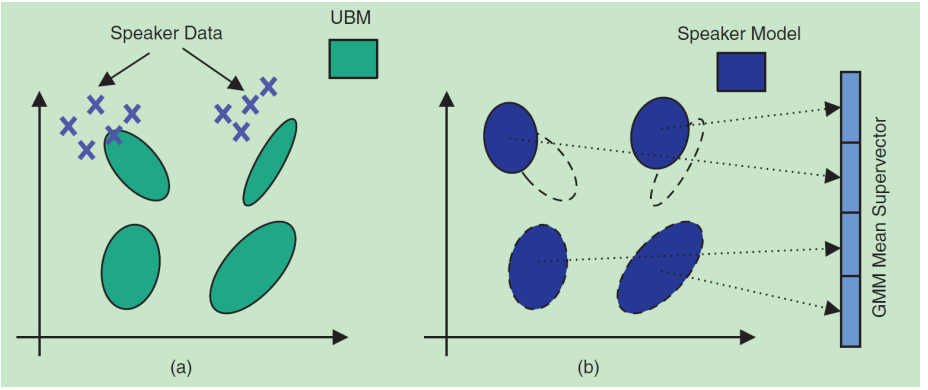


Figure 5: The GMM-UBM method with a four mixture GMM. (a) The trained UBM with measured speaker data. (b) The speaker model after MAP adaptation of the UBM model, as well as the concatenation of mixture means into the GMM supervector. See [HH15].

For large GMM models, the GMM supervectors can become large in dimensions, which can make standard processing algorithms impractical. A way to reduce the dimensionality of the supervectors is to use i-vectors.

2.2.4 i-vectors

The state of the art methods use intermediate sized vectors (i-vectors), which is a method to reduce the dimensionality of the GMM supervectors [HH15]. The i-vector approach is a factor analysis method, where a speaker and session dependant GMM supervector $\mathbf{m}$ is factorised as

$$\mathbf{m} = \mathbf{m_0} + \mathbf{Tw}, \tag{2.14}$$

where $\mathbf{m_0}$ is a speaker and session independent component, whereas $\mathbf{w} \sim \mathcal{N}(0, \mathbf{I})$ represent the total factors, which are latent variables, referred to as i-vectors.

2.2.4.1 Space projection

When the i-vectors have been computed, it is possible to perform an orthogonal space projection to further reduce dimensionality, but keep information along the directions in feature space that are the most discriminating of speakers. A method to use for this orthogonal projection is linear discriminant analysis

(LDA). Let $\mathcal{D}$ be the dataset of i-vectors $\mathbf{w_{s,i}}$, subscripts denoting the ith i-vector of speaker s. Let S be the total number of speakers in $\mathcal{D}$ and n_s be the number of i-vectors belonging to speaker s. Then the between and within class covariance matrices are given by

$$\mathbf{S_b} = \frac{1}{S}\sum_{s=1}^{S}(\bar{\mathbf{w}}_\mathbf{s} - \bar{\mathbf{w}})(\bar{\mathbf{w}}_\mathbf{s} - \bar{\mathbf{w}})^T, \tag{2.15}$$

$$\mathbf{S_w} = \frac{1}{S}\sum_{s=1}^{S}\frac{1}{n_s}\sum_{i=1}^{n_s}(\mathbf{w_{s,i}} - \bar{\mathbf{w}}_\mathbf{s})(\mathbf{w_{s,i}} - \bar{\mathbf{w}}_\mathbf{s})^T, \tag{2.16}$$

where $\bar{\mathbf{w}}_\mathbf{s}$ is the speaker dependent mean, and $\bar{\mathbf{w}}$ is the speaker independent mean vector given by

$$\bar{\mathbf{w}}_\mathbf{s} = \frac{1}{n_s}\sum_{i=1}^{n_s}\mathbf{w_{s,i}}, \tag{2.17}$$

$$\bar{\mathbf{w}} = \frac{1}{S}\sum_{s=1}^{S}\bar{\mathbf{w}}_\mathbf{s}. \tag{2.18}$$

The LDA procedure maximizes the between class covariance, while minimizing the within class covariance [HH15]. The LDA projection is made from projecting the i-vectors on the first k eigenvectors of the eigenproblem

$$\mathbf{S_b v} = \mathbf{\Lambda S_w v}. \tag{2.19}$$

where $\mathbf{\Lambda}$ is the diagonal matrix containing the eigenvalues.

The within class covariance normalization (WCCN) [HKS06, HH15] projection is computed as

$$\mathbf{\Phi}_{\mathrm{WCCN}}(\mathbf{w}) = \mathbf{A}_{\mathrm{WCCN}}^{\mathbf{T}}\mathbf{w}, \tag{2.20}$$

with $\mathbf{A}_{\mathrm{WCCN}}$ computed through Cholesky factorisation such that

$$\mathbf{S_w^{-1}} = \mathbf{A}_{\mathrm{WCCN}}\mathbf{A}_{\mathrm{WCCN}}^{\mathbf{T}}.$$

The WCCN projection has the advantage of forcing the data to have independent elements by forcing the covariance matrix to be an identity matrix. By having the data belong to independent Gaussian distributions is a useful attribute for the next step in the process, which is to compute a comparison score between i-vectors.

2.2.4.2 Comparison scores

To compute comparison scores between two i-vectors, probabilistic LDA (PLDA) can be used. By performing length normalization of the i-vectors to unit length

[GREW11, HH15], a Gaussian PLDA (GPLDA) method can be used, which is less computationally heavy than the otherwise used heavy tailed PLDA without performance degradation [GREW11]. The PLDA [BBM14] assumes that an i-vector $\mathbf{w}$ can be decomposed as

$$\mathbf{w} = \boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{y_s} + \boldsymbol{\varepsilon}. \tag{2.21}$$

Here $\boldsymbol{\mu}$ is a speaker independent factor, $\boldsymbol{\Phi}$ is a matrix representing the speaker dependent basis functions or eigenvoices [HH15], and $\mathbf{y_s}$ are latent variables representing the between speaker variability. $\boldsymbol{\varepsilon}$ denotes the residual. In GPLDA the it is assumed that $\mathbf{y_s} \sim \mathcal{N}(0, \mathbf{I})$, and the residual $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \boldsymbol{\Lambda})$. The parameters of the model can be computed using an expectation maximization algorithm [BBM14].

To compute the log likelihood ratio between two i-vectors $\mathbf{w_1}, \mathbf{w_2}$, the score is given

$$S_{\text{PLDA}}(\mathbf{w_1}, \mathbf{w_2}) = \log(\frac{p(\mathbf{w1}, \mathbf{w2}|H_0)}{p(\mathbf{w1}, \mathbf{w2}|H_A)}), \tag{2.22}$$

where the hypothesises are as in eq. (2.1), whether or not the hypothesis that they are i-vectors belonging to the same speaker.

For GPLDA it is possible to compute the comparison score analytically [BBM14]

$$S_{\text{PLDA}}(\mathbf{w_1}, \mathbf{w_2}) = \log \mathcal{N}\left(\begin{bmatrix}\mathbf{w_1}\\ \mathbf{w_2}\end{bmatrix} \middle| \begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{\text{tot}} & \boldsymbol{\Sigma}_{\text{ac}}\\ \boldsymbol{\Sigma}_{\text{ac}} & \boldsymbol{\Sigma}_{\text{tot}}\end{bmatrix}\right) \tag{2.23}$$

$$- \log \mathcal{N}\left(\begin{bmatrix}\mathbf{w_1}\\ \mathbf{w_2}\end{bmatrix} \middle| \begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{\text{tot}} & 0\\ 0 & \boldsymbol{\Sigma}_{\text{tot}}\end{bmatrix}\right), \tag{2.24}$$

where

$$\boldsymbol{\Sigma}_{\text{tot}} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathbf{T}} + \boldsymbol{\Lambda}, \tag{2.25}$$

$$\boldsymbol{\Sigma}_{\text{ac}} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathbf{T}}, \tag{2.26}$$

and are respectively the total covariance matrix and the across class covariance matrix.

This comparison score can then be used to perform a decision from the decision policy, for instance a threshold.

2.2.5 Score normalization

As the voice recordings used in speaker recognition can be recorded in many different environments, with many different conditions or nuisance factors, performing a normalization of the comparison score using information about the

environment is useful. Noise effects from the environment influences the speaker verification system, but the duration of the speech recorded also has an impact. Shorter recordings do not to the same degree deliver information to extract the necessary statistics for i-vector extraction [NRB+14].

One approach is to normalize the comparison score using cohort information. Cohort information is information stemming from other speakers or cohorts. The zero-normalization (z-norm) scheme normalizes the comparison score by using comparisons between references and cohorts, while test-normalization (t-norm) is performed by comparing the cohort set with the probe. State of the art PLDA systems use a symmetric fusion of z-norm and t-norm (s-norm) [NSRB15].

Using a subset of cohorts instead of using all cohort data improves robustness [NSRB15, MPK12]. Using a subset of the data to compare each reference and probe to is called adaptive symmetric normalization, AS-norm. By comparing the probe and reference to the cohort dataset, comparison scores are achieved – from where mean and standard deviation can be extracted from the best k cohorts. Then, the AS-norm can be computed from the PLDA score S as

$$S_{\text{AS}} = \frac{1}{2}\left(\frac{S-\mu_{\text{ref}}}{\sigma_{\text{ref}}} + \frac{S-\mu_{\text{prb}}}{\sigma_{\text{prb}}}\right), \tag{2.27}$$

where $\mu_{\text{ref}}, \sigma_{\text{ref}}$, denote the mean and standard deviation of the comparison scores between the reference and reference cohort set. Likewise $\mu_{\text{prb}}, \sigma_{\text{prb}}$ denote the mean and standard deviation of the comparison score between the probe and probe cohorts.

Disadvantages to using cohort information directly is that many comparisons are required, and therefore computation power, as the subset of cohorts is chosen per probe and reference comparison, but also that it requires that the cohort information is stored to be used at verification time. Storing data from other speakers is a privacy issue.

In [NSRB15, NSRB16] the use of quality information of the i-vectors in regards to the conditions of the environment in both duration and noise is proposed to select the subset of cohorts used for normalization. This quality information is computed through quality vectors.

2.2.6 Quality vectors

In [FBPS12], a unified approach for audio characterization (UAC) is proposed. Here, i-vectors are classified into different classes - one for each nuisance factor

or condition. A condition can be different noise type, noise amount or duration of the speech recordings. The approach in [FBPS12] is to model each condition or nuisance factor with a Gaussian distribution with mean as the mean of all i-vectors in that class. The covariance of all the Gaussians is forced to be identical and is estimated as the covariance of the i-vectors with the mean of the corresponding class subtracted.

An i-vector can then be fed to the classifier, which will give a likelihood probability of the i-vector belonging to each class or condition. Bayes' rule can then be used including prior information to produce a posterior probability of the i-vector belonging to each class. These posteriors then directly tell us something about what nuisances may be present in the sample, but it is also possible to normalize the probabilities, such that the sum of the posteriors are 1. This results in the probability of the sample belonging to each condition according to the classifier used.

Including quality information can increase the discriminative strength of methods [FBPS12, NSRB15], as the assumption that all samples are comprised of noise free data of sufficient length to perform speaker subspace estimation, is usually wrong, especially in real world settings.

2.3 Artificial Neural Networks

Artificial neural networks were first introduced as a way to model neurons in the human brain. The first foundations for neural networks were laid in the 1940s [Bis95], and they have since gone in and out of fashion multiple times. Currently, deep learning methods are performing very well in almost all applications. Deep neural networks are artificial neural networks with at least two hidden layers, which are now made feasible because of available computing power and large training datasets.

Neural networks are interesting for score normalization as the evaluation of a sample through a network is typically quick, as well as it not requiring to store cohort information to normalize the scores as in section 2.2.5.

Here, artificial neural networks (ANNs) are introduced, followed by sections detailing neural network components, such as activation functions and weights, as well as a section on training a network to fit data, as well as a section concerning data and overfitting.

2.3.1 Introduction to neural networks

Neural networks attempts to fit the underlying function of observations.

The architecture or complexity of the neural network is determined by the units, or neurons, included in the network. Neurons are collected into layers, where the units of each layer are connected through the network. Each of these connections are weighted, and these weights determine the behaviour of the network. A small network can be seen in fig. 6.

There always exists two layers in a network. These two layers are the input layer, where observations in the form of different features are fed into the network, as well as the output layer, which is the target or function value of the underlying function. Inputs can be anything measurable from raw images to extracted features of objects, such as weight, gender, price, or location. The output can be the probability of the data point belonging to a specific class, or a number belonging to a regression problem. Between the input and output layer are other layers, called hidden layers, as these are not modified or interacted with directly by the user (in form of input or output). Each layer consists of multiple units, also called neurons. For the hidden layers these are called hidden units.

Each unit is connected to units of the next layer and the output of one unit is weighted. At a unit, the weighted outputs of previous units are then summed together and fed through an activation function. This activation function introduces nonlinearity to the network.

If the connections of the network only run forward from input towards output, the network is called a feed-forward network. In recurrent neural networks, data information is also allowed to flow backwards through the network. Other extensions exists as well, such as convolutional neural networks, where linear convolution filters are trained in each layer. In this thesis only feed-forward networks are used. See an overview of a feed-forward neural network in Figure 6.

The process of propagating input data through the network is called forward-propagation, while the network is trained using back-propagation, which is to propagate errors (defined by a cost function, such as mean square error or cross-entropy) back through the network. In order to optimize the weights of the network to fit the data best, a gradient descent based optimization algorithm is used. For the optimization algorithm, back-propagation is used to compute the gradient of the cost function with respect to the weights of the network. With the back-propagation method it is not necessarily the global minimum of the cost function that is found, but it does not matter in practical applications [LBH15].

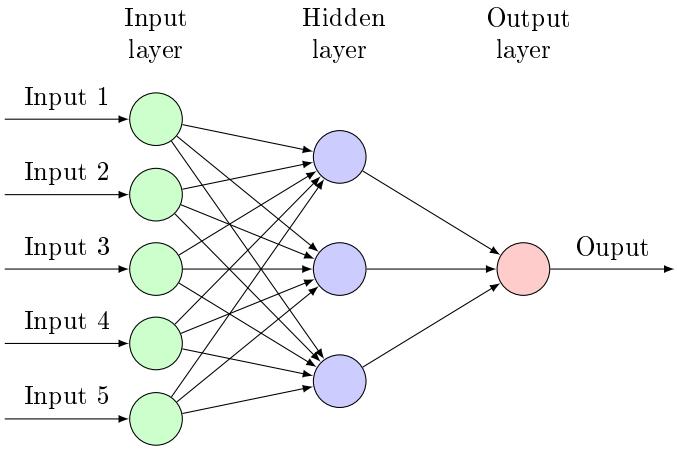


Figure 6: A simple feed-forward network.

2.3.2 Activation functions and weights

Going into closer detail of the neurons, the process of each neuron is seen in fig. 7.

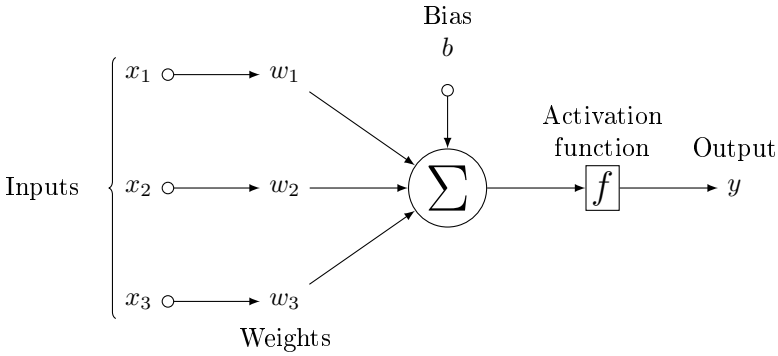


Figure 7: What happens in each neuron.

In the neuron the response is first computed, a linear combination of the output of previous neurons x_i and their weights in the network w_i as

$$r = \sum_{i=1}^{n} x_i \cdot w_i + b = \mathbf{w}^{\mathbf{T}}\mathbf{x} + b, \tag{2.28}$$

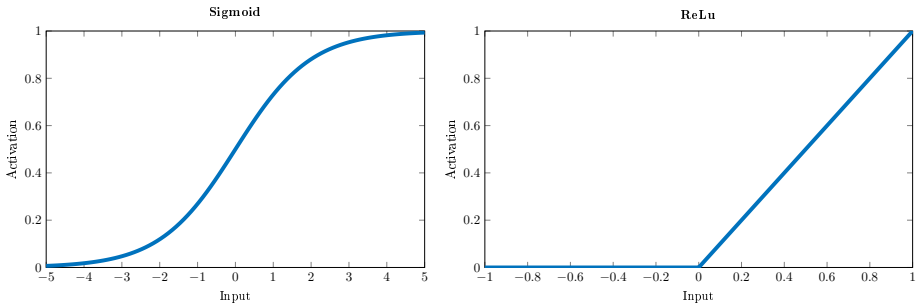


Figure 8: Activation functions.

with n being the number of incoming connections, and b a per neuron bias, introduced to be able to move the whole of the linear combination to one side or the other as input to the activation function. The input to the activation function is called the response, whereas the output is called the activation. The activation function is where nonlinearity is introduced to the network. Furthermore, the activation function has to be real-valued as well as differentiable, to be able to perform a gradient descent optimization algorithm. The activation function is a function determining whether or not the artificial neuron is active, or letting signal through. The most simple activation function showing on or off is the Heaviside step function, which is 0 if the input is below 0, and 1 otherwise. This is not a differentiable function however, and a nice and more smooth activation function is the sigmoid function given by

$$\text{sigmoid}(x) = \frac{1}{1 - \exp(-x)}. \tag{2.29}$$

Currently, the best performing and most widely used activation function is the linear rectifier unit, ReLU, and its extensions [MSM16, LBH15]. The ReLU function is not differentiable at $x = 0$, however the differential is here defined to be either 0 or 1 for practical purposes, in the Theano framework [The16], the derivative of the ReLU activation function is defined to be 1 at $x = 0$. In fig. 8 the sigmoid and ReLU activation functions are seen. The ReLU function is defined as

$$\text{ReLU}(x) = \max(0, x). \tag{2.30}$$

There are advantages and disadvantages of both sigmoid and ReLU units. Advantages of the sigmoid is that has a bounded output in $[0, 1]$, while ReLU on the other hand has outputs in $[0, \text{inf}]$, which makes (unbounded) regression possible. Using ReLU can therefore lead to exploding activations, which can lead to problems with overfitting or numerical problems. A risk of the sigmoid function

is that it has vanishing gradients – the further the input is from 0 in either direction, the smaller the gradient becomes, until it vanishes. This makes learning slower in the network, whereas learning using ReLU is faster, as the gradient is constant when the input is above zero, leading to learning in the network at all positive activations [Kar16]. Further advantages to the ReLU activation function are that it is quick to calculate, as only a comparison is needed, as well as it introducing sparseness in the network when the input is below zero. A problem of the ReLU is the so called death of neurons – if the learning rate is too high, the gradient through a neuron may be big enough to make all inputs to the neuron to be negative [Kar16]. This leads to all activations being zero – and therefore no more signal will pass through the neuron, nor the gradient be different. Therefore, the neuron will never activate. In order to alleviate this problem amongst striving for better performance, extensions to the ReLU activation has been proposed, first the Leaky ReLU (LReLU) [MHN13] which has a small activation, even when the units is not active. This was extended to the Parametric ReLU (PReLU) in [HZRS15] which is defined by

$$\text{PReLU}(x) = \begin{cases} \alpha x & \text{if} x < 0, \\ x & \text{otherwise}, \end{cases} \tag{2.31}$$

with α being a learnable parameter in the network. The PReLU is as such a scaling of the response below zero. Bias of the unit is already included in the neuron bias, which biases the response, and is included in the weights of the network.

2.3.2.1 Initialization of weights

Before training, the weights in the network has to be initialized. This initialization of weights in the network has an important role for the learning and therefore performance of the network, as large weights will result in large ReLU activationsas well as lead to vanishing gradients for sigmoid activations. However, too small weight initialization will lead to a very small gradient while training, as it is proportional to the weights. A compromise must then be made in some way. Initial weights in the network can reasonably be assumed to have a mean 0 and be distributed with equal amount negative and positive values [Kar16]. This is assuming that the input data is normalized around 0. If this is not the case, more effort will be spent on normalizing the input data through the first layer. Another important facet is that the weights should be different – if all initial weights are the same, the same change will propagate through the network, leading to a symmetric change in weights. In order to achieve asymmetric updates of weights, initial weights can be constructed from small random numbers, for instance from a normal $0.01\mathcal{N}(0, 1)$ or uniform distribution.

A problem with the normal distribution as initialization scheme is that the variance increases with the number of input signals or neurons. In [HZRS15], it is proposed to multiply each initial weights by $\sqrt{2 \cdot n_l}$, where n_l is the number of incoming connections to scale the distribution of initial weights while taking the ReLU into account. The argumentation is as follows as in [HZRS15]. The response of a unit in layer l is given by

$$\mathbf{y_l} = \mathbf{W_l}\mathbf{x_l} + \mathbf{b_l}, \tag{2.32}$$

where $\mathbf{x_l}$ is a n-by-1 dimensional vector with the inputs to the layer, $\mathbf{W_l}$ is a d-by-n matrix where d is the number of units in the layer, and each row of W are the weights of belonging to each unit. b_l holds the biases for the units. We have that

$$\mathbf{x_l} = f\left(\mathbf{y_{l-1}}\right), \tag{2.33}$$

with the activation function f. The elements of $\mathbf{W_l}$ are independently identically distributed (i.i.d), and it is assumed that the elements of $\mathbf{x_l}$ are also i.i.d, and that $\mathbf{W_l}$ and $\mathbf{x_l}$ are independent. Now y_l, w_l, and x_l can be written for the random variables of each element of the corresponding vectors and matrices. Then, with x_l having zero mean

$$\operatorname{Var}(y_l) = n_l \operatorname{Var}(w_l x_l) = n_l \operatorname{Var}(w_l) \operatorname{E}(x_l^2). \tag{2.34}$$

Let w_{l-1} belong to a symmetric distribution with mean zero, and $b_{l-1} = 0$, then y_{l-1} will belong to a symmetric distribution around zero as well, and using the definition of ReLU as being 0 when the response is equal or below zero, and equal to the response otherwise, as well as 2.33, giving $\operatorname{E}(x_l^2) = \frac{1}{2}\operatorname{Var}(y_{l-1})$. Inserting into 2.34 results in

$$\operatorname{Var}(y_l) = \frac{1}{2} n_l \operatorname{Var}(w_l) \operatorname{Var}(y_{l-1}). \tag{2.35}$$

Across L layers the variance become

$$\operatorname{Var}(y_L) = \operatorname{Var}(y_1) \prod_l = 1^L \frac{1}{2} n_l \operatorname{Var}(w_l). \tag{2.36}$$

As such, the signal is magnified or reduced with every layer with the initialization. It is therefore proper to have the variance of layer L to be equal to the variance of the input, as such

$$\frac{1}{2} n_l \operatorname{Var}(w_l) = 1 \forall l,$$

which can be achieved by initializing the weights of each layer with a standard deviation of $\sqrt{2/n_l}$.

2.3.3 Training a network

The process of fitting the network to input data is called training of the network. Neural networks are mostly used as supervised learning, which means that the real, measured output for a given input is known. So, the network is trained to fit the input data to the target values or classes in the best possible way. For the purpose of defining the best way to fit the network, the cost function is introduced. The cost function is also referred to as the objective function, depending on the author. The cost function measures the distance between the predicted value from inputs, and the actual target value. The goal of the training phase of a machine learning algorithm, here specifically neural networks, is to minimize this cost function.

One of the best known cost functions is the mean square error (MSE) function, which is used in many regression problems, given as

$$E_{\text{mse}}(\hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2\,, \tag{2.37}$$

with $\hat{\mathbf{y}}$ being the predicted targets, and $\mathbf{y}$ being the observed target values. The number of observations is given by n.

For classification problems, the MSE is not ideal however - there is no information about distinct classes included. For classification, the cross-entropy function is used. The MSE is derived [Bis95, Sec. 6.7] from the maximum likelihood principle by assuming that the target data follows a smooth deterministic function with Gaussian noise, whereas the binary cross-entropy function is derived assuming the target follows a Bernoulli distribution [Bis95] . The logistic or binary cross-entropy cost function is defined as

$$E_{\text{mce}}(\hat{\mathbf{y}}) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i))\,. \tag{2.38}$$

The cross-entropy is a measure of distance between the distribution of actual classes, and the probability distribution predicted by the neural network.

2.3.3.1 Backpropgation

For the purpose of updating the weights of the network according to data, a process called error back-propagation is used to compute a gradient with respect to the weights and bias parameters in a network. The computed gradient is then

used to perform a gradient descent optimization algorithm to minimize the cost function.

In order to derive the gradients [Bis95], the response eq. (2.32) is first rewritten to

$$r_j = \sum_i w_{ji} x_i, \tag{2.39}$$

where r_j is the response of unit j and x_i is inputs into this unit, weighted by w_{ji}. The bias parameter is here incorporated as an extra input fixed at 1, such that the bias is given by the corresponding weight. If We now use the differentiable activation function f and get the activation as

$$x_j = f(r_j). \tag{2.40}$$

From the outputs y of the network, the cost function is $E(\mathbf{y})$. Now, the interest is in finding the gradient of the cost in regards to the weights, which by the chain rule of derivatives is given as

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial r_j} \frac{\partial r_j}{\partial w_{ji}}. \tag{2.41}$$

The so called errors are introduced as

$$\delta_j = \frac{\partial E}{\partial r_j}. \tag{2.42}$$

eq. (2.39) results in

$$\frac{\partial r_j}{\partial w_{ji}} = x_i. \tag{2.43}$$

Substituting eqs. (2.42) and (2.43) into eq. (2.41) leads to

$$\frac{\partial E}{\partial w_{ji}} = \delta_j x_i, \tag{2.44}$$

which means that δ_j needs to be calculated for each unit in the network, and the gradient can be calculated. For output units, the error can be computed easily from the definition

$$\delta_k = \frac{\partial E}{\partial r_k} = f'(r_k) \frac{\partial E}{\partial y_k}. \tag{2.45}$$

The activation function usually has a nice analytical derivative, as does the cost function, and so the expression becomes easy to evaluate. For hidden units, the chain rule is used again and get the following expression

$$\delta_j = \frac{\partial E}{\partial r_j} = \sum_k \frac{\partial E}{\partial r_k} \frac{\partial r_k}{\partial r_j}, \tag{2.46}$$

where k are all the units, where unit j sends signal. Here, it is used that r_k is a function of r_j. That is, the activation of a unit varies only when the inputs to the units vary. By using eqs. (2.39), (2.40) and (2.42), the back-propagation formula is found as

$$\delta_j = f'(r_j) \sum_k w_{kj} \delta_k. \tag{2.47}$$

Getting the gradient of the cost function with respect to the weights of the network can therefore be computed by the following steps:

1. Perform forward propagation and get activations of all units in the network.
2. Calculate the error δ_k for all outputs using eq. (2.45).
3. Perform back-propagation of the error using eq. (2.47).
4. Get the derivative using eq. (2.42).

2.3.3.2 Gradient descent optimization.

We can optimize the weights in the network by using the gradient computed with back-propagation and perform a gradient descent based optimization method. The basic gradient descent is given by

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \alpha \nabla F(\mathbf{x}_{t-1}), \tag{2.48}$$

where t is the timestep, and F is the function to be minimized, here the cost function of the network. In gradient descent methods, the gradient is computed over the whole dataset, or batch, and then the parameter is updated. Another method is to perform stochastic gradient descent (SGD), where the parameter update is performed after each back-propagation of a data point. An advantage of the stochastic method is that it is a more noisy approximation to the true gradient of the function, and therefore can reduce the chance of landing in a local minima. Using the batch method however, can lead to faster and more stable descent into a minima, as an average gradient over many data points are used. A compromise between the batch and stochastic approach is to use mini-batches, where a small batch is used consisting of randomly chosen samples. Neural networks are trained on the dataset multiple times until convergence of the cost function, or until satisfaction of some stopping criteria. Each iteration over the whole dataset is called an epoch. Many extensions to the basic gradient descent method given here exist, one of the very well performing ones being the ADAM algorithm.

2.3.3.3 The ADAM optimization algorithm

The Adam optimization algorithm is a stochastic gradient-based optimization algorithm named after adaptive moment estimation. As focus is on high parameter spaces, higher order momentums are computationally expensive, the focus of the paper [KB14] is on using first order moments. The Adam algorithm is seen in Algorithm 1. In [KB14] the Adam algorithm performs at level or outperform other state of the art methods such as RMSprop and AdaGRAD.

Algorithm 1 The ADAM algorithm as seen in [KB14].

Require:

α ▷ Stepsize parameter.
$\beta_1, \beta_2 \in [0, 1)$ ▷ Exponential decay rates for moment estimates.
$f(\theta)$ ▷ Stochastic cost function with parameters θ
θ_0 ▷ Initial parameters.
1: $m_0 \leftarrow 0$ ▷ Initialize first moment vectors.
2: $v_0 \leftarrow 0$ ▷ Initialize second moment vectors.
3: $t \leftarrow 0$ ▷ Initialize timestep.
4: **while** θ_t not converged **do**
5: $t \leftarrow t + 1$
6: $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Get gradients at t
7: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) g_t$ ▷ Update first moment estimate.
8: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) g_t^2$ ▷ Update second moment estimate.
9: $\hat{m}_t \leftarrow m_t / (\beta_1^t)$ ▷ Bias correction.
10: $\hat{v}_t \leftarrow v_t / (\beta_2^t)$
11: $\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ ▷ Update parameter estimate.
return θ_t

In [KB14] recommended default values for parameters are given as $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. An optimization by changing the order of computations is also noted, by replacing line 9-11 of Algorithm 1 by $\alpha_t = \alpha\sqrt{1 - \beta_2^t}/(1 - \beta_1^t)$ and $\theta_t \leftarrow \theta_{t-1} - \alpha_t m_t / (\sqrt{v_t} + \epsilon)$. The ratio $\hat{m}_t / \sqrt{\hat{v}_t}$ decreases when there is a larger uncertainty of the direction of the first moment, which decreases the effective stepsize. This is usually the case near the optimum [KB14], leading to smaller step sizes closer to the optimum. The bias correction of the ADAM algorithm is included to correct for initializing the running average with zeroes.

2.3.3.4 Pretrained networks

Many networks are available on the internet, which have already been trained on a given dataset with a given model complexity. An example of this can be

a large convolutional neural network trained to classify objects in images or to perform face recognition. This way the network can be downloaded, and augmented with your own dataset (if preprocessing of images are the same) to reach much faster convergence of the network training, resulting in much more complex models being viable even with smaller datasets. In the case of normalization of speaker comparison scores, this is not a possible strategy, as there does not exist any pretrained networks on comparison scores and i- and q-vectors, or similar inputs. The augmenting of the network is made possible when the input to the neural network is of the same type as when it was first trained – for instance by using images of specific dimensions.

2.3.4 Data in machine learning

In machine learning in general, it is very important to keep a training and a test, or evaluation, dataset strictly separated. Training is performed on the training dataset, while the final performance of the system is evaluated on the test set. This way the system is evaluated on unseen data of the same type as the data used for training the system. The reason that performance should be evaluated on unseen data is that as data is used for training, it is possible for a model to overfit to the data. Overfitting means to fit to the data used for training very well, but generalize worse than other models may.

2.3.4.1 Overfitting

Overfitting is general problem with neural networks, as they can be arbitrarily complex, which means that special care and methods must be used to mitigate the risk of overfitting.
First and foremost there is a question about when to stop training the neural network. One possibility is to keep on training until the cost function does not decrease anymore on the training dataset, however this easily leads to overfitting - the network that fits the training dataset best, may not be the one that generalizes to unseen data best. A better method is to use a validation dataset, such that the network is trained on the training dataset, and the stopping criteria is to stop when there is no improvement in performance on the validation set. This way the network is chosen which best performs on a disjunct dataset- however the network is chosen, which performs best on the validation dataset – as such some care must be chosen to choose the validation dataset as to avoid implicit fitting to the validation dataset. This is usually done using cross-validation, where the dataset is split into multiple different sets of training and validation set. Cross-validation introduces a higher computational complexity

however, as for instance a 5-fold cross-validation (to split the data into 80% training data and 20% validation data) requires performing the training phase 5 times. The training phase is expensive using neural networks, and as such other methods are used to prevent overfitting, called regularization methods, of which some will be explained shortly. The validation split can also be performed by sampling randomly in the training dataset to perform the split between training and validation dataset.

Other issues regarding the dataset must also be considered. For a classification problem, a class bias can have large influences on the classifier – if a dataset has two classes, where 99% of the dataset is one class, the classifier will achieve very good performance by classifying all inputs to that class - while this may not be desired . Balancing the classes in the dataset is therefore usually done for a better performance of the classifier [Bis95, p. 45]. This can be done by weighting the classes in the cost function, such that the imbalances classes in total have the same amount of influence on the training of the classifier.
Furthermore, the data should be shuffled every epoch, unless the whole dataset is used as a batch before each gradient. This is done to avoid using a minibatch of highly correlated data (i.e. belonging to the same class), which will skew the approximated gradient direction in a different direction than the true gradient direction.
The classes or labels should also be distributed in the dataset, such that the network will not first train on one class, and then on another. Training on one class after each other will give problems with converging towards a solution that fits all classes equally well. When using a learning rate that decreases over time, where learning rate relates to the amount the gradient is traversed in the gradient descent method, the first training samples will have a larger effect on the training, as it is assumed that the parameters will be able to improve the most per training mini-batch in the beginning. If only one class is considered in the beginning of the training, this class will then have a larger impact on the solution than the classes coming later – where the learning rate may not be high enough to be able to correct for the initial biased gradient approximation.

2.3.4.2 Regularization

Regularization methods are used to help against overfitting by imposing constraints to the weights of the network, such as they are not able to vary wildly and easily overfit.

A regularization method is to add a regularization term in the cost function of the network, typically using L1 or L2 regularization, where L2. For L2 regular-

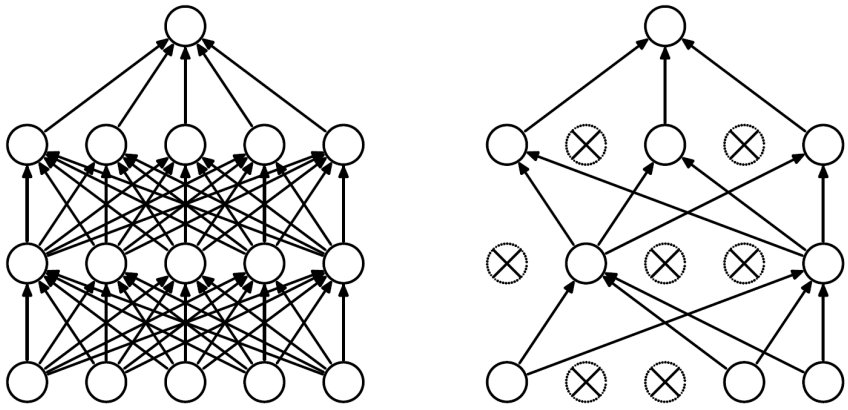
ization the cost function will become

$$E_{\mathrm{L2}}(\mathbf{y}, \mathbf{w}) = E(\mathbf{Y}) + \lambda \mathbf{w}^{\mathbf{T}} \mathbf{w}, \tag{2.49}$$

whereas L1 regularization uses the term $\lambda \sum_{i=1}^{n} |w_i|$, where λ is a regularization strength – which determines the influence of the regularization on the cost function compared to the term given by the outputs of the network. For L1 regularization the sum over all n weights in the network is used.

While L2 regularization penalizes large weights more than smaller weights, L1 regularization penalizes all weights equally. Using L1 therefore leads to sparsity in weights, while L2 regularization tends to lead to small, diffuse numbers. If the sparsity for feature selection is not needed, L2 usually performs better than L1 [Kar16].

A recent and effective method for reducing the risk of overfitting is called dropout [SHK+14, HSK+12]. When using dropout, connections are left out randomly during training with a probability of p each mini-batch. In the paper [SHK+14] a value of p between 0.2 and 0.5 is recommended. Using $p = 0.5$ for instance, half of the connections are used for training, while all connections are used for testing. When testing each Dropout functions as a way of ensemble learning – each mini-batch a slightly different network is trained, while testing functions as an average of all these smaller trained networks, as all connections are used. The dropout process can be seen in fig. 9.



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

Figure 9: A network without dropout, and the same network with dropout activated. Crossed out units are inactive. See [SHK+14].

A recent method is Batch normalization [IS15], where the responses of each layer are normalized for each mini-batch during training. According to [IS15], batch normalization makes it possible to use higher learning rates, as well as reducing the sensitivity of the learning process to initialization of weights. Furthermore, batch normalization functions as a regularization method, which can reduce the need of dropout. The batch normalization transform is seen in algorithm 2. Batch normalization adds two parameters, γ, β, to optimize per units where the batch normalization is performed, however, as the batch normalization method is differentiable [IS15], this optimization can be included in the back propagation and therefore also into the gradient descent optimization algorithm.

Algorithm 2 The batch normalization transform as seen in [IS15].

Require:

$\mathbf{x}$ ▷ Reponse values over a minibatch in a layer, size m

1: $\mu \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ ▷ Mean of mini-batch.

2: $\sigma^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$ ▷ Variance over mini-batch.

3: $\hat{x_i} \leftarrow \frac{x_i - \mu}{\sigma}$ ▷ Normalization.

4: $r_i \leftarrow \gamma\hat{x_i} + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$ ▷ Scale and shift response.

5: **return** $r_i = \mathrm{BN}_{\gamma,\beta}(x_i)$

2.3.5 Bayesian inference

Bayesian optimization is a method to optimize expensive black box functions in a Bayesian framework. In Bayesian optimization the objective function, f is modelled from pairs of input parameters, $\mathbf{x}$, and evaluated noisy function outputs, y, after which an acquisition function, u is used to predict the next best input parameters for objective function evaluation. The acquisition function has a tradeoff between exploitation and exploration to decide the region of search [GSA14]. Exploitation denotes the belief of the model that the objective function has low value, whereas exploration denotes parameter space regions where the model has high uncertainty.

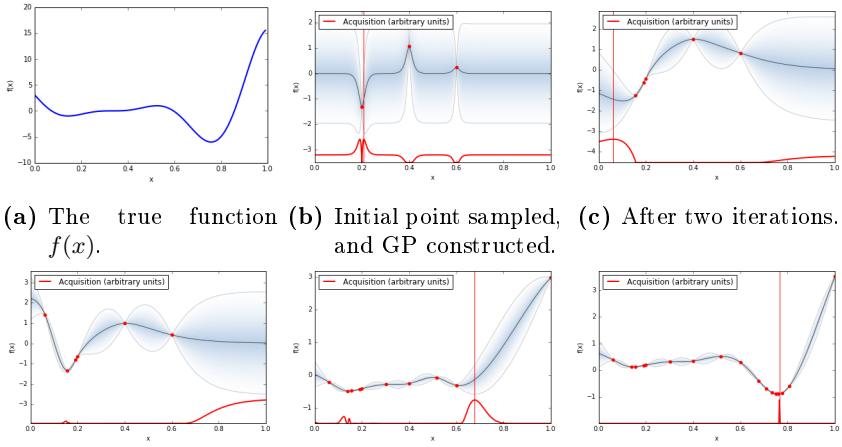
Typically, Gaussian processes are used to model or predict the objective function given data. A Gaussian process is an infinite dimensional multivariate Gaussian distribution, which can be interpreted as a function that at each point is given by a Gaussian distribution. A Gaussian process is defined by a mean and a variance function [SLA12]. These functions are defined from the hyperparameters of the Gaussian process, θ as well as previously acquired pairs of input parameters and output function values $\{\mathbf{x_n}, y_n\}$. The mean function is denoted $\mu(\mathbf{x}|\{\mathbf{x_n}, y_n\}, \theta)$ and the variance function $\sigma^2(\mathbf{x}|\{\mathbf{x_n}, y_n\}, \theta)$. Both will in the following be shortened to $\mu(\mathbf{x})$ and $\mu(\mathbf{x})$. The current best value is denoted

by $\mathbf{x}_{\text{best}} = \text{argmin}_{\mathbf{x_n}}(f(\mathbf{x_n}))$. In algorithm 3, the pseudo code for Bayesian optimization is seen as described in [BCDF10].

Algorithm 3 The Bayesian optimization algorithm as formulated in [BCDF10].

1: **for** $t = 1, 2, \ldots$ **do**
2: Find evaluation point $\mathbf{x_t}$ by optimizing the acquisition function over the Gaussian process: $\mathbf{x_t} = \text{argmin}_{\mathbf{x}}\, u(\mathbf{x}|\mathcal{D}_{1:t-1})$.
3: Evaluate objective function: $y_t = f(\mathbf{x_t}) + \varepsilon_t$.
4: Update Gaussian process with newly acquired data points.

In fig. 10, an example of a Bayesian optimization of a function is seen.



(a) The true function $f(x)$.

(b) Initial point sampled, and GP constructed.

(c) After two iterations.

(d) After three iterations.

(e) After seven iterations.

(f) After 15 iterations.

Figure 10: A 1D example of the Bayesian optimization approach, using an example from [aut16]. The acquisition function used is the expected improvement.

Different acquisition function exist. Four popular or well known choices are:

- **Probability of improvement:** Choose the next $\mathbf{x_t}$ by maximizing the probability of improvement in objective function over the current best value [SLA12, BCDF10]. The probability of improvement is given by

$$\mathrm{PI}(\mathbf{x}) = P(f(\mathbf{x} \leq f(\mathbf{x}_{\mathrm{best}}))) = \Phi(z(\mathbf{x})), \tag{2.50}$$

  with

$$z(\mathbf{x}) = \frac{f(\mathbf{x}_{\mathrm{best}}) - \mu(\mathbf{x}}{\sigma(\mathbf{x})},$$

  and Φ being the standard normal cumulative distribution function (CDF).

- **Expected improvement:** Probability of improvement is almost purely exploitation by not taking the amount of improvement possible into account – only the probability of an improvement happening. A different approach is to choose the next evaluation point by the best expected improvement. The expected improvement is the most popular acquisition function [EFH+13] as it is better behaved than probability of improvement, and does not need parameter tuning [SLA12]. The probability of

improvement can be evaluated analytically [GSA14]:

$$\begin{aligned} \mathrm{EI}(\mathbf{x}) &= \mathbb{E}(\max(0, f(\mathbf{x}_{\mathrm{best}})) - f(\mathbf{x})) && (2.51) \\ &= \sigma(\mathbf{x})(z(\mathbf{x})\Phi(z(\mathbf{x})) + \phi(z(\mathbf{x}))), && (2.52) \end{aligned}$$

with ϕ being the standard normal PDF.

- **Lower confidence bound:** It is possible to use the acquisition function using a lower bound on the function values:

$$\mathrm{LCB}(\mathbf{x}) = \mu(\mathbf{x}) - \kappa\ \sigma(\mathbf{x}), \tag{2.53}$$

with $\kappa \geq 0$.

- **Entropy search:** While the other acquisition functions look towards the function value, entropy search attempts to minimize the uncertainty of the location of optimal function value, $\mathbf{x}_*$ [GSA14]. This is done by choosing the next parameter step to minimize the entropy:

$$\mathrm{ES}(\mathbf{x}) = H(\mathbf{x}_*|\mathcal{D}) - H(\mathbf{x}_*|\mathcal{D}, \mathbf{x_t}, y_t), \tag{2.54}$$

where H is an entropy function. This function is not easy to evaluate in practise, as the probability distribution $p(\mathbf{x}_*|\mathcal{D})$ has no closed form [GSA14].

Impacts of the choice of acquisition function are not studied in this work due to time constraints, and Bayesian optimization will be performed solely by employing the expected improvement acquisition function.

Chapter 3

Quality informed score normalization utilizing neural networks

In this thesis it is proposed to use q-vectors, see section 2.2.6, in conjunction with i-vectors of reference and probe to perform a quality-informed score normalization. This chapter is composed of two sections, one giving a motivation for the method, and the second describing the configuration of the neural network.

3.1 Motivation

Degradation in sample completeness and sample quality is an issue for every biometric modality, but is especially easy to pick up for speaker recognition, as other sources in the vicinity produce sounds, which are harder to isolate, as sound waves propagate through the air. It is hard to construct a practical isolated speaker recognition system that does not allow environmental noise to be picked up, and it is therefore important to counteract the effects of nuisance factors, such as noise from the phone signal, the air conditioner in the office (AC), or someone speaking in the background (crowd). Crowd noise is especially challenging, as the noise contains biometric information from other subjects,

Condition	EER	C_{llr}^{min}
Clean, full duration	0.47%	0.022
Clean, 5 seconds	2.93%	0.115
0db AC, full duration	3.28%	0.121
0db Crowd, full duration	3.37%	0.140
0db Crowd, 5 seconds	23.91%	0.674

Table 1: The performance degradation of the PLDA algorithm for different duration and noise nuisance conditions. 0db AC is a 0db SNR of the signal versus air condition noise. Full duration is above 60 seconds. The PLDA algorithm is the baseline used in this thesis, see section 4.1.4.

which need to be distinguished between.
In addition to noise sources, the duration of the recorded signal is another important nuisance factor. Shorter durations do not give sufficient data to extract i-vector statistics satisfactorily. Performing well on short durations is attractive, as the subjects processed by the speaker recognition system are often not likely to want to give a long voice sample – for instance, the subject is rarely interested in speaking for a long time in the situation of phone banking, and would rather have the interaction finished as quick as possible. Forcing an individual to speak for a minute to unlock their smartphone is also inconvenient. Short duration samples are also an issue when there is not a cooperate enrolment – i.e. the subject does not want to be enrolled into the system. This could for instance be in a forensic scenario, where the suspect is unlikely to have supplied long durations of clean enrolment samples.

Nuisance factors such as noise and short duration decreases performance of the system, see table 1 for examples on the performance degradation for different quality reductions. In the most difficult situation with a SNR of 0 with crowd noise, as well as a 5 second duration, the algorithm performs more than 50 times worse in equal error rate than in the clean and full duration case, and the optimal performance of the system C_{llr}^{min} is more than 30 times worse. Being able to improve the performance in noisy conditions by a low relative amount therefore results in large absolute gains as well.

Current methods for score normalization focus on using cohort information directly to perform a score normalization, see section 2.2.5. The proposed method of using neural networks uses cohort information less directly, as the cohort information is used to train the network, but not needed to be stored or used for direct comparisons at the time of normalization. By not having to compare probes and references to cohort sets, the privacy issues of storing and using biometric data of other subjects are circumvented. Another motivation for using

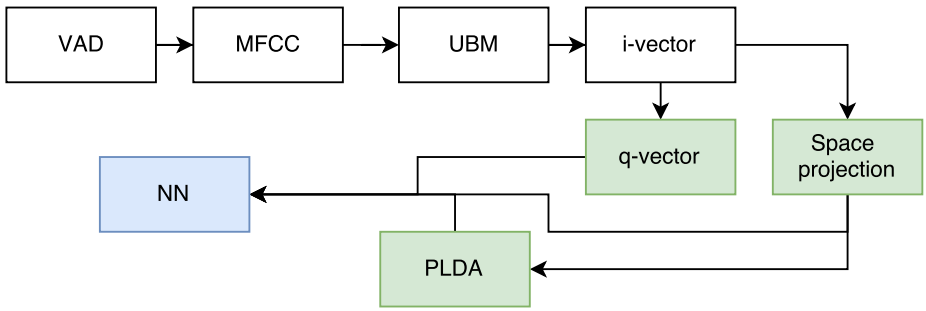


Figure 11: The elements in the well established processing chain in front of the proposed neural network (NN, marked blue) showing the inputs to the model in green.

neural networks is that the high computational cost of comparing each reference and probe to a set of cohorts is avoided, and substituted by the forward propagation in a neural network instead.

Neural networks have been used before in speaker recognition, both in the frontend for computing sufficient Baum-Welch statistics for i-vector extraction, as well as by using bottleneck features [YPS12]. In the backend process, a PLDA-RBM model (RBM being restricted Boltzmann machine) [NHS+16, SKSD12] has been used as an alternative to using PLDA for computing the comparison scores. Neural networks should therefore be able to perform score normalization, as well.

Even though the PLDA algorithm is here used for computing the scores, the proposed method does not directly use the PLDA algorithm, and therefore allows the use of any score computation algorithm in its place. The examination of neural networks for quality-informed score normalization is here using a fixed and well established processing chain that is assumed to be optimal to ensure a fair comparative setup. The processing chain is seen in 11. Perfect VAD is assumed, as VAD, especially energy based ones, perform badly in very noisy conditions [NBB16]. The processing chain is fixed and uses the same datasets for training when examining the robustness to noise, as the interest here lies in the sensitivity of the neural network to different quality degradation conditions, and not in the performance of the previous processing steps.

3.2 Neural network configuration

The neural networks studied in this work have common configurations, which are detailed here.

The inputs to the neural network are feature vectors consisting of $2 \cdot \#\text{ivecdims} + 2 \cdot \#\text{qvecdims} + 1$ elements, where #ivecdims is the number dimensions in the i-vector, and #ivecdims the number of dimensions in the q-vector. The feature vector consists of a concatenation of:

- **PLDA comparison score:** The comparison score to normalize. The comparison score is an unbounded scalar number.
- **Reference i-vector:** #ivecdims elements.
- **Reference q-vector:** One element for each condition in the dataset, #qvecdims in total.
- **Probe i-vector:** #ivecdims elements.
- **Probe q-vector:** #qvecdims elements.

The output of the neural network is a new comparison score, which uses the PLDA score as well as the information comprised in the i-vectors and the quality information from the quality vectors. The targets are 1 for a genuine comparison and 0 for an imposter comparison, leading to a binary classification problem.

The PLDA score is fed into the network and therefore, will an identity mapping of the PLDA score in the network lead to equal performance of the PLDA comparator. Reference and probe i-vector are included in the network, as these may contain discriminative information not explained by the PLDA algorithm, as well as, primarily for the probe, including the noise information that is sought counteracted – and therefore information about the noise as well. The q-vector of the probe includes information about which type(s) of quality reduction are present in the probe, which makes condition specific normalization possible. The reference q-vectors are included to account for different conditions during enrolment, for instance when cooperative enrolment is not possible. The inputs to the neural network are seen in fig. 11 in green.

A common configuration of all the neural networks is that within each neural network, each hidden layer has the same amount of hidden units, U.

Each network has a hidden layer of U units with linear activation functions as well directly after the input. This linear layer acts as a linear normalization

of the inputs, as well as being able to explain the linearity of the underlying function nonlinear, while the proceeding non-linear hidden layers can fit the non-linearity of the function. A downside of adding the linear layer is that it adds complexity and more weights to the network, which need to be trained.
In every, but the first, hidden layer, the ReLU activation function is applied. The output layer uses a sigmoid activation function to limit values between 0 and 1 for giving a score easily comparable to the binary classes. An unbounded comparison score is also possible like the PLDA score, in which case a linear activation function can be used. A ReLU activation in the output layer is a worse choice, as it is only unbounded in the positive domain, and there is no clear center of the output function which can denote an equal likelihood of both classes, leading to a regression problem with a lower bound. The weight initialization scheme for each layer is the scheme proposed by He, as described in 2.3.2.1.

Computationally, it is not possible to store all comparisons in RAM due to the amount of comparisons and the size of the feature vector per comparison, and impractical to store them on disk. In order to overcome this problem, the i-vector and q-vector for each reference and probe is concatenated, and loaded into RAM with the PLDA comparison scores
Then, each comparison is computed for each mini-batch by concatenating reference and probe with the corresponding PLDA comparison score into a full input feature vector. Each mini-batch consists of the comparisons between one probe and all the references, i.e 680 comparisons, refer to section 4.1.1.

As not all probes have a matching enrolled subject in the reference set, the usage of one probe for each mini-batch leads to not all mini-batches containing a genuine comparison. However, as the order of the probes are shuffled for each epoch of the training, it is assumed that this should not have an effect on the training of the network.
Given the nature of the comparisons, there is a relatively higher number of imposter comparisons compared to the amount of genuine comparisons. For the purpose of addressing class imbalance, the genuine comparisons are therefore weighted by a relatively high factor compared to the imposters, to achieve an effective class balance of 50% belonging to each class. The class balancing leads to mini-batches with a genuine score having a larger influence on the learning than a mini-batch with only imposters, however due to the shuffling of the order of the probes and the amount of probes, the network is effectively trained towards a scenario of equal genuine/imposter priors, representing that no prior information about the classes is available on comparison.

The training of the network is performed using the binary cross-entropy cost function, as the classification problem examined is binary. The ADAM algorithm is used for optimizing the weights in the network. The training process

is terminated after a maximum number of epochs, or if the stopping criteria is reached, which is to stop training if a validation error has not improved for a certain amount epochs. In the second case, only the best performing model is saved, i.e. the 4th to last model. The validation set is taken by sampling randomly from the development set and hold them out from the rest of the training. Each time a network is trained, a new random validation set is sampled.

By adding regularization to the network, it is expected to generalize better, i.e. perform slightly worse on development set, but better on the evaluation set or in unknown conditions. By adding L2 regularization and dropout, it is therefore expected that the network achieves better robustness.

While the networks examined here have common traits, they differ in size as well. It is expected that deeper networks with more hidden units are able to describe the data better, and therefore give a better performance.

It is possible to choose well performing hyperparameters, which are the parameters defining the neural network, such as amount of layers and units per layer, by empirical tuning. However, by feeding the performance and hyperparameters to a Bayesian optimization algorithm, a candidate for the optimum hyperparameters in the Bayesian framework can be achieved. The expectation is that the performance as a function of the hyperparameters of the networks achieves a relative smooth function with an uncertainty induced by the random factor of the initialization of the network, the shuffling of the input data, and the back propagation algorithm.

As the neural networks have an equal amount of hidden units in each hidden layer, the network configuration can be denoted shorter by layers, L, and units, U: (L, U), in addition to any extra information about regularization. An exemplary network with two hidden layers in addition to the first linear layer and 100 hidden units in each layer can be written as (2, 100), and is depicted in fig. 12.

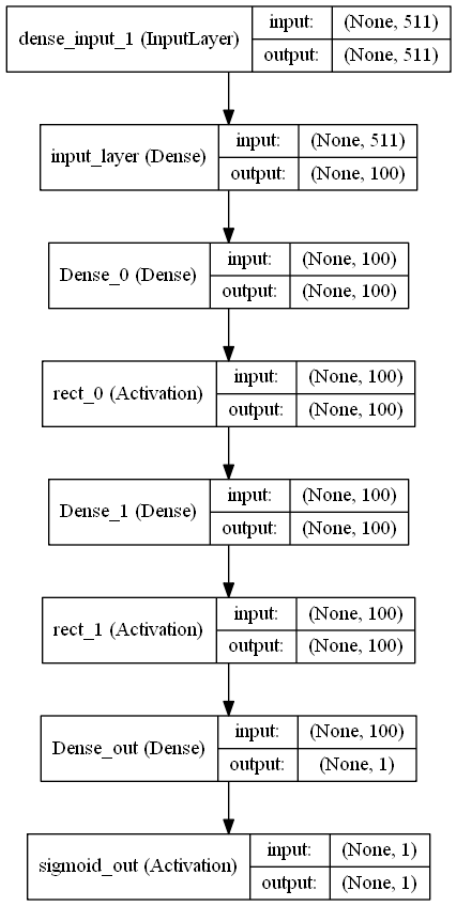


Figure 12: A (2, 100) neural network configuration. First column is the layer name and type, second column is the input and output dimensions. The None python datatype exists due to the interface with the Keras model object.

Chapter 4

Experiments

As motivated in chapter 1 and further detailed in section 3.1, using neural networks is an attractive approach for normalizing comparison scores. In order to be able support or falsify the research questions stated in section 1.2, experiments are introduced in section 4.1. The results of the experiments are presented and discussed in section 4.2 and summarized in section 4.3.

4.1 Experimental setup

In this section, the data used in this work is detailed, and experiments are formulated to be examined for each research question. The hard- and software used to perform the experiments are described as well.

4.1.1 Data

As in [NSRB15], the data used is the I4U file list for NIST SRE'12 [SLK+13] with induced nuisance factors. The dataset consists of data truncated to 5s, 10s, 20s, 40s, and full duration (of above 60 seconds), each being subjected

to air conditioner (AC) noise or crowd noise of 0dB, 5dB, 10dB, 15dB, 20dB. Conditions are present for no noise and duration truncation only, as well. In total 55 conditions are used of reduced duration and with signal degradation. Crowd noise is a different kind of noise compared to AC noise, as it has biometric information included. The different combinations of sample incompleteness and signal degradation will henceforth be denoted as noise conditions as well, even though some noise conditions include only duration truncation.
An overview of the noise and duration conditions are seen in table 2. The data used is only from male subjects.
The i-vectors are, as in [NSRB15], processed dependent on the noise condition by performing LDA projection from 400 to 200 dimensions, WCCN, and length normalization.

Condition	1	2	3	4	5	6	7	8	9	10	11 ... 30	31 ... 55
Duration	5 s	10 s	20 s	40 s	full	5 s					10 s ... full	5 s ... full
Noise	clean					AC						CROWD
SNR						0 dB	5 dB	10 dB	15 dB	20 dB	0 dB ... 20 dB	0 dB ... 20 dB

Table 2: Label scheme for combined duration and noise conditions, see [NSRB15].

The full duration conditions are from recordings of 60 seconds or longer duration. In the I4U dataset, some i-vectors belonging to the 40 second duration conditions have been wrongly included as full duration.

In table 3 the number of i-vectors in each dataset is depicted. Not all probes have an enrolled counterpart in the reference set, however for verification this is not relevant, and does not have any influence on the training of the proposed neural networks, other than adding robustness towards unknown subjects. The probes are divided equally between each of the 55 conditions.

A held out background dataset is used for training the LDA and WCCN parameters used to perform space projection on i-vectors in both the development and evaluation datasets.

4.1.2 Experiments

For the purpose of answering the research questions formulated in section 1.2, experiments are set up on a per question basis.

Dataset	#ref i-vecs	#probe i-vecs	#unknown	#comp.	#genuine comp.
Dev	680	357,269	93,434	242,942,920	263,835
Eval	723	388,278	114,356	280,724,994	273,922
Backgr.	–	300,256	–	–	–

Table 3: The number of reference and probe i-vectors included in the development and evaluation dataset, as well as the number of unknown probes, i.e. probes that do not have a corresponding reference, as well as the number of comparisons and the number of genuine comparisons. The number of i-vectors in the hold out dataset is also shown.

4.1.2.1 Performance of a simple network

To examine whether a simple two layer network can achieve better performance than the baseline, a network of two hidden layers of 100 hidden units each is trained on the data as the baseline neural network. Furthermore, regularization is used, with the expectation that the performance with regularization improves on the evaluation dataset.

For regularization, the (2, 100) network will be trained using L2 parameters $\lambda \in \{0.1, 0.01, 0.001, 0.0001, 0.00001\}$ to give an impression of the performance in different orders of magnitude of the regularization parameter. Batch normalization will furthermore be examined, as well as the best performing L2 regularization in conjunction with batch normalization. The L2 regularization will be performed on every weight in the network, except the bias weights in each hidden units, whereas the batch normalization is implemented by adding a batch normalization layer between the response and activation in each hidden layer, as recommended in [IS15].

To achieve class balance, the genuine comparisons are weighted by a factor of 920.81. The stopping criteria of the network are to stop training after either 30 epochs, or after there has been no improvement on a 20% validation set for 3 epochs. These stopping criteria are fixed for all experiments.

The success criteria of the research question is to achieve a better performance than the baseline, which is achieved by an identity mapping of the PLDA score, which is the first element of the input feature vector.

4.1.2.2 Gain of deeper architecture

To investigate the gain in performance obtained by using deeper neural networks, the best performing, of the tested, regularization schemes is chosen, and different networks with different number of hidden layers and number of hidden units are trained and evaluated. By using the (2, 100) network as starting point, combinations of halving and doubling the parameters are investigated. The combinations consist of one, two, and four hidden layers with 50, 100, and 200 hidden units.

The least complex network is therefore a (1, 50) network, whereas the deepest is (4, 200). The expectation is that the difference obtained with both an increase and a decrease in the parameters can be inferred from the halving and doubling of parameters. The expectation is that deeper networks outperform the shallower networks, as the deeper architectures have more degrees of freedom by having more parameters to tune.

When the most promising complexity has been found, it is investigated whether adding a 20% dropout improves performance or not. The 20% is chosen as it is the lower bound recommended in [SHK+14]. The dropout layer is added only for hidden layers – not for the input layer. Again, the assumption is that dropout improves robustness to noise and new conditions.

The success criteria is that deeper architectures has a relative performance gain over the simple (2, 100) network by 5%.

4.1.2.3 Confirmation by Bayesian optimization

The arbitrarily chosen halving and doubling of network parameters has a disadvantage in the choices of parameters being neat, ordered numbers. There is a risk that a better choice of parameters lies a slight distance from the chosen parameters. By using Bayesian optimization, the parameter space can be explored to find the optimum in a fashion described by the acquisition function. Here the expected improvement acquisition function is used, as it is one of the popular choices for acquisition functions, as well as being the default in the GPyOpt framework.
For confirming the parameters of the network, Bayesian optimization will be used for the parameters:

- Number of hidden layers. The number of layers is bounded to between one and 8 layers to limit the search space somewhat of the algorithm.

- Number of hidden units per layer. These are limited to a range from 10 units per hidden layer to 400.

The regularization scheme is fixed from the previously found best candidate.

The success criteria is that the optimal model complexity found by Bayesian optimization is within 5% of the model parameters of the best candidate found by the empirical network tuning performed for the previous research question.

4.1.2.4 Sensitivity towards unknown conditions

To investigate the robustness to unknown noise conditions, the model candidates from the previous experiments are trained on subsets of the dataset. The two types of unknown conditions studied are:

- **Unknown noise type** – Examined by excluding crowd noise conditions from the training dataset.
- **Good quality [NSRB16]** – Examined by excluding worse conditions of duration 5 and 10 seconds, and noise levels of 0dB and 5dB.

The success criteria in both cases is to perform to within 20% of the models trained using all noise conditions when excluding noise conditions to simulate unknown conditions.

As explained in chapter 3 the processing chain seen in fig. 11 is fixed. As such, even when investigating sensitivity to unknown conditions by excluding conditions from the dataset, the conditions are still present in the processing chain, as the i-vectors, q-vectors and the space projections all are trained on all data. However, performing the processing steps before the score normalization on the subsets of data does not ensure a fair comparison and examination of the robustness of the neural network approach, as the robustness of the method in this case depends on the robustness of all the earlier processing steps.

The training datasets for the two unknown types of signal degradation are reduced in size compared to training on the full dataset, to 30 conditions when examining noise type sensitivity, and to 21 conditions when evaluating robustness to worse conditions. As such, the training dataset is less than half the size when investigating the robustness to worse noise conditions. It is assumed that the datasets used are still large enough for proper training of the neural networks.

4.1.3 Performance comparison between models

The success criteria formulated for each research question all relate to a comparative performance, either compared to the baseline performance, or the performance of the model trained on a different data set. For this purpose we define the relative change in C_{llr}^{min} compared to another model in percent to be ΔC_{llr}^{min}. A ΔC_{llr}^{min} below zero means that the model compares favourably by having a lower C_{llr}^{min} than the model compared to. $\Delta C_{llr}^{min} > 0$ means a higher C_{llr}^{min} and therefore a worse optimal performance than the model which is compared to.
The average ΔC_{llr}^{min} over all conditions is used as a metric to decide the most promising models, as this number shows the average relative improvement of the model on all conditions, leading to a direct number for comparing models. The average ΔC_{llr}^{min} does not take into account that some models may perform better on certain conditions – if any specific conditions are weighted higher, it is possible to weight them both during the training of the network to ensure better performance on these conditions, or by weighting the metric in the average to ensure a biased decision.

C_{llr}^{min} is used to determine which model has the most promising performance, as the metric denotes the optimal performance of the classifier if perfectly calibrated. C_{llr}^{min} informs about the total discriminative power of the classifier, which is relevant for the examination of the optimal performance of the classifier. The best model candidates are therefore chosen as the ones with the lowest average ΔC_{llr}^{min}.

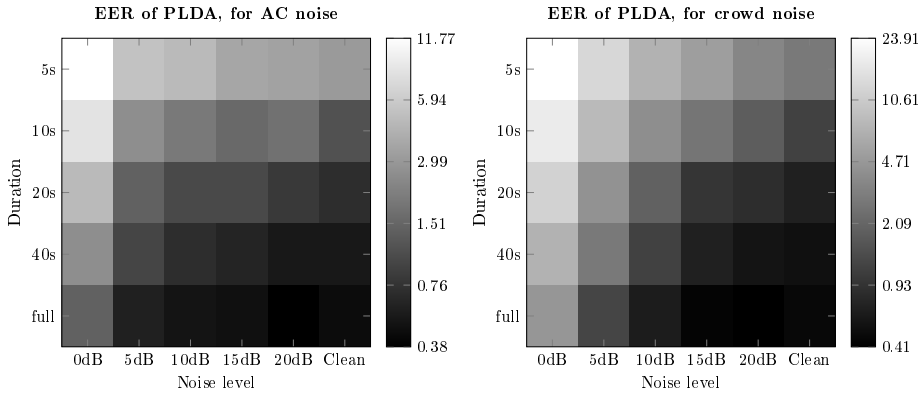
The metric C_{llr}^{min} is computed over a set of log likelihood scores and evaluates the system performance on this set of log likelihood scores. For the purpose of being able to compare and evaluate robustness of the neural network approach for unseen conditions, it is necessary to compute metrics on a per condition basis. This however measures the performance of disjoint systems for each noise condition, and not for one system for the pooled noise conditions. The performance on the pooled conditions is reported for completeness, as the neural networks are trained on the pooled data.

4.1.4 Baseline algorithm

Originally, the intended baseline algorithm was the quality-informed AS normalization algorithm described in [NSRB15], which would allow a direct comparison of quality-informed neural network score normalization to quality-informed cohort normalization in regards to improvement over the basic PLDA algorithm.

However, when attempting to verify and achieve the data results from [NSRB15] it proved that the basic PLDA comparison scores were improved compared to what reported, to having basically equal performance to the AS-normalization approach. The AS-normalization algorithm has been run again with the same dataset, same codebase, and same hardware as used in [NSRB15]. The only difference is the Matlab version used to run the code, where the algorithm was originally run on Matlab 2014a, and the attempt to reproduce the results was run on Matlab 2016b. Further investigation into why the PLDA scores suddenly have improved performance have not been carried out due to time constraints.

The baseline algorithm used is therefore the PLDA approach on the processed i-vectors, using the results from the re-computation. As already noted in section 3.1, the performance of the PLDA algorithm degrades on noisy data. The degradation of the performance of the PLDA algorithm is depicted in fig. 13 for the different noise levels and durations. There is a gradient of performance degradation regarding both duration and noise level, and the PLDA performs worse overall on the crowd noise type compared to AC noise.



(a) Performance of PLDA as duration versus AC noise level.

(b) Performance of PLDA as duration versus crowd noise level.

Figure 13: The EER of the PLDA algorithm depicted for duration versus noise levels for the two types of noise. Note the intensity map is logarithmic and different for the two plots. Lower intensity or darker colour is better.

4.1.5 Hardware and software frameworks

All computations are carried out on a machine with 512 GB of RAM and two Intel Xeon E5-2698 v3 CPUs running at 2.30GHz. Each CPU has 16 cores with

32 threads total. GPU acceleration is to be desired when training neural networks, but the machines available have too old GPUs to install current versions of the CUDA 8.0 and CuDNN 5.1 frameworks. The GPUs do not have compute capability higher than 2.0, and does therefore not fully support neural network frameworks of theano or Tensorflow.

Matlab 2016b and Python 3.5.2 were both used.
Matlab was used in conjunction with the Bosaris toolkit[BdV13] to calculate metrics. The baseline PLDA algorithm was furthermore implemented in Matlab. Python was used to train and evaluate the neural networks using the Keras [Cho16] v. 1.1.1 framework as an abstraction layer to theano v. 0.9.0.dev1 [The16]. The theano version used is a version optimized by engineers at Intel to perform better on the Intel Xeon processors.

For the Bayesian optimization, the GPyOpt v. 1.0.3 Python framework is used [aut16]. GPyOpt is based on Gaussian processes. It also supports Python 3.

4.2 Evaluation results

The results from each experiment as described in section 4.1.2 are reported and discussed in this section with the same structure as in section 4.1.2.

4.2.1 Performance of a simple network

The (2, 100) neural network without regularization performs close to random chance, with an EER of close to 50% and a $C_{\text{llr}}^{\text{min}}$ of close to 1 in all conditions, which shows that the network does not learn anything about the data, even though an identity mapping of the first input leads to baseline performance. The weights corresponding to the connections from the input layer to the first linear layer of the network are depicted in fig. 14.
A possible explanation for the performance of the network is exploding activations of the ReLU units, or the phenomena of dead units, where a large gradient early in the training leads to a neuron never firing (always evaluating to 0), from where there is no possibility of a gradient passing through and learning or activating the unit. Merit to this hypothesis is that the weights in the network are of large magnitudes.
There are large magnitudes of the weights associated with multiple conditions of the reference q-vector features, even though these should have nonzero values only in the clean, full duration condition, as well as the clean 40s condition (due

to some of the vectors being included in the wrong condition, see section 4.1.1). The reference q-vector elements belonging to other conditions may have low non-zero values as well, due to the tails of the GMMs used to model the qualities. One possibility of the bad performance is that the network fits only to noise, or the utmost tails of the GMM used for q-vector training, and therefore has a performance breakdown. Another plausible explanation for the large magnitude of weights leading from these inputs into the network is that the values of these inputs are 0 for the whole dataset, and therefore the gradient of the cost function in regards to these weights is ill defined, or zero, as the cost function is completely invariant to these inputs. The pattern of the weights of the probe q-vectors is desired to be nonzero, as it is here the quality information of the probe lies. In this case there is a very symmetric pattern, which is unexpected, but plausible, as there are an equal amount of probes belonging to each of the 55 conditions.

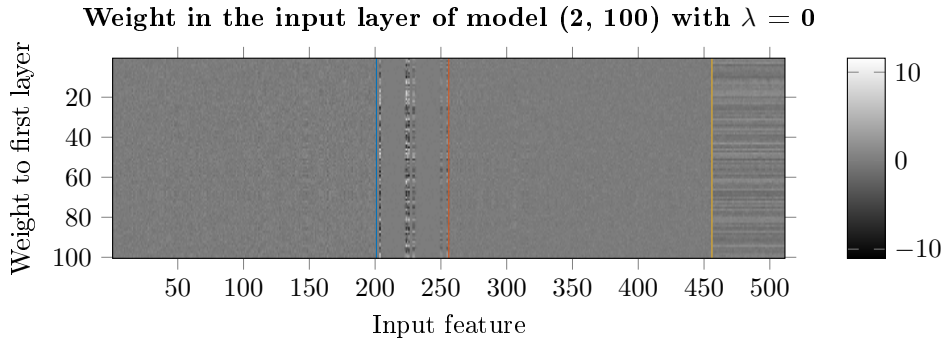


Figure 14: The weights belonging to the first linear hidden layer, i.e. the weighting of the inputs. The weights are shown for each connection of input feature on the first axis to each unit of the first hidden layer on the second axis. First column corresponds to the PLDA score. The blue vertical line denotes the transition from reference i-vector to reference q-vector, the red line the transition from reference q-vector to probe i-vector, and the yellow line denotes the transition from probe i-vector to q-vector.

Performing regularization is expected to improve the performance of the network, as weights are forced towards zero and large activations are avoided. In table 4 the average improvement in C_{llr}^{min} over the baseline is shown for different regularization schemes. For $\lambda = 0.1$ it is seen that the performance follows the baseline completely, and as the value of the regularization parameter decreases, the performance improves for all noise types.
From the average ΔC_{llr}^{min} over all conditions, the most promising regularization scheme is L2 regularization with $\lambda = 0.00001$ with $\Delta C_{llr}^{min} = -5.68$. The batch

normalization regularization approach does not on average perform better than the baseline, which is unexpected as the method achieves good performance in the introduced paper [IS15].

For completeness, combinations of batch normalization and L2 regularization has been examined as well to investigate whether any positive interactions between the two methods exist. While the performance is improved compared to using batch normalization alone, the performance is still worse than using L2 regularization exclusively. The average $\Delta C_{\text{llr}}^{\text{min}}$ compared to baseline over all conditions is for batch normalization 18.43%, while it in combination with $\lambda = 0.0001$ L2 regularization is 1.19% and with $\lambda = 0.00001$ L2 regularization 5.10%, while L2 regularization with $\lambda = 0.00001$ alone results in -5.68%. It is noted that the interplay between the schemes is not simple; batch normalization performs worse with the most promising choice of L2 parameter than with the second most promising choice of L2 regularization.

	$\lambda = 0.00001$	$\lambda = 0.0001$	$\lambda = 0.001$	$\lambda = 0.01$	$\lambda = 0.1$	BN
Clean	**-5.32**	0.16	-0.127	-0.13	0.01	28.68
AC	**-4.89**	-1.11	-0.27	-0.10	0.00	20.11
Crowd	**-6.55**	-3.11	-0.37	-0.09	0.00	14.70
All	**-5.68**	-1.90	-0.30	-0.10	0.00	18.43

Table 4: Average $\Delta C_{\text{llr}}^{\text{min}}$ compared to baseline for each noise type and L2 regularization parameter λ. Results shown for different regularization schemes on a (2, 100) model. BN is batch normalization.

In fig. 15a the sum of absolute weights moving outwards of each of the input units of the network is shown for $\lambda = 0.1$. The sum of weights is high for the first input unit, which is the PLDA score, and close to zero for every other input unit. This network therefore performs an identity mapping of the PLDA.

The weights from the input layer to the first linear hidden layer are depicted in fig. 15b for L" regularization parameter $\lambda = 0.00001$. The expectation is that there is variability in especially the probe i-vector and q-vector, as the noise, which the probes have been subjected to, appears here. However, there are larger weights appearing for the reference i-vector than for the probe i-vector for every hidden unit, and as the contents of the i-vectors follow the same distribution, the reference i-vector has a larger influence on the classification result than the probe. The reference i-vector having a larger influence on the classification result than the probe may lead to the neural network performing a PLDA augmented with i-vector information not fully utilized in the PLDA, instead of performing quality-informed score normalization. As noted in section 4.2.1, there is some non-zero weights in the area of the reference q-vector, which should have a gradient of 0 regarding the cost function. By adding regularization however,

these weights should move towards zero to minimize the regularization term of the cost function. The weights for the reference q-vector being non-zero may hint at the early stopping criteria is too strict with a patience of 3 epochs – further training may remove these weights and improve the performance of the network.

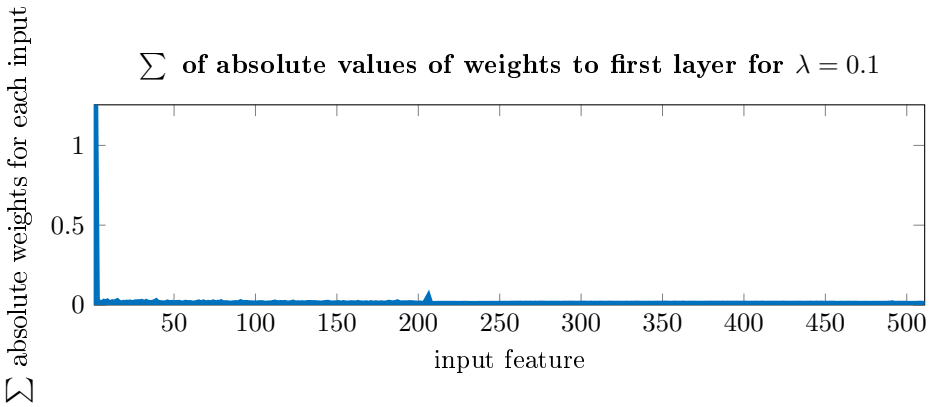
In fig. 16, the performance of three regularization schemes is shown for each condition. The most promising parameter, $\lambda = 0.00001$ outperforms $\lambda = 0.0001$ in close to every condition. There is no performance gain over the baseline PLDA algorithm solely in conditions of 20 second duration and 10dB noise for both AC and crowd noise types. It is difficult to explain this behaviour, yet it is the same duration and noise level for two different noise types, so a pattern could be present, possibly stemming from the i-vector extraction or q-vector GMM training. The noise conditions are the exact median conditions in both duration and noise level, for both noise types.
The batch normalization regularization scheme performs worse than baseline in almost all conditions. However, batch normalization has a worse performance in better conditions than in noisy conditions, and combined with L2 regularization with $\lambda = 0.0001$, the performance on the difficult conditions improves over the baseline, for instance with a $\Delta C_{\text{llr}}^{\text{min}}$ of -5.79 for 5 second duration and 0dB crowd noise and -6.83 for 10 second and 0dB crowd noise conditions. Batch normalization may therefore be applicable in very noisy situations.

The time elapsed to train and classify the comparisons is shown in table 5. The amount of epochs needed to converge given the stopping criteria is seen to be much lower than the maximum of 30 defined as the maximum number of epochs to run, signifying that the early stopping criteria is reached in every case. There does not seem to be a clean correlation between training time and regularization parameter.

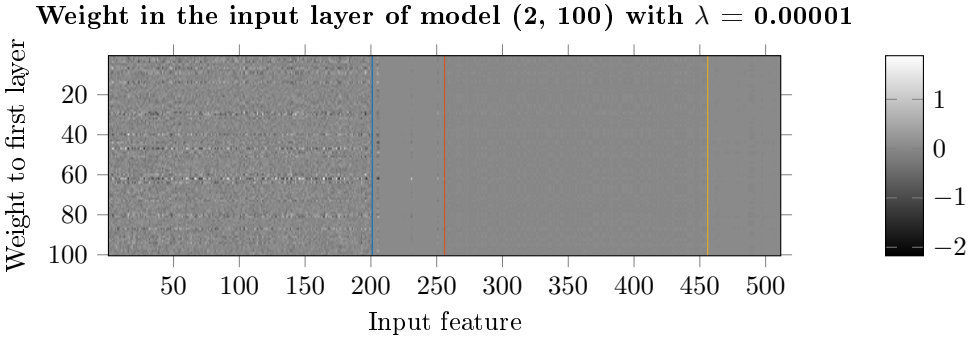
4.2.2 Gain of deeper architecture

Fixing the regularization scheme to L2 regularization with $\lambda = 0.00001$ as the most promising parameter from the previous experiment, the performance of different network size configurations is shown in table 6. The performance metric is $\Delta C_{\text{llr}}^{\text{min}}$, which leads to a better performance than baseline if negative, and worse if positive, respectively.

Models (1, 50), (2, 100), and (4, 100) all have very promising performance of an increased performance of between 5.16% and 6.15% averaged over all conditions, compared to the baseline PLDA. The (1, 50) network has an average $\Delta C_{\text{llr}}^{\text{min}}$ 8.27% lower than the (2, 100) network. The condition with the highest

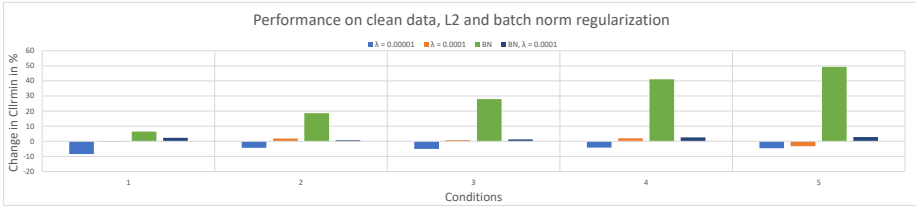


(a) The sum of absolute weights belonging to each input of the network.

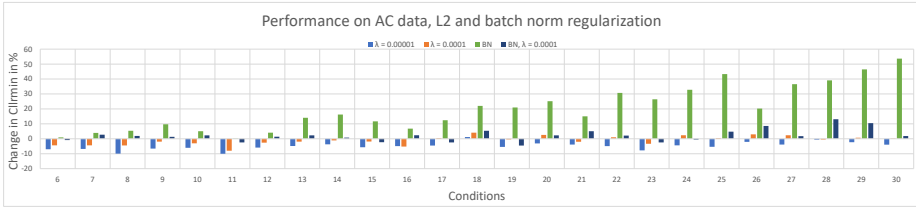


(b) The weights belonging to the first linear hidden layer, i.e. the weighting of the inputs. The weights are shown for each connection of input feature on the first axis to each unit of the first hidden layer on the second axis. First column corresponds to the PLDA score. The blue vertical line denotes the transition from reference i-vector to reference q-vector, the red line the transition from reference q-vector to prove i-vector, and the yellow line denotes the transition from probe i-vector to q-vector.

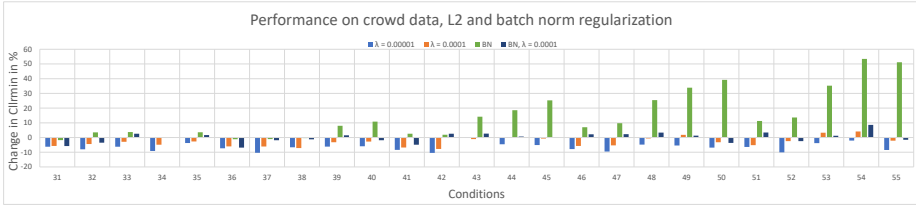
Figure 15



(a) The ΔC_{llr}^{min} compared to baseline for three regularization schemes on clean conditions.



(b) The ΔC_{llr}^{min} compared to baseline for three regularization schemes on AC conditions.



(c) The ΔC_{llr}^{min} compared to baseline for three regularization schemes on crowd conditions.

Figure 16: The ΔC_{llr}^{min} of the models using three different regularization schemes, compared to baseline. λ denotes the L2 regularization parameter. Lower ΔC_{llr}^{min} is better – negative ΔC_{llr}^{min} denotes better performance than baseline. The condition labels correspond to the conditions seen in table 2

λ	0	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	BN	BN, 10^{-4}	BN, 10^{-5}
#Epoch	4	11	**1**	3	**1**	2	11	4	10
t_{train}	16:21	31:41	2:14	13:18	**2:11**	2:46	15:54	7:33	42:31
$t_{\text{train}}^{\text{predict}}$	0:25	0:52	**0:16**	0:46	**0:16**	**0:16**	0:32	0:32	1:08
t_{test}	0:28	0:58	**0:20**	1:08	**0:20**	**0:20**	0:37	0:37	1:21
t / epoch	2:20	2:15	0:33	2:13	**0:32**	0:33	1:08	1:04	3:16
Comp./s	165k	80k	228k	68k	228k	**230k**	126k	126k	57k

Table 5: The number of epochs required to train the neural network using different regularization schemes, as well as the total time spent on training the network, the time required for prediction on the training dataset as well as on the test set. The time per epoch is also included, note that there are three epochs trained more than the number of epochs required caused by the patience of the stopping criteria mentioned in section 3.2. Time format is hours:minutes. Furthermore, the amount of comparisons performed per second on the test dataset is reported, where k denotes kilo, or 1000. $\lambda = 0$ denotes the model without regularization.

improvement of the (1, 50) model compared to baseline, has a performance gain of 11.5%, while the worst faring condition has an improvement in performance of 0.4%.

It is noted that there is a larger relative performance gain on crowd noise conditions over AC conditions, and no network performs worse than PLDA averaged over crowd noise conditions, even when the networks have positive $\Delta C_{\text{llr}}^{\text{min}}$ for clean and AC conditions.

Adding 20% dropout to the hidden units of the two most promising network configurations decreases the average performance of the network substantially in all noise types. The expectation of using dropout is that the network should become more robust towards overfitting. There is a low probability that the networks of these dimensions are able to overfit on the large dataset used in the project. As such, the models including dropout are kept for nearer study in the following experiments as well to examine whether this is true or not.

The standard deviation of the $\Delta C_{\text{llr}}^{\text{min}}$ of all conditions is depicted in table 6 as well. It is noticed that low standard deviation and good model performance are correlated. Well performing models therefore also perform consistently well. The (1, 50) with 20% dropout has a very low standard deviation as well, leading to a consistent, yet slightly worse than baseline performance.

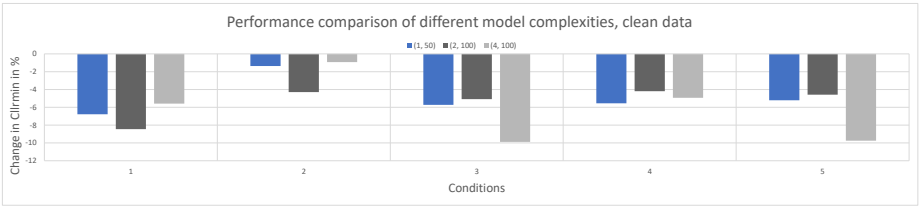
	None	AC	Crowd	All	σ
(1, 50)	-4.93	**-5.62**	**-6.92**	**-6.15**	2.43
(1, 100)	7.17	2.81	-1.15	1.41	6.58
(1, 200)	-1.51	-1.16	-2.99	-2.02	3.50
(2, 50)	0.80	-0.81	-4.05	-2.13	4.27
(2, 100)	-5.32	-4.89	-6.55	-5.68	2.57
(2, 200)	3.84	1.77	-0.83	0.78	4.43
(4, 50)	-1.02	-1.91	-4.29	-2.91	3.06
(4, 100)	**-6.21**	-4.59	-5.52	-5.16	2.87
(4, 200)	1.67	-0.09	-2.25	-0.91	3.78
(1, 50), DO	0.10	0.81	2.07	1.32	**1.62**
(2, 100), DO	8.20	5.41	3.71	4.89	4.04

Table 6: Average ΔC_{llr}^{min} compared to baseline for each model complexity for each noise type. σ is the standard deviation over all conditions. DO denotes the model has an included 20% dropout on all hidden units.

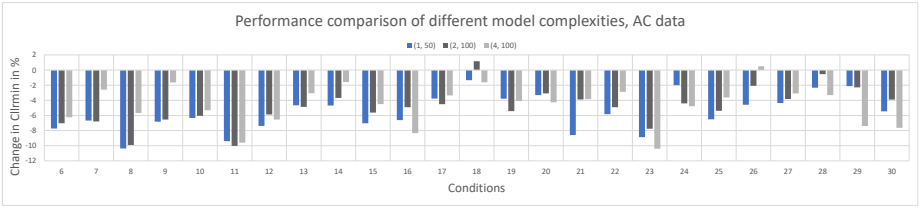
In fig. 17 the ΔC_{llr}^{min} is depicted for the three most promising network configurations (1, 50), (2, 100), and (4, 100). The consistency of the networks discussed above is noted, as seen from the performance of the (4, 100) network, which performs less consistent than the others – performing very well in some conditions, yet worse than baseline in full duration and 0dB conditions for both noise types.

The pattern of the conditions of 20 second duration and 10dB noise seen in the regularization parameter tuning persists for the size configurations. The issue is mitigated by choosing a network with a different size configuration than (2, 100), which results in a performance gain, yet less than for other conditions.

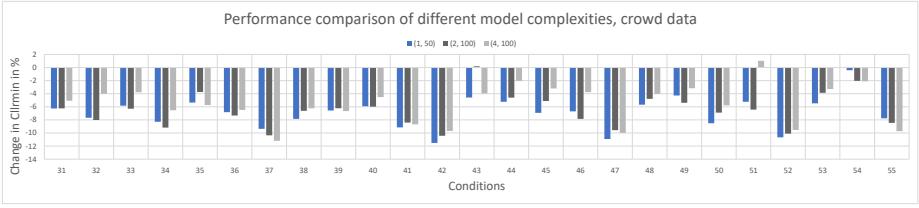
The absolute biometric performance for six different conditions, three difficult and 3 good, is seen in table 7. The performance for full duration and 20dB noise level performs better than the full and clean condition in all metrics for both noise types. An explanation for this phenomena is that there are more conditions that are slightly noisy with low noise levels or low duration reductions, than the one condition that is of full duration and clean of noise. Therefore, there is a much larger proportion of the dataset that may look alike to the slightly noisy conditions.
The (2, 100) model has the lowest C_{llr}^{min} in all conditions depicted in table 7, however when compared to (1, 50) it is solely because of the choice of conditions for the table. The (1, 50) model has lower C_{llr}^{min} than (2, 100) in 33 conditions, whereas the (2, 100) outperforms the (1, 50) in 19 conditions.



(a) The ΔC_{llr}^{min} compared to baseline for 3 model architectures on clean conditions.



(b) The ΔC_{llr}^{min} compared to baseline for 3 model architectures on AC conditions.



(c) The ΔC_{llr}^{min} compared to baseline for 3 model architectures on crowd conditions.

Figure 17: The ΔC_{llr}^{min} compared to baseline for each condition, for three promising network configurations. The regularization used is L2 with parameter $\lambda = 0.00001$.

The difference between the actual C_{llr} and the optimal C_{llr}^{min} is the calibration loss. The PLDA has a better calibrated threshold than the neural network for most conditions, except the worst condition included in table 7 of 5 second duration and 5dB crowd noise. Calibration is left for future work, however an actual C_{llr} of less than one is well calibrated and can be used in to increase the performance in ensemble systems. The PLDA algorithm is therefore not well calibrated on noisy data, and should therefore not be included in an ensemble approach for this condition without performing calibration of the comparison scores, as it would decrease the overall performance of the ensemble method. The difference between actual and optimal C_{llr} shows that there is a great potential in calibration of the comparison scores for the neural network approach. From table 7 it can be seen that the performance is improved by a greater amount on noisy conditions than good conditions across all metrics.

dur	EER			FMR100			C_{llr}^{min}			C_{llr}		
	PLDA	(1, 50)	(2, 100)	PLDA	(1, 50)	(2, 100)	PLDA	(1, 50)	(2, 100)	PLDA	(1, 50)	(2, 100)
$clean^{5}s$	2.93	2.70	**2.61**	6.97	**6.64**	6.78	0.115	0.107	**0.105**	**0.13**	0.76	0.76
$clean^{f}$	0.47	**0.43**	0.47	**0.35**	0.36	0.38	0.022	**0.021**	**0.021**	**0.08**	0.74	0.75
AC_{5dB}^{5s}	7.93	**7.00**	7.06	30.38	27.55	**26.26**	0.274	0.248	**0.246**	**0.45**	0.79	0.79
AC_{20dB}^{f}	0.38	0.38	0.38	0.32	**0.30**	0.32	0.020	0.020	**0.019**	**0.08**	0.74	0.74
Cr_{5dB}^{5s}	17.20	**15.15**	15.30	58.92	57.29	**56.25**	0.528	0.492	**0.489**	1.31	0.86	**0.85**
Cr_{20dB}^{f}	**0.41**	0.42	**0.41**	0.30	**0.28**	0.30	0.021	**0.019**	**0.019**	**0.07**	0.74	0.74

Table 7: Biometric performance on difficult and good conditions for two neural network configurations and the baseline PLDA. The conditions are noted as noise type with duration in superscript and noise level in subscript.

The biometric performance with all conditions pooled, of the two most promising networks, (1, 50) and (2, 100), as well as the networks with 20% dropout applied to the hidden layer, is shown in table 8 for the development set, and in table 9 for the evaluation dataset. Here, the discrepancy between computing the biometric metrics on a condition basis and computing them on the pooled conditions is seen. While the (1, 50) model has an average decrease in C_{llr}^{min} of 6.15%, the decrease on the pooled conditions is 8.66%.

From table 9 is is seen that even though the average ΔC_{llr}^{min} of the neural networks including dropout is above zero, i.e. worse than baseline, the EER of the (1, 50) with dropout is 2.84 versus the 2.94 of the PLDA on the development set. Likewise, the (1, 50) network has a higher average ΔC_{llr}^{min} than the (2, 100) network, yet on the pooled conditions the (2, 100) network has the lowest C_{llr}^{min} on the development set. Dropout adding better generalization can be seen by the (2, 100) with 20% dropout, which on the development set performs worse than the PLDA algorithm, yet has better performance on the evaluation dataset.

Model	EER	FMR100	C_{llr}^{min}	C_{llr}
PLDA	2.94	6.97	0.115	**0.13**
(1, 50)	2.71	6.65	0.107	0.76
(2, 100)	**2.61**	6.78	**0.105**	0.76
(1, 50), DO	2.84	**6.64**	0.109	0.77
(2, 100), DO	3.34	7.27	0.120	0.77

Table 8: Biometric performance for 4 networks as well as PLDA on development dataset. All conditions are pooled.

Model	EER	FMR100	C_{llr}^{min}	C_{llr}
PLDA	3.43	6.96	0.127	**0.16**
(1, 50)	**3.00**	6.46	**0.116**	0.76
(2, 100)	3.13	**6.32**	0.120	0.77
(1, 50), DO	3.35	7.01	0.123	0.77
(2, 100), DO	3.20	6.77	0.122	0.77

Table 9: Biometric performance for 4 networks as well as PLDA on evaluation dataset. All conditions are pooled.

In fig. 18, the DET curves for the PLDA, (1, 50), and (2, 100) models are shown on the evaluation dataset. From the DET plot it is noted that the neural network approach improves the PLDA performance for every security level. The larger network (2, 100), in green, is seen to perform better than the (1, 50) network, in red, in secure situations, i.e. a FMR of less than 2%, whereas the (1, 50) network performs better in less secure situation of higher FMR above 2%.

The required time to train networks of different size is seen in table 10, as well as the evaluation time for both the development and evaluation datasets. The required number of epochs to reach the stopping criteria is generally low – at or below 5 – except for the (2, 100) network which requires 11 epochs. The time per epoch is expected to go up for deeper networks, which is also observed for the 1 layer networks. The networks with 100 units and 2 or 4 hidden layers however use a much longer time per epoch, 3.5 hours for the (4, 100) network compared to 26 minutes for the (1, 50) network. Furthermore, the time needed to train the (1, 50) and (2, 100) networks with dropout is close to equal. The networks have been trained on two different machines with exact same hardware and software setup, however the difference in time is not explained by one machine outperforming the other.
The expected fastest network to train and to perform evaluations of is the smallest network with the fewest units and weights to feed forward through. The network performing the most comparisons per second is, as expected, the (1, 50) network, with 260 thousand comparisons per second. This speed is fast

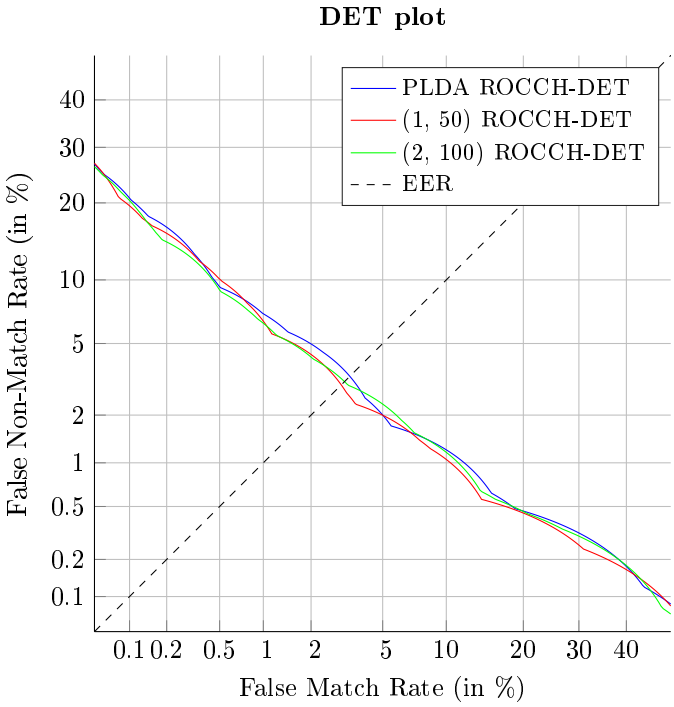


Figure 18: DET curves for evaluation dataset for two promising neural network configurations, as well as the PLDA. The EER line is also depicted.

enough to be immaterial for most real world applications.
The storage space needed to store all the weights of the networks is below 1MB for the largest (4, 200) network, and at 120kB for the (1, 50) network, which is negligible for most modern devices.

Model	#Epoch	t_{train}	$t_{\text{train}}^{\text{predict}}$	t_{test}	t / epoch	Comp./s
(1, 50)	**1**	**1:45**	**0:14**	**0:18**	**0:26**	**260k**
(1, 100)	2	2:38	0:15	0:19	0:31	245k
(1, 200)	**1**	2:30	0:16	0:21	0:37	220k
(2, 50)	**1**	1:59	0:15	**0:18**	0:29	254k
(2, 100)	11	31:41	0:52	0:58	2:15	80k
(2, 200)	5	6:05	0:20	0:23	0:45	196k
(4, 50)	2	2:41	0:16	0:20	0:32	227k
(4, 100)	5	28:05	1:09	1:10	3:30	66k
(4, 200)	**1**	4:15	0:26	0:30	1:03	153k
(1, 50), DO	2	4:43	0:41	0:54	0:56	86k
(2, 100), DO	2	4:42	0:49	0:58	0:56	80k

Table 10: The number of epochs required to train the neural network of different size, as well as the total time spent on training the network, the time required for prediction on the training dataset as well as on the test set. The time per epoch is also included, note that there are three epochs trained more than the number of epochs required caused by the patience of the stopping criteria mentioned in section 3.2. Time format is hours:minutes. Furthermore, the amount of comparisons performed per second on the test dataset is reported, where k denotes kilo, or 1000.

4.2.3 Confirmation by Bayesian optimization

The Bayesian optimization algorithm is initialized using the 9 previously trained networks and their computed performance. The performance metric to optimize with the Bayesian optimization approach is the average $\Delta C_{\text{llr}}^{\text{min}}$ over all conditions. The neural network is therefore not trained using the same cost function as is optimized during the Bayesian optimization. Instead, the best evaluation point found using the Bayesian optimization denotes the network with the best average increase in optimal biometric performance.

The Bayesian optimization results in the best network size being (1, 52) after 30 network evaluations, which is within the 5% of the empirically found network size. The (1, 52) network has an average $\Delta C_{\text{llr}}^{\text{min}}$ over all conditions of -6.331, whereas the (1, 50) model has -6.147.

In fig. 19a, the convergence information of the Bayesian optimization algorithm is shown. The distances between the seeded input have a regular distance pattern, whereas the algorithm from iteration 10 examines points very close together. The best value found by the optimization is shown on the right in fig. 19a, where it is seen that the first, smallest network (1, 50) is the most promising network until iteration 17, where the average ΔC_{llr}^{min} is increases by choosing the (1, 52) network.

In fig. 19b the Gaussian process is shown for the two dimensions, as well as the acquisition function. The exploration of the search space, defined by the acquisition function, is not as high as wanted – it quickly converges to a limited search region of both parameters, and never explores larger networks with many hidden layers and/or hidden units per layer. A reason for the relatively slow exploration is that the output of the neural network does not follow a smooth function nicely or closely, as it is stochastic for each training process of the neural network, due to the randomness of initialization and data sequence. As a result of this, there is no clear, well defined change in performance by changing the amount of hidden layers or units per layer, which causes the standard deviation of the Gaussian process to not decrease much in the areas already exploited, see the scale on the centre figure in fig. 19b.

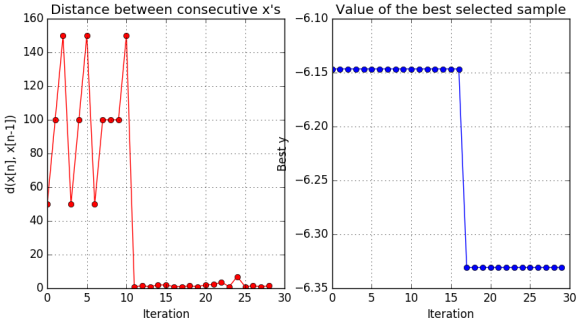
In table 11 the biometric performance of the (1, 52) on the pooled conditions is reported on the evaluation dataset, as well as for the (1, 50) and PLDA models. Here, the discrepancy between the metrics on the pooled conditions and condition wise are again distinct. Even though the (1, 52) network has a lower average ΔC_{llr}^{min} than the (1, 50) model, the latter has better performance metrics on the pooled conditions.

Model	EER	FMR100	C_{llr}^{min}	C_{llr}
PLDA	3.43	6.96	0.127	**0.16**
(1, 50)	**3.00**	**6.46**	**0.116**	0.76
(1, 52)	3.05	6.64	0.121	0.76

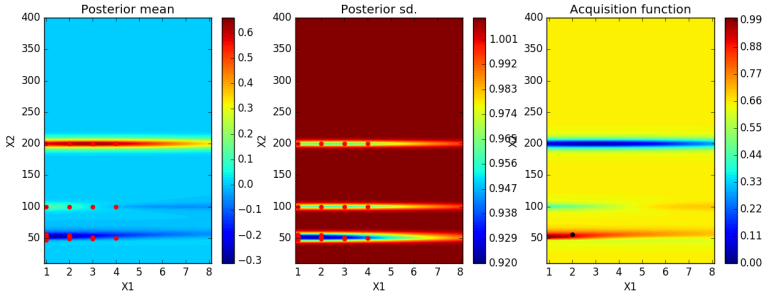
Table 11: Biometric performance for the (1, 50), PLDA, and the optimal found by Bayesian optimization, (1, 52), on evaluation dataset. All conditions are pooled.

4.2.4 Sensitivity towards unseen conditions

The sensitivity to two different types of unseen conditions is examined, and reported separately.



(a) The convergence information of the Bayesian optimization algorithm. Left is shown the Euclidean distance between consecutive evaluations, and right is seen the average $\Delta C_{\text{llr}}^{\text{min}}$ over all conditions for the best evaluation point.



(b) The posterior mean of the Gaussian process, posterior standard deviation, and the acquisition function computed for the final iteration of the Bayesian optimization algorithm. The first axis is the amount of hidden layers and the second axis shows the amount of hidden units per layer.

4.2.4.1 Unseen noise type

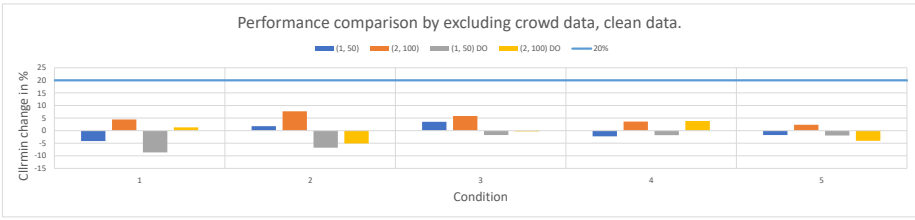
When training only on clean and AC data, the expected outcome is that the model should perform worse on the conditions with crowd noise compared to the model trained on all data, while having around equal performance on clean and AC conditions. The performance on clean or AC data may improve, as there is no crowd data to fit, and the network therefore better can fit to the fewer conditions. However, the performance may also decrease, as there may be information in the crowd conditions that is descriptive of the clean and AC conditions as well.

In fig. 20, the ΔC_{llr}^{min} compared to the model using all training data is seen for four models trained on only clean and AC data. The comparison is therefore not to the baseline PLDA, but the network trained on the whole dataset.

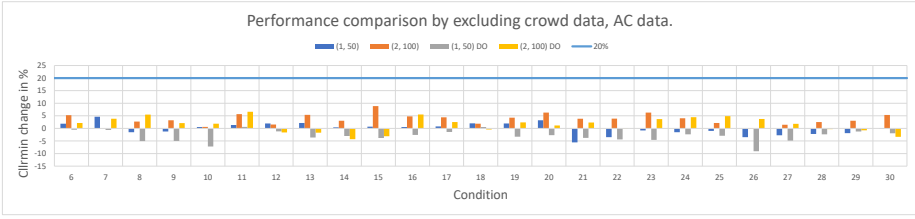
For clean conditions the (1, 50) with dropout has an increase in performance for every clean condition, while the (2, 100) network has a decrease. Furthermore, the (1, 50) with dropout performs better in the less noisy AC conditions. This can be due to not having to fit crowd conditions leading to less variability in the data that needs to be explained or fitted. Even for the best improvement in C_{llr}^{min} of 9.1%, the (1, 50) model with dropout does still not outperform the PLDA algorithm.

The (1, 50) model performs slightly better on the crowd conditions of longer duration, ie. 40 seconds and full duration, than when training on the full dataset. An explanation for this is that the clean conditions constitutes a larger proportion of the total conditions used to train the network, leading to a larger proportion of the dataset having low noise, and therefore a bias in the direction of fitting cleaner data. The (2, 100) model does not improve in performance in any condition by excluding crowd data from the training dataset. This model may have large enough discriminative force to extract common information from crowd conditions that can be used to improve performance on clean and AC conditions as well.

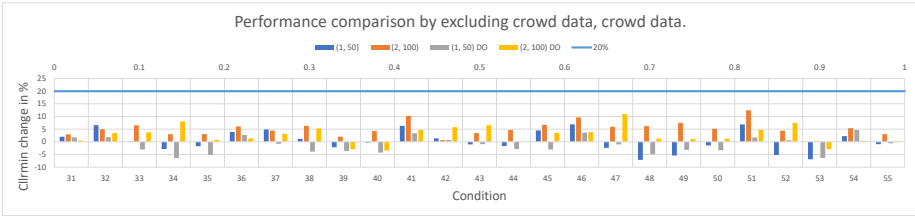
The largest decrease in performance is for the (2, 100) network in the crowd noise condition of full duration and 0dB noise, with $\Delta C_{llr}^{min} = 12.5\%$, satisfying the 20% threshold defined as the success criteria for research question 4, see section 1.2.



(a) For clean data.



(b) For AC data.



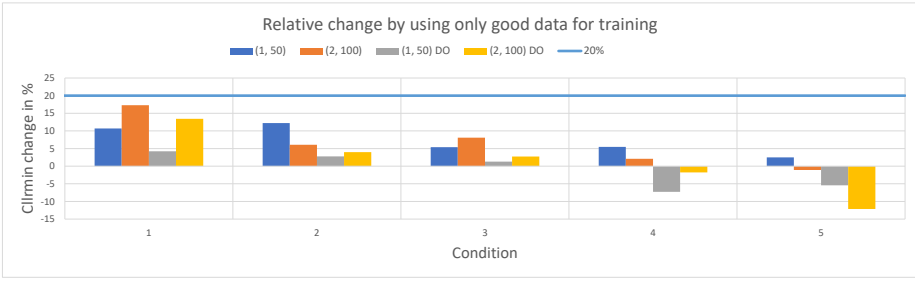
(c) For crowd data, on which the networks have not been trained.

Figure 20: The ΔC_{llr}^{min} of models trained on clean and AC data compared to the model trained on the full dataset, for each condition, showing four network configurations. Performance on the evaluation dataset.

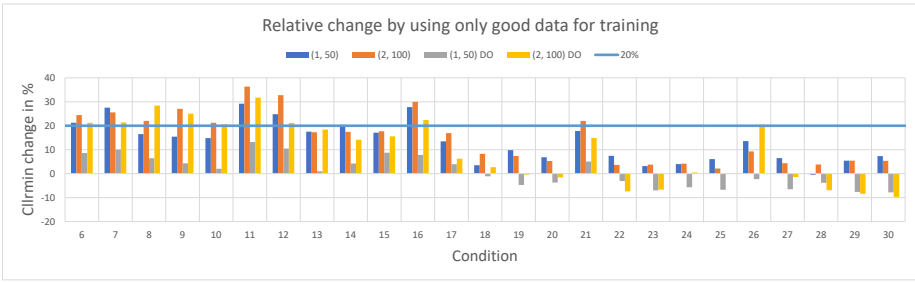
4.2.4.2 Worse noise conditions

The ΔC_{llr}^{min} of the networks trained on only good conditions compared to the networks trained on the whole dataset is depicted in fig. 21.

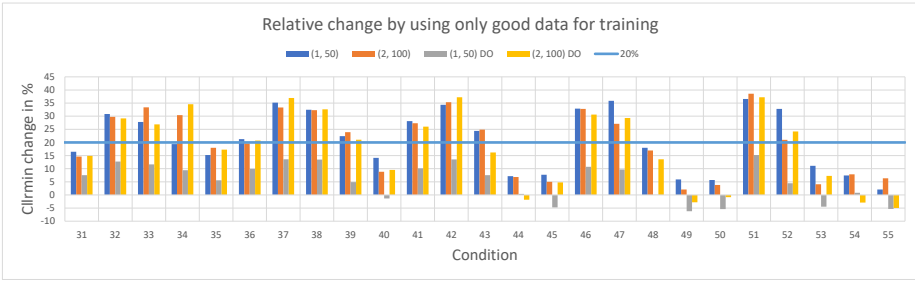
Generally, there is a decrease in performance in all conditions by excluding the worse noise conditions from the training dataset, except for the models using dropout, where an improvement is seen in the good conditions. This increase in performance, appearing when reducing the training dataset size, may indicate that the networks with dropout do not have enough discriminative power to discriminate between all the conditions when trained on the whole dataset, and that the networks with dropout are therefore too small in this case.
It is observed that the models using dropout are more robust to noisier conditions, and have a lower ΔC_{llr}^{min} , than their non-dropout counterparts. The expected result from dropout is a better generalization, and therefore less performance degradation on data with noise present. This is true for all conditions. Yet, the performance of the models with dropout do still not exceed the performance of the baseline PLDA.



(a) For clean data.



(b) For AC data.



(c) For crowd data.

Figure 21: The ΔC_{llr}^{min} of models trained on good conditions compared to the model trained on the full dataset, for each condition, showing four promising network configurations. Performance on evaluation dataset.

In table 12, the minimum, maximum, and mean ΔC_{llr}^{min} of the model trained on good conditions compared to the model trained on all conditions are shown. The largest improvement of more than 12% is seen for the (2, 100) model with dropout for the clean data of full duration, which coincides with the dataset being more biased towards cleaner data, and therefore, the model can better fit the subset of the dataset, as the data is more homogeneous in noise characteristics. While all models are below the 20% increase in C_{llr}^{min} on average, only the (1, 50) model with dropout is below 20% on all conditions. In the worst cases, the ΔC_{llr}^{min} is above 36% for every other model.

	(1, 50)	(2, 100)	(1, 50), DO	(2, 100), DO
min	-0.47	-1.09	-7.81	**-12.15**
max	36.54	38.56	**15.20**	37.23
mean	16.33	16.42	**3.14**	12.83

Table 12: The minimum, maximum and mean ΔC_{llr}^{min} of using only good conditions for training compared to using all, over all conditions.

In figs. 22 to 25, the ΔC_{llr}^{min} is shown comparing the models (1, 50) and (2, 100) with and without dropout trained on the good conditions only, to the models trained on the full dataset. The conditions used for training are the 3×4 rectangle in the bottom right corner of each figure. For each condition outside the training set, the robustness is seen by the intensity – the lower intensity, the lower the ΔC_{llr}^{min} and therefore higher robustness to the specific noise condition.
From fig. 22, it is seen that the (1, 50) model is robust to either higher AC noise levels, or lower duration, but not both higher AC noise level and lower duration conditions simultaneously. For crowd data, the robustness decreases, such that the model is only robust to shorter duration with low crowd noise levels. The robustness to higher crowd noise levels is very low – the ΔC_{llr}^{min} is high.

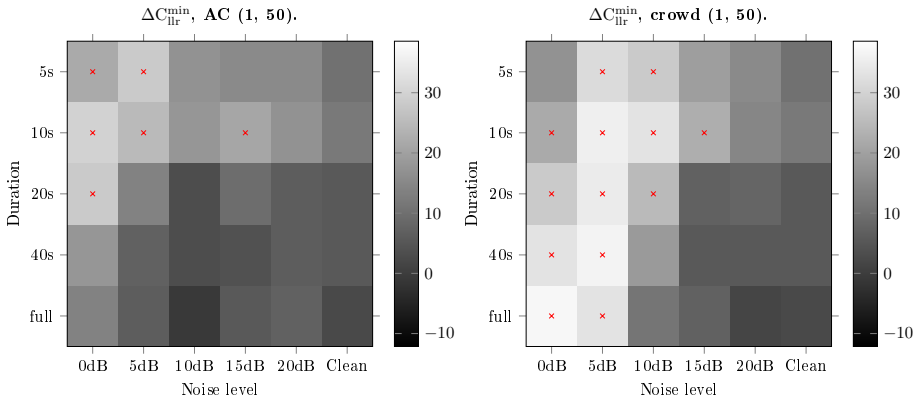


Figure 22: The ΔC_{llr}^{min} of the (1, 50) model, solely trained on good quality data compared to the model trained on all conditions. The red crosses denote conditions where $\Delta C_{llr}^{min} > 20\%$, and which conditions the model is not robust to.

For the (2, 100) model, the sensitivity to unknown conditions is seen in fig. 23. The (2, 100) model is robust to lower durations if the condition is without noise, and for AC noise the model is robust to an increase to 5dB noise or a reduction to 10 second duration, but not both simultaneously. For crowd noise conditions, the model is solely robust to lower duration situations with low noise levels. Adding 20% dropout results to an equal robustness pattern regarding conditions, as seen in fig. 24, yet the ΔC_{llr}^{min} values are lower, showing a gain in robustness from adding dropout.

As noted, the (1, 50) model with dropout is robust to every unknown noise condition, i.e. the performance loss is less than 20% in all conditions. The performance of the (1, 50) model with dropout is depicted in fig. 25.

The expectation is that the worse the noise condition, the more sensitive the model is to the condition. However, for the most difficult condition of 0dB crowd noise of 5 second duration, all the models show robustness. An explanation is that the C_{llr}^{min} is quite high at 0.60 for all models using the full training dataset, and that the performance is explained by underlying elements already present in good quality conditions, where there is not much further discriminative power to be found in slightly less noisy conditions.

Excluding noisy conditions from the training dataset may introduce errors in the current setup by risking weighting the q-vector elements without any information (in the training set) highly, as observed in section 4.2.1 and section 4.2.2.

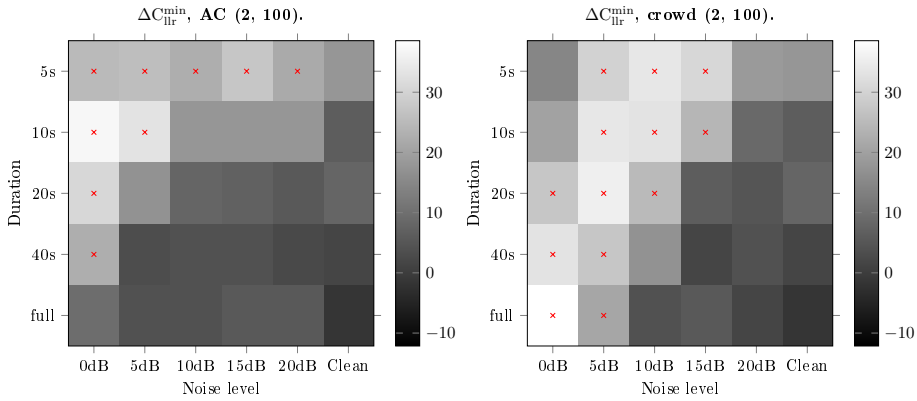


Figure 23: The ΔC_{llr}^{min} of the (2, 100) model, solely trained on good quality data compared to the model trained on all conditions. The red crosses denote conditions where $\Delta C_{llr}^{min} > 20\%$, and which conditions the model is not robust to.

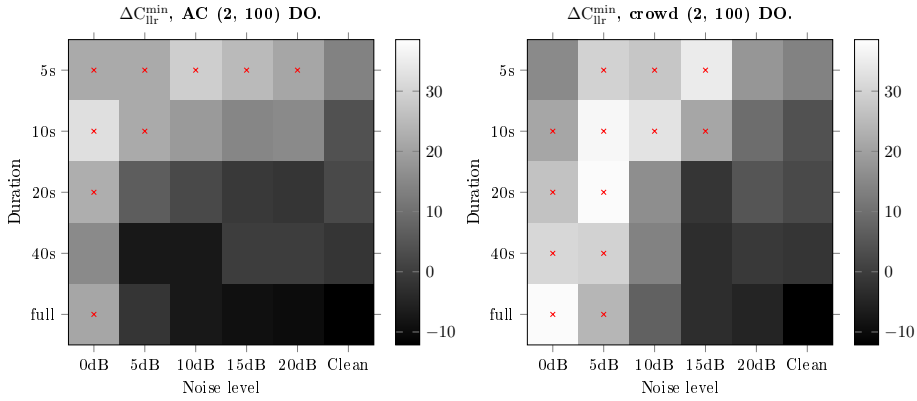


Figure 24: The ΔC_{llr}^{min} of the (2, 100) model with 20% dropout, solely trained on good quality data compared to the model trained on all conditions. The red crosses denote conditions where $\Delta C_{llr}^{min} > 20\%$, and which conditions the model is not robust to.

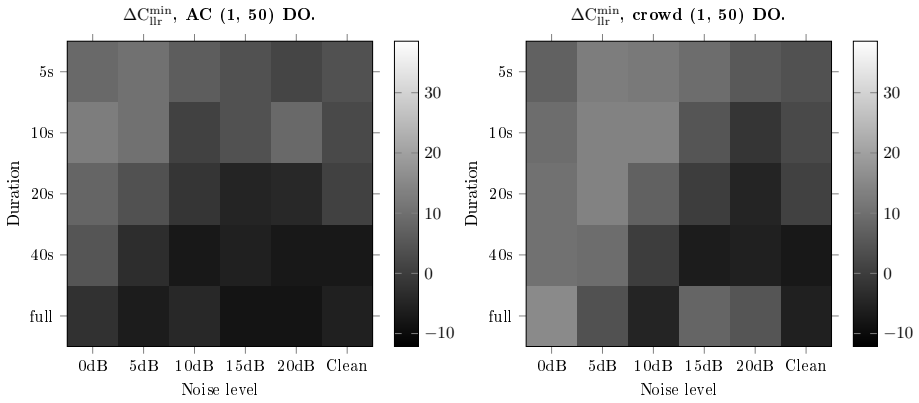


Figure 25: The ΔC_{llr}^{min} of the (1, 50) model with 20% dropout, solely trained on good quality data compared to the model trained on all conditions. The red crosses denote conditions where $\Delta C_{llr}^{min} > 20\%$, and which conditions the model is not robust to.

This may introduce large weights in the network, which do not influence the performance on the development set at all, yet have a large influence on the test dataset including these conditions. The issue is that the network attempts to extract information from inputs, where there is none – as the information is solely present in the test dataset. In realistic examples the new, unseen conditions are by nature unknown, and therefore cannot be included as the input. Then, the unseen conditions are included in a mixture of the already known conditions, which the network can then extract information from by training and recognizing a pattern in the condition, given by a mixture of q-vector elements.
The q-vectors including too many conditions may lead to models including less conditions in the q-vector to perform more robustly on unknown conditions, depending on the ability of the network to ascertain the patterns in the input data.

4.3 Summary

To summarize the finding of the previous section, a brief recapitulation is made per experiment.

4.3.1 Performance of a single network

The simple network, without regularization, of 2 hidden layers and 100 hidden units per layer does not perform much better than a coin toss, yet the performance increases dramatically by adding regularization. The most promising performance is reached by using L2 regularization with regularization parameter $\lambda = 0.00001$, where an average ΔC_{llr}^{min} over all conditions of -5.68 is reached. Increasing the regularization parameter reduces the performance of the network until an identity mapping of the PLDA input is achieved. Batch normalization does not give a performance gain over the baseline PLDA.

4.3.2 Gain of deeper architectures

There is not necessarily a gain in performance to be had by increasing the complexity of the feed-forward neural network. On the contrary, the most promising network architecture consists of 1 hidden layer of 50 hidden units, which reaches an improvement of 6.15% in C_{llr}^{min} averaged over all conditions compared to the baseline PLDA, which is a relative increase of 8.27% compared to the (2, 100) model. There is an improvement in performance for every type of signal degradation. Furthermore, the smaller network also has a higher consistency in performance by having a lower standard deviation of ΔC_{llr}^{min} over the conditions. Using dropout does not improve performance in the condition wise metric, yet on the pooled conditions using a 20% dropout on the (1, 50) network achieves improved performance over the baseline.

4.3.3 Confirmation by Bayesian optimization

Using Bayesian optimization, an optimal network size of (1, 52) if found in 30 iterations. The average C_{llr}^{min} over all conditions for the model is 6.33% better than the baseline. Even though the (1, 52) network has a lower average ΔC_{llr}^{min} than the (1, 50) model, the latter has the better biometric performance on the pooled conditions. The stochastic nature of training neural networks results in

a Bayesian optimization with bad exploration using the Expected improvement acquisition function – large regions of the search space remains unexplored.

4.3.4 Sensitivity to unknown noise conditions

When training on clean conditions and conditions with AC noise, the neural network approach is robust to the unknown noise type of crowd noise.

When training solely on good conditions, i.e conditions of 20 second duration and above and a noise level less than 5dB, the models show larger robustness to unknown, worse noise conditions on AC data. On crowd noise robustness is shown towards short duration for the clean and almost clean conditions. The models show more robustness towards shorter duration than to higher noise levels. The models with a 20% dropout show a lower sensitivity to unknown noise conditions, with the (1, 50) model with dropout being robust to all conditions examined.

Chapter 5

Conclusion and future work

By performing comparison score normalization using quality information, the performance of the baseline model is increased in situations of sample incompleteness and signal degradation.

Employing a neural network approach utilizing a network of three hidden layers, where one is linear, and 100 hidden units per layer, as well as L2 regularization increases the performance of the baseline by 5.68%.
Using deeper networks than the model of three hidden layers of 100 hidden units to perform the score normalization does not increase the performance gain, however. Using a network with one non-linear hidden layer and 50 hidden units per layer, an increase in the relative performance by 8.27% over the two layer model is achieved, reaching an EER of 3% compared to the 3.43% of the baseline on the pooled dataset, a relative gain of 13.3%. The performance gain exists in every noise condition, yet the improvement is largest in difficult noise conditions, which is a challenge for speaker recognition.
By performing Bayesian optimization, the optimal network size is found to be of 1 hidden layer with 52 hidden units, which confirms the empirically found network size to within 5% error. This network size does not improve the performance on the pooled dataset over the model of one non-linear layer found previously.
The neural network approach is robust to new noise types, for instance the crowd noise type. The performance does not degrade more than 20% by omit-

ting crowd noise data from the training set of the network.
By solely training the network on good noise conditions, the neural network approach is found to be more robust to lower duration than to higher noise levels, while the performance degradation is larger than 20% for many noise conditions. Therefore, the network approach is less sensitive to new noise types than it is to more noise introduced of already known noise types.

The proposed method achieves an evident gain in performance in all situations, including clean conditions as well as incomplete sample duration and signal degradation.

The proposed neural network approach for score normalization results in a great biometric performance increase of the PLDA comparator. To increase the performance even further, there are several extensions and examinations that are relevant for future investigation.
The calibration of scores has not been performed in this work, and the examination of the actual performance gain by score calibration is left for future work as well, and should as earlier noted result in a good actual performance increase.
other regularization schemes are of further interest as well, as using dropout increases the robustness towards unknown conditions, and batch normalization seem to perform better in worse conditions.
In this work, the enrolment is assumed cooperative and performed in good conditions. Examining the robustness to uncooperative enrolment or noisy enrolment is attractive from a real-world usability case. If the enrolment is performed in good conditions, removal of the reference q-vector from the input may improve performance – as well as examining whether the proposed method is simply an augmentation of the PLDA algorithm with i-vector information, or actually extracts quality information directly from the i-vectors.
Other quality measures are relevant to examine as well – one approach is to use less conditions in the q-vectors and investigate whether the neural network can find sufficient patterns in the smaller dimensional q-vectors to perform the normalization, another is utilizing neural networks for extracting quality information, which, given the great performance obtained in this work, is very probable to perform well. Using neural networks for quality information extraction leads to a possible entire framework in one deep neural network, consisting of i-vector extraction (by e.g. bottleneck features), q-vector extraction, comparison score (e.g. PLDA-RBM), and score normalization (the method proposed in this thesis). The whole process in one deep neural network is promising for inter-system learning and flexibility.
As the proposed method has the comparison score as input, the sensitivity to different comparators is interesting as well – does the proposed method result in a performance gain when exchanging the PLDA comparator for another comparator, and how robust is the network to the other comparators without retraining the network?

Performing quality-informed score normalization with neural networks achieves a considerable performance gain for every and all signal degradation types, and has great promise for even further improvement by future work, as well as making score normalization possible in mobile devices by having fast evaluation times and low storage requirements.

Bibliography

[aut16] The GPyOpt authors. GPyOpt: A bayesian optimization framework in python. http://github.com/SheffieldML/GPyOpt, 2016.

[BBM14] Pierre-Michel Bousquet, Jean-François Bonastre, and Driss Matrouf. Exploring some limits of gaussian plda modeling for i-vector distributions. In *Odyssey: The Speaker and Language Recognition Workshop*, 2014.

[BCDF10] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[BdP06] Niko Brümmer and Johan du Preez. Application-independent evaluation of speaker detection. *Computer Speech & Language*, 20(2):230–275, 2006.

[BdV13] Niko Brümmer and Edward de Villiers. The bosaris toolkit: Theory, algorithms and code for surviving the new dcf. *arXiv preprint arXiv:1304.2865*, 2013.

[Bis95] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[Brü10] Niko Brümmer. *Measuring, refining and calibrating speaker and language information extracted from speech*. PhD thesis, University of Stellenbosch, 2010.

[Cho16] François Chollet. Keras. https://github.com/fchollet/keras, 2016.

[EFH+13] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, pages 1–5, 2013.

[Faw06] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[FBPS12] Luciana Ferrer, Lukas Burget, Oldrich Plchot, and Nicolas Scheffer. A unified approach for audio characterization and its application to speaker recognition. In *In Proc. Odyssey*, pages 317–323, 2012.

[GREW11] Daniel Garcia-Romero and Carol Y Espy-Wilson. Analysis of i-vector length normalization in speaker recognition systems. In *Interspeech*, pages 249–252, 2011.

[GSA14] Michael A Gelbart, Jasper Snoek, and Ryan P Adams. Bayesian optimization with unknown constraints. *arXiv preprint arXiv:1403.5607*, 2014.

[HH15] John HL Hansen and Taufiq Hasan. Speaker recognition by machines and humans: a tutorial review. *IEEE Signal Processing Magazine*, 32(6):74–99, 2015.

[HKS06] Andrew O Hatch, Sachin S Kajarekar, and Andreas Stolcke. Within-class covariance normalization for svm-based speaker recognition. In *Interspeech*, 2006.

[HSK+12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[ISO11] ISO/IEC JTC SC37 Information technology – Biometric data interchange formats – Part 1: Framework. Standard, International Organization for Standardization, 2011.

[JFR07] Anil Jain, Patrick Flynn, and Arun A Ross. *Handbook of biometrics*. Springer Science & Business Media, 2007.

[Kar16] Andrej Karpathy. Course notes; cs231n: Convolutional neural networks for visual recognition. http://cs231n.github.io/, 2016. Accesed 2016-12-11.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KEY+16] Tomi Kinnunen, Nicholas Evans, Junichi Yamagishi, Kong Aik Lee, Md Sahidullah, Massimiliano Todisco, and Héctor Delgado. Asvspoof 2017: Automatic speaker verification spoofing and countermeasures challenge evaluation plan. *Training*, 10(1508):1508, 2016.

[KGS+14] Patrick Kenny, Vishwa Gupta, Themos Stafylakis, P Ouellet, and J Alam. Deep neural networks for extracting baum-welch statistics for speaker recognition. In *Proc. Odyssey*, pages 293–298, 2014.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[MPK12] Amin Merati, Norman Poh, and Josef Kittler. User-specific cohort selection and score normalization for biometric systems. *IEEE Transactions on Information Forensics and Security*, 7(4):1270–1277, 2012.

[MSM16] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of cnn advances on the imagenet. *arXiv preprint arXiv:1606.02228*, 2016.

[NBB16] Andreas Nautsch, Reiner Bamberger, and Christoph Busch. Decision robustness of voice activity segmentation in unconstrained mobile speaker recognition environments. In *Biometrics Special Interest Group (BIOSIG), 2016 International Conference of the*, pages 1–7. IEEE, 2016.

[NHS+16] Andreas Nautsch, Hong Hao, Themos Stafylakis, Christian Rathgeb, and Christoph Busch. Towards plda-rbm based speaker recognition in mobile environment: Designing stacked/deep plda-rbm systems. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5055–5059. IEEE, 2016.

[NRB+14] Andreas Nautsch, Christian Rathgeb, Christoph Busch, Herbert Reininger, and Klaus Kasper. Towards duration invariance of i-vector-based adaptive score normalization. In *Odyssey speaker and language recognition workshop*, 2014.

[NRSB15] Andreas Nautsch, Christian Rathgeb, Rahim Saeidi, and Christoph Busch. Entropy analysis of i-vector feature spaces in duration-sensitive speaker recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4674–4678. IEEE, 2015.

[NSRB15] Andreas Nautsch, Rahim Saeidi, Christian Rathgeb, and Christoph Busch. Analysis of mutual duration and noise effects in speaker recognition: benefits of condition-matched cohort selection in score normalization. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[NSRB16] Andreas Nautsch, Rahim Saeidi, Christian Rathgeb, and Christoph Busch. Robustness of quality-based score calibration of speaker recognition systems with respect to low-snr and short-duration conditions. Odyssey, 2016.

[SHK+14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SKSD12] Themos Stafylakis, Patrick Kenny, Mohammed Senoussaoui, and Pierre Dumouchel. Plda using gaussian restricted boltzmann machines with application to speaker verification. In *INTERSPEECH*, pages 1692–1695, 2012.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[SLK+13] Rahim Saeidi, Kong-Aik Lee, Tomi Kinnunen, Tawfik Hasan, Benoit Fauve, P-M Bousquet, Elie Khoury, PL Sordo Martinez, Jia Min Karen Kua, CH You, et al. I4u submission to nist sre 2012: A large-scale collaborative effort for noise-robust speaker verification. In *INTERSPEECH*. International Speech Communication Association, 2013.

[SNP98] Martin JJ Scott, Mahesan Niranjan, and Richard W Prager. Realisable classifiers: Improving operating performance on variable cost problems. In *BMVC*, pages 1–10, 1998.

[The16] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[YPS12] Sibel Yaman, Jason Pelecanos, and Ruhi Sarikaya. Bottleneck features for speaker recognition. In *Odyssey 2012-The Speaker and Language Recognition Workshop*, 2012.