UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Voice gender identification using deep neural networks running on FPGA

Marc Palet Gual

Director: Francisco Javier Hernando Pericas (UPC-TSC)

Co-director: Daniel Jiménez-González (UPC-AC)

Ponent: Carlos Álvarez Martínez (UPC-AC)

Defence date: April 29th, 2016

Bachelor of Computer Engineering
Specialization in Computer Architecture

Course 2015-16 2nd semester

Abstract

The use of biological features, biometrics, is widely used in personal identification systems. Automatic speaker identification and speech recognition are established key aspects of these systems, in which voice gender identification plays an important role.

Personal identification systems running on embedded devices often lack enough computational power to satisfy concurrent or high throughput application demands. This project studies the relative improvement of a low powered Field Programable Gate Array (FPGA) design of a voice gender identification system, in comparison to a general embedded architecture. The main hypothesis is that the FPGA design will yield a superior performance in terms of throughput, latency and energy efficiency.

This work presents a voice gender identification system based on a deep neural network implemented on an FPGA device. The system is based on a signal feature extractor able to obtain vocal information suitable for speaker gender identification. Those features were used to design, train and test a set of deep neural networks able to classify the gender of a voice speech signal. These networks were implemented using a general purpose embedded architecture focusing on the performance of the system. Afterwards, the networks were implemented on an FPGA device. Finally, the relative performance gain between both architectures in terms of response time, throughput and energy consumption was assessed.

The studied design confirmed that a voice gender identification system based on an FPGA can exhibit a higher performance, compared to a system based on a standard embedded architecture. That renders the design useful for applications which require a high throughput or that need to classify multiple voice signals concurrently in real time.

Resum

Les característiques biològiques, o biomètriques, són àmpliament utilitzades en sistemes d'identificació personal. La identificació personal automàtica a través de la veu i el reconeixement automàtic de la parla són aspectes claus d'aquests sistemes, en els quals hi juga un paper important la identificació del gènere de l'interlocutor a partir de la veu.

Els sistemes d'identificació personal basats en dispositius encastats sovint manquen de la suficient potència de càlcul per satisfer les necessitats d'aplicacions concurrents o d'alt rendiment. El present projecte estudia la millora relativa d'un sistema d'identificació del gènere de l'interlocutor a partir de la veu basat en un dispositiu FPGA de baix consum, en comparació amb una arquitectura encastada generalista. La principal hipòtesi és que el sistema basat en FPGA gaudirà d'un major rendiment, una menor latència i d'una eficiència energètica superior.

Aquest document presenta un sistema d'identificació del gènere de l'interlocutor a partir de la veu basat en una xarxa neuronal profunda implementada en un dispositiu FPGA. El sistema està basat en un extractor de característiques vocals aptes per la identificació del gènere de l'interlocutor. Aquestes característiques s'han utilitzat per dissenyar, entrenar i posar a prova un conjunt de xarxes neuronals profundes capaces de classificar el gènere de l'interlocutor partint d'un senyal de veu. Les diferents xarxes neuronals han estat implementades utilitzant una arquitectura encastada primant el rendiment del sistema. A continuació, les xarxes s'han implementat en un dispositiu FPGA. Finalment, s'han examinat les diferències relatives de rendiment, latència i eficiència energètica entre ambdues arquitectures.

El disseny objecte d'estudi ha confirmat que un sistema d'identificació del gènere de l'interlocutor a partir de la veu basat en FPGA pot presentar un rendiment superior comparat amb un sistema basat en una arquitectura encastada. Això confirma la utilitat del disseny per a aplicacions d'alt rendiment o que requereixin classificar múltiples senyals de veu de forma concurrent en temps real.

Resumen

Las características biológicas, o biométricas, son ampliamente utilizadas en sistemas de identificación personal. La identificación personal automática a través de la voz y el reconocimiento automático de el habla son aspectos clave de estos sistemas, en los cuales juega un papel importante la identificación vocal del género del interlocutor.

Los sistemas de identificación personal basados en dispositivos empotrados a menudo carecen de la suficiente potencia de cálculo para satisfacer las necesidades de aplicaciones concurrentes o de alto rendimiento. El presente proyecto estudia la mejora relativa de un sistema de identificación vocal del género del interlocutor basado en un dispositivo FPGA de bajo consumo, en comparación con una arquitectura empotrada generalista. La principal hipótesis es que el sistema basado en FPGA disfrutará de un mayor rendimiento, una menor latencia y una eficiencia energética superior.

Este documento presenta un sistema de identificación vocal del género del interlocutor basado en una red neuronal profunda implementada en un dispositivo FPGA. El sistema esta basado en un extractor de características vocales aptas para la identificación vocal del género del interlocutor. Estas características se han utilizado para diseñar, entrenar y poner a prueba un conjunto de redes neuronales profundas capaces de clasificar el género del interlocutor partiendo de una señal de voz. Las diferentes redes neuronales se han implementado usando una arquitectura empotrada primando el rendimiento del sistema. A continuación, las redes se han implementado en un dispositivo FPGA. Finalmente, se han examinado las diferencias relativas de rendimiento, latencia y eficiencia energética entre ambas arquitecturas.

El diseño objeto de estudio ha confirmado que un sistema de identificación vocal del género del interlocutor basado en FPGA puede presentar un rendimiento superior comparado con un sistema basado en una arquitectura empotrada. Esto confirma la utilidad del diseño para aplicaciones de alto rendimiento o que requieran clasificar múltiples señales de voz de forma concurrente en tiempo real.

Contents

Nomenclature

$\mathbb{F}_0$	Fundamental frequency range
$\mathbb{T}_0$	Fundamental sample period range
$f_{0,t}$	Fundamental frequency of a signal x_t at a given time t
f_s	Sampling frequency of a discrete-time signal
AC	Autocorrelation
AGC	Automatic gain control
AMDF	Average magnitude differential function
ANN	Artificial neural network
ASIC	Application specific integrated circuit
ASR	Automatic speech recognition
BProp	Back-Propagation training algorithm
BRAM	Block random access memory
DBN	Deep belief network
DFT	Discrete Fourier transform
DNN	Deep neural network
DSP	Digital signal processor
F0	Fundamental frequency
FF	Flip-flop
FFT	Fast Fourier transform
FPGA	Field programmable gate array
GPU	Graphics processor unit
HDL	Hardware description language

HLS High level synthesis

IDFT Inverse discrete Fourier transform

IFFT Inverse fast Fourier transform

II Initiation interval

LPC Linear predictive-coding coefficients

LUT Lookup table

MFCC Mel-frequency cepstral coefficients

NN Neural network

PDA Pitch detection algorithm

PL Programmable logic

PS Processing system

Q$m.f$ Signed fixed point number format with m integer bits (including the sign bit) and f fractional bits, also notated as $Q_m = m$ and $Q_f = f$

QProp Quick Propagation training algorithm

RProp Resilient back-Propagation training algorithm

RTL Register transfer level

Sigm Sigmoid function

SigmSym Sigmoid symmetric or hyperbolic tangent function

Step Unit step function

StepSym Symmetric unit step function

STFT Short term Fourier transform

ZCR Zero-crossing rate

List of Figures

List of Tables

Part I

Project definition

Chapter 1

Introduction

In the last decade, machine learning has provided new solutions to highly abstract tasks. Data identification and classification tasks, formerly believed to be human dependent, have been addressed by expert systems inspired by how the human brain processes information. Many of these systems demand challenging computing resources often solved by highly specific computing approaches. Graphical Processor Units (GPUs), Field Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuit (ASIC) designs are usually used to deal with these demanding calculation requirements.

This project focuses on voice gender identification as the data identification problem. The identification method will be a deep neural network model whose inputs will be multiple relevant features extracted from a vocal audio signal. The present work aims to contribute to how FPGA designs can improve voice gender identification.

1.1 Context and problem justification

1.1.1 Problem justification

Vocal characteristics can identify speaker dependent attributes such as gender, age or mood. The first step of the identification process is the extraction and conditioning of a set of features from multiple vocal input audio signals believed to convey information from the speaker [1, Ch. 16]. Features from vocal signals are used to build or train a model which maps vocal characteristics to signal features. The purpose of this model is the identification of voice characteristics from any possible speaker. To achieve this, the inputs of the model are the features extracted from an audio signal and the outputs are the identified voice characteristics.

The project studies the relative improvement of an FPGA implementation of a voice gender identification algorithm from a human voice recording.

Voice gender identification is potentially useful in Automatic Speech Recognition (ASR). Gender-dependent models perform better than independent ones [2] therefore, voice gender

identification is needed in order to apply these models. In the same fashion, gender-based speech audio coders achieve better performance than independent ones [3], [4]. In speaker recognition, gender information can be used to limit the search space to speakers from the same gender. Finally, in context based multimedia indexing and retrieval, the speaker's gender can be used as a label for the indexed data.

Knowing the possible applications of voice gender identification, the FPGA approach can have further benefits. The expected FPGA improvements, in terms of energy efficiency and high throughput, may be useful for any large-scale multimedia information retrieval system. Embedded, battery powered or miniaturized systems can benefit from the low power consumption and small size of the FPGA. Finally, speech coders and ASR systems can benefit from the low latency of the FPGA.

Another potential application for an FPGA-accelerated embedded voice gender identification system is personal identification. The use of biological features, biometrics, is becoming largely accepted as a new paradigm for security systems. Automatic speaker identification and, subsequently, voice gender recognition are areas of interests in personal identification methods. The use of an embedded and low powered FPGA device can perform concurrent identification of voice streams in real time [5]. This has an important potential for surveillance, access control, and portable data collection systems.

1.1.2 Stakeholders

This section describes the actors who have an interest in the project or could either affect or be affected by the project outcomes.

Developer

I am the only developer. My functions will be to carry out the investigation and development of the project.

Project director

The project's director is Francisco Javier Hernando Pericas who is part of the Department of Signal Theory and Communications of Barcelona School of Telecommunications Engineering (ETSETB). His role will be to supervise the accomplishment of the project objectives and schedule, and to guide the developer. Additionally, he will advise as the neural networks and signal processing expert.

Project co-director

The project's co-director is Daniel Jiménez González who is part of the Department of Computer Architecture of Barcelona School of Informatics (FIB). He will share the supervision role along with the project director. Additionally, he will advise as the FPGA expert.

Project rapporteur

The project's rapporteur will be Carlos Álvarez Martínez who is part of the Department of Computer Architecture of Barcelona School of Informatics (FIB). He will monitor the project along with the director and co-director. Additionally, he will advise as the FPGA and signal processing expert.

AXIOM project

AXIOM[1] is a project partnership between the University of Siena (Italy), Barcelona Supercomputing Center (Spain), Herta Security S.L. (Spain), Evidence Srl (Italy), FORTH-ICS (Greece), SECO Srl (Italy) and VIMAR Srl (Italy), funded by European Union's Horizon 2020 program. Among the project's objectives is to design a small, energy efficient and modularly scalable board based on a multicore ARM architecture, integrated with an FPGA as a System on Chip (SoC). They are interested in using my project's work as a benchmark test for their hardware platform.

Beneficiaries

Beneficiaries of the project could be multimedia information indexing and retrieval platforms. Some of these platforms rely on real-time processing and the expected response time improvement may be of their interest. Data mining over large sets of recorded voices could also benefit from the project outcomes due to the expected superior throughput and power efficiency of the FPGA implementation. Furthermore, the work could be useful for embedded voice characteristics dependent applications. For example, security and surveillance systems, domotic or human-machine voice interaction interfaces can benefit from the low power needs and high performance of FPGA-based neural network implementation. Automatic speech recognition systems and speech coders could improve their performance by applying gender dependent optimization. Therefore, they could be interested in any improvements in this field.

Users

There are no direct users for the project outcomes. The investigated technologies are intended to serve as a new efficient way to solve existing technological needs in a broad field

[1] Agile, eXtensible, fast I/O Module for the cyber-physical era. http://www.axiom-project.eu/

of actual end user applications.

1.2 Objectives

The main goal of this project is to implement a vocal gender identification model, as described in section 1.1.1. This model will run on a general purpose embedded architecture and on a low power, energy efficient, small FPGA device. The inherent parallelism of neural networks and the methodologies used for extraction of features makes the FPGA implementation suitable for the case study. This project's hypothesis is that, in terms of implementation, an FPGA is capable of delivering a superior performance to that of a general purpose computer. Finally, the project's objective is to achieve a substantial relative improvement in terms of response time, throughput and energy efficiency when the model is running on an FPGA compared to an embedded general purpose system.

1.3 Scope

In order to accomplish the aforementioned objectives, the project will comprise the following activities:

1. Design a voice audio signal feature extraction system, able to extract vocal information suitable for speaker gender identification.
2. Use the feature extractor system to design and train a deep neural network.
3. Implement the neural network algorithm using the feature extraction system and the trained deep neural network on an embedded general purpose architecture. While focusing on maximizing performance, minimizing the problem complexity and operating in real-time.
4. Port the neural network algorithm to an FPGA device. While exploiting the intrinsic characteristics of the architecture and focusing on the parallelizability of the algorithm, the optimization of response time and the minimization of power consumption.
5. Assess the relative performance gain between both architectures in terms of response time, throughput and direct and indirect power consumption.

1.3.1 General methodology

Following the scope definition, the general methodology of this project may be summarized as:

1. Voice feature extractor: a study of pitch detection algorithms. Implementation of a voice fundamental frequency estimator and a Mel-frequency cepstrum coefficient extractor. Assessment of the feature extractors performance.

2. DNN voice gender identification: a study of the best DNN designs and training methods. Training and assessing of the voice gender identification DNN designs.

3. DNN implementation: functional implementation of the designed DNNs. Assessment of the effects of reducing numeric precision and using fixed point arithmetic. Evaluation of the consequences of approximating the DNN activation functions.

4. FPGA DNN implementation: implementation of the designed DNNs on the target FPGA device. Study of the parallelizability and pipelining capabilities of the algorithm.

5. Performance assessment: profiling of the FPGA and the embedded system implementations. Evaluation of the relative performance gain between both systems. Measurement of the power consumption on both systems.

1.4 Tools and resources

1.4.1 Hardware

Zynq ZC702

The FPGA used is Zynq®-7000 AP SoC ZC702 evaluation board. Zynq-7000 family is defined as an All Programmable System on Chip (AP SoC). It includes the XC7Z020 CLG484-1 AP SoC which implements a dual-core ARM Cortex-A9 based processing system (PS) and Xilinx programmable logic (PL) FPGA in a single device. It also includes 1GB DDR3 RAM and I/O interfaces such as flash storage, USB, Ethernet, HDMI video output and ADCs and DACs.

The FPGA programmable logic features 46 200 look-up tables, 92 400 flip-flops, 160 programmable DSP blocks and 380 KB of Block RAM. The ARM system features two 32 KB L1 caches (data and instructions) per core, a shared 256 KB L2 cache and 256 KB of on-chip memory.

Arvei cluster

Arvei is a processing cluster from the DAC department of FIB school. It is composed of 193 mixed types Intel Xeon processing nodes. The cluster was used to perform the training of most of the ANN designs of this project.

1.4.2 Software

Vivado HLS

Xilinx Vivado HLS is a high-level synthesis (HLS) tool which can transform a C or C++ specification into a register transfer level (RTL) implementation synthesizable into an FPGA. This tool has been used during the developing process of the FPGA algorithm to assess the performance of various optimization strategies. The results synthesis results shown in chapter 6 are extracted from HLS performed by this software.

Xilinx SDSoC

Xilinx SDSoC is a development environment for the Xilinx Zynq SoC devices. It features some functionalities of Vivado HLS and it also may be used to perform HLS. SDSoC facilitates the implementation of code using FPGA accelerated functions automatizing the hardware synthesis and interfacing process.

1.4.3 Voice corpus

The 2000 NIST Speaker Recognition Evaluation corpus [6] has been used to train the voice gender identification DNNs and to assess the performance of the voice fundamental frequency extractor. It also has been used and as input data for the FPGA performance assessment.

The 2000 NIST Speaker Recognition Evaluation is part of an ongoing series of yearly evaluations conducted by NIST. They are intended to be of interest to all researchers working on the general problems of speaker verification, speaker segmentation and tracking, speaker identification or speech recognition.

The corpus is divided into training and test datasets. This project only used the training dataset namely the 2000 NIST SRE dataset throughout this work. The used dataset consists of telephonic conversational speech classified by speaker and gender. The information is organized in 1004 single channel SPHERE files encoded in 8 KHz@8-bit mu-law containing one side of a two-way telephonic conversation. The dataset contains 32,8 hours of speech partitioned into 17,9 hours and 547 female speakers and into 14,8 hours and 457 male speakers. The whole dataset has been converted to 8 KHz@16 bit PCM WAV format to ease the use of the corpus.

1.5 Contributions

This project will contribute to the study of ANN-based voice gender identification systems, which is an area with limited studies. The FPGA implementation will presumably contribute to improving the computation and power efficiency of voice gender identification and rep-

resents a novel contribution to the field. Finally, the project outcomes may be suitable as part of real-time applications or systems that process large amounts of voice information concurrently.

Chapter 2

Background

2.1 State of the art

This section describes the state-of-the-art of the technologies used in the project along with how they will be adopted or how they might be improved by the present work.

2.1.1 Artificial neural networks

Artificial neural networks (ANNs), commonly referred to as neural networks (NNs), are a pattern recognition method inspired by how biological nervous systems process information, in particular, the brain. Kohonen defined ANNs as "massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organizations which are intended to interact with the objects of the real world in the same way as biological nervous systems do" [7].

ANNs are formed by local processing individual units, which combine with many other simple behavior units to produce a complex, nonlinear global behavior. Each one of these single units is referred to as a "neuron" which, in turn, is connected to a set of other similar neurons. The network is defined by the behavior of each neuron, the numerical weights associated with the connections between neurons and the interconnection design. ANNs are usually organized in layers which share a common neuron behavior and interconnection design. By tuning the network weights based on the knowledge of a given problem, the ANN can adaptively learn from the problem inputs and is able to produce a correct output when the inputs are new to the network. Associative memory is represented in the network weights, therefore pattern recognition operations are reinforced [8, Ch. 1-3], [9, Ch. 1].

The use of ANNs for pattern recognition began in the 1940's. In 1943, McCulloch and Pitts designed the first artificial network [10]. ANNs simulated by computers were developed during the 1950's [11] and the perceptron algorithm, capable of learning linearly separable patterns, was invented by Rosenblatt in 1957 [12]. Backpropagation algorithm was developed during the 1960's and was applied as a supervised learning method in the context of

ANNs in 1974 by Werbos [13].

Despite promising initial results, ANNs went out of favor during the 1970's. This is because serious computational demands were needed given the contemporary technology. The initialization of the training process affected the training convergence time and there was no defined criterion on how to determine the initial parameters. Furthermore, other pattern recognition methods, such as support vector machines, appeared [14]. The combination of all these factors led to the decrease of research in ANN.

Deep neural networks

In the 2000's, research interest in the field is restored due to new learning algorithms, superior computational power and the advent of GPUs in training and computing ANNs efficiently. Moreover, the initialization problem was addressed by unsupervised learning algorithms successfully introduced by deep belief networks (DBNs) [15]. The terms "deep learning" and deep neural network (DNN) gained attention when Hinton showed how a many layered ANN could be effectively trained [16]. Nowadays, DNNs with multiple layers are trained with large amounts of data and produce state-of-the-art results on many pattern recognition tasks. Computer-vision, automatic speech recognition, audio recognition, image classification or natural language processing tasks have been successfully implemented using DNNs causing a real impact on the machine learning industry [17].

This project uses the DNN paradigm to implement the voice gender identification system.

2.1.2 FPGA-based deep neural networks

Due to highly parallel computation schemes, general purpose processors are often not efficient for DNN implementations and, consequently, they can seldom meet the performance requirements of real-time or high-throughput applications [18]. Hence, various approaches based on GPU, FPGA or ASIC have been studied in order to improve the performance of DNN applications [19], [20].

FPGA-based designs have been proven to have good performance, high energy efficiency, fast development cycles and reconfiguration capabilities [21], [22], [23]. Most implementations are static ANN designs intended to gain performance via dedicated hardware and parallelism [24]. Fewer are the designs that use the run-time reconfiguration capabilities of FPGA to adaptively modify the ANN network [25], [26]. Other approaches focus on accelerating ANN training process [27].

This project falls into the category of a static DNN implementation. The design will be focused on a static DNN computing architecture easily expandable to implement an expert system consisting of multiple DNNs.

2.1.3 Voice gender identification using artificial neural networks

The human voice provides information about the semantics of the spoken words as well as other nonsemantic attributes such as language or speaker characteristics. Currently, automatic speech recognition (ASR) is concerned in non-verbal speaker information such as language, age, gender and emotion. Gender identification can improve the performance of ASR by limiting the speaker search domain to only one gender. Furthermore, spoken audio compression can benefit from accuracy improvements when gender dependent speech coders are used [3], [4].

Despite the numerous machine learning approaches to gender identification found in the literature, few designs use ANNs. These designs mainly differ on what features are extracted from the signal to serve as the ANN inputs. Konig and Morgan used linear predictive-coding coefficients (LPC) to design a multi-layer perceptron as a voice gender classifier [28]. The algorithm described by Meena et al. [29] uses short time energy, zero crossing rate and energy entropy as signal features feeding a combined fuzzy logic and ANN design. The work of Sas and Sas [30] uses fundamental frequency (usually denoted F0) estimation and Mel-frequency cepstral coefficients (MFCC) to design an ANN classifier.

This project uses the latter two signal features as the main base to implement a voice gender classifier. The reasons for this are the high accuracy achieved in Sas and Sas [30], and the ease of implementation of these algorithms in an FPGA due to their parallelizability and arithmetic simplicity.

2.2 Voice features

The first step of the gender classification process is the extraction of a set of features from vocal audio signals believed to convey information from the speaker. This section summarizes different acoustic features used for voice gender identification.

2.2.1 Pitch

Voice pitch is proven to be a good discriminator between males' and females' voices, both in biological and perceptual terms [31]. Pitch has been defined as that perceptual attribute of sound that can be ordered on a scale from low to high [32]. In general, sounds which have a periodic or nearly-periodic waveform are perceived as having a pitch associated with them, and sounds whose waveform is non-periodic are perceived as having no pitch [33, Ch. 3]. Pitched periodic sounds can be decomposed as a series of repeating cycles whose repetition rate is called fundamental frequency, also commonly notated as F0. Complex pitched sounds, such as the human voice or many musical instruments, can be broken down as a sum of simple sinusoidal components whose frequencies are related in a harmonic fashion. These types of sounds are designated as having a harmonic spectra and each sinusoidal component is called partial or harmonic. F0 also corresponds to the lowest partial of a harmonic

spectra.

Although the measurement of pitch and F0 are respectively subjective and objective, a measurement of pitch can be generally assumed to be equal to the F0 [33, Ch. 3]. The perception of the pitch of a sound varies as its F0 is changed. Moreover, a human listener can compare any given pitched sound with a simple sine wave by adjusting its frequency until both of the pitches are perceived as equal.

Pitch detection algorithms

Pitch detection algorithms (PDAs) estimate the F0 of a given signal, usually speech or monophonic musical instruments. Different algorithms can be classified into time-domain and frequency-domain detectors. The following sections describe various analyzed PDAs.

2.2.2 Time-domain PDAs

Time-domain detectors look at the input signal as an oscillating amplitude over time and try to find repeating patterns in order to extract information about the signal periodicity. These types of detectors are usually easy to implement and computationally inexpensive. However, audio signals are highly stochastic and noise can mask the relevant signal leading to poor performance results.

Zero-crossing rate

Zero-crossing rate (ZCR) measures the rate of change of a signal from positive to negative during a given time frame. ZCR of a discrete time signal x_t may be defined as

$$\text{zcr}_t = \frac{1}{2W} \sum_{j=t+1}^{t+W} |\text{sgn}(x_j) - \text{sgn}(x_{j-1})| \tag{2.1}$$

where zcr_t is the zero-crossing rate at time index t, W is the length of the time frame and $\text{sgn}(\cdot)$ is defined as

$$\text{sgn}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \tag{2.2}$$

F0 can be calculated from ZCR by assuming that each signal period would have two zero-crossings. This assumption is considerably weak and noisy signals or harmonic signals which partials are stronger than the fundamental would produce poor pitch estimations. However, this algorithm can be effective as an initial or heuristic estimator due to its low $O(n)$ complexity.

ZCR is also useful to separate between voiced and unvoiced parts of a speech signal when combined with the short-time energy calculation. Voiced sections often have lower zero-

crossing rates and higher short-time energy than unvoiced sections [34].

Autocorrelation

Correlation functions measure similarity between two signals. The autocorrelation function (AC) compares the similarity of the signal with a delayed version of the same signal. The short-time autocorrelation of a discrete signal x_t for a given time frame may be defined as

$$r_t(\tau) = \sum_{j=t}^{t+W-\tau-1} x_j x_{j+\tau} \tag{2.3}$$

where $r_t(\tau)$ is the autocorrelation of delay τ at time index t and W is the length of the analyzed time frame.

If the input signal is harmonic $r_t(\tau)$ will be periodic and will have peaks in the sample period of F0 and its multiples. F0 estimation consists of searching the maxima of $r_t(\tau)$ within a range of delays to find the smallest non-zero delay. This method requires at least two periods of the analyzed signal to produce reliable results, thus determining the range of possible delays as a function of the analysis frame length W. Equation 2.4 defines the AC F0 estimator based on eq. 2.3.

$$f_{0,t} = \frac{f_s}{\tau_{max}}, \quad r_t(\tau_{max}) = \max_{\tau} r_t(\tau), \quad \tau \in (0, W/2) \tag{2.4}$$

where f_s is the sampling frequency of the input signal.

Large values of delay τ imply less terms in the summation of eq. 2.3, thus tapering the envelope of the AC towards zero. This can be solved by normalizing the AC function, dividing each value of $r_t(\tau)$ by the number of terms of the summation ($W - \tau$). Figure 2.1 shows an example of autocorrelation and normalized autocorrelation for a harmonic, noisy input signal. AC method looses precision when F0 is high whereas the normalized AC method looses precision for small values of F0 [35].

Autocorrelation methods are effective for voice pitch detection due to the limited range of possible frequencies. For broader spectrum signals the computational cost $O(n^2)$ increases significantly.

Average magnitude differential function

Average magnitude differential function (AMDF) is a variation of the autocorrelation function. Instead of correlating the signal at various delays, it differentiates the signal with a delayed version of itself and it takes the absolute magnitude at each delay value. The short-time AMDF of a discrete signal x_t for a given time frame may be defined as

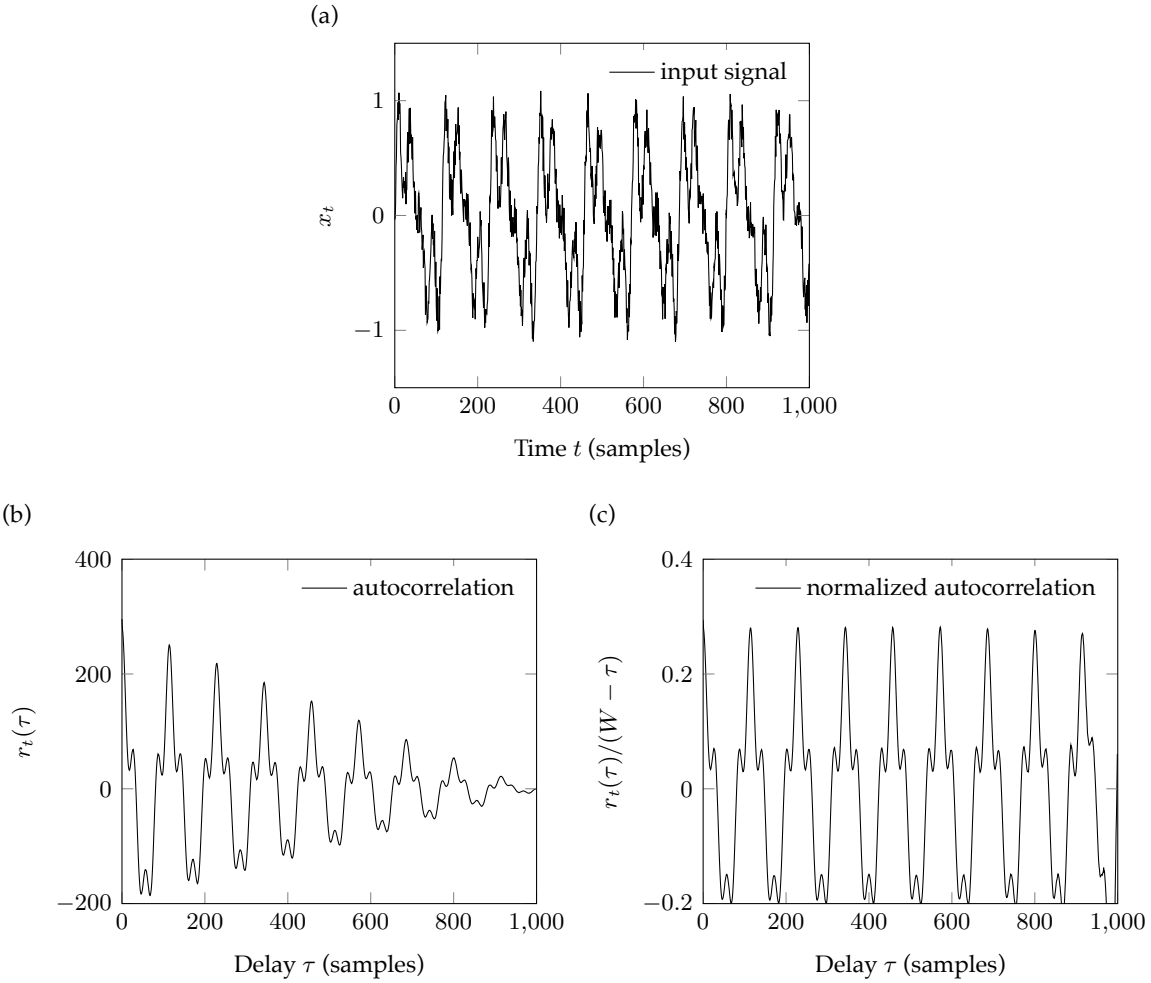


Figure 2.1: Example of autocorrelation. (a) 1000 samples of a harmonic, noisy input signal. (b) AC according to eq. 2.3 where tapering of the function envelope can be observed. (c) Normalized AC without tapering. ($W = 1000$)

$$m_t(\tau) = \frac{1}{W} \sum_{j=t}^{t+W-1} |x_j - x_{j+\tau}| \tag{2.5}$$

where $m_t(\tau)$ is the AMDF of delay τ at time index t and W is the length of the analyzed time frame.

AMDF will exhibit the same periodicity as the AC function when the input signal is harmonic. However, instead of having peaks it will exhibit nulls at the sample period of the F0 and its multiples. Figure 2.2 shows an example of AMDF using the same input signal as figure 2.1a.

F0 may be estimated in a manner similar to that of AC methods, as is defined by the expression

$$f_{0,t} = \frac{f_s}{\tau_{min}}, \quad m_t(\tau_{min}) = \min_{\tau} m_t(\tau), \quad \tau \in (0, W/2) \tag{2.6}$$

The AMDF calculation requires no multiplications making it less computationally expen-

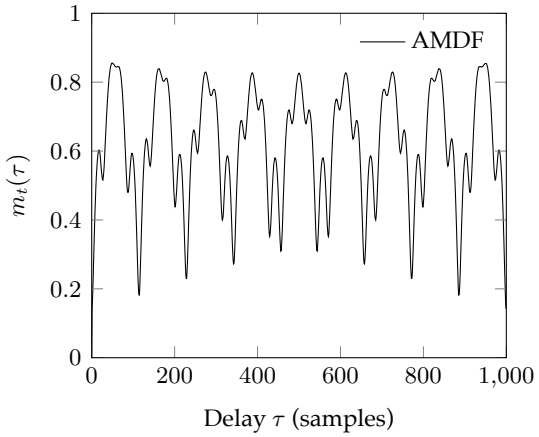


Figure 2.2: Example of AMDF using the same input signal as fig. 2.1a. ($W = 1000$)

sive than AC, while still having $O(n^2)$ complexity.

2.2.3 Frequency-domain PDAs

Frequency-domain detecting consists of dividing the input signal into small frames and getting the short-term Fourier transform (STFT). The Fourier transform will show peaks in the F0 and its multiples for periodic signals; frequency-domain PDAs define ways to find which peak correspond to the F0. The operating principle consists of dividing the audio spectrum into M bins which are equally spaced by a number of N Hz. Due to the logarithmic nature of human pitch perception, this means that low pitches may be detected less accurately than higher pitches. This can be solved by increasing the number of frequency bins by increasing the length of the STFT which, in turn, increases the computational complexity. Another drawback is that increasing the signal frame size implies decreasing the pitch detection temporal accuracy, due to a coarser division of the input signal.

Cepstrum

Cepstrum is an anagram of spectrum coming from the observation that the spectrum can be treated as a time-domain signal and used for further analysis. The cepstrum can be understood as the rate of change across the frequency bands of the spectrum. The cepstrum of a discrete signal $x(n)$ is defined as

$$c_x(\tau) = \mathcal{F}^{-1}\{\log|\mathcal{F}\{x(n)\}|\} \tag{2.7}$$

where $c_x(\tau)$ is the cepstrum of the signal $x(n)$, $\mathcal{F}$ represents the discrete Fourier transform (DFT) and $\mathcal{F}^{-1}$ the inverse discrete Fourier transform (IDFT).

Short-time cepstrum analysis is a common way to estimate the F0 of a signal [36]. The

operating principle behind cepstral F0 estimation is based on modeling of the human voice as a linear time-invariant system: the speech signal is modeled as an excitation signal, produced by the vocal folds, modified by a filter which characterizes the vocal tract. This model can be expressed in the discrete-time domain as

$$x(n) = s(n) \otimes h(n) \tag{2.8}$$

where $x(n)$ is the speech signal, $s(n)$ is the excitation signal, $h(n)$ is the filter which characterizes the vocal tract and $\otimes$ is the convolution operation. Because the excitation and filter signals are convolved they cannot be easily broken down in the time domain.

The speech model can also be represented in frequency domain as

$$X(\omega) = S(\omega) \cdot H(\omega) \tag{2.9}$$

where $X(\omega)$, $S(\omega)$ and $H(\omega)$ are the DFTs of the variables from eq. 2.8. Note that the convolution operation in time-domain has been transformed into a multiplication.

If we take the log magnitude of eq. 2.9 we obtain

$$\log\left(|X(\omega)|\right) = \log\left(|S(\omega)|\right) + \log\left(|H(\omega)|\right) \tag{2.10}$$

The log magnitude of the frequency spectrum has converted the vocal model into a linear combination of the excitation and filter components in the frequency domain.

If we take the inverse discrete Fourier transform of eq. 2.10 we have

$$c_x(\tau) = c_s(\tau) + c_h(\tau) \tag{2.11}$$

where $c_x(\tau)$, $c_s(\tau)$ and $c_h(\tau)$ are the cepstrums of the signals $x(n)$, $s(n)$ and $h(n)$ respectively as if expressed by eq. 2.7. The IDFT of the log spectrum it is said to transform back to "quefrency" or cepstral domain which is similar to time domain.

If we look at the signal cepstrum, the excitation signal is characterized by a peak which time sample matches the F0 period. Whereas, the vocal tract filter is characterized by a series of peaks at the very beginning of the cepstrum signal. Figure 2.3 exemplifies the cepstral processing of a female voice depicting the described steps. On subfigure (c) there is the log spectrum of the excitation signal and vocal tract filter. On subfigure (d) the separated cepstral components $c_s(\tau)$ and $c_h(\tau)$ are marked along with the peak corresponding to the F0.

F0 estimation can be implemented as a maxima search within a sample range which lower bound is high enough to ignore the peaks caused by the vocal tract filter. The cepstral F0 estimation algorithm for a discrete signal $x(n)$ is defined as follows

$$f_0 = \frac{f_s}{\tau_{max}}, \quad c_x(\tau_{max}) = \max_{\tau} c_x(\tau), \quad \tau \in (\tau_0, \tau_1) \tag{2.12}$$

where τ_0 and τ_1 are suitable lower and higher bounds, respectively, that can ignore the effect of the vocal tract filter and are inside the Fourier transform limits.

Frequency-domain autocorrelation

The autocorrelation function can be calculated in the spectral domain as the inverse Fourier transform of the power spectrum of the input signal [37]. Let $X_t(k)$ be the DFT of the input signal $(x_t, \ldots, x_{t+W})$ zero padded to twice its length, the autocorrelation function may be defined as

$$r'_t(\tau) = \sum_{k=0}^{W-1} |X_t(k)|^2 \cos\left(\frac{2\pi k\tau}{W}\right) \tag{2.13}$$

Compared to AC in time domain, frequency domain AC complexity is $O(n \log_2 n)$ when implemented using fast Fourier transforms (FFTs) and inverse FFTs (IFFTs). However, the cost of calculating the transforms is only beneficial for large values of W compared to time-domain AC.

2.2.4 Mel-frequency cepstral coefficients

Mel-frequency cepstral coefficients (MFCC) are a common feature used in automatic speech recognition and speaker identification [38], [39]. They have also been successfully used in gender identification problems [30].

The Mel-frequency cepstrum is a representation of the signal cepstrum on a non-linear scale of frequency. A mel-frequency cepstrum (MFC) coefficient represents the cepstrum of a signal for a narrow frequency band. The MFCC is the composition of various MFC corresponding to evenly spaced frequency bands on a mel-frequency scale. Frequency bands spaced according to the mel-frequency transformation approximates the human auditory system's response closely than linear-spaced frequency bands.

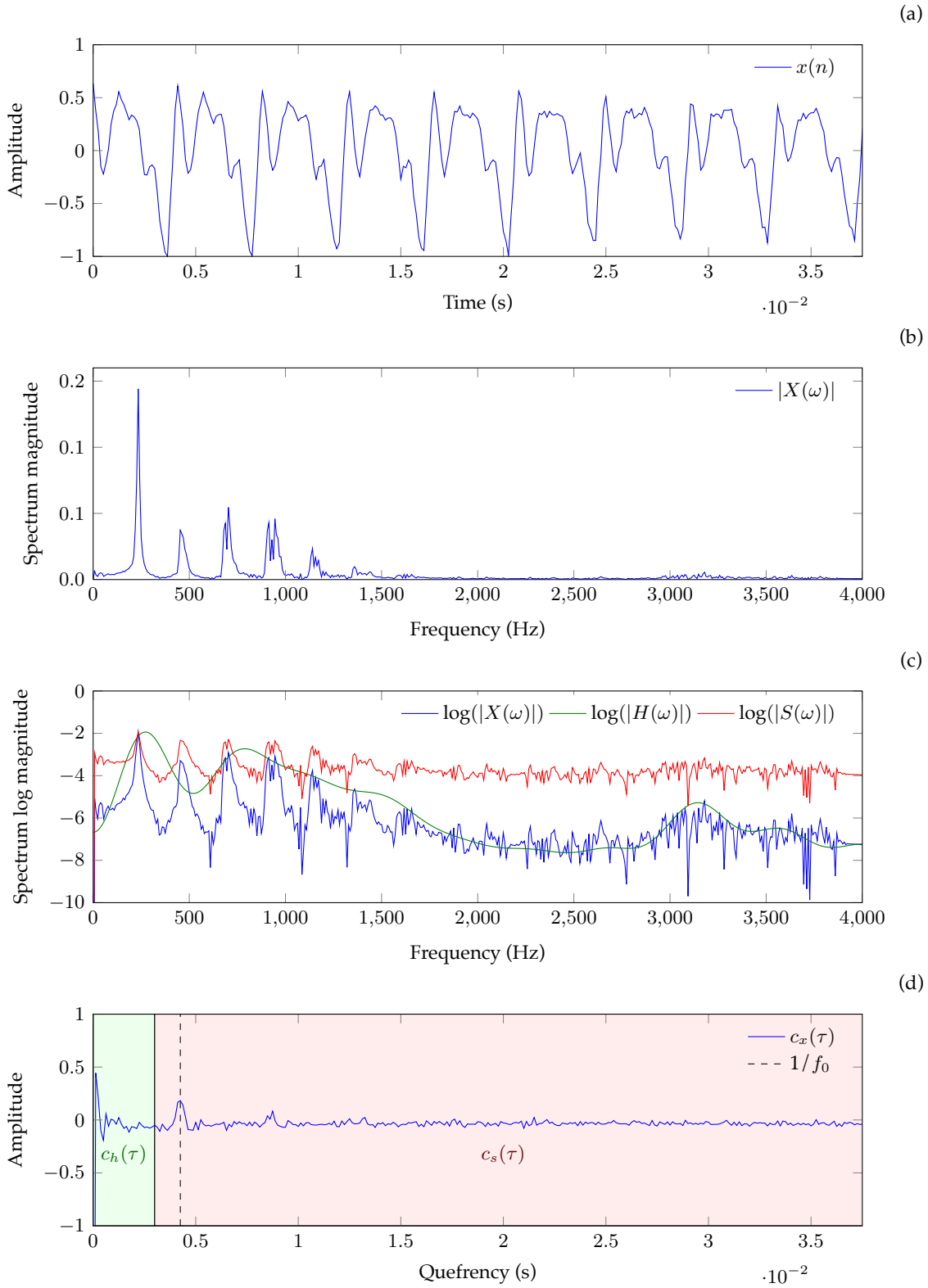


Figure 2.3: Cepstrum example. (a) Female voice input signal with a $f_0 = 235$ Hz. (b) Spectrum magnitude of the input signal calculated using the DFT. (c) Spectrum log magnitude of the input signal showing the excitation signal and the voice tract filter extracted from the cepstrum calculation. (d) Cepstrum of the input signal showing the peak in "quefrency" $\tau = 4.25 \cdot 10^{-3}$ s corresponding to the F0 and the cepstral components of the excitation signal and the voice tract filter.

2.3 Artificial neural networks

This section describes the artificial neural network model.

Notation

Figures, equations, and text will use the following notation in this section and throughout the whole memoir:

Scalars: lower-case *italic* letters: x, y, z

Constants: capital *ITALIC* letters: R, S, T

Vectors: lower-case ***bold-italic*** letters: $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$

Matrices: capital ***BOLD-ITALIC*** letters: $\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}$

2.3.1 Artificial neuron model

The basic computation neuron model, namely unit or neuron, is an abstraction of how a biological neuron operates. It receives a set of inputs with an associated weight w per input. The output of the unit is the computation of an activation function f of the weighted sum of its inputs. Figure 2.4 shows a diagram of the artificial neuron and, for a unit labeled i, the model can be expressed as

$$a_i = f(n_i) \tag{2.14}$$

$$n_i = \sum_j w_{ij} x_j + b_i \tag{2.15}$$

where a_i is the unit's output, n_i is the weighted sum of the inputs or net value of the unit and f is the unit's activation function. Regarding n_i: x_j is the j-th unit's input, w_{ij} is the weight of the input j and b_i is a bias value.

This model can also be expressed in matrix form as

$$a_i = f(\boldsymbol{W}\boldsymbol{x} + b_i) \tag{2.16}$$

where $\boldsymbol{W}$ is the weight matrix and $\boldsymbol{x}$ is the input vector. In this case the input matrix has only one row and is expressed as $\boldsymbol{W} = [\ w_{i,1} \quad w_{i,2} \quad \cdots \quad w_{i,R}\ ]$. The input vector is a column vector notated as $\boldsymbol{x} = [\ x_1 \quad x_2 \quad \cdots \quad x_R\ ]^{\mathrm{T}}$.

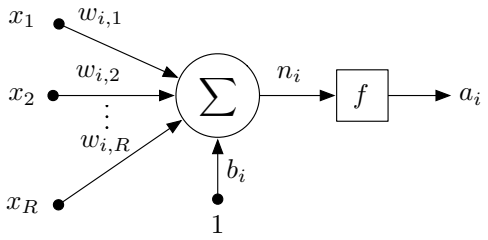


Figure 2.4: Artificial neuron model

2.3.2 Activation functions

The activation or transfer function is an abstract model of the behavior of a neuron given a set of inputs. Biologically, it represents the rate of input action potential that fires a cell. Activation functions are often non-linear in order to introduce a non-linearity into the NN so that their outputs cannot be expressed as a linear combination of their inputs. Activation functions are chosen to satisfy specific particularities of the problem that the NN is trying to solve.

The activation functions used in this work are defined in the following subsections.

Sigmoid

Sigmoid, log-sigmoid or logistic function is one of the most common activation functions. It is a continuous function which range is within $[0, 1]$. It may be defined as

$$\text{Sigm}(x) = \frac{1}{1 + e^{-\theta x}} \tag{2.17}$$

θ is the steepness parameter which is often 1.

Symmetric sigmoid

Symmetric sigmoid or hyperbolic tangent function is a modification of the sigmoid function into an output range within $[-1, 1]$. It may be defined as

$$\text{SigmSym}(x) = \frac{2}{1 + e^{-2\theta x}} - 1 = \tanh(\theta x) = 2\text{Sigm}(2x) - 1 \tag{2.18}$$

Unit step

Unit step, threshold or hard limit function is a non-continuous piecewise function which range is within $[0, 1]$. It may be defined as

$$\text{Step}(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{2.19}$$

The step function may also be understood as a sigmoid function with an infinity steepness parameter θ.

Symmetric step

Symmetric step, symmetric threshold or symmetric hard limit function is a modification of the step function which range is within $[-1, 1]$. It may be defined as

$$\text{StepSym}(x) = \begin{cases} -1, & x < 0 \\ +1, & x \geq 0 \end{cases} \tag{2.20}$$

The symmetric step function may also be understood as a symmetric sigmoid function with an infinity steepness parameter θ.

Summary of functions

Table 2.1 shows a summary of the discussed activation functions and their properties.

2.3.3 Layer of neurons

Single neurons are connected to other units operating in parallel over the same set of inputs. This group of parallel operating units is called a "layer".

Figure 2.5 shows a single neuron layer with S units. Each element of the input vector is connected to each unit resulting in a weight matrix of dimension $S \times R$. The weight matrix

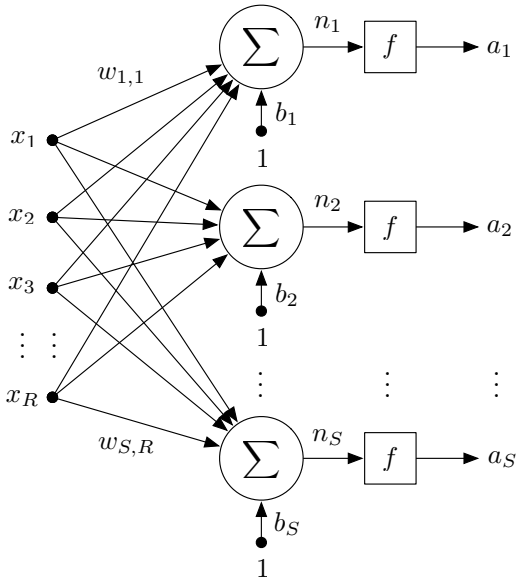
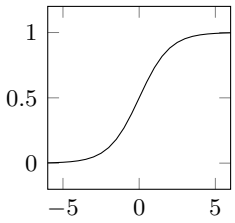
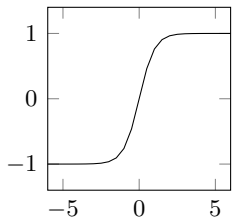
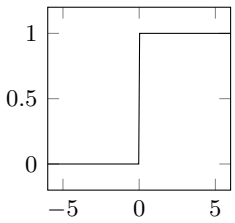
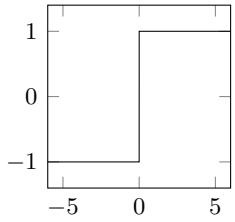


Figure 2.5: Layer of neurons

Table 2.1: Activation functions

Name	Definition	Plot ($\theta = 1$)	Properties
Sigmoid (Sigm)	$f(x) = \frac{1}{1 + e^{-\theta x}}$		Continuous Derivable $f(0) = 0.5$ $\lim_{x \to +\infty} f(x) = 1$ $\lim_{x \to -\infty} f(x) = 0$
Symmetric sigmoid (SigmSym)	$f(x) = \frac{2}{1 + e^{-2\theta x}} - 1$		Continuous Derivable $f(0) = 0$ $\lim_{x \to \pm\infty} f(x) = \pm 1$
Unit step (Step)	$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$		Non-continuous Non-derivable $f(0) = 1$ $\lim_{x \to +\infty} f(x) = 1$ $\lim_{x \to -\infty} f(x) = 0$
Symmetric step (StepSym)	$\mathrm{f}(x) = \begin{cases} -1, & x < 0 \\ +1, & x \geq 0 \end{cases}$		Non-continuous Non-derivable $f(0) = 1$ $\lim_{x \to \pm\infty} f(x) = \pm 1$

$\boldsymbol{W}$ is now defined as

$$\boldsymbol{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix} \tag{2.21}$$

and the calculation of the layer can be expressed as

$$\boldsymbol{a} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{2.22}$$

where $\boldsymbol{a}$ is the output vector and $\boldsymbol{b}$ is the bias vector. Both vectors are defined as $\boldsymbol{a} = [\ a_1 \quad a_2 \quad \cdots \quad a_S\ ]^{\mathrm{T}}$ and $\boldsymbol{b} = [\ b_1 \quad b_2 \quad \cdots \quad b_S\ ]^{\mathrm{T}}$.

The units in a layer may have different activation functions, however, this is outside the scope of the study. For the purposes of the analysis, every neuron layer will always share the same activation function across its units.

2.3.4 Feedforward neural networks

A feedforward neural network is an ANN composed of multiple layers of neurons where the connections between the units never form a closed loop. A feedforward NN has a minimum of three layers: the first layer, called input layer, represents the input values of the network. The neurons in this layer do not have any input connection, nor weights, nor activation functions and their output values codify the input data of the network. The second layer, called hidden layer, receives the data from the input layer and is connected to the third layer. The third layer, called output layer, processes the data coming from the hidden layer and represents the output of the network. The output of each unit codifies an output value of the network. If the network has more than three layers, those between the input and output are called hidden layers.

Each layer has its own bias vector and is connected to the preceding layer through its own weight matrix and, according to the layer definition, each layer may have its own activation function.

To notate this type of network superscripts are used to indicate each layer: L^0 always refers to the input layer and L^{N-1} to the output layer for a network with N layers. The input vector now is notated $\boldsymbol{x} = [\ x_1 \quad x_2 \quad \cdots \quad x_R\ ]^{\mathrm{T}}$ and the output vector $\boldsymbol{y} = [\ y_1 \quad y_2 \quad \cdots \quad y_S\ ]^{\mathrm{T}}$. The i-th layer net value, output and bias vectors are notated as $\boldsymbol{n}^i$, $\boldsymbol{a}^i$ and $\boldsymbol{b}^i$ accordingly. f^i indicates the i-th layer activation function. The weight matrices are referred to as $\boldsymbol{W}^i$ meaning that they connect the outputs of the $(i-1)$-th layer to the inputs of the i-th layer. Likewise, the weight matrices values are expressed as $w^i_{j,k}$ where j indicates the unit of the i-th layer and k the unit of the $(i-1)$-th layer. Finally, the number of units per layer is expressed

as $\left|L^i\right| = S^i$ except for the input and output layers which are expressed as $\left|L^0\right| = R$ and $\left|L^{N-1}\right| = T$.

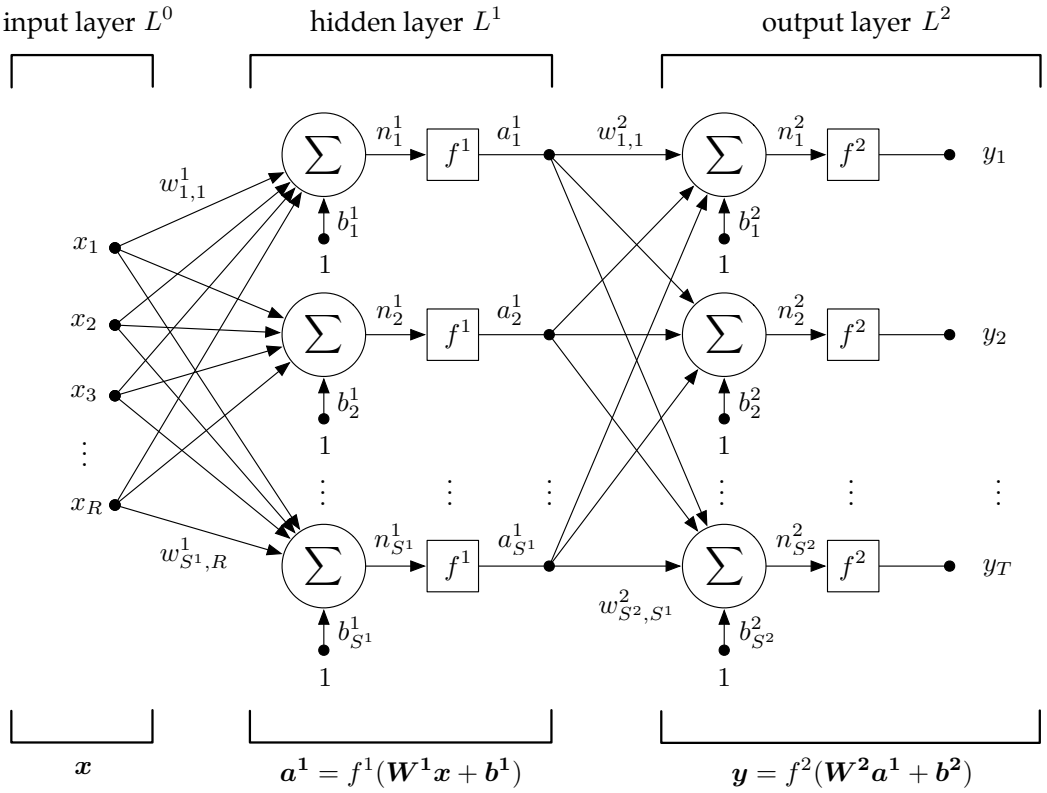


Figure 2.6: Three-layer feedforward neural network

A diagram of a three-layer feedforward network is shown in figure 2.6. The computation of this network can be expressed in matrix form as:

$$\boldsymbol{y} = f^2\left(\boldsymbol{W}^2 f^1\left(\mathbf{W}^1\boldsymbol{x} + \boldsymbol{b}^1\right) + \boldsymbol{b}^2\right) \tag{2.23}$$

In general, the computation of any feedforward NN with N layers can be defined by the following recursion:

$$\begin{cases} \boldsymbol{a}^0 = \boldsymbol{x} \\ \boldsymbol{a}^i = f^i\left(\boldsymbol{W}^i\boldsymbol{a}^{i-1} + \boldsymbol{b}^i\right) \\ \boldsymbol{y} = \boldsymbol{a}^{N-1} \end{cases} \tag{2.24}$$

2.3.5 Deep neural networks

A deep neural network (DNN) is a kind of ANN architecture with multiple hidden layers between the input and output layers of the network. DNNs are often implemented as feedforward networks as well as many other topologies. The main advantage of DNNs is that the extra hidden layers may compose new features from their previous layer, giving the network the potential of classifying complex data with fewer units per layer than a shallower

network [40].

A DNN can be trained using the same training strategies and algorithms than other kind of ANN. However the training convergence time may be higher than a conventional network. Additionally, another common issue of deep architectures is overfitting the training data instead of learning genuine patterns correlated with a classification of the input data.

2.3.6 Signal classification using ANNs

Artificial neural networks can be used as classifiers for time-varying signals. There are many approaches on how to map the signal to the network inputs. The ANNs used in this work are intended to produce a time-varying classification, sampling the input signal at certain times.

The signal classification system, shown in figure 2.7, divides the input discrete-time signal $x(n)$ into frames of length W notated as $x^W(i) = [x_i, \ldots, x_{i+W-1}]$. A frame is processed by a feature extractor which produces a set of features. Those features make up the input vector $\boldsymbol{x}$. Then, the network is computed in order to obtain the output vector $\boldsymbol{y}$. The output vector is processed by a classifier which obtains the class of the frame. The result of the classifier over time is represented by the classification signal $s(n)$.

The input signal can be divided into adjacent frames however, it is often divided into overlapping frames in order to improve the time resolution of the classification signal. The overlapping is defined by the hop size H which indicates the number of samples between two consecutive frames. Commonly, the overlapping is also expressed by the overlapping factor which is $(W - H)/W$. Note that the time resolution of the classification signal is H times smaller than the input signal. The value of $s(i/H)$ is the class of the frame x^W starting at sample time i. Additionally, for a system operating in real time the minimum latency is W samples.

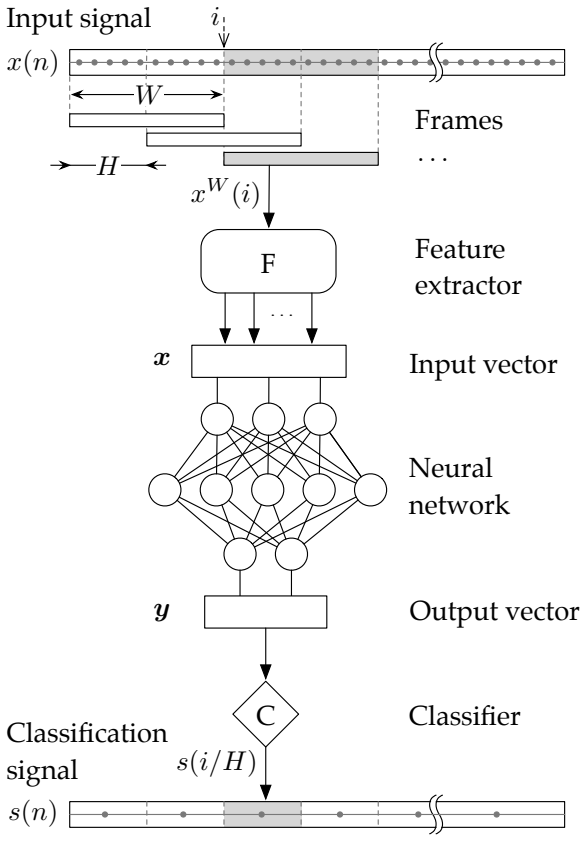


Figure 2.7: Diagram of signal classification using an ANN

Frame grouping

A common hypothesis in ANN signal classification is that the network may improve its performance if it is run over features from multiple frames. The reason for that is that the network can learn feature variation patterns over a certain period of time. To achieve this, the input signal is divided into frames of length W and the network is fed by the features from a group of G frames. The frame overlapping method may still be used.

The network samples the signal's class over a specific time, regardless of the frame grouping. One common approach is to take an odd value of G and to count the central frame time as the classifier signal output time. The network input vector is composed by the current time frame plus $G/2 - 1$ "past" and "future" frames.

Figure 2.8 shows a diagram of the frame grouping method with $G = 3$. The frame $x^W(i)$ is the central frame at sample time i and $x^W(i - H)$ and $x^W(i + H)$ are the past and future frames. They are processed by the features extractors composing an input vector $\boldsymbol{x}$ formed by a concatenation of three feature vectors $\boldsymbol{x'}_{i-H}$, $\boldsymbol{x'}_i$ and $\boldsymbol{x'}_{i+H}$ corresponding to each frame of the group. Finally, the classified network output is added to the classification signal at sample point i/H.

Note that with frame grouping the length of the signal used by the ANN is $H(G-1)+W$

and the minimum latency of the system is $H(G-1)+W-H(G-1)/2$.

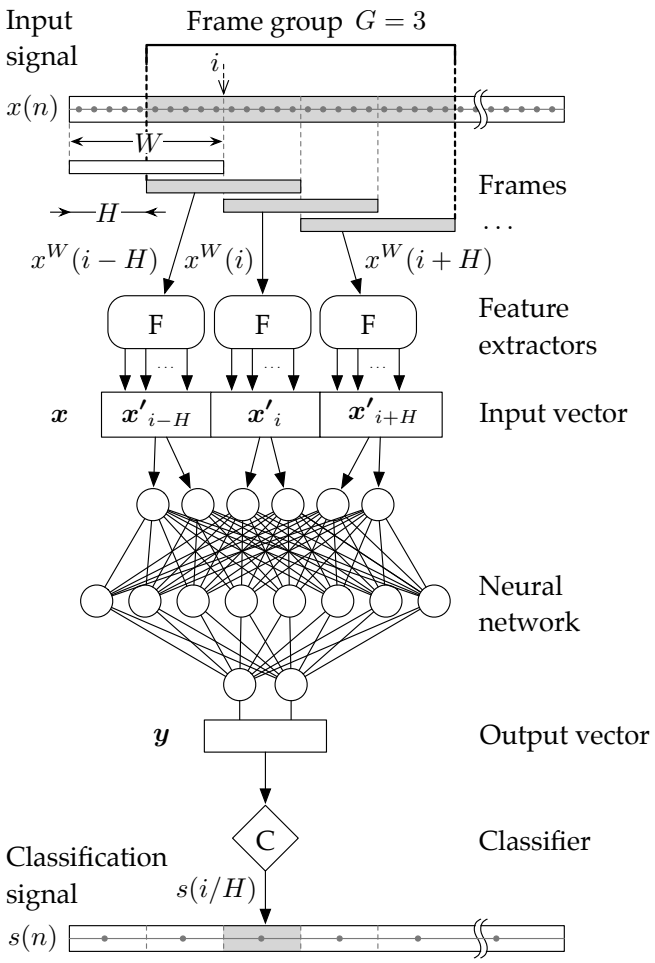


Figure 2.8: Diagram of signal classification using an ANN with frame grouping

Delta coefficients

Another common technique to improve the performance of an ANN signal classifier is to use the delta and delta-delta coefficients of the input vector. The delta, or differential, and delta-delta, or acceleration, coefficients describe the variations of the features over time. They can also be thought of as the trajectories of the features.

Given a scalar feature σ_t extracted from a signal at time t the delta coefficient is calculated as

$$\delta(\sigma_t) = \frac{\sum_{n=1}^{N} n\,(\sigma_{t+n} - \sigma_{t-n})}{2\sum_{n=1}^{N} n^2} \tag{2.25}$$

where $\delta(\sigma_t)$ is the delta coefficient of feature σ at time t and N is the differentiation window. A typical value for N is 2.

Similarly, the delta-delta coefficient is calculated as:

$$\delta^2(\sigma_t) = \frac{\sum_{n=1}^{N} n\left(\delta\left(\sigma_{t+n}\right) - \delta\left(\sigma_{t-n}\right)\right)}{2\sum_{n=1}^{N} n^2} \tag{2.26}$$

The delta and delta-delta coefficients are often calculated for the entire input vector resulting in a coefficient for each scalar value. For a feature vector $\boldsymbol{x} = [\ \sigma_t^1 \quad \dots \quad \sigma_t^M\ ]^{\mathrm{T}}$ formed by M scalar features σ_t^i, the inclusion of the delta and delta-delta coefficients is done by extending the input vector. For instance:

$$\boldsymbol{x'} = [\ \sigma_t^1 \quad \cdots \quad \sigma_t^M \quad \delta\left(\sigma_t^1\right) \quad \cdots \quad \delta\left(\sigma_t^M\right) \quad \delta^2\left(\sigma_t^1\right) \quad \cdots \quad \delta^2\left(\sigma_t^M\right)]^{\mathrm{T}}$$

2.3.7 Training strategies

The ANN used in this work are trained to extract data patterns from a given signal using supervised learning.

Supervised learning is the task of inferring a pattern from labeled training data. The training data consists of a set of data examples associated to the desired output value used during the learning process. A neural network trained using a supervised learning algorithm is expected to be able to correctly determine the expected output from an unseen input.

A supervised learning method applied to an ANN may be divided into the following steps:

1. Determine a dataset representative to the data classification problem being solved.
2. Label each element of the dataset with its expected output value or category
3. Establish a representative set of features which accurately represents the data being classified. These features will compose the input feature vector used by the network.
4. Run a learning algorithm on the ANN using the data features extracted from the dataset and their expected outputs.
5. Test the accuracy of the learned function by feeding the ANN with unseen data and measuring the classification performance.

Many different supervised algorithms can be used to effectively train an ANN. The following subsections briefly describe the training algorithms used in this work.

Backpropagation

The backpropagation algorithm is a well known supervised learning method. This algorithm is based on an iterative process which can be described by the following steps:

1. A training example is calculated by the network obtaining an output value. This value is compared to the expected output and a difference or error signal δ between the network output and the desired value is calculated.

2. The error signal is back-propagated to the previous layers using the same weight coefficients employed in the network computation. At the end of this operation, each unit will have its own error value. This step can also be expressed as running the computation of the ANN backwards without calculating the unit's activation functions.

3. Once the error signal is back-propagated throughout the entire network, the weight matrix is recalculated using the following expression for each matrix value

$$w*^{i}_{j,k} = w^{i}_{j,k} + \eta\delta\frac{df(x)}{dx}a^{i-1}_{j} \tag{2.27}$$

   where η is the learning rate and $f(x)$ is the unit's activation function.

The learning rate parameter controls the convergence speed of the algorithm to stable training results.

QProp and RProp

Quick Propagation (QProp) and Resilient backPropagation (RProp) are optimizations of the backpropagation algorithm by dynamically adjusting the value of the learning rate. These algorithms provide better convergence speeds than the backpropagation algorithm and, often, converge to steadier training results.

Steepness training

The backpropagation algorithm and other alike algorithms require that the activation functions are differentiable. That is not possible for networks using step or symmetric step functions. In order to train those networks, an initial network design is defined using either sigmoid or sigmoid symmetric functions in exchange to their counterparts. Once the desired classification performance is achieved, every activation function of the network increases its steepness parameter and the training process is started over again.

Successive steepness increase makes the activation functions closer to step or symmetric step functions. Once the network training process converges, the ANN activation functions are switched to step or symmetric sigmoid accordingly.

2.4 Used technologies

2.4.1 Field-Programmable Gate Array

A Field-Programmable Gate Array (FPGA) is an integrated circuit that can be reprogrammed to the desired functionality after manufacturing. FPGA are based on a matrix of Configurable Logic Blocks (CLB) connected by programmable interconnects. Each CLB can consist of logical cells, adders, selection circuits, and flip-flops. A CLB can act as a combinatorial logic circuit, register or RAM. Connections between CLBs and I/O devices are provided by flexible interconnection routing, configurable on programming time. Most FPGAs expand this basic architecture by adding higher level functionalities into the integrated circuit. Examples include ALU, DSP blocks, I/O interfaces, memory and embedded processors.

FPGAs are configured using Hardware Description Languages (HDL), such as VHDL or Verilog, or via a schematic circuit design. In order to simplify the design of certain systems, predefined common function circuits are often provided in the form of libraries. These circuits are called Intellectual Property (IP) cores, which are tested and optimized for the current architecture. Most IPs are licensed by FPGA vendors and third parties, although open source licensed IPs are also available.

2.4.2 High-Level Synthesis

High-Level Synthesis (HLS) is an automated design process that interprets an algorithmic description of the desired behavior and creates digital hardware that implements that behavior [41]. HLS is generally capable of interpreting subsets of C, C++, Matlab and other languages. The process analyzes the code, constrains the algorithm to the architecture of the device used in the design and creates a description of a circuit using an HDL language. These descriptions can be used in conventional logic synthesis processes in order to create a circuit or FPGA implementation.

Part II

Designs and results

Chapter 3

Pitch detection

This chapter discusses describes the pitch detection method that was developed.

3.1 Signal pre-processing

From section 2.2.2 in chapter 2, we know that, for a harmonic signal, the AMDF function sampled over different values of τ produces local minima at the sample periods of the F0 and its multiples. The depth of the minima is proportional to the amplitude of the partials contained in the signal. This property can be used to estimate the F0 of a signal.

The limits of F0 has been established inside the 80-350 Hz range with an average fundamental frequency of 120 Hz and 220 Hz for men and women respectively. This assumption is based on common pitch distribution of adult speakers in western languages [42], [31] and pitch distribution of the 2000 NIST SRE dataset extracted using the designed AMDF algorithm as shown in figure 3.1. The F0 range will be notated as

$$\mathbb{F}_0 = [f_0^{min}, f_0^{max}], \quad f_0^{min} = 80\,\text{Hz}, \quad f_0^{min} = 350\,\text{Hz} \tag{3.1}$$

and, by extension, the fundamental sample period range will be notated as

$$\mathbb{T}_0 = [T_0^{min}, T_0^{max}], \quad T_0^{min} = \left\lfloor \frac{f_s}{f_0^{max}} \right\rfloor, \quad T_0^{max} = \left\lceil \frac{f_s}{f_0^{min}} \right\rceil \tag{3.2}$$

where f_s is the sampling frequency of the signal.

However, speech is very rich in harmonics often having partials stronger than the fundamental in the range of 240-850 Hz due to resonances of the vocal tract [42]. This can lead to errors in AMDF pitch tracking caused by the selection of minima corresponding to some harmonic instead of the fundamental frequency. This makes some pre-processing necessary in order to improve the reliability of the system.

The following subsections describe the different pre-processing techniques applied.

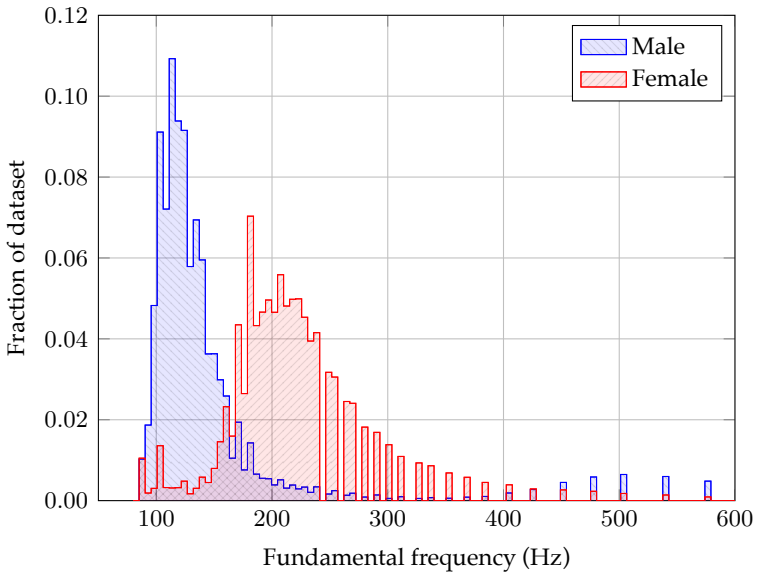


Figure 3.1: Speech fundamental frequency histogram of the 2000 NIST data set extracted using the AMDF algorithm.

Frequency spectrum range narrowing

The first step of pre-processing is to limit the signal spectrum into the $\mathbb{F}_0$ frequency range, therefore, a band pass filter would be useful. The system implements a 5th order Butterworth FIR filter whose cutoffs are f_0^{min} and f_0^{max} respectively. Figure 3.2 shows the filter's frequency response.

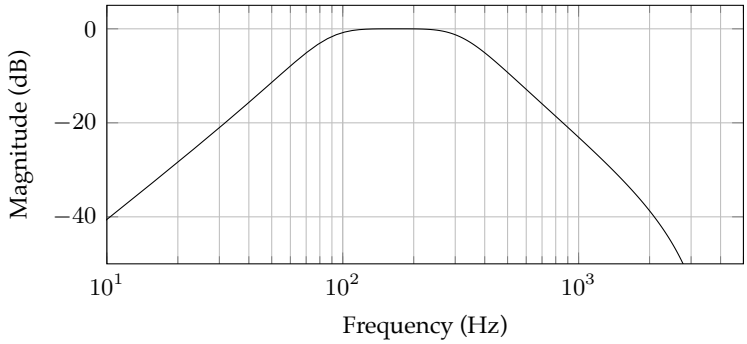


Figure 3.2: Frequency response of the 5th order Butterworth FIR bandpass filter. Cutoff frequencies$f_0^{min} = 80$ Hz and $f_0^{max} = 350$ Hz.

Non-linear processing

Non-linear processing has proven to be successful in further reducing the vocal track resonances structure in the shape of the AMDF vector [43]. The system uses a non-linear tech-

nique called center-clipping defined as

$$y(n) = clc[x(n)] = \begin{cases} 0, & |x(n)| < C_L \\ x(n), & \text{otherwise} \end{cases} \tag{3.3}$$

where $x(n)$ and $y(n)$ are the input and output signals respectively, $clc[\cdot]$ is the center-clipping function and C_L is the clipping threshold which has to be set around the 30% of the input signal maximum amplitude. The reliability of this method depends on the selection of the clipping threshold as a function of the input signal level.

Auto-gain control

Center-clipping processing is highly dependent on the relationship between the level of the input signal and the clipping threshold. Since the input signal is unknown, it is necessary to apply some sort of dynamic processing. The proposed solution is based on auto-levelling the input signal while maintaining a static clipping threshold, for this reason, an automatic gain control (AGC) has been implemented.

An AGC is a closed loop circuit whose purpose is to provide a constant amplitude in its output, despite variations of the input signal's amplitude. Figure 3.3 describes the AGC circuit where $x(n)$ and $y(n)$ are the input and output of the system, the z^{-1} block means a time delay of one sample and R and α are constants of the system. The value of R is the expected root mean square amplitude of the output and α is a parameter that regulates the feedback loop controlling the speed of the gain changes.

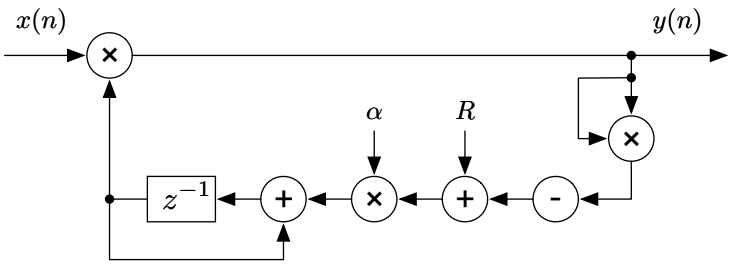


Figure 3.3: AGC circuit.

The AGC circuit operates by subtracting the power of the output signal to a reference power value R. If the output is too high (low) a positive (negative) value is fed back to the delay path which controls the input gain. The α parameter controls the amount of signal fed into the delay path controlling how fast the system reacts to level fluctuations of the input signal.

3.2 Pitch tracking

3.2.1 Short-time AMDF pitch estimator

The short-time AMDF filter, as described in chapter 2 eq. 2.5, differentiates two frames of length W separated by a sample delay τ. An AMDF pitch estimator may be implemented according to eq. 2.6. The first possible optimization is to limit the sampling range of the AMDF function over values of τ inside the $\mathbb{T}_0$ range. The pitch estimator may be expressed as

$$f_{0,t} = \frac{f_s}{\tau_{min}}, \quad m_t(\tau_{min}) = \min_{\tau} m_t(\tau), \quad \tau \in \mathbb{T}_0 \tag{3.4}$$

Modified short-time AMDF

Multiple τ values have to be sampled inside the $\mathbb{T}_0$ range to determine the pitch of the input signal. The problem with this approach is that the necessary frame length of the input signal is a function of τ and its minimum length has to be $W + T_0^{max}$. This variable frame length is not desirable in real-time processing, therefore, the AMDF filter has been modified to operate with a limited frame of fixed length. The modified AMDF may be expressed as

$$m'_t(\tau) = \frac{1}{W} \sum_{j=0}^{W-1} \left| x_{t+j} - x_{t+[(j+\tau) \bmod W]} \right| \tag{3.5}$$

where mod is the modulo operation.

This modification yields similar results to the original definition when $\tau \ll W$ because multiple periods of length τ may be contained in a signal of length $W - \tau$. As τ gets closer to $W/2$ (which is the upper limit of F0 estimator τ resolution, ch. 2 eq. 2.6), the modified AMDF function produces shallower minima than the original definition. Thus, making it necessary to define a frame length W bigger than T_0^{max}.

Pitch estimator harmonic corrections One of the major weaknesses of the AMDF pitch estimator, as discussed in section 3.1, is that it is prone to select a multiple of the F0. The narrowing of the frequency spectrum, done in the pre-processing step, minimizes this problem. However, there are still chances that the pitch estimator chooses a multiple of the F0 which falls inside the $\mathbb{F}_0$ range. This is particularly true in the case of the male speakers.

One optimization of the pitch estimator is to check if multiples of the selected τ value are also local minima of the AMDF vector. This case implies that the estimator has chosen a partial of the voice spectrum instead of the F0. Due to the limited range of the $\mathbb{F}_0$ interval and the filtering of the input signal in the pre-processing stage, the implemented optimization only takes account of the first multiple of the τ value.

The pitch estimator using the modified short-time AMDF with harmonic corrections applied may be expressed as

$$
\begin{aligned}
f_{0,t} &= \frac{f_s}{\tau_n} \\
\tau_n &= \begin{cases} 2\tau_{min}, & m'_t(2\tau_{min}) < m'_t(\tau_{min})\gamma \\ \tau_{min}, & \text{otherwise} \end{cases} \\
m'_t(\tau_{min}) &= \min_{\tau} m'_t(\tau), \quad \tau \in \mathbb{T}_0
\end{aligned}
\tag{3.6}
$$

where γ is the subharmonic factor which is a measure of how shallow the local minimum at $m'_t(2\tau_{min})$ can be with respect to the minimum of the AMDF vector, for $2\tau_{min}$ to be considered as the sample period of the F0.

3.2.2 Voiced/unvoiced decision

The AMDF pitch estimator is effective only when fed with a harmonic signal. One of the problems of the pitch tracking algorithm is to discriminate whether the current processing frame contains an actual harmonic speech signal or it contains silence, noise or non-harmonic content. This problem falls into the category of voice-activity detector algorithms or voiced/unvoiced classifiers. These methods divide the speech signal into voiced and unvoiced regions:

"Voiced speech consists of more or less constant frequency tones of some duration, made when vowels are spoken. [...] About two-thirds of speech is voiced and this type of speech is also what is most important for intelligibility. Unvoiced speech is non-periodic, random-like sounds, [... such] as when consonants are spoken. Voiced speech, because of its periodic nature, can be identified, and extracted." [44]

The designed pitch tracking method implements a voiced/unvoiced classifier based on the following facts:

1. The energy of voiced regions is one order of magnitude higher than unvoiced regions [45].

2. The zero-crossing rate (ZCR) distribution of voiced and unvoiced regions are non-overlapping [46].

3. The envelope of the AMDF function of a harmonic signal sampled over a certain range of delays correlates to voiced/unvoiced sections.

Energy-based classification

To determine the energy of the signal within the processing frame the short-term energy is calculated based on the following definition

$$E_t = \sum_{j=0}^{W-1} x_{t+j}^2 \tag{3.7}$$

Since the input signal has already been pre-processed with the AGC, the voiced/unvoiced decision is based on a static threshold E_L. The value of E_L is empirically set to be the 60% of the average short-term energy of the signal after applying the AGC. Frames which comply with $E_t > E_L$ are considered voiced under this criteria.

ZCR-based classification

ZCR is supposed to be twice the F0 of the signal under the assumption that a periodic signal will cross the origin twice each period. Voiced sections contain harmonic speech signal which F0 is supposed to be inside the $\mathbb{F}_0$ range. Thus, their ZCR is supposed to be small and close to twice the fundamental frequency. On the other hand, unvoiced sections contain noise, broadband spectrum or high pitched sound thus, their ZCR is supposed to be bigger than the voiced sections'. In order to simplify the voiced/unvoiced classifier, the positive zero-crossing is used according to the definition

$$\text{pzc}_t = \sum_{j=t+1}^{t+W} \mathbb{I}\{(x_{j-1} \leq 0) \wedge (x_j > 0)\} \tag{3.8}$$

where $\mathbb{I}\{A\}$ is 1 if the argument A is true and 0 otherwise. Positive zero-crossing is supposed to be half of the full-wave zero crossing.

Voiced and unvoiced ZCR distributions have been empirically observed to crossover around 650 Hz. The voiced/unvoiced decision is based on a static threshold Z_L set to match the positive zero-crossing of a 650 Hz sine wave. Frames which comply with $\text{pzc}_t < Z_L$ are considered voiced. However, the ZCR classification is not reliable on its own as discussed in chapter 2, section 2.2.2.

AMDF-based classification

The AMDF function of a harmonic signal, sampled over a certain range of delays τ, yields nulls at the sample periods of the F0 and its partials. The stronger the harmonic structure of the signal, the deeper the nulls. A good estimator of the voiced/unvoiced status of the frame is the ratio between the minimum and the maximum of the AMDF function over any possible value of τ. Voiced frames will yield low values while unvoiced frames will have

higher ratios. This estimator is named frame confidence and may be defined as

$$\text{conf}_t = \frac{m'_t(\tau_{min})}{m'_t(\tau_{max})} \tag{3.9}$$
$$m'_t(\tau_{min}) = \min_{\tau} m'_t(\tau), \quad m'_t(\tau_{max}) = \max_{\tau} m'_t(\tau), \quad \tau \in (0, W)$$

and the voice-unvoiced decision is based on a static threshold called K_L. Frames which comply with $\text{conf}_t < K_L$ are considered voiced.

One of the advantages of the short-time AMDF pitch estimator is that the AMDF function only needs to be sampled for values of τ inside the $\mathbb{T}_0$ range. Therefore, looking for the maxima of the AMDF function sampled over all possible values of τ would counteract this benefit. One good approximation is to assume that the local maxima of the AMDF will be found in the sample period of the lowest frequency inside the $\mathbb{F}_0$ range (T_0^{max}). The modified frame confidence may be expressed as

$$\text{conf}'_t = \frac{m'_t(\tau_{min})}{m'_t(T_0^{max})}, \quad m'_t(\tau_{min}) = \min_{\tau} m'_t(\tau), \quad \tau \in \mathbb{T}_0 \tag{3.10}$$

The performance of the modified frame confidence has been tested over the 2000 NIST dataset yielding a voice-unvoiced classification difference of 0.1% over the frame confidence defined in eq. 3.9.

Combined voiced/unvoiced classifier

A combination of the three described classifiers has been implemented to provide for a robust voiced/unvoiced classification. The combination consists of a weighted sum of the ratio (or the complementary ratio) between each classifier and its threshold. The combined classifier of a given frame may be defined as

$$V_t = \frac{E_t}{E_L} w_E + \left(1 - \frac{\text{pzc}_t}{Z_L}\right) w_Z + \left(1 - \frac{\text{conf}'_t}{K_L}\right) w_K \tag{3.11}$$

where w_E, w_Z, w_K are the weights of the short-time energy, positive zero-crossing and confidence classifiers respectively. When $V_t \geq 1$ the frame is considered voiced and unvoiced otherwise.

3.2.3 Pitch tracking method

After pre-processing the speech signal, the modified short-time AMDF F0 estimator is applied in a frame-basis to extract the pitch contour over a period of time. The designed system divides the input signal into frames of length W overlapped by $2/3$, in other words, two consecutive frames are separated by $2W/3$ samples.

The combined voiced/unvoiced classifier is also calculated for each frame. If a frame is

considered unvoiced, its pitch value is set to zero.

3.3 Pitch tracking post-processing

The described pitch tracking method is still error-prone. The tendency of the pitch estimator to select multiples of the F0 and inaccurate voiced/unvoiced pitch hypotheses can render the pitch tracking method noisy and unreliable. Therefore, a noise removal filtering of the pitch data is desirable to increase the dependability of the system. A median filter is used in order to smooth out spurious events from the pitch tracker output.

The median filter uses a moving window of length $2L + 1$ over the output sequence of the pitch tracker. The value at point n is the median of the data points from $n - L$ to $n + L$.

3.4 Results

In order to test the accuracy of the algorithm, many comparisons of voice spectrograms and the results of the pitch tracking were studied. The observations of the spectrograms showed that the median filtering combined with the harmonic corrections method had a great effect on the tracking accuracy. As well, the voiced/unvoiced classifier was found to be reliable without the use of the zero-crossing rate which was redundant when combining the energy of the frame and the AMDF confidence value. Therefore, it has been removed from the final algorithm.

The effect of preprocessing the signal had a minor improvement effect when used with noise-free and dynamically steady signals. However, when the algorithm was fed with signals with high or low-frequency noise, the band pass filter proved to be useful for obvious reasons. Additionally, the AGC proved to be useful for signals with a high dynamic range.

Figure 3.4 exemplifies the pitch tracker results without preprocessing nor voiced/unvoiced classifier. The graphic shows the analyzed waveform and two spectrograms of the signal where the pitch estimation is outlined. The second one include harmonic corrections while the first does not. It is observable that the pitch tracking algorithm produces acceptable results most of the time when there is a voiced signal. The median filtering is capable of smoothing out spurious pitch tracks happening at the beginning of words or when consonants occur. The combination of the median filtering plus the harmonic corrections produces a more robust pitch estimation which track tends to remain close by the actual fundamental frequency of the signal.

Figure 3.5 exemplifies the pitch tracker results when applying preprocessing and the voiced/unvoiced classifier. The graphic analyzes the same waveform than the previous example including the results of various of the voiced/unvoiced classifiers. It is observable that the pitch tracking algorithm discriminates between voiced and unvoiced sections limiting the voiced sections when the signal has a strong harmonic content.

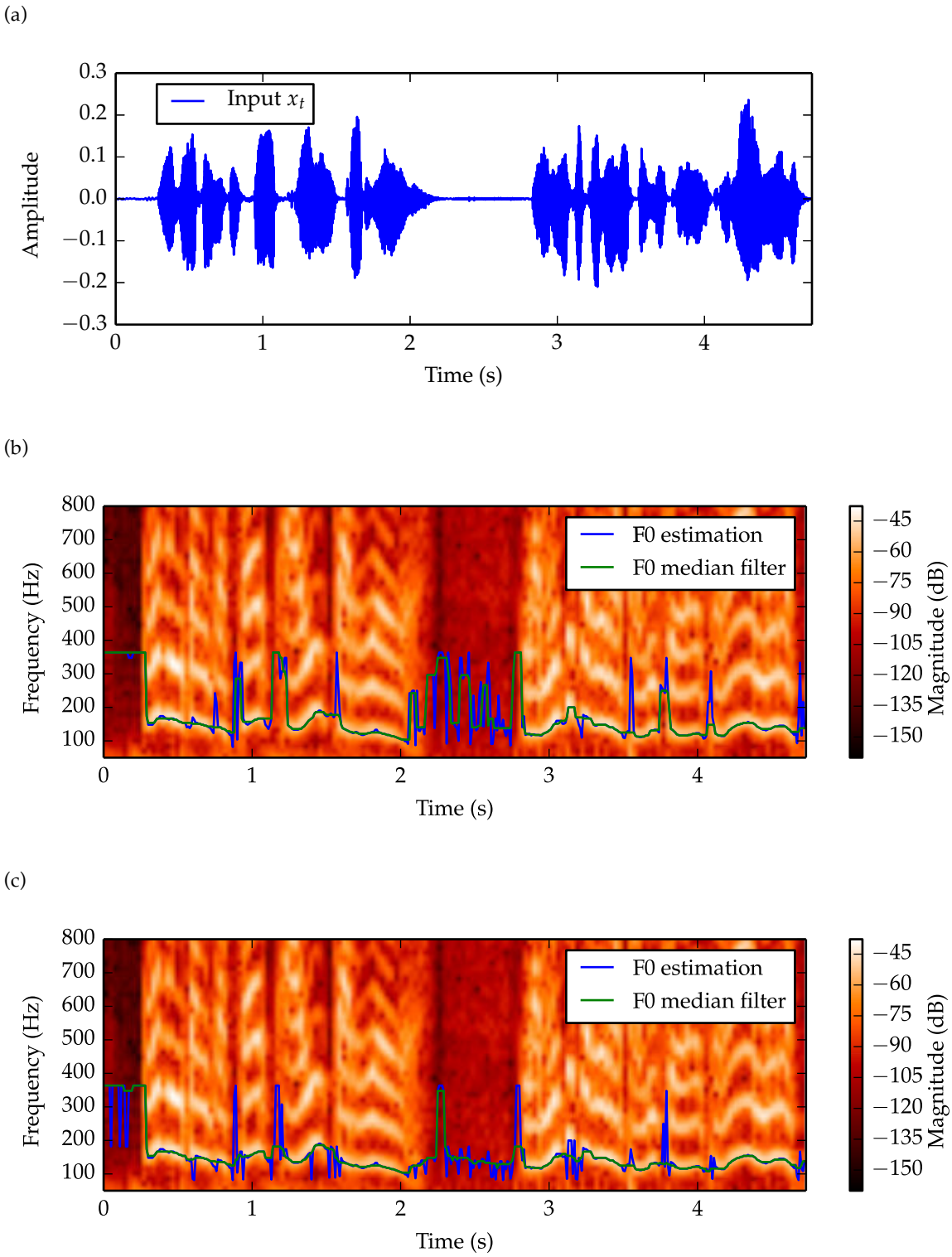


Figure 3.4: Pitch tracker without pre-processing nor voiced/unvoiced classifier. (a) Waveform of the input signal. (b) Spectrogram of the input signal with the pitch tracker results without harmonic corrections. (c) Harmonic corrections added.

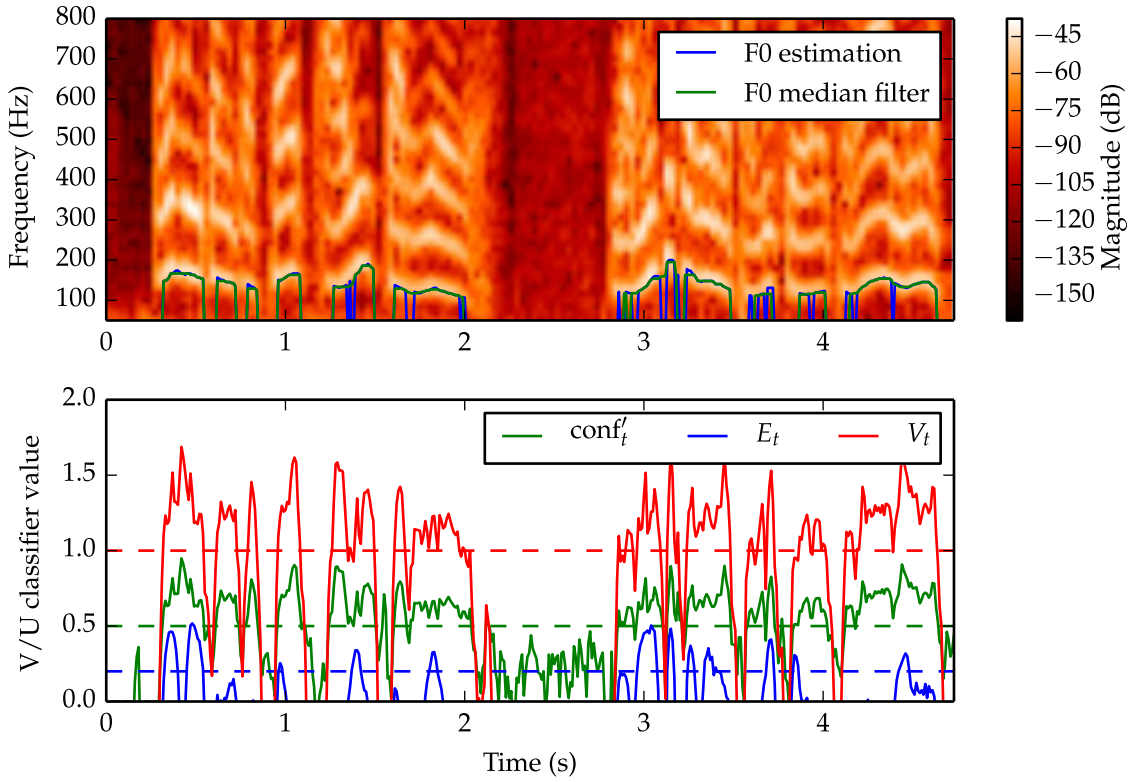


Figure 3.5: Pitch tracker with pre-processing, harmonic corrections and voiced/unvoiced classifier. Top: spectrogram of the input signal with the pitch tracker results. Bottom: voiced/unvoiced normalized classifier values and thresholds (dashed).

Chapter 4

Gender identification using DNNs

4.1 Neural network design

The neural networks used in this work are feedforward DNNs with three hidden layers. The hidden units can use the sigmoid (Sigm), hyperbolic tangent (SigmSym), step (Step) or symmetric step (StepSym) activation functions. The output units can use the sigmoid or step activation functions.

Each of the three hidden layers has the same number of units per layer varying according to the size of the input feature vector. The output layer codifies the guessed speaker gender from the input feature vector either with one or two units. The output encoding of the training and test examples is 1-of-N-1 for the networks with one output and 1-of-N for the networks with two outputs. The former case codifies $y = [0]$ as male and $y = [1]$ as female and the latter codifies $y = [10]$ as male and $y = [01]$ as female.

Network inputs

The input layer consists of a feature vector extracted from $N = 11$ adjacent overlapped frames of the voice signal. Each frame is 30 ms long overlapped with a hop size of 10 ms yielding a total of 110 ms voice signal for any DNN input instance.

The available features are F0 and MFCC calculated for each frame. The F0 features consist of 1 scalar value per frame and MFCC features consist of $K = 26$ scalar values per frame where K is the number of cepstral coefficients. Both features can be extended by the calculation of the delta (differential) and delta-delta (acceleration) coefficients of each scalar value between adjacent frames. The composition of the feature vectors may be any concatenation of the F0 or MFCC features and their delta and delta-delta coefficients.

The feature vector composed by the F0 features is defined as

$$\text{F0} = [f_{0,t}, f_{0,t+H}, \ldots, f_{0,t+H(n-1)}] \tag{4.1}$$

where $f_{0,t}$ is the F0 of the voice signal at a given sample time t, and H is the sample hop size between frames. Similarly, the delta and delta-delta features of the F0 are defined as

$$\Delta_{\text{F0}} = [\delta(f_{0,t}), \delta(f_{0,t+H}) \ldots, \delta(f_{0,t+H(N-1)})] \tag{4.2}$$

$$\Delta^2_{\text{F0}} = [\delta^2(f_{0,t}), \delta^2(f_{0,t+H}), \ldots, \delta^2(f_{0,t+H(N-1)})] \tag{4.3}$$

where $\delta(\cdot)$ and $\delta^2(\cdot)$ are the delta and delta-delta coefficients.

The MFCC feature vector is defined as

$$\text{MFCC} = [C_{1,t}, \ldots, C_{K,t}, C_{1,t+H}, \ldots, C_{K,t+H} \ldots, C_{1,t+H(n-1)}, \ldots, C_{K,t+H(n-1)}] \tag{4.4}$$

where $C_{k,t}$ is the k-th Mel-frequency cepstral coefficient at time t. And the delta Δ_{MFCC} and delta-delta Δ^2_{MFCC} features are defined following the equations 4.2 and 4.3 substituting the F0 features with the MFCC features.

4.2 Training methodology

4.2.1 Dataset

Each DNN has been trained using the 2000 NIST SRE dataset described in chapter 1, section 1.4.3. The dataset has been divided into training and test partitions, 60% and 40% respectively. The train partition is the dataset from where the network learns about the classification problem. The test partition is a new dataset, which is unknown to the network in order to assess its performance. All DNNs have been trained and tested using one or multiple features extracted from the whole dataset following the defined train and test partition scheme.

4.2.2 Features

The dataset has been processed to extract F0 and MFCC features. Both F0 and MFCC features have been post-processed to extract their delta and delta-delta coefficients.

Fundamental frequency

The F0 features have been extracted using the AMDF pitch detection algorithm discussed in chapter 3. The fundamental voice frequencies have been constrained inside the 80-350 Hz range. All data points have been normalized into the $[0, 1]$ range. The pitch detection algorithm has also been used to determine the voiced/unvoiced regions of the audio signal.

Mel-frequency cepstral coefficients

The MFCC features consists of $K = 26$ cepstral coefficients calculated using a Mel filterbank spanning between 0 and 4 KHz. Since MFCC do not have a constrained range of values, the dataset has been used either without normalization, standardized to have zero mean or standardized to have zero mean and unit variance. The networks trained with non-normalized features presented the best results among the three options.

Feature vectors

To produce the feature vectors, the train and test partitions were divided into male and female speakers. Then, the raw voice signals were divided into overlapping frames of 30 ms separated by a hop size of 10 ms. Each frame was processed to extract their F0 and MFCC features along with the voiced/unvoiced status of the frame and the unvoiced frames were discarded from the features sets. Once all features were extracted and the unvoiced frames removed, the delta and delta-delta coefficients of the F0 and MFCC features were calculated.

To produce the F0 feature vectors, the calculated feature sets were divided into overlapping groups of 11 adjacent frames. Each group was 1 frame apart from each other. This yielded a total of $3\,474\,782$ frame groups for the whole dataset. The feature vectors were generated as the F0, Δ_{F0} Δ^2_{F0} features corresponding to each of these frame groups. The vectors along with their gender annotation were stored in separate files for the F0, Δ_{F0} and Δ^2_{F0} features. Each file size was around 600 MB.

A second version of the F0 feature vectors was produced dividing the feature sets into non-overlapping groups of 11 adjacent frames. This yielded a total of $316\,350$ frame groups. The file sizes for the F0, Δ_{F0} and Δ^2_{F0} features were around 60 MB in this new version.

During the training process, it was observed that the high data redundancy due to overlapping in the first version of the feature vectors was not necessary and both versions were producing similar results. Therefore, the first version of the feature vectors was discarded and all calculations were performed using the second version.

To produce the MFCC feature vectors, the last approach was used. The vectors along with their gender annotation were stored in separate files for the MFCC, Δ_{MFCC} and Δ^2_{MFCC} features. Each file size was around 1 GB.

4.2.3 Training process

The DNNs have been trained using the fast artificial neural network[1] (FANN) library. This library features different functions to help the implementation of Back-Propagation (BProp), Quick back-Propagation (QProp) and Resilient Propagation (RPprop). A training tool named

[1] http://leenissen.dk/fann/

fann_train has been developed implementing the different training algorithms in parallelized forms. It also supports the training of DNNs with step activation functions by increasing the steepness of the sigmoid function between epochs. The tool details can be found in appendix A.

The training data set has been shuffled to fairly reproduce the test set and to avoid any bias derived from the order of the data presented to the network.

Training systematization

Multiple feature vector combinations, hidden layer and output layer sizes and activation function combinations have been systematically trained in order to find the best tradeoff between performance and network complexity.

The different network parameters and their possible values are listed as:

H Hidden units count

$S_H = \{5, 10, 20, 50, 100, 200, 500\}$

O Output units count

$S_O = \{1, 2\}$

A_{hid} Activation functions of the hidden units

$S_{A_{hid}} = \{\text{Sigm}, \text{SigmSym}, \text{Step}, \text{StepSym}\}$

A_{out} Activation functions of the output units

$S_{A_{out}} = \{\text{Sigm}, \text{SigmSym}\}$

F Features

$F = \{\text{F0}, \Delta_{\text{F0}}, \Delta^2_{\text{F0}}, \text{MFCC}, \Delta_{\text{MFCC}}, \Delta^2_{\text{MFCC}}\}$

V Feature vector

V can be any concatenated combination of features in set F. The concatenation of n features Z_1 to Z_n will be notated as $Z_1 + \cdots + Z_n$.

T Training algorithm

$S_T = \{\text{BProp}, \text{QProp}, \text{RProp}\}$

The following test scenarios were studied:

1. The combination of all parameters constraining $V = \{\text{F0}, \text{F0}+\Delta_{\text{F0}}, \text{F0}+\Delta_{\text{F0}}+\Delta^2{}_{\text{F0}}\}$ were tested in order to determine the combination of parameters O, H, A_{hid}, A_{out}, F and T that yield the best results using the F0 features.

2. The same test was repeated constraining $V = \{\text{MFCC}, \text{MFCC}+\Delta_{\text{MFCC}}, \text{MFCC}+\Delta_{\text{MFCC}}+\Delta^2_{\text{MFCC}}\}$ and O, A_{hid} and A_{out} to the best values of the last test. The purpose of this test was to analyze the DNNs using the MFCC features.

3. Keeping the same parameters than the second test, the DNNs were tested constraining V to any possible concatenation of the elements in F to determine the best feature combination using both F0 and MFCC features.

Performance test

The test metric of the training is the classification error ε defined as the fraction of the data points where the gender classification failed. The classification error is only considered in the test partition as a measure of the performance of the network to unknown data. For networks with one output the classification criteria is defined as

$$\begin{aligned} 0 \leq y_1 < 0.5 &\Rightarrow \text{male} \\ 0.5 \leq y_1 \leq 1 &\Rightarrow \text{female} \end{aligned}$$

and for networks with two outputs

$$\begin{aligned} y_1 \geq y_2 &\Rightarrow \text{male} \\ y_1 < y_2 &\Rightarrow \text{female} \end{aligned}$$

Training system

The different training operations were executed on the SmartVeu and Arvei clusters. First, the optimum level of parallelism of the fann_train application was measured in different processing nodes and for different problem sizes. A general optimum level of 16 threads was found to be reasonable in most of the cases. Then, batch scripts were developed to systematize the different network parameter and feature vectors combinations into training jobs running concurrently on single nodes.

A total of 1332 training jobs were executed. The time it took for the different training instances to converge to a stationary classification error varied depending on the complexity of the network. The average execution time of each job was 6 hours.

4.3 Testing methodology

In order to assess the performance of the designed DNNs the dataset was classified using a simplified technique. The results of this classification were used as a baseline for comparing the DNNs' performance.

A second test was performed where the network has to identify the gender of multiple speakers using a continuous stream of data.

4.3.1 Baseline classification

A simple threshold function classification method based on F0 features was analyzed as a baseline for assessing the DNNs gender classification performance. The classification function consists of a threshold frequency f_{th} in which each F0 feature below this threshold will be classified as belonging to a male speaker and, if above, belonging to a female.

Both train and test F0 feature partitions were analyzed to find the best threshold frequency while taking into account the grouping of multiple adjacent frames. For a grouping of $G = 1$ frame, each frame F0 is evaluated by the threshold function. For a grouping of $G > 1$ frames, the average of a group of G adjacent frames are evaluated by the threshold function. The grouping mechanism overlaps each group by one frame, thus processing a set with the threshold function will result in the same number of outputs regardless the value of G.

The best threshold frequency was 151 Hz regardless of the dataset partition or the grouping length. Figure 4.1 shows the classification error of the threshold function for $f_{th} = 151$ Hz applied to the train and test F0 feature set. For $G = 1$ frame the baseline test classification error is $\varepsilon_b(G{=}1) = 11.06\%$, for $G = 11$ frames the baseline test classification error is $\varepsilon_b(G{=}11) = 9.52\%$. The train and test partitions were divided into male and female signals therefore, the ε_b value only reflects the ability to classify the data when a single gender is presented to the classifier.

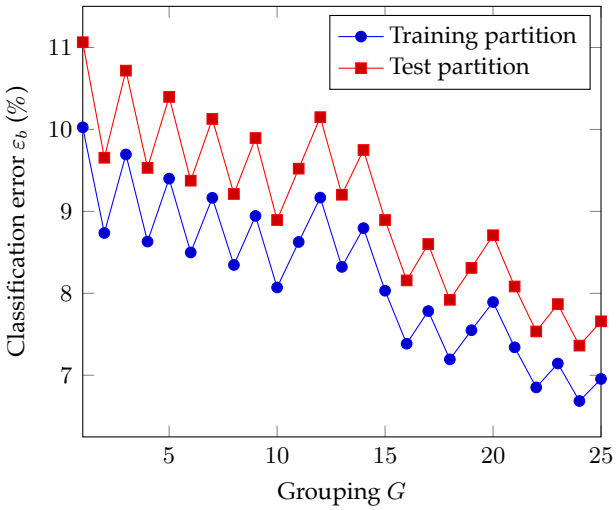


Figure 4.1: Threshold function classification error vs grouping ($f_{th} = 151$ Hz)

4.3.2 Continuous classification

The studied DNNs classification error ratio corresponds to the performance of the network when presented with a single feature vector. Any of those feature vectors represents only

11 audio frames adding up a total length of 110 ms. Supposing an application case such as speaker gender identification in a conversation, a single given speaker is likely to talk longer than 110 ms.

In order to test such situation, a test case scenario was set as follows: the test partition was divided into male and female signals. A new dataset was produced interleaving the male and female signals simulating a conversation. Finally, the feature vectors of the new dataset were calculated.

To simulate the conversation the male/female signals were interleaved using a random pattern based on the following distributions: according to [47] and [48] the conversational sentence length in words follow a normal distribution with sample mean $\overline{X}_l = 11.82$ and standard deviation $s_l = 14.01$. The speaking rate in words per second also follows a normal distribution with sample mean $\overline{X}_r = 3.7$ and standard deviation $s_r = 0.48$. Accordingly, the signal length for interleaving the male and female speech was calculated as

$$T = \frac{l}{r}, \quad l \sim N(\overline{X}_l, s_l), \quad r \sim N(\overline{X}_r, s_r) \tag{4.5}$$

where T is the signal length in seconds, l is the word-length of the sentence and r is the speaking rate both sampled from a normal distribution $N(\mu, \sigma)$.

4.4 Results

This section describes the voice gender classification results of the trained DNNs.

4.4.1 DNNs using F0 features

In order to find the best DNN designs, different parameters have been studied in isolation trying to find their optimal values. Some general parameters have been optimized and kept over the next DNN designs. Some parameters, such as the hidden layer sizes, have been assessed multiple times due to the high variability of the results when modifying other parameters.

Output units

Varying the number of output units between one and two delivers similar classification error performance. For Step or StepSym activation functions, DNNs with one output outperforms DNNs with two outputs by a relative ε improvement factor of 1.142 in average.

When switching the output unit activation function between Sigm or SigmSym and Step or StepSym functions, the former delivers a higher relative ε improvement factor of 1.331 in average.

The subsequent DNNs analyzed use one output unit with Sigm or SigmSym activation

functions.

Training algorithm

RProp algorithm delivers the best classification error compared with the training algorithms described in section 4.2. Table 4.1 shows the average performance comparison between the three training algorithms for DNNs with $O = 1$, $A_{out} =$ Sigm, $H \in S_H$, $A_{hid} =$ SigmSym and V equal to concatenations of F0, Δ_{F0} andΔ^2_{F0}.

Table 4.1: Comparison between training algorithms

Feature vector	ε by T (%)			RProp ε rel. improvement	
	Bprop	QProp	RProp	vs Bprop	vs Qprop
F0	13.14	9.50	7.64	1.72	1.243
F0+Δ_{F0}	11.82	11.16	7.48	1.56	1.492
F0+Δ_{F0}+Δ^2_{F0}	11.51	9.35	7.51	1.533	1.245
Average	12.16	10.00	7.54	1.612	1.326

Hidden units activation functions

The hidden unit activation functions can either be Sigm, SigmSym, Step or StepSym. Between Sigm and SigmSym, the latter achieves a better classification rate. Table 4.2 shows the average performance comparison between both activation functions for DNNs trained using RProp with $O = 1$, $A_{out} =$ Sigm, $H \in S_H$ and V equal to concatenations of F0, Δ_{F0} and Δ^2_{F0}.

Table 4.2: Comparison between hidden unit activation functions

Feature vector	ε by A_{hid} (%)		SigmSym ε rel. improvement
	SigmSym	Sigm	
F0	7.22	8.06	1.117
F0+Δ_{F0}	6.86	8.10	1.181
F0+Δ_{F0}+Δ^2_{F0}	6.76	8.26	1.221
Average	6.95	8.14	1.172

The same behavior is assumed for Step versus StepSym functions. These two functions are a special case of Sigm and SigmSym functions with an infinite steepness parameter. The

subsequently analyzed DNNs do not use the Sigm or Step function in their hidden units.

The classification error is increased when StepSym functions are used. However, the computational complexity of the DNN is largely reduced due to avoiding the computation of the SigmSym function. Following the same DNN designs in table 4.2, the networks using StepSym functions have an average relative ε loss factor of 1.097 compared to the networks using SigmSym functions.

Number of hidden units

The optimal number of hidden units depends on the feature vector used. However, it has been found to be constrained between 10 and 50 units per hidden layer. Figure 4.2 shows the classification error versus H for DNNs with A_{hid} = SigmSym, over the different F0 feature vectors.

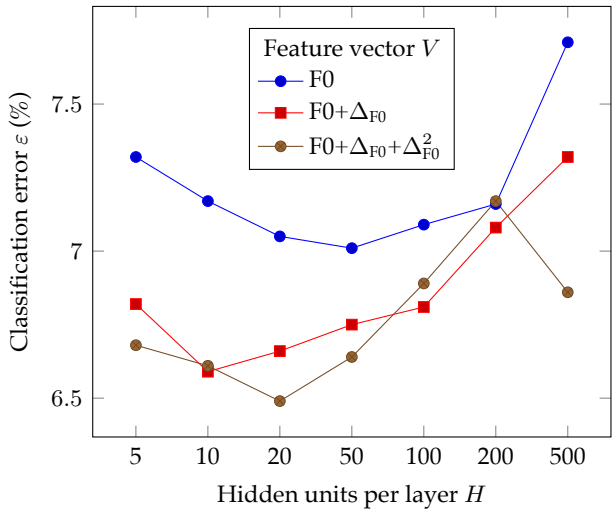


Figure 4.2: Classification error vs hidden units per layer with A_{hid} = SigmSym

DNNs using StepSym activation function show similar behavior regarding the optimal number of hidden units.

Feature vector

The addition of the delta coefficients to the feature vector results in an average relative ε improvement factor of 1.052 over the DNNs using only F0. The addition of the delta and delta-delta coefficients results in an average ε relative improvement factor of 1.068. However, these improvements depend on the number of hidden units. The networks with fewer hidden units are the most improved by the addition of the delta and delta-delta coefficients. Figure 4.3 shows this behavior.

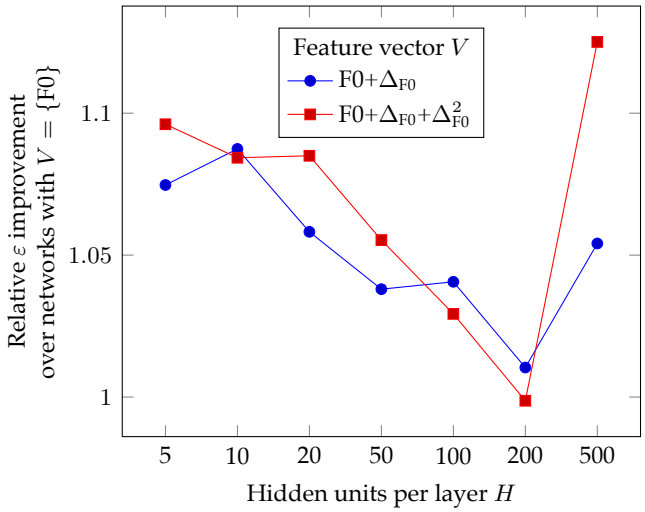


Figure 4.3: Classification error improvement over F0 vs hidden units per layer with A_{hid} = SigmSym

Best networks

The different training parameter analysis showed that the best performing networks are trained using RProp algorithm and have one output unit with sigmoid activation functions. Table 4.3 summarizes the best performing networks ordered by the input feature vector, the number of hidden units and the activation function of the hidden units. The best DNNs classified by the hidden units activation function are marked in bold.

Table 4.3: Best networks for F0 features

Feature vector	A_{hid}	H	ε (%)	$\varepsilon_b(G{=}11)/\varepsilon$
F0	SigmSym	50	7.01	1.358
	StepSym	10	**7.38**	1.23
F0+Δ_{F0}	SigmSym	10	6.59	1.445
	StepSym	100	7.44	1.28
F0+Δ_{F0}+Δ^2_{F0}	SigmSym	20	**6.49**	1.467
	StepSym	50	7.56	1.259

4.4.2 DNNs using MFCC features

The same general behavior observed for the DNNs using F0 features applies for the DNNs using MFCC features. The major differences are the higher number of units required in the hidden layers, the increased performance of other training algorithms different than RProp and the lack of improvement when adding the delta-delta coefficients. The number of units in the hidden layers have been found to be optimal in the range between 100 and 300 de-

pending on the input feature vector.

Best networks

The different training parameter analysis showed that the best performing networks have one output unit with sigmoid activation functions. Table 4.4 summarizes the best performing ordered by the input feature vector, the number of hidden units and the activation function of the hidden units. As can be seen in table 4.3, the DNNs using F0 features outperform the ones using MFCC features.

Table 4.4: Best networks for MFCC features

Feature vector	A_{hid}	T	H	ε	$\varepsilon_b(G{=}11)/\varepsilon$
MFCC	SigmSym	BProp	200	6.93	1.373
	StepSym	RProp	200	8.04	1.184
MFCC+Δ_{MFCC}	SigmSym	BProp	200	6.88	1.384
	StepSym	RProp	100	8.98	1.06

4.4.3 Combination of F0 and MFCC features

Two strategies have been attempted to combine the F0 and MFCC features in a DNN design. Firstly, training DNNs by combining F0 and MFCC data in a single feature vector. Secondly, joining two DNNs to calculate a linear combination of their outputs.

DNNs using F0 and MFCC features

Training a DNN with a feature vector combining F0 and MFCC does not diminish the classification error compared to the best networks described in section 4.4.1. Table 4.5 shows the best networks combining F0 and MFCC features, classified by the activation function of the hidden units.

Table 4.5: Best networks using F0 and MFCC features

Feature vector	A_{hid}	H	ε (%)	$\varepsilon_b(G{=}11)/\varepsilon$
F0+Δ_{F0}+MFCC	SigmSym	100	6.78	1.404
F0+Δ_{F0}+MFCC	StepSym	100	6.96	1.368

Linear combination of DNNs

This approach consists of combining two DNNs using a weighted sum of their outputs. Different network pairs were studied in order to find the best weight combination which yields the smallest classification error. Table 4.6 shows the best network combinations using this approach. As expected, the best combinations occur when joining the best overall networks (as showed in tables 4.3 and 4.6) and when they combine F0 and MFCC features.

Table 4.6: Best network combinations

Parameters	DNN1	DNN2	Combined
Feature vector	F0+Δ+Δ^2	MFCC+Δ	
A_{hid}	SigmSym	SigmSym	
H	100	200	
Best weights	0.5	0.51	
ε (%)	6.49	6.88	5.77
$\varepsilon/\varepsilon_{combined}$	1.124	1.192	
$\varepsilon_b(G{=}11)/\varepsilon$	1.467	1.383	1.65
Feature vector	F0	MFCC	
A_{hid}	StepSym	StepSym	
H	100	200	
Best weights	0.5	0.51	
ε (%)	7.38	8.04	7.07
$\varepsilon/\varepsilon_{combined}$	1.044	1.137	
$\varepsilon_b(G{=}11)/\varepsilon$	1.29	1.184	1.347

4.4.4 Network post-processing

A method for increasing the gender identification performance on continuous classification is to calculate a moving average of the network outputs over M adjacent audio frames. The value of M depends on how long is a speaker talking until another speaker of opposite gender answers. For instance, if there is only one speaker, the bigger M is, the higher the performance due to the slow response to changes in the moving average. Conversely, when there are multiple alternate gender speakers, high values of M can lead to poor results.

Table 4.7 shows the gender identification performance of the best networks when applying a moving average of length M on their outputs under the continuous classification test conditions.

Table 4.7: Network post-processing performance

DNN parameters		ε (%)		M	Mov. avg. ε rel. improvement
		No process	Moving average		
V	F0+Δ_{F0}+Δ^2_{F0} & MFCC+Δ_{MFCC}	5.77	3.05	42	1.892
A_{hid}	SigmSym				
H	100 & 200				
V	F0 & MFCC				
A_{hid}	StepSym	7.07	3.54	41	1.997
H	100 & 200				
V	F0+Δ_{F0}+MFCC				
A_{hid}	SigmSym	6.78	4.07	38	1.666
H	100				
V	F0+Δ_{F0}+MFCC				
A_{hid}	StepSym	6.96	3.7	46	1.881
H	100				

Chapter 5

Limited precision ANNs

5.1 Fixed-point ANNs

This section describes the use of fixed-point arithmetics in order to optimize the ANN computations. The objective hardware, where the ANNs will be calculated, is the Zynq ZC702 device processing system (PS) which features an ARM A9 cortex dual core CPU.

The motivation behind using fixed-point arithmetics is because of the higher throughput of integer arithmetics compared to floating point. Figure 5.1 shows a benchmark comparing arithmetic operations for various integer and floating-point data types. The ZC702 PS exhibits a higher throughput in integer arithmetics than the same data bit-length floating-point operations for additions and multiplications.

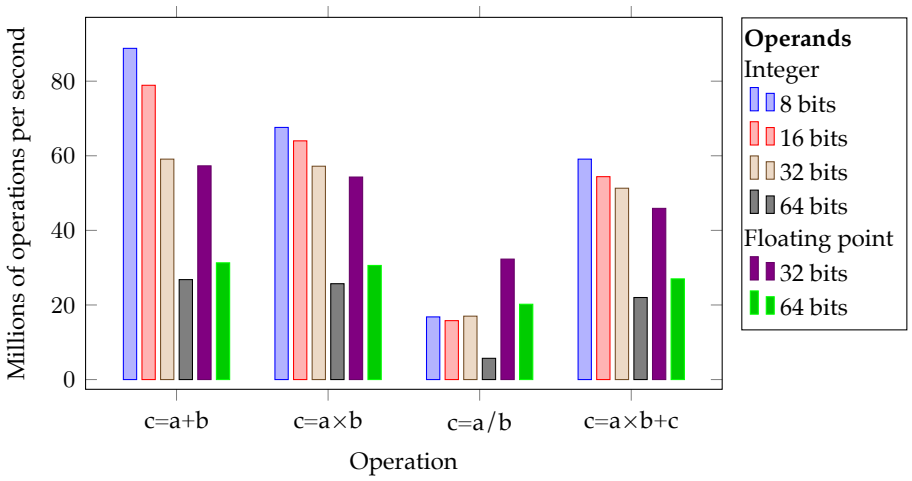


Figure 5.1: Zynq ZC702 PS arithmetic benchmark

Calculating any of the designed ANNs implies a number of multiply-accumulate operations for the computation of the weight matrix which is bounded by $O(3H^2)$, where H is the number of units in a hidden layer. Therefore, it is expected that the use of fixed-point arithmetics implemented over the architectures' integer operations will yield a higher throughput than a floating-point implementation.

Fixed-point notation

Signed fixed-point numbers format will be notated as Q$m.f$ where m is the number of bits on the left of the decimal point including the sign bit and f is the number of bits after the decimal point. These values will be also notated as $Q_m = m$ and $Q_f = f$.

A Q$m.f$ signed fixed-point number has a decimal range of $[-2^m, 2^m - 2^{-f}]$ and its resolution is 2^{-f}.

5.1.1 ANN precision

The designed ANNs detailed in chapter 4 were implemented using signed 32-bit floating data operations. In order to implement the ANN computation algorithm in fixed point, each operation's dynamic range was evaluated.

The ANN algorithm was instrumented in order to obtain the numeric range of every variable involved in the computation. Then, the different ANN designs were run to obtain their maximal ranges. Finally, the bit-length of the fixed-point number able to contain the maximal range of each operation was calculated.

The calculation of a network's layer, as described in section 2.3.3, can be expressed by the following algorithm

```
for (i = 0; i < |L_K|; ++i) {
    sum = 0.0;
    for(j = 0; j < |L_K - 1|; ++j) {
        sum += in[j] * w[i][j];
    }
    sum += w[i][|L_K|]; //Bias
    out[i] = f(sum); //Activation function
}
```

where $in[\cdot]$ and $out[\cdot]$ are the layer's input and output vectors, $w[\cdot][\cdot]$ is the weights matrix, $|L^K|$ is the layer's number of units, $|L^{K-1}|$ is the preceding layer's number of units and $f(\cdot)$ is the activation function. Figure 5.2 represents a flow diagram of the layer computation algorithm showing the fixed-point unsigned bit length b_{var} of each variable able to contain the maximal range for the ANN computation.

From the last observation, one can assume that the fixed-point implementation will need a data-type of, at least, 43 bits to meet the required precision. However, ANN computations were found to be resilient to precision losses. In order to determine whether the networks could be computed with less precision, the only metric taken into account was the classification error. The best networks from sections 4.4.1 and 4.4.2 were tested using different fixed-point formats Q$m.f$, where $m + f = 32$, in order to measure classification performance loss. Whereas the fixed-point numbers may be stored in 32 bits, the arithmetic operations were executed in 64 bits in order to prevent overflow. The activation functions were calculated in floating point and converted back to fixed point.

Figure 5.3 shows the relative average classification error loss factor between the floating-point ANN implementations and the fixed-point implementations using various Q$m.f$ for-

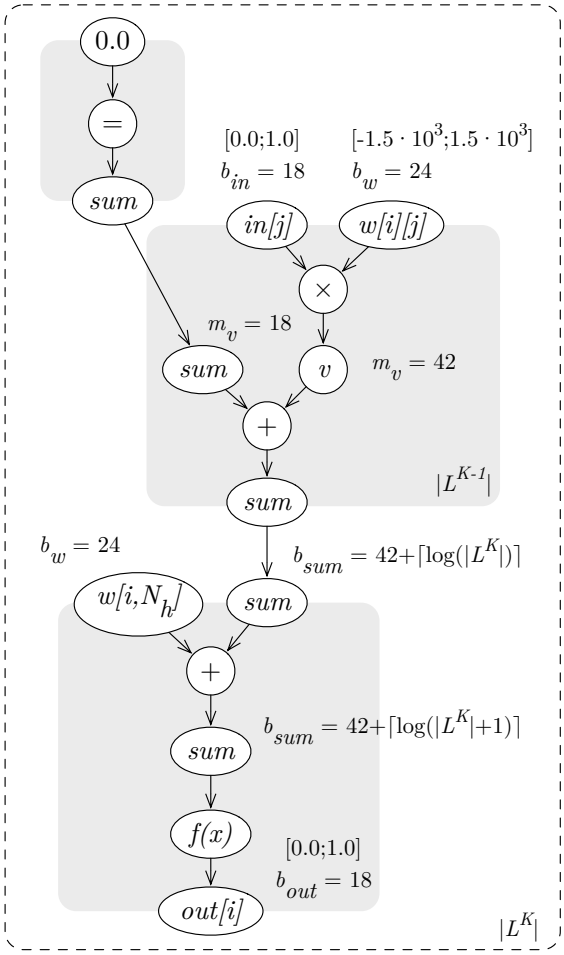


Figure 5.2: Flow diagram of the network layer computation algorithm

mats. From these observations, one can see that networks using 32-bit fixed-point arithmetics yields results comparable to floating-point implementations. For networks using symmetric sigmoid activation functions, the best fixed-point format is Q18.14 with an average classification error loss factor of 1.00182. For networks using symmetric step functions, the best fixed-point format is Q19.13 with an average classification error loss factor of 1.00006. However, the latter has almost the same performance for the formats from Q17.15 to Q22.10.

5.1.2 Fixed-point implementation

The implementation of the fixed-point ANN algorithm is analogous to the floating-point implementation. The only difference is that the multiply-add operation to calculate each unit net input has to be implemented as

```
...
    sum += (in[j] * w[i][j]) >> Q_f;
...
```

in order to keep the number format after the multiplication. That is because a multiplication

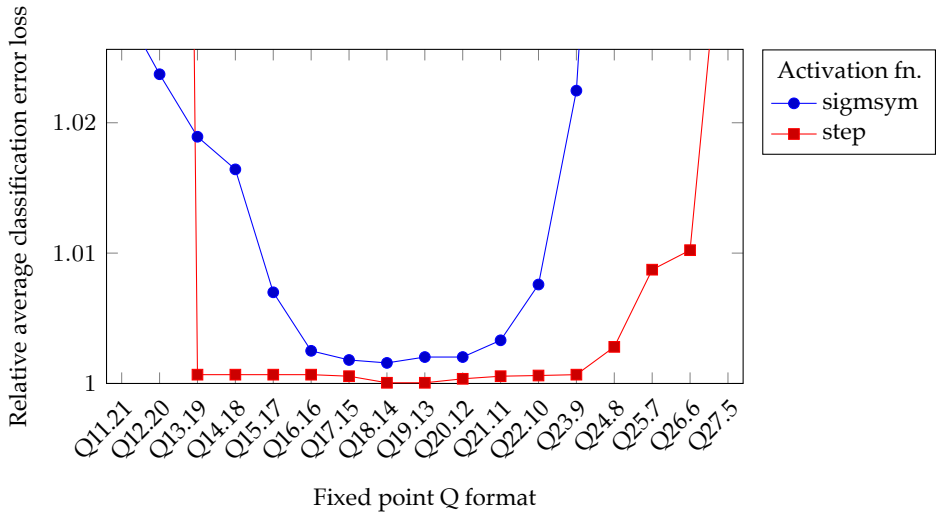


Figure 5.3: Relative average ANN precision loss factor for fixed-point implementations vs fixed-point format

of two $Qm.f$ fixed-point numbers results in a $Q2m.2f$ number. Therefore, a fixed-point implementation using $Qm.f$ numbers where $m + f = 32$ may use 32-bit addition operations but needs 64-bit multiplications to avoid overflow.

The multiplication operation is prone to overflow. As well, the expansion of the summation, which calculates each unit's net input, can also result in overflow. Therefore, 64-bit operations should be used throughout the algorithm to obtain the classification error performance described in the previous section. To minimize the number of 64-bit operations, the algorithm has been modified to implement saturated addition and multiplication. The multiplications are still executed using 64 bits but their results are saturated to the 32-bit $Qm.f$ range. The additions are executed using only 32 bits also saturating the results.

The saturated multiplications are implemented according to the following C algorithm

```
int32 sat_mulfps32b(int32 x, int32 y) {
    int64 res = (int64) x * (int64) y;
    res = res >> Q_f;

    //Calculate overflowed result. INT_MAX= 2^31 - 1
    uint32 res2 = ((uint32) (x ^ y) >> 31) + INT_MAX;

    int32 hi = (res >> 32);
    int32 lo = res;
    if (hi != (lo >> 31)) res = res2;

    return res;
}
```

and the saturated sums are implemented according to:

```
int32 sat_adds32b(int32 x, int32 y) {
    uint32 ux = x;
    uint32 uy = y;
    uint32 res = ux + uy;

    //Calculate overflowed result. INT_MAX= 2^31 - 1
    ux = (ux >> 31) + INT_MAX;

    //Force compiler to use conditional movs
    if ((s32b) ((ux ^ uy) | ~(uy ^ res)) >= 0) {
        res = ux;
    }

    return res;
}
```

Precision of the implementations

Implementations using saturated 64-bit multiplications and saturated 32-bit additions yield classification errors of the same order of magnitude than the 64-bit implementations (fig. 5.3). The best $Qm.f$ formats remain unchanged from the ones discussed in section 5.1.1. Additionally, the performance for the best Q formats is equal to the 64-bit implementations. Non-saturated implementations using 64-bit multiplications and 32-bit additions have a smaller computational cost at the expense of a slightly higher classification error. Table 5.1 shows the relative average classification error loss factor for the best Q formats and the different implementations. According to these results, the fixed-point implementation which does not use saturated arithmetic is considered to be precise enough and will be used by default.

Table 5.1: Fixed-point best implementations average relative classification error compared to 32-bit floating-point ones

Activation fn. & format	Relative avg. classification error loss compared to 32-bit float		
	64b mul & add	64b sat. mul & 32b sat. add	64b mul & 32b add
SigmSym Q18.14	1.00182	1.00182	1.00262
StepSym Q17.15	1.00006	1.00006	1.00047

5.2 Activation functions optimization

The calculation of the activation functions may be computationally expensive. Each unit of the network (except the input units) must evaluate its activation function. That implies that computational complexity is linearly proportional to the network's size. This section discusses the optimization of the activation functions referred in chapter 2 section 2.3.2.

5.2.1 Step functions

The first trivial optimization of an ANN is to use the step or symmetric step functions. Those functions are computed using a single sign compare operation. Furthermore, each unit calculation can be stored into a single bit value.

Using the step or symmetric step functions usually implies a tradeoff between computational cost and accuracy. Networks using those functions tend to perform worse than ones using sigmoid or other continuous non-linear functions. Moreover, the training process may become harder and take longer to converge to the required results. Regarding the networks described in chapter 4, the average relative performance loss factor of the networks using StepSym versus SigmSym functions is 1.097.

5.2.2 tanh approximations

The evaluation of the symmetric sigmoid or tanh function accounts for a substantial part of the ANN computation. This section presents various approximations of the tanh functions which can be evaluated with a lower cost.

The tanh function may be expressed using the exponential function as

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{5.1}$$

and some of the approximations are based on the following properties:

$$\tanh(-x) = -\tanh(x) \tag{5.2}$$

$$\lim_{x \to 0} \tanh(x) = x \tag{5.3}$$

$$\lim_{x \to \pm\infty} \tanh(x) = \pm 1 \tag{5.4}$$

Squash function

The first approximation is the "squash" or fast sigmoid function first proposed by [49]. This function can be expressed as:

$$f_{sqsh}(x) = \frac{x}{1 + |x|} \tag{5.5}$$

The maximal error between the squash and the tanh function is 3.063×10^{-1} at $|x| = 1.616$ and the average absolute error for $x \in [-5, +5]$ is 2.197×10^{-1}. Figure 5.4 shows the squash function compared to tanh and the absolute difference between both.

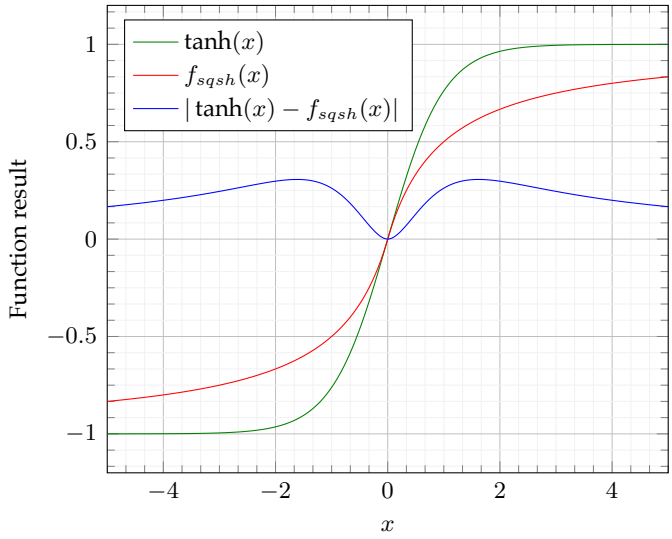


Figure 5.4: squash vs tanh function

Piecewise tanh

A second approximation is to use a piecewise function which lineally interpolates various segments of the tanh function. The domain of the function is divided into three intervals $(-\infty, r_0)$, $[r_1, r_K)$ and $[r_K, +\infty)$. The approximation assumes $\tanh(x) \simeq -1$ or $\tanh(x) \simeq +1$ for the first and third interval by eq. 5.4. The second interval is divided in $K-1$ equally sized subintervals in which the linear interpolation of the tanh function is evaluated.

The piecewise interpolated tanh approximation may be expressed as

$$f_{pcw}(x) = \begin{cases} +1, & x > r_1 \\ \text{lint}(x, r_1, r_2), & r_2 < x \leq r_1 \\ \quad\vdots & \quad\vdots \\ \text{lint}(x, r_{K-1}, r_K), & r_K < x \leq r_K \\ -1, & x \leq r_K \end{cases} \tag{5.6}$$

where $\text{lint}(x, r_a, r_b)$ is the linear interpolation between $\tanh(r_a)$ and $\tanh(r_b)$ at point $r_a < x \leq r_b$.

$$\text{lint}(x, r_a, r_b) = \tanh(r_a) + (\tanh(r_b) - \tanh(r_a)) \frac{x - r_a}{r_b - r_a} \tag{5.7}$$

For $K = 6$, $r_1 = -2.647$ and $r_6 = 2.646$, the maximal error is 6.64×10^{-2} at $|x| = 0.979$ and the average absolute error for $x \in [-5, +5]$ is 1.406×10^{-2}. Figure 5.5 compares the defined function to tanh.

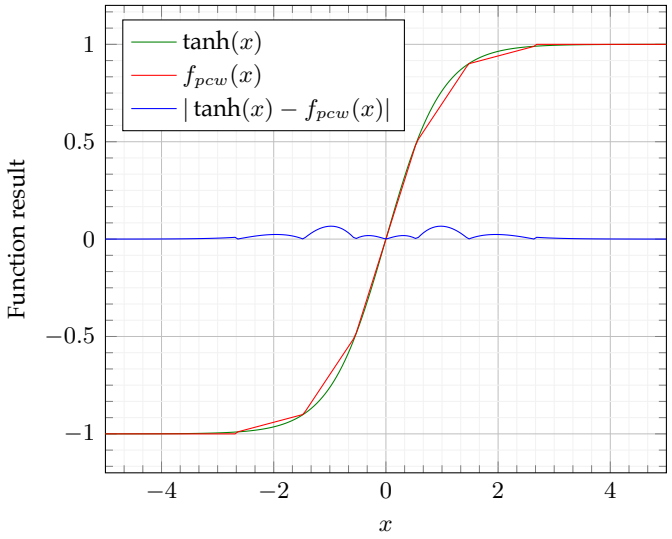


Figure 5.5: Piecewise interpolated tanh vs tanh function. $K = 6$, $r_1 = -2.647$ and $r_6 = 2.646$.

Taylor series

A third approximation is to use a limited number of terms of the tanh Taylor series' expansion. However, the convergence interval of the tanh Taylor series is (-π/2,π/2) and the approximation error around these bounds is excessive, even with a high number of terms. The proposed approach uses the exponential function Taylor series to approximate the tanh function as expressed by 5.1. The K-term Taylor series expansion of the exponential function is defined as

$$\exp_{tylr}(x) = \sum_{k=0}^{K} \frac{x^k}{k!} \tag{5.8}$$

The exponential function Taylor series' converges for any real value. However, the expansion error is always higher for negative values of x, especially when the expansion has fewer terms. To improve the accuracy, the proposed approximation takes advantage of the odd symmetry (eq. 5.2) of the tanh function as

$$f_{tylr}(x) = \begin{cases} \dfrac{2}{1 + \exp_{tylr}(-2x)} - 1, & x < 0 \\ \\ \dfrac{-2}{1 + \exp_{tylr}(2x)} + 1, & x \geq 0 \end{cases} \tag{5.9}$$

For $K = 3$ terms, the approximation maximal error is 4.841×10^{-2} at $|x| = 1.615$ and the average absolute error for $x \in [-5, +5]$ is 2.495×10^{-2}. Figure 5.6 compares the previous approximation to tanh.

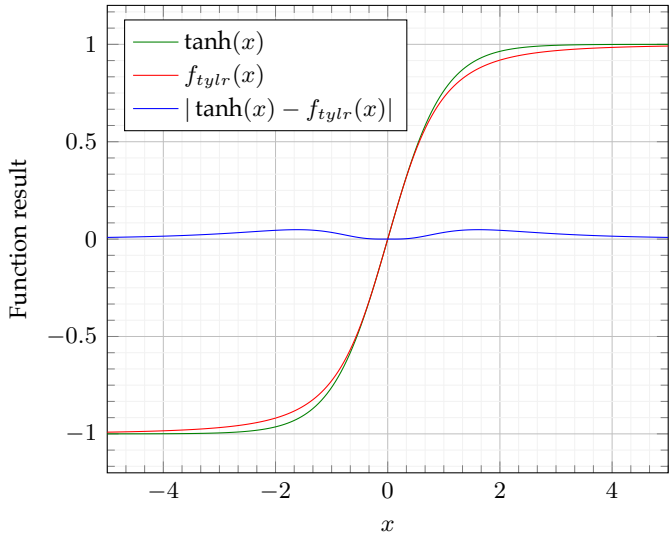


Figure 5.6: Taylor series approximation vs tanh function. $K = 3$ terms.

5.2.3 tanh lookup table

Another tanh approximation is to use a lookup table (LUT), which stores values of the function for specific values of x. This section adopts the tanh LUT design of [50] and discusses its precision.

According to the odd symmetry defined by eq. 5.2, it is only necessery to store the values of $\tanh(x)$ for $x \geq 0$. From eq. 5.3, $\tanh(x) \simeq x$ can be assumed for small values of x. Similarly, $\tanh(x) \simeq \pm 1$ for large values of x can assumed by eq. 5.4.

From the previous observations, a lookup table with a maximum error ϵ can be defined storing the values of $\tanh(x)$ for $\alpha \leq x \leq \beta$ where α and β are limiting values derived from the error value. The α value can be obtained from eq. 5.3 and 5.2 such as

$$|\tanh(\alpha) - \alpha| \leq \epsilon \tag{5.10}$$

Similarly, the β value can be derived from eq. 5.4 and 5.2 such as

$$1 - |\tanh(\beta)| \leq \epsilon \tag{5.11}$$

The α and β values can be determined given a value of ϵ based on the following:

1. The inequality 5.10 can be solved by approximating $\tanh(\alpha)$ by its Taylor series' expansion limited to a few terms. Then the value of α can be obtained by finding the roots of the resulting polynomial.

2. The inequality 5.11 can be solved as $\beta \geq \tanh^{-1}(1 - \epsilon)$ for $\epsilon \in (0, 2)$.

Lookup table definition

Once known the α and β limits for a given maximum error ϵ, a lookup table of size N can be built defining N adjacent subdomains S_i from the interval $A = [\alpha, \beta)$. The proposed solution divides A into $N + 1$ evenly spaced points such as

$$\begin{cases} a_i & = \alpha + i\dfrac{(\beta - \alpha)}{N} \\ S_i & = [a_i, a_{i+1}) \end{cases}, \quad 0 < i < N - 1 \tag{5.12}$$

Then, the lookup table is built evaluating the tanh function for the central point of each subdomain as

$$y_i = \tanh\left(\frac{a_i + a_{i+1}}{2}\right) \tag{5.13}$$

Finally, the lookup function is defined as

$$\text{lut}(x) = \begin{cases} x, & |x| < \alpha \\ \text{sgn}(x), & |x| \geq \beta \\ y_i \text{sgn}(x), & \{y_i \,|\, |x| \in S_i\}, \quad \text{otherwise} \end{cases} \tag{5.14}$$

where $\text{sgn}(\cdot)$ is the sign function

$$\text{sgn}(x) = \begin{cases} +1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \tag{5.15}$$

To ensure that $\text{lut}(x)$ has a maximum error value of ϵ, the subdomain division must fulfill that $|\tanh(x) - y_i| \leq \epsilon$ for any value of x inside the subdomain S_i. This condition may be expressed as

$$|\tanh(a_i) - y_i| \leq \epsilon \wedge |\tanh(a_{i+1}) - y_i| \leq \epsilon \tag{5.16}$$

Fixed-point lookup table design

The linear division of the lookup interval A could be used to optimize the lookup function $\text{lut}(x)$ by defining a mapping of the floating-point x value to addresses of the lookup table.

The $\tanh(x)$ lookup table, for a fixed-point representation $Q_m.Q_f$ of the x value, is defined as follows:

1. After defining a desired maximum error ϵ and a lookup range $[\alpha, \beta)$, the bottom limit of the range is substituted by their closest fixed-point representations α_{fp} conforming $\alpha_{fp} \leq \alpha$.

2. A suitable subdomain division N is chosen such as N is a power of two and $\log_2(N) \leq$

Q_F (N is the size of the lookup table).

3. The subdomain limits definition (eq. 5.12) is modified such as

$$a_i = \alpha_{fp} + i2^{-Q_f+L} \tag{5.17}$$

for $L \geq 0$ where 2^{-Q_f+L} is the step or distance between two adjacent subdomains. Now the value of a_N defines the upper limit of the lookup range and $\beta_{fp} = a_N$ is set accordingly. Note that β_{fp} may be bigger than β, which would then make it necessary to choose a combination of N and L values that results in α_{fp} and β_{fp} values as close to α and β as possible.

4. The lookup table is built following the procedure defined in the last section and fulfilling the condition of eq. 5.16. If this condition is not met, it would be necessary to use a smaller error value ϵ, a smaller L value or a bigger lookup table size N.

Now the lookup table entry for a given input value x can be calculated using a subtraction and a left shift.

$$i = \frac{(x - \alpha_{fp})}{2^L} \tag{5.18}$$

The fixed-point lookup algorithm may be expressed in C code as follows:

```
//Lookup table
fixed_t lut[N] = { ... };

//Lookup function. α_fp, β_fp, Q_f, N and L are constants
fixed_t tanh_lut(fixed_t x) {
    fixed_t xabs;
    unsigned char sign;
    unsigned int i;

    sign = (x < 0);
    xabs = (sign ? -x : x);

    if (xabs < α_fp)
        return x;
    if (xabs >= β_fp)
        return (sign ? 1 << Q_f : -1 << Q_f);

    i = (unsigned int) (xabs - α_fp) >> L;

    return (sign ? lut[i] : -lut[i]);
}
```

Table 5.2 shows a lookup table with parameters $\epsilon = 0.08$, $N = 8$, $L = 13$ for Q18.14 fixed-point format. The lookup interval is from $\alpha = 0.655$ to $\beta = 1.589$ and their fixed-point counterparts are $\alpha_{fp} = 0.625$ and $\beta_{fp} = 1.625$. Figure 5.7 shows the lookup table compared to the tanh function and the absolute difference between both. The maximal error is 4.03×10^{-1} for $|x| = \alpha_{fp}$ and the average absolute error for $x \in [-5, +5]$ is 1.224×10^{-2}.

Table 5.2: tanh fixed-point lookup table example. $\epsilon = 0.08$, $N = 8$

LUT index i	Limits of S_i subdomains		lut(x)	Max. error
	a_i	a_{i+1}		
0	0.625	0.750	0.595	4.027×10^{-2}
1	0.750	0.875	0.700	3.438×10^{-2}
2	0.875	1.000	0.733	2.885×10^{-2}
3	1.000	1.125	0.785	2.386×10^{-2}
4	1.125	1.250	0.829	1.95×10^{-2}
5	1.250	1.375	0.864	1.578×10^{-2}
6	1.375	1.500	0.892	1.266×10^{-2}
7	1.500	1.625	0.915	1.01×10^{-2}

5.3 Results

This section describes the performance and accuracy results of the discussed designs. All tests were performed using the Zynq ZC702 device processing system.

5.3.1 Fixed point ANNs performance

The performance of the fixed-point implementation was tested using the candidate networks from chapter 4. Timing evaluation was performed using the whole test dataset (section 4.2.1) calculating the average ANN execution time. The code was instrumented using Linux clock() system call to perform the timing measurements. A second pass of the tests was performed with code instrumented to measure the power of the processing system (PS) detailed in appendix B. The results of the second test were joined with the timing measurements in order to estimate the energy use of the algorithms.

The tests compare between the 32-bit floating-point ANN and the non-saturated 32-bit addition and 64-bit multiply fixed-point ANN algorithms. Fixed-point algorithms use the Q18.14 format for networks with SigmSym activation functions and Q19.13 for networks with StepSym functions. The StepSym function is implemented using the same code in both algorithms. The SigmSym function is implemented using the standard C float expf() according to eq. 5.1. The fixed-point algorithm has to convert the net output of each unit to floating-point to calculate the activation function. Then, transform the result back to fixed-point. The conversion is necessary to compare the algorithm performance's while keeping numerical precision because there is no fixed-point SigmSym precise implementation (section 5.3.2 discusses the implementation of approximated fixed-point SigmSym functions). This may provoke some performance skew for ANNs with fewer hidden units due to a higher code

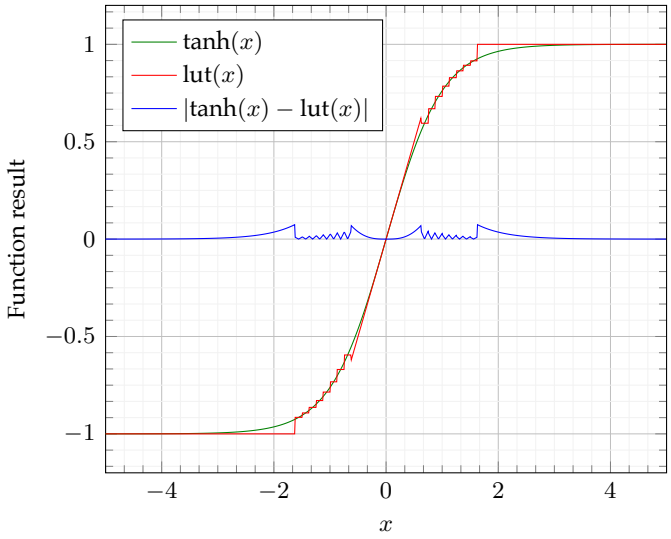


Figure 5.7: tanh fixed-point lookup table vs tanh function. $\epsilon = 0.08$, $N = 8$

fraction dedicated to calculating the activation functions.

Table 5.3 shows the algorithm comparison for the best networks from sections 4.4.1, 4.4.2, and 4.4. It compares the average network execution time, the Zynq ZC702 processing system energy use and the classification error between both implementations. The results show that fixed-point implementations yield superior timing results while using a similar amount of power. Thus, fixed-point has a smaller energy use than floating-point implementations. The timing speed-up is similar to the energy efficiency gain. The classification error increment is consistent with the results discussed in section 5.1.2 and it is considered non-significant for the purposes of the ANN.

5.3.2 Activation functions performance

This section discusses the classification error and performance of the ANNs implemented using the approximated SigmSym functions.

Approximated functions classification error

The use of the different activation function approximations increases the network classification error. In order to quantify this effect, the best networks of sections 4.4.1 and 4.4.2 were tested using the proposed SigmSym approximations.

Table 5.4 shows the average classification error ε loss factor of the different approximations compared to the precise SigmSym implementation in 32 bits floating point. Both piecewise tanh and Taylor approximations yield similar results whose losses approximately halves when increasing the K parameter. The piecewise function with $K = 6$ divisions and the Taylor approximation with $K = 3$ terms are considered to produce acceptable results.

Table 5.3: Floating-point vs fixed-point ANN performance comparison

	Network properties					
Feature vector	F0+Δ_{F0}+Δ^2_{F0}	MFCC+Δ_{MFCC}	F0	MFCC	F0+Δ_{F0}+MFCC	
Hidden units H	20	200	10	200	100	
Activation fn. A_{hid}	SigmSym	SigmSym	StepSym	StepSym	SigmSym	StepSym
	Average Execution time (s)					
Floating-point	2.318×10^{-5}	1.793×10^{-2}	5.544×10^{-6}	1.196×10^{-2}	1.112×10^{-3}	8.788×10^{-4}
Fixed-point	1.399×10^{-5}	1.072×10^{-2}	3.959×10^{-6}	8.435×10^{-3}	6.661×10^{-4}	6.188×10^{-4}
Speed-up	1.658	1.673	1.401	1.419	1.669	1.420
	PS energy use (mW·s)					
Floating-point	2.773×10^{-2}	21.42	6.497×10^{-3}	14.01	1.33	1.03
Fixed-point	1.681×10^{-2}	12.89	4.644×10^{-3}	9.591	8.006×10^{-1}	7.049×10^{-1}
Efficiency gain	1.649	1.664	1.397	1.459	1.661	1.459
	Classification error ε					
Floating-point	6.493	6.884	7.382	8.04	6.782	6.961
Fixed-point	6.524	6.913	7.391	8.053	6.813	6.982
Relative loss	1.004	1.004	1.001	1.002	1.005	1.003

Table 5.4: Approximated activation functions classification error

Function	Approximation fn. error		Avg. Rel. ε loss
	Maximum	Abs. avg. $x \in [-5, 5]$	
Floating point 32 bits			
$f_{sqsh}(x)$	3.063×10^{-1}	2.197×10^{-1}	1.219
$f_{pcw}(x)$ $K = 6$	6.64×10^{-2}	1.406×10^{-2}	1.018
$f_{pcw}(x)$ $K = 12$	3.497×10^{-2}	7.401×10^{-3}	1.007
$f_{tylr}(x)$ $K = 3$	4.841×10^{-2}	2.495×10^{-2}	1.025
$f_{tylr}(x)$ $K = 6$	4.442×10^{-3}	2.081×10^{-3}	1.002

Execution time

The performance of the described activation functions was tested using the following benchmark: the program calculates the activation functions for $x \in [-5, 5]$ using 100 different pseudorandom numbers. The performance metric is the average running time of each function over 10^9 executions. The code is instrumented using the Linux clock() system call to measure the total execution time. A second execution was performed with power measurement code instrumentation.

The performance test includes measures for 32-bit floating-point and Q18.14 fixed-point implementations of the symmetric sigmoid activation functions. The performance baseline is the evaluation of the tanh function according to eq. 5.1 using the standard C expf() function. Table 5.5 shows the average execution time per function call and the speedup relative to the baseline. It also shows the power consumption differentiating between the board's total and that of the PS, the average PS energy use per function call and the PS efficiency gain relative to the baseline.

Table 5.5: Symmetric sigmoid activation functions performance

Function	Avg. execution time		Power		PS energy use	
	Time (ns)	Speed-up	Total (mW)	PS	Energy (mW·s)	Efficiency gain
Floating point 32 bit						
Baseline	286.1	-	2436	1196	3.423×10^{-4}	-
$f_{sqsh}(x)$	66.1	4.328	2398	1170	7.733×10^{-5}	4.426
$f_{pcw}(x)\ K=6$	67.2	4.257	2418	1182	7.946×10^{-5}	4.308
$f_{tylr}(x)\ K=3$	136.6	2.094	2418	1182	1.615×10^{-4}	2.119
Fixed point Q18.14						
$f_{sqsh}(x)$	120.2	2.38	2366	1173	1.411×10^{-4}	2.427
$f_{pcw}(x)\ K=6$	61.3	4.667	2340	1173	7.191×10^{-5}	4.760
$f_{tylr}(x)\ K=3$	214.1	1.336	2398	1202	2.573×10^{-4}	1.330
lut$(x)\ N=8$	32.6	8.776	2427	1208	3.940×10^{-5}	8.688
lut$(x)\ N=1024$	32.7	8.749	2371	1207	3.946×10^{-5}	8.673

For the floating-point implementations, the piecewise sigmoid function has the best trade-off between precision and execution time. For the fixed-point ones, the lookup table yields the best performance results with a clear advantage over the others due to its constant execution time regardless of the LUT size (N). This makes it possible to increase the precision while maintaining the performance. The energy efficiency gain is closely related to the execution time speed-up, however, functions with a higher count of memory accesses have a

slightly higher power rating.

Activation functions vs network size

The computation of an ANN is comprised of the matrix weight multiplication and the calculation of the activation functions. The complexity of the matrix weight multiplication for the designed networks is $O(3H^2)$ where H is the number of hidden units per layer. However, the complexity of the activation function is $O(3H)$. This means that the optimizations of the activation functions would have a diminishing impact on the overall performance as the number of hidden units grow.

A performance test was performed to compare the execution time of an ANN using various activation functions when varying the number of hidden units. The tested ANN has 33 inputs, 3 hidden layers with H units per layer and 1 output unit. The test was performed using 32-bit floating-point and Q18.14 fixed-point implementations of the network. The best activation functions were compared for both implementations using $K = 6$ for the piecewise functions and $N = 1024$ for the lut functions. The baseline for comparison uses the standard C expf() function to calculate the symmetric sigmoid function.

Figure 5.8 shows the execution time speed-up compared to the baseline of the different activation functions. Fixed point implementations yield a superior performance over the floating-point ones by an average factor of 1.75. Additionally, the use of step functions provides the best performance, especially for the floating-point implementations.

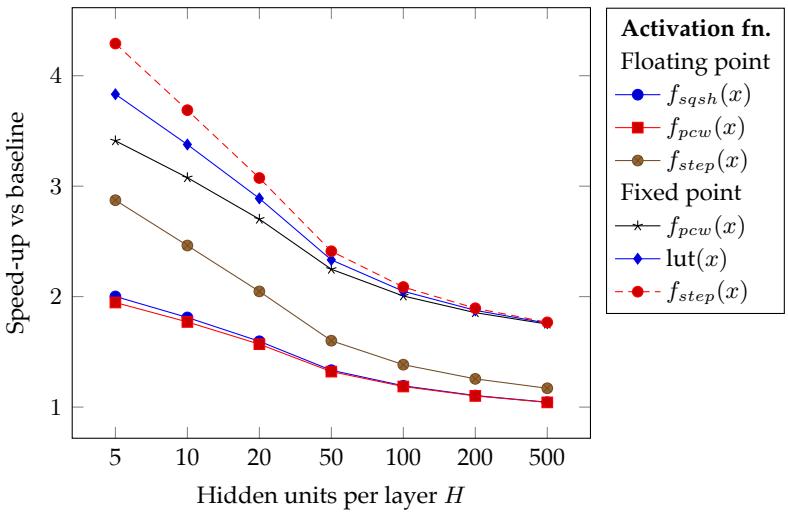


Figure 5.8: Activation functions vs hidden units

The behavior described in the figure has been also observed in ANNs using a different number of inputs. There were only marginal differences between the networks with less than 20 hidden units per layer. For the remaining cases, varying the number of inputs does not produce substantial differences in the results. Moreover, the energy efficiency gain of the different activation functions is almost identical to the speed-up that was observed in the previous section.

Chapter 6

FPGA ANN implementation

This chapter describes the design and implementation of the ANNs from chapter 4 on a FPGA device.

The ANN algorithms were implemented using C++ code, then synthesized into a register transfer level design (RTL) described in an HDL, and finally synthesized into the FPGA physical layout.

6.1 Experimental setup

The target architecture was Xilinx Zynq-7000 SoC and the specific device used for the implementation was the XC7Z020 chip. This hardware has been tested using the Xilinx ZC702 development board. The development tools used were Xilinx Vivado HLS and Xilinx SDSoC.

The testing procedure between the general architecture and FPGA performance consisted of the comparison of the execution times and energy efficiency between the processing system (PS) and the programmable logic (PL) of the Zynq ZC7Z020 device.

In order to measure the execution time and energy efficiency of the ANN algorithms running on the processing system, the following steps were followed:

1. The ANN algorithms under test were included in a test program which executes many instances of an ANN. The network input data is the testing partition used during the ANN training phase (section 4.2.1). The test program is instrumented to measure the execution time and energy rating of the algorithms under test. It also calculates the ANN outputs in order to check the accuracy of the computations.
2. The test program was implemented in C++ and compiled using the SDSoC sds++ compiler. The target architecture was an ARM A9 processor implemented by the ZC7Z020 PS. The compilation outcomes was an elf binary.
3. The test program was run on the XC7Z020 PS using the ZC7Z020 development board. The OS running on the board was Linux using the 3.19.0-xilinx-apf kernel.

4. Time execution measurements of the ANN algorithms were done using the sds_clk_counter() routine from the SDSoC development environment which counts the number of the PS clock cycles.

5. Power rating measurements of the ANN algorithms were measured by concurrently polling the ZC702 power controller while executing the algorithms under test. The estimation of the energy consumption was done by multiplying the power rating by the execution time. Further information about the power rating measurements can be found in the appendix B.

The measurements of the execution time and energy efficiency of the ANN algorithms running on the FPGA PL were done according to the following steps:

1. The ANN C++ algorithms under test were synthesized using Vivado HLS. During the development process, various optimization strategies were studied using the Vivado HLS pragma directives. The metrics for the comparison of different synthesis results were the execution latency and the resource usage of the FPGA.

2. The ANN C++ algorithms including the synthesis optimization pragmas were inserted into the test program, encapsulated into a single C function call named nn_5_run(). The test program was compiled using the SDSoC sds++ compiler marking the nn_5_run() function as synthesizable. This step generates the synthesis of the nn_5_run() and the FPGA bitstream implementing the synthesized function. As well, it generates an elf binary implementing the rest of the test program code interfaced with the FPGA accelerated nn_5_run() function.

3. The test program was run on the XC7Z020 PS and PL using the ZC7Z020 development board. The bitstream was loaded into the FPGA using the tools provided by the SDSoC development tools. The OS running on the board was the same than one used in the PS measurements.

4. Test executions were performed comparing the ANN results between the PS implementation and the FPGA accelerated ones. Any FPGA implementation which produced inconsistent results compared to the PS ANN algorithm was discarded.

5. Time execution and power rating measurements were measured following the same methodology than the PS ones. However, time execution measurements counted the FPGA execution plus the data transfer between the PS and the FPGA as the total execution time.

The versions of Vivado HLS and Xilinx SDSoC were 2015.4. The sds++ compiler and the synthesis tools were the ones bundled with those packages.

6.2 Design

The voice gender identification system is implemented using both the Zynq ARM A9 processing system (PS) and the FPGA device. The proposed design uses the ARM processor to handle the input and output of the data streams and the FPGA to accelerate the ANN calculations. The present work only implements the ANN calculations based on voice features which have been calculated previously. Future work for this project should be the design of the entire voice gender identification system implemented on the FPGA device.

ANN algorithm design

The base ANN algorithms were the same ones used in chapter 5. However, they were adapted to the HLS workflow. While the used HLS tools support a wide range of the C language, some operations, syntaxes and constructs are not supported or fail during the synthesis process.

The first relevant constraint of HLS designs is that arrays and other constructs must be of a fixed size. Additionally, it is not possible to dynamically allocate memory on a HLS function. The ANN algorithm uses dynamic memory to allocate the different data structures which store the required data and intermediate results of the network calculations. In order to avoid using dynamic memory, the ANN algorithm was implemented on a C++ template class which specifies fixed bounds for the input and output vectors, the weight matrix and the network data representation. The template class facilitates the implementation of different network designs while using the same code. This approach allows the synthesis tool to implement the whole ANN data structure using block RAM memories embedded in the FPGA thus, maximizing the performance of the design. The template class also allows the specification of different data types for the ANN data structures in order to provide a flexible way to modify the precision of the calculations.

A second design strategy was to implement the ANN weight matrix using ROM blocks. The Zynq 7000 SoCs does not feature any ROM device nor the ZC702 board. However, static arrays can be synthesized into ROMs implemented on the FPGA's block RAMs or as a distributed ROM made of the LUTs from several slices. The block RAM approach requires synchronous access to the data and makes use of the limited on-chip memory resources to store static data. On the other hand, the distributed LUT ROM is purely combinatorial and may be accessed asynchronously. However, the weight matrices dimensions make the LUT ROM approach impractical due to the high count of slices required which results in a latency higher than that of the block RAM.

A major drawback of this approach is that the network size is limited to the number of memory available inside the FPGA. Networks with more than 100 hidden units per layer did not fit inside the memory limits of the XC7Z020 chip.

Another requirement of the design is to provide an interfacing method between the FPGA and the ARM PS. The PS provides the FPGA with the ANN input vector and has to get the output vector back from the FPGA. The HLS design of this data passing interface is a function call whose arguments represent the data transferred between both devices. The FPGA has to access to external shared memory and copy the input data to local storage. Likewise, once the computation is finished it has to store the results back into external memory. The used HLS tools provide various interfaces optimized to achieve high data transfer rates between the PS and the FPGA, however, access to memory is a serious bottleneck of the design due to access latency and transfer speed.

The designed ANN have a number of inputs ranging from 10 to 858 and one single output. Every input or output value is stored as a 32-bit scalar. These numbers make it impractical to process single instances of an ANN due to the high latency of the data transferring compared to that of the ANN computation. Accordingly, the proposed design transfers N data frames in order for the FPGA to process many instances of the ANN in a single data transfer.

6.3 Optimizations

This section describes the HLS optimizations applied to the ANN algorithm. The performance assessment was done using the results of the synthesis executed by Vivado HLS. These results, showing the calculation latency and resource usage are provided for each optimization step. In order to establish a common base for the optimizations comparison, ANNs with the following parameters will be tested:

- Input units (IN): 308
- Hidden layers: 3
- Hidden units per layer (HN): 100
- Output units (ON): 1
- Output layer activation function: Sigmoid

The number of frames per FPGA processing instance will be $N = 1024$. The datatypes and hidden unit's activation functions will be switched between 32-bit floating point, 32-bit fixed-point, SigmSym and StepSym in order to provide performance results based on those parameters. The SigmSym function will be the piecewise version in the floating-point implementations and the LUT approximation from section 5.2.3 in the fixed-point ones. Finally, the synthesis clock will be 100 MHz.

In order to assess the synthesis results, the following parameters will be considered:

- Total latency: this is the number of cycles required to load the input vectors, compute N ANNs and write the output vectors.

- ANN latency: this is the number of cycles required to finish an ANN computation.

- ANN initiation interval (II): this is the number of cycles that must elapse between two consecutive ANN computations.

- Block RAM usage (BRAM): this is the used fraction of the device's 18KB block RAMs.

- DSP usage: this is the used fraction of the device's DSP arithmetic units.

- Flip-flop usage (FF): this is the used fraction of the device's Flip-flops.

- LUT usage: this is the used fraction of the device's look-up tables.

6.3.1 Base algorithm

The ANN algorithm is implemented in the following top-level function

```
template<typename w_t, typename i_t, typename h_t, typename o_t,
typename act_h, typename act_o>
void nn_5<w_t,i_t,h_t,o_t,act_h,act_o>::run(const i_t input[IN], o_t output[ON]) {
    //neurons
    h_t nh1[HN];
    h_t nh2[HN];
    h_t nh3[HN];

    //hidden layer 1
    compute_layer<w_t,h_t,IN,HN,act_h>(input,w1,nh1);

    //hidden layer 2
    compute_layer<h_t,h_t,HN,HN,act_h>(nh1,w2,nh2);

    //hidden layer 3
    compute_layer<h_t,h_t,HN,HN,act_h>(nh2,w3,nh3);

    //output layer
    compute_layer<h_t,o_t,HN,ON,act_o>(nh3,w4,output);
};
```

where w_t, i_t, h_t and o_t are the weights, input units, hidden units and output units data types, act_h and act_o are classes implementing the hidden and output unit's activation functions, and nh#[·] are the arrays storing the neuron net output values.

Each compute_layer function calculates one of the network's layers. This function is implemented by the following code

```
template<typename li_t, typename lo_t, int IS, int OS, typename act_f>
void nn_5< ... >::compute_layer(const li_t input[IS],
 const w_t weights[OS*(IS+1)], lo_t output[OS]){
    w_t sum;
    int i, j;

outer:
    for (i=0; i<OS; ++i)
    {
        sum = 0.0;
inner:
        for(j=0; j<IS; ++j)
        {
            sum += w_t(input[j]) * weights[i*(IS+1)+j];
        }
        sum += weights[i*(IS+1)+IS]; //Bias

        output[i] = (lo_t)act_f::f(sum);
    }
};
```

where li_t and lo_t are the input and output layer data types, IS is the layer input size, OS is the layer output size, and act_f is the class implementing the layer's activation function.

The results of the synthesis from the base algorithm are shown in table 6.1. These values will be used as the baseline for measuring the relative gains of the different optimizations.

Table 6.1: Base algorithm synthesis results

ANN Parameters	Timing (cycles)			Resources (% of total)			
	Total Lat.	ANN Lat.	ANN II	BRAM	DSP	FF	LUT
Float SigmSym	540041217	527071	527071	47	25	16	50
Fixed SigmSym	324221953	316306	316306	29	2	0.7	2
Float StepSym	462916609	451745	451745	46	6	2	10
Fixed StepSym	200112129	195109	195109	27	0.9	0.3	1

6.3.2 Layer pipelining

The ANN computation can be divided into four steps which calculate the three hidden layers and the output layer respectively. The input of a network's layer is the immediate output of its previous layer. That implies a data dependency which forces a sequential computation of the layers. However, once a layer is computed, the logic used to perform the calculations is liberated. This enables the pipelining of the ANN algorithm by dividing the algorithm into four steps corresponding to each layer. Figure 6.1 shows a time diagram of the pipelined layer ANN design.

This optimization increases the ANN computation throughput when calculating multiple networks sequentially. The basis of the optimization is the reduction of the initiation interval down to the computation delay of the biggest layer. However, more FPGA resources are

required because each layer computation must be synthesized into a unique circuit. On the contrary, the base algorithm reuses the same circuit for each hidden layer computation due to their algorithmic equivalency.

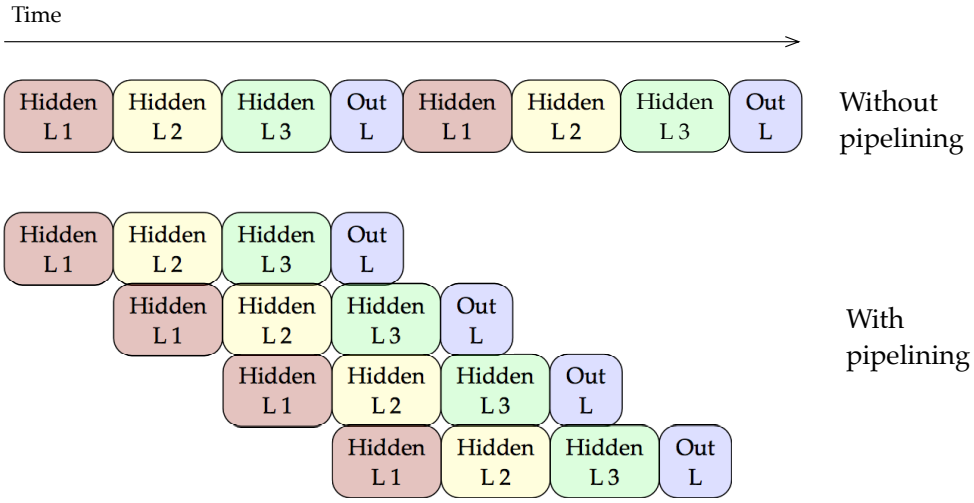


Figure 6.1: ANN layer pipelining time diagram

Table 6.2 shows the synthesis results for this optimization compared to the baseline results of table 6.1. Layer pipelining achieves a higher throughput, especially for the networks implemented in fixed point, while using a reasonable amount of resources compared to the achieved speed-up.

Table 6.2: Layer pipelining synthesis results

ANN Parameters	Timing (cycles)			Resources (% of total)			
	Total lat.	ANN lat.	ANN II	BRAM	DSP	FF	LUT
Float SigmSym	321224049	527071	314002	48	36	22	68
Speed-up	1.681	1	1.679	0.97	0.694	0.727	0.735
Fixed SigmSym	158157849	316306	154602	30	5	0.7	2
Speed-up	2.050	1	2.046	0.967	0.4	1	1
Float StepSym	388352860	451745	379937	47	8	3	10
Speed-up	1.192	1	1.189	0.979	0.721	0.672	1
Fixed StepSym	157850949	205016	154302	28	0.4	0.7	1
Speed-up	1.268	0.952	1.264	0.964	2.250	0.429	1

6.3.3 Inner loop pipelining

A second optimization can be designed taking advantage of the potential parallelism of the compute_layer function. Both function's nested loops do not have any write-read data de-

pendency enabling a full parallelization of the layer computation. However, the stages of the computation grows exponentially with the number of units per layer. This makes it impractical to synthesize a fully pipelined function, feasible within the FPGA resources.

The design approach was to pipeline the inner loop of the compute_layer function which can be implemented using, practically, the same amount of resources than the previous design. The inner loop computation can be divided into the four stages:

1. Read input[j] and weights[i*(IS+1)+j] from block RAM. These accesses can be executed concurrently.

2. Multiply the input and weight values.

3. Accumulate the result of the multiplication into the sum register

4. Write the sum register to block RAM.

The pipeline design executes these steps concurrently, achieving a loop iteration initiation interval equal to the longest step latency. Generally, the multiply operation has the largest latency throughout the different ANNs except for the networks using StepSym activation functions where the multiplication can be substituted by a sign change operation.

Figure 6.2 shows a time diagram of the inner loop computation with and without pipelining. Table 6.3 shows the synthesis results for this optimization. The inner loop pipelining achieves significant speed-ups, especially for the fixed-point implementations due to a lower latency of the multiply-accumulate operation which reduces the initiation interval of the pipeline.

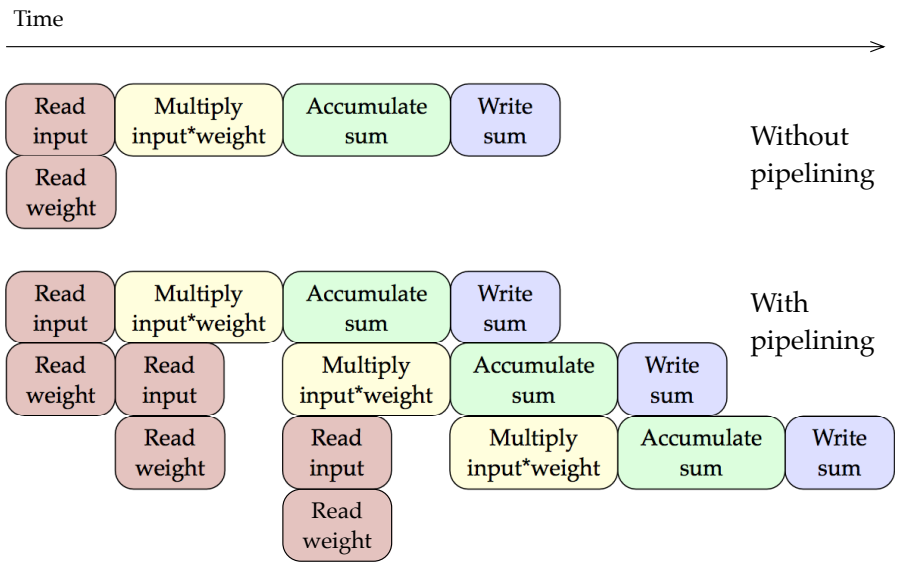


Figure 6.2: Inner loop pipelining time diagram

Table 6.3: Inner loop pipelining synthesis results

ANN Parameters	Timing (cycles)			Resources (% of total)			
	Total Lat.	ANN Lat.	ANN II	BRAM	DSP	FF	LUT
Float SigmSym	126100098	203962	123265	48	46	41	99
Speed-up	4.283	2.584	4.276	0.979	0.543	0.390	0.505
Fixed SigmSym	32533449	54311	31802	30	5	0.7	2
Speed-up	9.966	5.824	9.946	0.967	0.4	1	1
Float StepSym	88477945	175913	86409	47	8	4	20
Speed-up	5.232	2.568	5.228	0.979	0.721	0.523	0.5
Fixed StepSym	32328849	52508	31602	28	0.4	0.7	2
Speed-up	6.190	3.716	6.174	0.964	2.25	0.429	0.5

6.3.4 Inner loop unrolling

A third optimization takes further the compute_layer function inner loop parallelization. The previous optimization increases the throughput of the computation by reducing the inner loop initiation interval to a fraction of a single iteration computation. However, the FPGA still has enough resources to completely parallelize multiple iterations from the function's inner loop. To achieve that, the inner loop is unrolled by a factor U to process this number of units in a single operation. The pipelining of the inner loop is still viable when applying this optimization. This design may have the same initiation interval of the previous optimization however, it increases the throughput by computing U units concurrently in each step of the pipeline.

This optimization requires replicating the resources allocated for each step of the pipeline by a factor of U. For steps 2 and 3 (multiply and accumulate), this is only limited by the combinatorial and arithmetic resources of the FPGA. However, steps 1 and 3 (read and write) implies U concurrent memory reads to the same block RAM to get the input values. This produces a bottleneck because each block RAM has only two read ports. The proposed design solves this limitation by interleaving the input values across $U/2$. This results in a reading step in which latency does not limit the initiation interval of the pipeline.

This optimization is not feasible for the networks implemented in floating point using SigmSym activation functions. This is due to a shortage of resources to parallelize the multiplication-addition operation by a factor of U.

The selection of the unrolling factor (U) depends on the pipeline performance and the gain achieved by the concurrent execution of multiple iterations. Increasing U implies decreasing the pipeline depth and the increased concurrency also extends the latency of the multiply-accumulate steps. This causes a plateau on the throughput when increasing the unrolling factor and, eventually, the performance gets worse if the U value is further increased. A

good general case unrolling factor has been found to be $U = 8$.

Table 6.4 shows the synthesis results for the inner loop unrolling optimization. One can observe that the floating point implementation using SigmSym has a poorer performance compared to the previous optimization. However, the rest of the implementations achieve the highest speed-up relative to the baseline.

Table 6.4: Inner loop unrolling synthesis results

ANN Parameters	Timing (cycles)			Resources (% of total)			
	Total Lat.	ANN Lat.	ANN II	BRAM	DSP	FF	LUT
Float SigmSym	132787449	223474	129801	48	36	25	71
Speed-up	4.067	2.359	4.061	0.979	0.694	0.64	0.704
Fixed SigmSym	6549249	17823	6401	72	43	2	5
Speed-up	49.505	17.747	49.415	0.403	0.047	0.35	0.4
Float StepSym	22878156	48700	22444	123	35	6	33
Speed-up	20.234	9.276	20.128	0.375	0.173	0.34	0.3
Fixed StepSym	5628549	15722	5501	72	7	1	4
Speed-up	35.553	12.410	35.468	0.375	0.129	0.3	0.25

6.4 Results

The best FPGA ANN implementations described in section 6.3 were compared to the ones running on the ARM A9 PS. The ANN number of hidden units were modified due to significant precision losses of the FPGA ANN compared to the PS one. However, the FPGA design produced satisfactory results when used with ANN with fewer number of hidden units. Further work should be done to determine the causes of this drawback.

Two ANN were tested with the following parameters:

ANN parameters	Test 1	Test 2
Feature vector	F0	F0+Δ_{F0}+Δ^2_{F0}
Units per hidden layer	10	20
Inputs	11	33
Hidden layer activation fn.	StepSym	SigmSym

For the first test, the PS ANN implementation was the floating-point design from section 5.1.2. The FPGA ANN implementation was the floating-point design from section 6.3.4. The number of frames per FPGA call was $N = 1024$.

For the second test, the PS ANN implementation was the floating-point design from section 5.1.2 using the expf() SigmSym function. The FPGA ANN implementation was the fixed-point design from section 6.3.4 using the LUT approximation from section 5.2.3. The number of frames per FPGA call was $N = 1024$. The reason of comparing fixed-point and floating-point arithmetics is because the PL implementation makes use of the Xilinx fixed-point data type. This data type is optimized for FPGA algorithm synthesis, but it fails to produce optimal implementations when executed outside the FPGA.

The code was instrumented in order to measure the average execution time of a single ANN computation. Once the timing tests were performed, a second test pass was run with code instrumentation to measure the power rating of the board. The power measurement procedure is detailed in the appendix B.

Table 6.5 shows the comparison between both implementations. As expected, the FPGA design achieves a substantial speed-up, especially for the network using SigmSym functions. The energy efficiency gain is slightly lower than the execution time speed-up due to the extra energy used by the FPGA not accounted in the PS energy measurements. However, the energy used by the FPGA is approximately 5 times smaller than the used by the PS resulting in an energy efficiency gain close to the magnitude to that of the execution time speed-up.

Table 6.5: FPGA vs ARM A9 ANN implementation tests 1 and 2

Test	Avg. execution time		Energy use (mW·s)		Efficiency gain
	Time (μs)	Speed-up			
Test 1					
ARM A9	7.679		PS	9.902×10^{-3}	
FPGA	3.682	2.085	PS+FPGA	4.917×10^{-3}	2.014
Test 2					
ARM A9	58.947		PS	6.696×10^{-2}	
FPGA	9.695	6.080	PS+FPGA	1.401×10^{-2}	4.780

Part III

Project management

Chapter 7

Project planning

This chapter defines the work plan of the project: the division of tasks, the resource management and the time schedule.

7.1 Stages

This project will be divided into four stages listed as follows.

Analysis and definition

The objective is to analyze the project's topic in order to obtain a well-founded definition.

Development

This stage consists of the development of any source code and tools necessary for the project. It will be the core section of the project.

Documentation

This stage comprises of reporting the project in any written form, especially the final thesis writing.

Defence

This stage consists of the final project defence and presentation.

7.2 Resources

This section lists all the essential resources required for the project.

7.2.1 Human

Developer (Dev): He is the responsible for carrying out the project development, documentation and defence.

Project's Director (PD): He is the responsible for supervising the accomplishment of the project and guiding the developer. Planning-wise, he is the responsible for providing the DNNs used on the project's implementations.

7.2.2 Hardware

Development Computer (DC): It is the computer the developer will use to carry out all the development and documentation needs.

Baseline System (BS): It is the computer that will be used to run the C prototypes. It will serve as the baseline system for the performance gain assessment. It will be an Intel Core I7-3770K with 16 GB DDR3 running Ubuntu 15.04.

Zynq Board (Zynq): the FPGA used will be one of the Xilinx Zync-7000 SoC family boards.

Power Meter (PM): Model Yokogawa WT300. It is the hardware that will be used to measure the power consumption of both baseline system and Zynq board.

7.2.3 Software

Vivado Design Suite and Vivado HLS (Vivado): It is the software used to design and simulate the FPGA implementations. It is the only commercial software used.

Development tools: The C development tools will be Sublime Editor, GCC compiler and GDB debugger. I will also use the Fast Artificial Neural Network (FANN) library to validate results of the DNN implementation. To profile the C code, I will use GNU gprof and OProfile.

Documentation tools: To document the project, I will user LyX for text editing and Inkscape and different LaTeX packages for graphics. Python code will be used to produce some graphs as well.

Project management: To manage the project, I will use Gantter as a Gantt chart planner, GIT as a source repository and Trello and Evernote as project management tools.

7.3 Tasks

7.3.1 Summary

The methodology for developing and assessing the FPGA voice gender identification algorithm will be executed in two iterative cycles where a voice feature will be used as the input

of the DNN. Every cycle will have the following steps.

1. Feature extractor: implementation of the voice feature extraction algorithm using C.
2. DNN: training and implementation of the DNN using the prior feature extractor using C.
3. Prototyping: functional implementation of the voice characteristics identification system using the feature extractor and the DNN in C.
4. FPGA port: implementation of the prototype in the FPGA.
5. Assessment: evaluation of the relative performance gain between the FPGA implementation and the prototype.

The first cycle feature extractor will be fundamental frequency estimation and the second will be MFCC. Once both cycles are completed, both neural networks will be joined in a single system following steps 3 to 5.

The following subsections break down the project's stages into specific work tasks. Each task is assigned a set of resources and an estimated time necessary for completion.

7.3.2 Analysis and design

AD1 State of the art: This task comprises of the necessary research necessary to obtain an updated state-of-the-art on the project's topics and technologies.

Estimated time: 16 h - Resources: Dev, DC

AD2 Project definition: This task consists of defining the project's problem, based on the results of task AD1.

Estimated time: 8 h - Resources: Dev, DC

AD3 Project planning: This task consists of planning the project's tasks during the initial stage of the project.

Estimated time: 8 h - Resources: Dev, DC

7.3.3 Development

This project's development will be divided into two main iterative cycles. There will be a third final cycle intended to combine the results of the two previous cycles together.

Dev1 First iteration

Dev1.1 F0 feature extractor: I will perform an investigation on the most suitable voice fundamental frequency (F0) estimator focusing on efficient, highly parallelizable, accurate and robust methods. The source of the investigation will be comparative studies on fundamental frequency estimation techniques. Once the suitable method is

selected, I will implement the feature extractor using C.

Estimated time: 20 h - Resources: Dev, DC

Dev1.2 F0 DNN training: the provided DNN will be trained using a voice corpus analyzed with the F0 estimator feature extractor. The design and training processes along with the voice corpus will be provided by the project's director.

Estimated time: 10 h - Resources: Dev, PD, DC

Dev1.3 DNN implementation: I will implement the trained DNN in C using arbitrary arithmetic precision. I will investigate how precision can be limited to optimize and simplify the problem complexity. Integer, floating point and fixed point arithmetics will be assessed in terms of complexity and accuracy.

Estimated time: 16 h - Resources: Dev, DC

Dev1.4 Prototype: I will implement a fully functional voice gender identification system including the F0 estimator feature extractor and the DNN implementation in C.

Estimated time: 8 h - Resources: Dev, DC, BS

Dev1.5 DNN FPGA design: I will design and simulate a DNN on chip architecture based on literature designs. I will port the DNN to the FPGA architecture using High-Level Synthesis (HLS) tools first and fine tune the Hardware Descriptor Language (HDL) implementation if necessary. Finally, I will design an interface to communicate the FPGA and the ARM processor.

Estimated time: 30 h - Resources: Dev, DC, Vivado

Dev1.6 F0 estimator FPGA design: I will design and simulate the F0 estimator using HLS tools and fine tune the HDL implementation if necessary. I will combine the DNN FPGA implementation with the feature extractor and I will adapt the communication interface between the FPGA and the ARM processor.

Estimated time: 30 h - Resources: Dev, DC, Vivado

Dev1.7 FPGA port: I will combine the DNN and F0 estimator FPGA designs and I will implement them on the FPGA development board.

Estimated time: 30 h - Resources: Dev, DC, Vivado, Zynq

Dev1.8 Implementation assessment: I will evaluate the relative performance gain between the FPGA implementation and the C prototype.

Estimated time: 8 h. Resources: Dev, DC, BS, Vivado, Zynq, PM

Dev2 Second iteration

Dev2.1 MFCC feature extractor: I will implement an MFCC feature extractor in C.

Estimated time: 20 h - Resources: Dev, DC

Dev2.2 MFCC DNN training: the provided second DNN will be trained using the same voice corpus as in Dev1.2 using the MFCC feature extractor.

Estimated time: 8 h - Resources: Dev, PD, DC

Dev2.3 Prototype: I will modify the first prototype from task Dev1.4 to use the first and second DNNs together following the same steps as in steps Dev1.3 and Dev1.4.

Estimated time: 8 h - Resources: Dev, DC, BS

Dev2.4 DNN FPGA design: I will design and simulate the MFCC DNN design from task Dev2.3 following the same steps as in task Dev1.5.

Estimated time: 30 h - Resources: Dev, DC, Vivado

Dev2.5 MFCC estimator FPGA design: I will design and simulate the MFCC estimator following the same steps as in task Dev1.6.

Estimated time: 30 h - Resources: Dev, DC, Vivado

Dev2.6 FPGA port: I will combine the DNN and F0 estimator FPGA designs from tasks Dev2.3 and Dev2.4 and I will implement them on the FPGA development board.

Estimated time: 30 h - Resources: Dev, DC, Vivado, Zynq

Dev2.7 Implementation assessment: I will evaluate the relative performance gain between the FPGA implementation from task Dev2.5 and the C prototype from task Dev2.3.

Estimated time: 8 h. Resources: Dev, DC, BS, Vivado, Zynq, PM

Dev3 Third iteration

Dev3.1 Prototype: I will combine the prototypes from tasks Dev1.4 and Dev2.3 in order to make a system using both DNNs.

Estimated time: 20 h. Resources: Dev, DC, BS

Dev3.2 FPGA design: I will combine the FPGA designs from tasks Dev1.5, Dev1.6, Dev2.4, Dev2.5. I will simulate the design until it meets the expected behavior.

Estimated time: 30 h - Resources: Dev, DC, Vivado

Dev3.3 FPGA port: I will implement the design from task Dev3.2 on the FPGA board.

Estimated time: 30 h - Resources: Dev, DC, Vivado, Zynq

Dev3.4 Implementation assessment: I will evaluate the relative performance gain between the FPGA implementation from task Dev3.3 and the C prototype from task Dev3.1.

Estimated time: 8 h - Resources: Dev, DC, BS, Vivado, Zynq, PM

7.3.4 Documentation

Doc1 GEP deliverables: This task consists of writing all the deliverables for the GEP (Project Management) course. This work will be included as a part of the final thesis.

Estimated time: 24 h - Resources: Dev, DC

Doc2 GEP presentation: This task consists of preparing and making a presentation about the project. It also includes the creation of some support slides.

Estimated time: 8 h - Resources: Dev, PD, DC

Doc3 Thesis: This task consists of writing a document including the purpose, the previous research literature, the methods used and the findings of the project.

Estimated time: 40 h - Resources: Dev, DC

7.3.5 Defence

Def1 Defence preparation: This task consists of preparing the defence itself and the supporting slides.

Estimated time: 20 h - Resources: Dev, DC

Def2 Project's defence: This task consists of preparing a 30 minutes defence of the project.

Estimated time: 2 h - Resources: Dev, DC

7.4 Work plan

7.4.1 Duration

The table 7.1 shows the overall estimated duration of the project stages and sub-stages.

Table 7.1: Project duration

Stage	Duration
AD - Analysis and Design	32 h
Dev - Development	347 h
Dev1 - First iteration	152 h
Dev2 - Second iteration	134 h
Dev3 - Third iteration	88 h
Doc - Documentation	72 h
Def - Defence	22 h
Total	500 h

7.4.2 Schedule

This project started on July 27th of 2015 and it is expected to end before January 29th of 2016. I will work 20 hours per week on average with a vacation period from August 31st to September 7th. There are two deadlines on October 18th for Doc1 and October 27th for Doc2.

The project's hardware and software resources will be available at any time, the PD availability will be checked in advance for the required tasks.

Task dependencies

This project's tasks dependencies can be outlined in a hierarchical manner:

- Dev stage depends on AD.
- Def depends on Doc and Dev.
- Dev3 depends on Dev1 and Dev2.
- Each task in every development iteration is dependent on the previous task as they are ordered in section 7.3.
- Doc1 and Doc2 depend on AD.
- Doc3 depends on Dev stage.

Gantt diagram

The diagram in figure 7.1 shows the project agenda along with the task dependency relationships.

Plan monitoring

Each stage and development iteration will be assessed by the PD. Additionally, regular reports on the project's status will be sent to the PD for an up-to-date project monitoring.

7.4.3 Plan feasibility

The project's planning can be adapted based on the deviation of the tasks duration. If any task finishes earlier than it is predicted, I will start the following task based on the project agenda. If any task is delayed, I can increase the work hours up to a limit of 30 hours per week. The latter case sets a maximum positive time deviation up to the 50%. The project's tasks division is fine enough to ensure a realistic time estimation. Additionally, more time has been allotted to the tasks in which I have less experience. Therefore, a possible leeway of 50% ensures a high probability of completion.

However, if by any chance the project is delayed more than a 50%, I will abandon the second and third development iterations (as stated in section 7.5) which account for the 44.4% of the estimated project time. In that case, the expected completion time could be increased by 79.9% without compromising the feasibility of the project's plan.

The plan alternatives detailed above have no impact on how the rest of resources except for the developer will be used, therefore they have not been taken into account.

The plan's foremost plausible disturbances and their respective probabilities are detailed as follows:

Task time underestimation

Highly probable. Addressed on the plan alternatives discussed above.

Delays on Dev1.2 and/or Dev2.2

Moderately probable. In that case I will use an arbitrary DNN to advance into the following tasks and I will re-adapt the development once the delayed tasks are completed. This would imply extra time, but it would be unavoidable due to the fact that Dev1.2 and Dev2.2 are on the task's critical path.

Unavailability of Zynq board

Low probability. Delays due to the lack of the board could not be solved. If the delay increases to an inadmissible point I will consider renting or buying a development board.

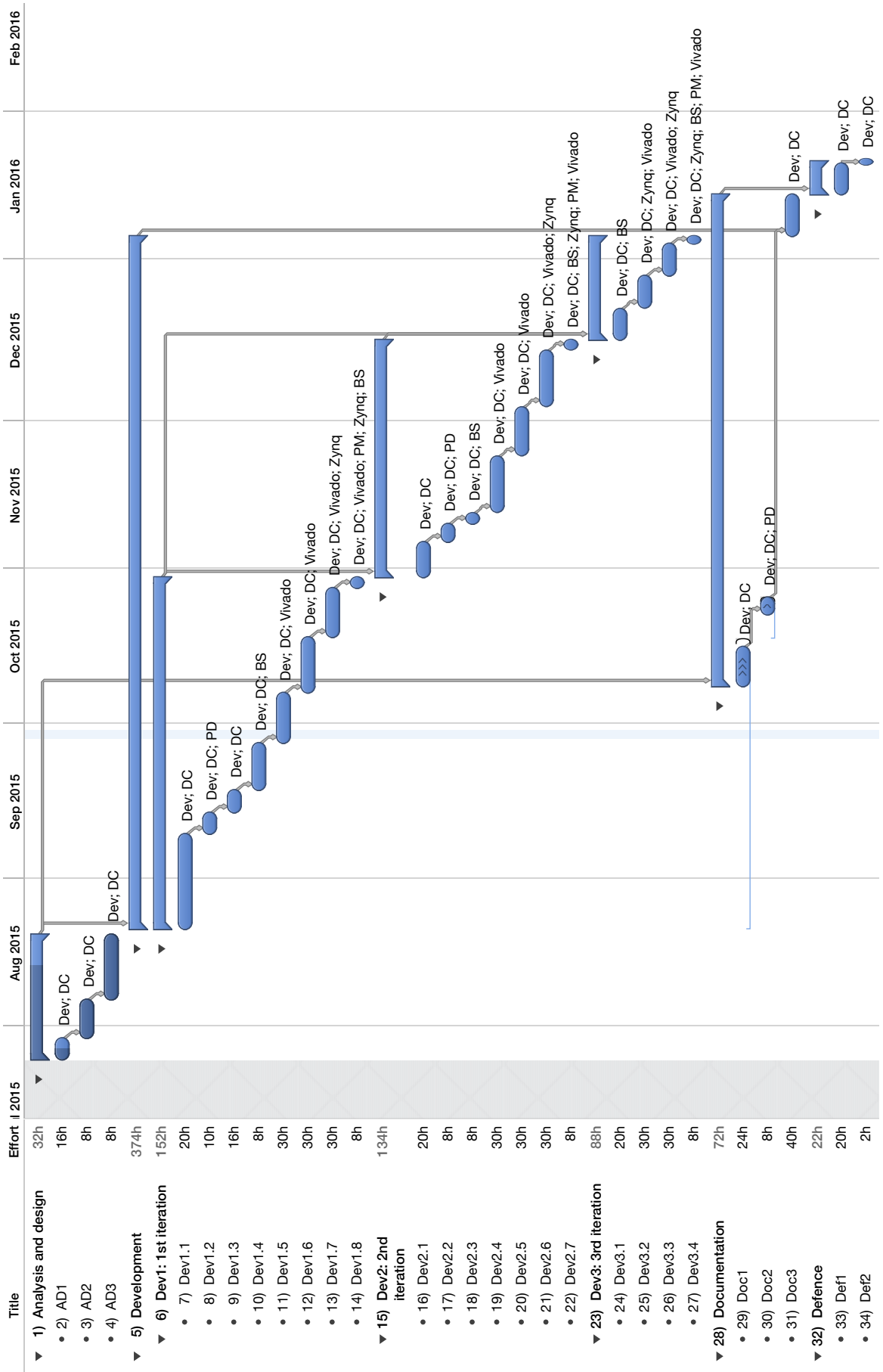


Figure 7.1: Project's Gantt diagram.

7.5 Limitations and risks

There are some difficulties which can hinder this project:

- I have no experience on FPGA development and I can not predict precisely how long will it take to develop the system on this platform. Furthermore, I am also not experienced in DNNs though I can predict more precisely the development time of the C code based on the overall progress of the project. If I run out of time I will discard one of the development cycles and I will only implement the fundamental frequency estimator version.

- I have working knowledge on the digital signal processing basics needed to implement the signal feature extractors, but I lack some theoretical foundations. That may result in inaccurate algorithm implementations. I will address this obstacle by comparing the feature extraction algorithms with state-of-the-art feature extractors.

- I am dependent of the DNN design which will be provided. If I am not able to obtain the network design I will use a minimal voice corpus and DNN automated training software to get a suboptimal design suitable to work with. If necessary, the sacrifice in accuracy will still allow me to work on improving system efficiency which is the main focus of the project.

- It may be difficult to measure the power consumption of the used devices using direct current consumption measuring. In this case, I will use an approach based on software profiling analysis using the power specifications provided by the device manufacturer.

7.6 Cost estimation

This section presents the estimated budget of the whole project. Expenses are divided in the following categories: human resources, hardware amortization, software amortization and general expenses. Each category has an estimated cost and a contingency cost. Furthermore, an unforeseen costs category has been added in order to provide some flexibility. This category represents the 5% of the sum of costs of the last categories. Table 7.2 shows the overall budget of the project. The calculation details of this data can be found in the following sections of the present chapter.

Table 7.2: Project's overall budget

Category	Cost	Contingency	
Human resources	11,161.63 €	1,345.75 €	
Hardware amortization	402.37 €	50.70 €	
Software amortization	254.66 €	32.09 €	
General expenses	4.08 €	0.51 €	
Unforeseen costs	591.14 €		
			Project Total
Totals	12,413.88 €	1,429.05 €	**13,842.93 €**

7.6.1 Human resources

All the human resources costs are assumed to be direct costs due to the ephemeral nature of this project. These expenses are estimated based on the work plan, taking into account some degree of variability to allow for any plausible deviations from the predicted plan.

The human resources of the project, as defined in section 7.2, are the Developer (Dev) and Project's Director (PD). However, in order to provide a more realistic budget estimation, the Dev resource has been divided into three roles according to different professional expertise profiles. Each project's task is assigned a role according to the professional skills needed for such task. The role definitions and salaries are based on independent Spanish IT market analysis using 2014 and 2015 real market data [51], [52]. Gross hourly wage is calculated based on the average annual salary divided by Spanish 2014 average annual worked hours (1,689 hours) [53]. Table 7.3 shows the professional roles and associated salaries.

Human resources cost estimation is calculated according to the project's plan task durations, as defined in section 7.3. Each task is assigned a professional role and a deviation factor to adjust for the possible divergences on its duration. These factors are set accordingly to the uncertainty of each task's time estimation, as is defined in the work plan. Table 7.4 shows the role-to-task assignations and the estimated human resources budget.

Table 7.3: Human resources salaries

Role (abbreviation)	Resource	Gross hourly wage
Project Director (PD)	PD	23.68 €/h
Project Manager (PM)	Dev	23.68 €/h
Architect/Analyst (AA)	Dev	21.91 €/h
Developer (D)	Dev	18.95 €/h

Table 7.4: Human resources budget

				Contingencies	
Task	Role	Hours	Cost	Deviation	Cost
Analysis and design					
AD1 State of the art	PM	16	378.92 €	7%	26.52 €
AD2 Project definition	AA	8	175.25 €	7%	12.27 €
AD3 Project planning	PM	8	189.46 €	7%	13.26 €
Analysis and design subtotal			743.64 €		52.05 €
Development - 1st iteration					
Dev1.1 Feature extractor	AA	20	438.13 €	10%	43.81 €
Dev1.2 DNN training	AA	10	219.06 €	10%	21.91 €
	PD	10	236.83 €	10%	23.68 €
Dev1.3 DNN impl.	D	16	303.14 €	10%	30.31 €
Dev1.4 Prototype	AA	8	175.25 €	15%	26.29 €
Dev1.5 DNN FPGA design	AA	30	657.19 €	15%	98.58 €
Dev1.6 Est. FPGA design	AA	30	657.19 €	15%	98.58 €
Dev1.7 FPGA port	D	30	568.38 €	15%	85.26 €
Dev1.8 Impl. assessment	AA	8	175.25 €	10%	17.53 €
Development - 2nd iteration					
Dev2.1 Feature extractor	AA	20	438.13 €	10%	43.81 €

Table 7.4: Human resources budget

Task	Role	Hours	Cost	Contingencies Deviation	Contingencies Cost
Dev2.2 DNN training	AA	8	175.25 €	10%	17.53 €
	PD	8	189.46 €	10%	18.95 €
Dev2.3 Prototype	AA	8	175.25 €	15%	26.29 €
Dev2.4 DNN FPGA design	AA	30	657.19 €	15%	98.58 €
Dev2.5 Est. FPGA design	AA	30	657.19 €	15%	98.58 €
Dev2.6 FPGA port	D	30	568.38 €	15%	85.26 €
Dev2.7 Impl. assessment	AA	8	175.25 €	10%	17.53 €
Development - 3rd iteration					
Dev3.1 Prototype	AA	20	438.13 €	15%	65.72 €
Dev3.2 FPGA design	AA	30	657.19 €	15%	98.58 €
Dev3.3 FPGA port	D	30	568.38 €	15%	85.26 €
Dev3.4 Impl. assessment	AA	8	175.25 €	10%	17.53 €
Development subtotal			8,305.51 €		1,119.54 €
Documentation					
Doc1 GEP deliverables	AA	24	525.75 €	7%	36.80 €
Doc2 GEP presentation	PM	8	189.46 €	7%	13.26 €
Doc3 Write thesis	AA	40	876.26 €	10%	87.63 €
Documentation subtotal			1,591.47 €		137.69 €
Defence					
Def1 Defence preparation	PM	20	473.65 €	7%	33.16 €
Def2 Project's defence	PM	2	47.37 €	7%	3.32 €
Defence subtotal			521.02 €		36.47 €
TOTAL			**11,161.63 €**		**1,345.75 €**

7.6.2 Hardware

All hardware requirements are assumed to be indirect since all the devices used are not exclusively employed in this project. In order to quantify the hardware costs, each item is assigned an estimated lifespan and a related amortization cost based on its usage time. The usage times of the hardware resources are based on the project's task durations, which can be found in section 7.3. Hardware lifespans are based on the author's estimations and are expressed in years. So that lifespan and usage times can be related, I assume each resource is employed half the Spanish average annual worked time [53] per year. Additionally, a deviation factor is used in order to account for the possible time divergences on the task durations. This factor is the average of the deviation factors from table 7.4 weighted by the time fraction of each task over the total project time. Table 7.5 shows the hardware resources budget.

Table 7.5: Hardware resources budget

Resource name	Model	Price	Amortization cost	Usage (hours)	Lifespan (years)
Dev. Computer (DC)	MacbookPro 12,1	1,649.00 €	325.44 €	500	3
Baseline Sys. (BS)	N/A	1,500.00 €	35.52 €	60	3
Zynq Board (Zynq)	Xilinx Z7-ZC702-G	795.00 €	35.77 €	114	3
Power Meter (PM)	Yokogawa WT300	990.50 €	5.63 €	24	5
		Price Total	Amortization Total	Contingencies Deviation	Contingencies Cost
		4,934.50 €	402.37 €	12.6 %	50.70 €

7.6.3 Software

All hardware requirements are assumed to be indirect since all the used software is not exclusively employed in this project. The software used will be mainly open source, thus without an accountable cost. The only software resource with an associated cost will be Vivado Design Suite and Vivado HLS. Table 7.6 shows the software resources budget calculated in the same way as section 7.6.2.

Table 7.6: Software resources budget

Resource retail name	Price	Amortization cost	Usage (hours)	Lifespan (years)
Vivado Design Suite System Edition	4,267.00 €	254.66 €	252	5

	Price Total	Amortization Total	Contingencies Deviation	Contingencies Cost
	4,267.00 €	254.66 €	12.6 %	32.09 €

7.6.4 General expenses

Electric energy consumption by hardware resources has been accounted as the sole general indirect expense associated with this project. Its cost has been calculated by counting the energy consumption of each resource based on its usage time and multiplied by the Spanish average electricity energy price (0.15€/kWh). A deviation factor is used in the same fashion as sections 7.6.2 and 7.6.3. Although the electric energy cost is negligible, it is shown to account for the energy consumption footprint of the project. Table 7.7 shows the electric energy consumption budget.

Table 7.7: Electric energy consumption budget

Resource name	Energy cost	Usage (hours)	Consumption Power (W)	Consumption Energy (kWh)
Development Computer (DC)	2.25 €	500	30	15.00
Baseline System (BS)	1.80 €	60	200	12.00
Zynq Board (Zynq)	0.03 €	114	2	0.23
Power Meter (PM)	0.18 €	24	50	1.2

	Energy cost Total	Contingencies Deviation	Contingencies Cost	Energy Total (kWh)
	4.08 €	12.6 %	0.51 €	28.43

7.6.5 Control mechanisms

The budget control mechanism will consist of the analysis of deviations between the estimated direct costs and the actual costs. This analysis will be performed once every project task is finished. Since any indirect cost is derived from the direct costs, deviations will be propagated to indirect costs.

The only direct cost is human resources and its calculation is based on the tasks' duration from the work plan. In order to control the cost deviations, all tasks will be logged registering the time spent on each task. Any task that exceeds its assigned cost will be included in the task's contingency item. Consequently, indirect costs will be recalculated and surpluses will be included in their respective contingency items. If any contingency limit is reached, the exceeding cost will be included in unforeseen costs.

7.7 Identification of laws and regulations

Voice features derived from human speech, like any other form of biometric data, are considered personal information and should be protected and treated with the utmost privacy. Any system storing human voice features must fulfill the applicable data privacy and protection regulations. This is an imperious necessity when the biometric data is part of an identification system granting access to other sensitive information.

The voice data used in this project is part of the 2000 NIST Speaker Recognition Evaluation database. The use of this data is subject to the Linguistic Data Consortium user agreement for non-members license[1] which allows the use of their data for the purposes of non-commercial linguistic education, research and technology development.

7.8 Project plan development

The original project plan has been followed to a great extent, with only two major modifications:

Firstly, the project end date has been extended from January 2015 to April 2016 due to personal reasons.

Secondly, the DNN design and training process, originally planned to be out of the scope of this project, has been incorporated into the TFG. This lead to a substantial use of the original allotted time, which limited the available time for development. As a response to this situation, the original development plan has been modified to exclude the implementation of the feature extractor system on the FPGA device.

This plan modification does not interfere with the project main objective of measuring the performance improvement of using a FPGA implementation. However, it excludes the

[1] https://catalog.ldc.upenn.edu/license/ldc-non-members-agreement.pdf

secondary objective consisting of measuring the performance improvement of the whole system including the feature extractor. Regarding the human resources cost estimation, development time extension was already foreseen inside the risks and contingencies plans.

7.8.1 Methodology modifications

The method of measuring the power consumption of the FPGA device has been simplified to the use of the onboard digital power controllers incorporated in the Zynq 7000 APSoC systems. The use of a power measuring device is not longer necessary eliminating the amortization cost of this device initially included in the hardware budget.

Chapter 8

Sustainability analysis

This section assesses the project's sustainability according to the work plan. This assessment analyzes the economic, social and environmental dimensions of the project. Each dimension is assigned a score from zero to ten in order to provide for an overall sustainability score.

8.1 Economic dimension

Human resources cost represents almost 90% of the project's total budget. This cost is assessed based on the project's plan however, there is a high degree of uncertainty due to my lack of experience in FPGA development. To rectify this issue, a contingencies budget is calculated based on my level of knowledge on each task. This item represents a surplus of 23% of the project's budget. With this in mind, the budget has a comfortable margin to allow for any deviations. However, the contingency cost could be decreased if the project was carried out by someone with the appropriate experience.

Hardware and software resources represent 5.3% of the project's total budget and are strictly needed. Therefore, it is not possible to decrease this budget line. Furthermore, all devices and software used are only accounted for based on their amortization costs. There will not be any purchases during the project. Costs related to this budget line, such as repairs, are not counted but they are highly unlikely and, if some were to happen, they could be counted as unforeseen costs.

Project viability is not studied due to a lack of knowledge of the market and the uncertainty of the applicability of the project outcomes. Once the project goals would be achieved, it will be possible to assess its viability.

Finally, economic sustainability is supported by a fine-grained and detailed work plan and an associated budget, which allows for the management of uncertainties. However, due to this very uncertainties there are shortcomings regarding the project viability and resource savings. For these reasons, I give a 7 over 10 to the planning's economic sustainability.

8.2 Social dimension

The investigative nature of this project implies a leap between the expected project outcomes and their potential applications. However, is it possible to hypothesize on some scenarios where this project could benefit certain social groups. For instance, one possible outcome of the project is the improvement of Automatic Speech Recognition applications. Beneficiaries of such improvement could be a broad range of disabled people which rely on human-computer interaction to satisfy their basic needs.

Due to the uncertainties about the project applicability, it is difficult to define whether it will be substantial social benefits or damages. However, the most likely scenario is that there may be some benefits and no damages. For this reason, I give a 4 over 10 to the planning's social sustainability.

8.3 Environmental dimension

The material resources used during the project are and will be reused for other applications. Additionally, the environmental footprint of their use will be comparable to any average personal technology usage footprint. The energy consumption of the hardware resources of the project is estimated to be in the range of 30 kWh yielding 18 kg of CO_2 emissions based on the Spanish electricity sources mix.

The main concern about environmental sustainability is the expected efficiency gain of the Deep Neural Network FPGA implementation over a standard general purpose system. Power efficiency gains are expected to fall between 10 and 100 times, thus yielding a substantial emission reduction in any large-scale application that could benefit from the project's outcomes. One possible example is large-scale data mining systems over voice recordings.

Due to the expected power efficiency gains and low environmental footprint of the project, I give a 9 over 10 to the planning's environmental sustainability.

Part IV

Conclusions

Chapter 9

Conclusions

The project fulfilled the goal of designing and testing a voice gender identification system. The tests compared the system's performance running on an embedded general purpose architecture and a low power, energy efficient, small FPGA device. The main hypothesis of the project was that the FPGA device would yield a superior performance in terms of throughput, latency and energy efficiency. The proposed designs have confirmed this hypothesis in analytical terms and the tests performed in the target architectures have accomplished the expected results.

9.1 Summary of results

This project consisted of the design of a voice signal feature extraction system, the design of a voice gender identification system based on an artificial neural network, the implementation and optimization of the neural network designs on a general purpose architecture and on an FPGA.

This section summarizes the most relevant conclusions of each area of the project.

9.1.1 Voice signal feature extractor

The described work was a fundamental frequency estimator algorithm for speech signals. According to the design and observations from the experiments, the AMDF method provided acceptable results. The combined techniques of signal preprocessing and AMDF harmonic corrections provided a robust way to improve the pitch estimation accuracy. However, the signal preprocessing effect was overrun by the advantages of the AMDF harmonic corrections. Additionally, the voiced/unvoiced segment detector proved to be effective in improving the overall estimation performance.

9.1.2 ANN voice gender identification system

The work presented a voice gender identification system based on an ANN. Several fundamental frequency and MFCC were studied as the network input signal. The system was trained and tested using telephonic voice signal annotated with the gender of the speaker. These signals were preprocessed using the designed voice pitch estimator and an MFCC extractor to make up the network's features. The designed systems were based on deep neural networks with three hidden layers.

The studied networks used symmetric sigmoid and symmetric step activation functions. Through the observing of test results, networks using symmetric sigmoid functions had better classification performance than the ones using step functions. However, the latter had a relative performance loss not higher than 13%, which may make them worth considering due to their smaller computational complexity.

Networks using voice fundamental frequency as their features, were found to provide a good gender classification performance. The classification error of these type of networks was tested to be as low as a 6.49% using a low number of units per hidden layer. Networks using MFCC features were found to perform slightly worse than the previous ones, while requiring a higher hidden unit count to achieve their best performances.

Using the delta and delta-delta coefficients of the network features have proven to be useful to slightly improve the system performance, at the expense of a higher count of hidden units. However, the relative improvement over simpler networks was often not higher than 2%.

Networks combining fundamental frequency and MFCC features did not result in any improvement. However, linear combinations of two networks using those features were observed to produce significant relative improvements up to 14%.

As a final consideration, the system performance was tested using the neural network to classify a continuous voice signal over time. Post-processing the network outputs using a moving average filter was found to be effective to improve gender classification performance. However, the length of the filter is a parameter dependent on the characteristics of the signal fed to the system.

9.1.3 ANN general purpose implementation

The studied ANN designs were implemented on a general embedded architecture. The goal of the implementation was to optimize the ANN computation algorithm. The study focused on different techniques which reduce the numerical precision of the computation while trying to keep the classification error.

The study of the numerical range of the ANN computations showed that the designed networks are resilient to precision losses. Fixed-point implementations with a limited numerical range had a relative classification performance loss not higher than 2%.

Fixed point ANN implementations were found to produce a significant diminishment of the execution time in the tested architecture, over floating-point ones. Observations on various test network designs showed a speed-up factor ranging from 1.4 to 1.6.

Another focus of the implementation was to optimize the symmetric sigmoid activation functions. Piecewise approximation using linear interpolation was found to provide a good tradeoff between complexity and accuracy for networks implemented in floating point. A lookup table method combined with an analytical approximation of the sigmoid function was found to be the most efficient method for fixed-point implementations.

The ANN speed-up when applying the previous optimizations was found to be highly dependent on the network design. The complexity of the activation functions is linear to the number of units in the network. On the other hand, computing the network's net inputs is subject to a complexity proportional to the square of the network's units. The divergence between both complexities implies that the program fraction which calculates the activation functions diminishes as the network increases. Thus, decreasing the speedups achieved by optimizing the activation functions.

The last focus of the study was to analyze the energy efficiency of the implementations. The measured power ratings of the tested device were observed to be almost constant between each test. Therefore, the energy efficiency of the implementations were proportional to the execution time.

9.1.4 ANN FPGA implementation

The studied ANN implementations were ported to an FPGA architecture using HLS synthesis tools. The goal of the FPGA design was to achieve a better throughput, latency and energy efficiency than the general purpose architecture implementations. The FPGA design was based on the previous ANN implementation synthesized using HLS techniques. Various parallelization strategies were used to exploit the parallelizability of the ANN.

Pipelining various stages of the ANN computation resulted in a significant throughput increase. The performance limit of the pipelining was greatly determined by the latency of the multiply-accumulate operation. Another remarkable optimization was the concurrent computation of many units per layer. This was found to produce speed-ups one order of magnitude higher than other optimizations. The concurrency limit was imposed by the FPGA's available resources and the number of memory ports available for parallel memory accesses.

The FPGA design was tested and compared to a general purpose ANN implementation. The test results showed a significant speed-up and higher energy efficiency. However, the test scenario was limited to a single network design not representative of each of the studied ANNs. While synthesis results were consistent with the expected performance increase, further work should be done to validate the performance results over different network designs.

Another major design drawback was that the network dimension is limited to the local memory available inside the FPGA. The proposed design was unable to fit networks with more than 100 hidden units per layer. It could be possible to store the network data using memory external to the FPGA, however, the performance of the design would reduce to sub-optimal levels due its dependency on high-bandwidth in-chip memory accesses. This calls into question the suitability of the FPGA design for a real-time voice gender identification which only requires processing a single signal. Both the FPGA and general purpose architecture were able to achieve the real-time latency constraints when implementing networks small enough to fit inside the FPGA. Therefore, it would be desirable that the FPGA design could contain bigger ANNs which a general purpose architecture would fail to compute in real time. However, the FPGA design still achieves better performance than a general purpose implementation. That renders the proposed designs useful for applications with high throughput requirements which need to concurrently classify multiple voice signals in real time.

9.2 Personal observations

This project supposed the discovery of many interesting areas previously unknown to me. To implement the pitch estimator necessary for the feature extractor system I had to learn many signal processing concepts. Machine learning and artificial neural networks were also a new field. Finally, I had no experience in FPGA development nor hardware designs.

This project allowed me to join many abilities learned throughout my bachelor studies. Computer architecture courses have provided me with the necessary insight to understand the process of FPGA development, as well as the knowledge to design, optimize and test the proposed system. Operating system courses and PAR were also helpful to understand concurrent programming and optimal ways to instrument the developed code. Finally, PCA course helped me to understand the optimization process of an algorithm and to properly quantify the achieved results.

The heterogeneity of the project was an exciting challenge, however, the huge scope of the project delayed its objectives longer than I initially expected. This is going to be a useful experience to properly dimension new projects.

Lastly, I would like to thank the guidance and support brought by many people who supported the work on this project. Especially my family and the project's co-directors.

References

[1] C. Müller, *Speaker Classification I: Fundamentals, Features, and Methods*, 7th ed. Springer, 2007.

[2] A. Acero and X. Huang, "Speaker and gender normalization for continuous-density hidden Markov models," in *Acoustics, Speech, and Signal Processing, 1996. Vol. 1.*, 1996, pp. 342 – 345.

[3] D. Marston, "Gender adapted speech coding," *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, vol. 1, 1998.

[4] I. Potamitis, N. Fakotakis, and G. Kokkinakis, "Gender-dependent and speaker-dependent speech enhancement," *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. I–249–I–252, 2002.

[5] P. Ehkan, T. Allen, and S. F. Quigley, "FPGA implementation for GMM-based speaker identification," *International Journal of Reconfigurable Computing*, vol. 2011, 2011.

[6] M. Przybocki and A. Martin, *2000 NIST Speaker Recognition Evaluation LDC2001S97*. Philadelphia: Linguistic Data Consortium, 2001.

[7] T. Kohonen, "An introduction to neural computing," *Neural Networks*, vol. 1, no. 1, pp. 3–16, jan 1988.

[8] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, 3rd edition*, 3rd ed. Pearson Higher Education, 2009.

[9] S. Haykin, *Neural networks: a comprehensive foundation*, 1st ed. Macmillan, 1994.

[10] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[11] B. Farley and W. Clark, "Simulation of self-organizing systems by digital computer," *IRE Professional Group on Information Theory*, vol. 4, no. 4, 1954.

[12] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, pp. 386–408, 1958.

[13] P. J. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," Ph.D. dissertation, Harvard University, 1974.

[14] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and other kernel based learning methods*. Cambridge university press, 2000.

[15] G. E. Hinton, G. E. Hinton, S. Osindero, S. Osindero, Y. W. Teh, and Y. W. Teh, "A fast learning algorithm for deep belief nets." *Neural computation*, vol. 18, no. 7, pp. 1527–54, 2006.

[16] G. E. Hinton, "Learning multiple layers of representation," *Trends in Cognitive Sciences*, vol. 11, no. 10, pp. 428–434, 2007.

[17] D. Yu, L. Deng, and D. Yu, "Deep Learning: Methods and Applications," *Foundations and Trends R in Signal Processing*, vol. 7, pp. 3–4, 2013.

[18] A. R. Omondi, *FPGA Implementations of Neural Networks*, 2006.

[19] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," p. 247, 2010.

[20] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, 2014, pp. 269–284.

[21] A. Majumdar, S. Cadambi, M. Becchi, S. T. Chakradhar, and H. P. Graf, "A Massively Parallel, Energy Efficient Programmable Accelerator for Learning and Classification," pp. 1–30, 2012.

[22] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for Convolutional Networks," in *FPL 09: 19th International Conference on Field Programmable Logic and Applications*, 2009, pp. 32–37.

[23] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A Massively Parallel Coprocessor for Convolutional Neural Networks," in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2009, pp. 53–60.

[24] C. Zhang, Y. Guan, P. Li, B. Xiao, G. Sun, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks."

[25] A. Pérez-Uribe and E. Sanchez, "FPGA implementation of an Adaptable-Size neural network," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1996, vol. 1112 LNCS, pp. 383–388.

[26] A. Gomperts, A. Ukil, and F. Zurfluh, "FPGA An optimum implementation of Neural Network on FPGA," in *AAAI Spring Symposium: Embedded Reasoning*, 2010.

[27] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," *arXiv preprint arXiv:1502.02551*, 2015.

[28] Y. Konig and N. Morgan, "GDNN: a gender-dependent neural network for continuous speech recognition," *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 2, 1992.

[29] K. Meena, K. Subramaniam, and M. Gomathy, "Gender classification in speech recognition using fuzzy logic and neural network," *Int Arab J Inf Technol*, vol. 10, no. 5, 2013. [Online]. Available: http://ccis2k.org/iajit/PDF/vol.10,no.5/4476-7.pdf

[30] J. Sas and A. Sas, "Gender recognition using neural networks and ASR techniques," *Journal of Medical Informatics & Technologies*, vol. 22, pp. 1642–6037, 2013.

[31] H. Traunmüller and A. Eriksson, "The frequency range of the voice fundamental in the speech of male and female adults," *Department of Linguistics, University of Stockholm*, vol. 97, pp. 1 905 191–5, 1994.

[32] ANSIl, "Acoustical Terminology," *America*, vol. 1994, p. 58, 2004.

[33] D. M. Howard, *Acoustics and Psychoacoustics*, 4th ed. Focal Press, 2009.

[34] R. G. Bachu, S. Kopparthi, B. Adapa, and B. D. Barkana, "Advanced Techniques in Computing Sciences and Software Engineering," K. Elleithy, Ed. Dordrecht: Springer Netherlands, 2010, ch. Voiced/Unv, pp. 279–282.

[35] A. de Cheveigné and H. Kawahara, "YIN, a fundamental frequency estimator for speech and music." *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.

[36] A. M. Noll, "Short-Time Spectrum and "Cepstrum" Techniques for Vocal-Pitch Detection," *The Journal of the Acoustical Society of America*, vol. 36, no. 2, p. 296, 1964.

[37] A. Klapuri, "Qualitative and quantitative aspects in the design of periodicity estimation algorithms," *Proceedings of the European Signal Processing {…}*, no. 2, pp. 2–5, 2000. [Online]. Available: http://www.eurasip.org/Proceedings/Eusipco/Eusipco2000/SESSIONS/FRIAM/PO1/CR1908.PDF

[38] W. H. Abdulla, "Robust Speaker Modeling Using Perceptually Motivated Feature," *Pattern Recogn. Lett.*, vol. 28, no. 11, pp. 1333–1342, 2007.

[39] T. Kinnunen, E. Karpov, and P. Franti, "Real-time speaker identification and verification," in *IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, no. 1, 2006, pp. 277–288.

[40] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[41] P. Coussy and A. Morawiec, *High-Level Synthesis*. Springer Netherlands, 2008.

[42] J. Laver, *Principles of Phonetics*, ser. Cambridge Textbooks in Linguistics. Cambridge University Press, 1994.

[43] L. R. Rabiner, "Detection," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, no. 1, pp. 24–33, 1977.

[44] J. K. Lee, "Wavelet speech enhancement based on voiced / unvoiced decision," *Noise Control Engineering*, pp. 4149–4156, 2003.

[45] M. Jalil, F. A. Butt, and A. Malik, "Short-time energy, magnitude, zero crossing rate and autocorrelation measurement for discriminating voiced and unvoiced segments of speech signals," *2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering, TAEECE 2013*, no. m, pp. 208–212, 2013.

[46] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, ser. Prentice-Hall signal processing series. Prentice-Hall, 1978.

[47] D. Binnenpoorte, C. V. Bael, E. den Os, and L. Boves, "Gender in everyday speech and language: A corpus-based study," *Proceedings of Interspeech*, pp. 2213–2216, 2005. [Online]. Available: http://www.ldc.upenn.edu/myl/llog/binnenpoorte2005.pdf

[48] V. Matoušek and P. Mautner, *Text, Speech and Dialogue: 10th International Conference, TSD 2007, Pilsen, Czech Republic, September 3-7, 2007, Proceedings*, ser. LNCS sublibrary: Artificial intelligence. Springer, 2007. [Online]. Available: https://books.google.es/books?id=XBWD7oKSpg8C

[49] D. L. Elliott, "A Better Activation Function for Artificial Neural Networks. ISR Technical Report TR 93-8," Institute for Systems Research. University of Maryland, Tech. Rep., 1993.

[50] P. K. Meher, "An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks," *Proceedings of the 2010 18th IEEE/IFIP International Conference on VLSI and System-on-Chip, VLSI-SoC 2010*, pp. 91–95, 2010.

[51] PagePersonnel, "Estudios de remuneración 2015 TECNOLOGÍA," Tech. Rep., 2015. [Online]. Available: http://www.pagepersonnel.es/sites/pagepersonnel.es/files/er{_}rrhh16.pdf

[52] Tecnoempleo.com, "Informe Empleo Informática." [Online]. Available: http://www.tecnoempleo.com/informe-empleo-informatica.php

[53] OECD, "OECD.Stat Average annual hours actually worked per worker." [Online]. Available: http://stats.oecd.org/Index.aspx?DatasetCode=ANHRS

Part V

Appendices

Appendix A

Developed tools

This appendix describe the developed tools used in this project.

A.1 Voice tools

A.1.1 pitch_extractor

This python scrips implements the AMDF pitch extraction algorithm described in chapter 3. It takes a input wave file and prints the estimated pitch of each frame of the signal into the standard output.

Usage

usage: amdf_f0.py [-h] [-d] [-u] file

Extract F0 using AMDF algorithm

positional arguments:

file file to process

optional arguments:

-h, –help show this help message and exit

-d, –drawplot draw plot

-u, –includeunvoiced include unvoiced frames

A.1.2 mfcc_extractor

This python scrips implements the MFCC feature extraction algorithm described in chapter 2. It takes a input wave file and prints the mfcc features of each frame of the signal into the standard output.

Usage

usage: mfcc.py [-h] [-d] [-u] file

Extract MFCC

positional arguments:

file file to process

optional arguments:

-h, –help show this help message and exit

-d, –drawplot draw plot

-u, –includeunvoiced include unvoiced frames

A.1.3 delta

This python script implements the delta feature transformation algorithm described in chapter 2. It takes a input feature file and prints the delta features of each frame of the signal into the standard output.

Usage

usage: delta.py [-h] [-d] input

Reads feature files and produces delta files.

positional arguments:

input path of the feature file to calculate deltas from

optional arguments:

-h, –help show this help message and exit

-d, –deltadelta calculates delta-delta (acceleration) values

A.2 Fann tools

A.2.1 fann_train

This C program implements the ANN training algorithms using a custom modified version of the FANN library. Its inputs are the train and test data files containing the annotated feature vectors. Its output is the trained network. The tool provides information about the training process.

The number of threads used for the training algorithms can be controlled by the environmental variable FANN_THREADS.

Usage

Usage: fann_train [OPTION...] train_data test_data out_network hidden_units

fann_train – Trains a neural network using fann library

-a, –algorithm=incremental | batch | rprop | quickprop

Changes training algorithm (default is rprop)

-d, –dropout-fraction=#.# Sets the dropout fraction of the network (default is 0.0)

-e, –error=#.# Sets test set desired error (default is 0.001)

-h, –hidden-activation=sigm | sigmsym | step | stepsym

Sets the hidden units activation function (default

is sigmsym)

-l, –learning-rate=#.# Sets the training algorithm learning rate

-m, –momentum=#.# Sets the training algorithm momentum

-o, –output-activation=sigm | step

Sets the hidden units activation function (default

is sigm)

-t, –train-errfunc=linear | tanh

Changes training error function (default is tanh)

-x, –max-epochs=# Sets the maximum number of epochs to train

(default is 1000000)

-?, –help Give this help list

–usage Give a short usage message

Mandatory or optional arguments to long options are also mandatory or optional

for any corresponding short options.

A.2.2 fann_format

This python script reads pitch files from a male and female paths and produce labelled FANN train and test partitions for supervised learning.

The outputs trainfile and testfile contain the ANN input feature vectors.

Usage

usage: pitch_fann_format.py [-h] [-p] [-m] [-s] [-n] pathmale pathfemale trainfile testfile frames outputs

Reads pitch files and produces fann train and test data.

positional arguments:

pathmale path of male MFCC files directory

pathfemale path of female MFCC files directory

trainfile output train file

testfile output test file

frames number of audio frames of the ANN input vector

outputs number of output units of the ANN

optional arguments:

-h, –help show this help message and exit

-p, –packed reduces the output files sizes by not overlapping the analysis frames

-m, –meannorm normalize the coefficients by the population mean

-s, –meanstdevnorm normalize the coefficients by the population mean and standard deviation

-n, –minmaxnorm normalizes by minimum (80) and maximum (350) pitches

A.2.3 mfcc_fann_format

This python script reads MFCC files from a male and female paths and produce labelled FANN train and test partitions for supervised learning. The outputs trainfile and testfile contain the ANN input feature vectors.

Usage

usage: mfcc_fann_format.py [-h] [-m] [-s] [-p] [-d] pathmale pathfemale trainfile testfile frames

outputs coeff

Reads MFCC files and produces fann train and test data.

positional arguments:

pathmale path of male MFCC files directory

pathfemale path of female MFCC files directory

trainfile output train file

testfile output test file

frames number of audio frames of the ANN input vector

outputs number of output units of the ANN

coeff number of MFCC per frame

optional arguments:

-h, –help show this help message and exit

-m, –meannorm normalize the coefficients by the population mean

-s, –meanstdevnorm normalize the coefficients by the population mean and standard deviation

-p, –packed reduces the output files sizes by not overlapping the analysis frames

-d, –discardfirst discards the first MFCC of each frame

A.2.4 join_fann_data

This python script join two FANN feature vector files in order to produce a composited feature vector file.

Usage

usage: join_fann_data.py [-h] input1 input2 output

Joins fann data files.

positional arguments:

input1 path of the first file to join

input2 path of the second file to join

output path of the joined file

optional arguments:

-h, –help show this help message and exit

Appendix B

Zynq ZC702 power measurements

This appendix describes the methodology employed to measure the power rating of the Zynq ZC702 board.

According to Xilinx "Zynq-7000 AP SoC Low Power Techniques part 2"[1] technical documentation:

"The ZC702 board uses power regulators and a PMBus compliant system controller from Texas Instruments to supply core and auxiliary voltages to the Zynq 7000 APSoC.

There are 5 switching regulators (PTD08D210W) and 1 linear regulator which generate different voltages required for the Zynq-7000 APSoC as well as the on-board components present on the ZC702 board. The voltage and currents supplied by these voltage regulators are continuously measured and monitored by three Texas Instruments digital power controllers (UCD9248) available on the ZC702 board."

The Zynq-7000 power domains are distributed according to the table:

Name	**Description**
VCCINT	1.0V nominal supply of Zynq 7000 that powers all of the PL internal logic circuits.
VCCPINT	1.0V nominal supply that powers all of the PS internal logic circuits.
VCCAUX	1.8V nominal supply that powers all of the PL auxiliary circuits.
VCCPAUX	1.8V nominal supply that powers all of the PS auxiliary circuits.
VCCADJ	Supplies power to the VCCADJ power net on the ZC702 board. The list of components powered by VCCADJ net apart from the Zynq 7000 on ZC702 board is listed in the Appendix A: List of Components using the ZC702 power supplies section.

[1] http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+Low+Power+Techniques+part+2+-+Measuring+ZC702+Power+using+TI+Fusion+Power+Designer+Tech+Tip

Name	Description
VCC1V5PS	Supplies power to the VCCO_DDR power domain of the Zynq 7000 as well as the 4 Micron DDR3 (MT41J256M8HX-15E) components on the ZC702 board. This is a 1.5V nominal supply that supplies the DDR I/O bank input, output drivers and termination circuitry.
VCC_MIO	Supplies power to the VCC_MIO power net on the ZC702 board.VCC_MIO supplies power to PS_MIO power domain of the 1.8V nominal supply that supplies power to the PS_MIO [53:0], PS_CLK, and PS_POR_B I/Os banks of Zynq 7000 as well few components on the ZC702 boards.The list of components that are powered by VCC_MIO supply apart from the Zynq 7000 on ZC702 board is listed in the Appendix: List of Components using the ZC702 power supplies section.
VCCBRAM	VCC3V3 PTD08D210W(U21) VoutA The VCC3V3 supplies 3.3v power to the components available on the board which also includes the UCD9248 power controller itself. The list of components that are powered by VCC3V3 on ZC702 board is listed in the Appendix A: List of Components using the ZC702 power supplies section. VCC2V5 PTD08D210W(U21) VoutB The VCC2V5 supplies 2.5v power to few PL IO banks as well as the components available on the board. The list of components that are powered by VCC2V5 on ZC702 board is listed in the Appendix A: List of Components using the ZC702 power supplies section.
VCC3V3	The VCC3V3 supplies 3.3v power to the components available on the board which also includes the UCD9248 power controller itself. The list of components that are powered by VCC3V3 on ZC702 board is listed in the Appendix A: List of Components using the ZC702 power supplies section.
VCC2V5	The VCC2V5 supplies 2.5v power to few PL IO banks as well as the components available on the board. The list of components that are powered by VCC2V5 on ZC702 board is listed in the Appendix A: List of Components using the ZC702 power supplies section.

The UCD9248 can be accessed using the I2C device driver controllers of the Linux kernel. The driver provides information on voltage, intensity and power rating of each rail.

In order to measure the code power rating, instrumentation was performed implementing a parallel thread concurrent to the process intended to measure which reads the data from the power controller. This thread is run every 5 milliseconds gathering 50 consecutive reads of the power controller. Once the power measurement has finished, the average of all the measurements for each rail is calculated.

According to the power domains table the following rail assignments was defined for each measurement.

Processing system (PS): rails VCCPINT, VCCPAUX, VCC1V5PS.

Programmable logic (PL): rails VCCINT, VCCAUX, VCCBRAM.

Board components: rest of the rails.