Discriminative Acoustic Features for Deployable Speech Recognition

Arlo Faria

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-199
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-199.html

December 13, 2016

Discriminative Acoustic Features for Deployable Speech Recognition

by

Arlo M Faria

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Nelson Morgan, Chair
Professor Jerome A Feldman
Professor Keith A Johnson

Fall 2014

Discriminative Acoustic Features for Deployable Speech Recognition

Abstract

Discriminative Acoustic Features for Deployable Speech Recognition

by

Arlo M Faria

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Nelson Morgan, Chair

This work explores discriminative acoustic features: audio signal representations produced by multilayer perceptrons, such as Tandem and bottleneck features used in state-of-the-art automatic speech recognition systems. Experimental results highlight the factors that influence performance in terms of accuracy and speed; novel approaches are introduced to provide improvement in both regards. The overall emphasis is on discovering techniques that are suitable for practical deployment, translating effectiveness beyond a traditional research setting. Applications include real-time low-latency audio stream processing on mobile devices – as well as systems that are built with low-quality training data and must encounter the diverse complications of "real-world" use cases.

In memory of Cromwell Daisaku Mukai (1917 - 2001)

Honoris causa inter silbas academi restituere justitiam

University of California, Berkeley
December 13, 2009

Contents

List of Figures

List of Tables

Acknowledgments

Thank you, Morgan, for allowing me to pursue so many varied interests during my time at ICSI. This dissertation cannot capture the breadth and depth of knowledge that I have gained since we first met in 2003. I am forever grateful that you accepted me.

It was crucially important that you connected me with David Gelbart that summer. To the extent that I can call myself a researcher, I credit it all to his mentorship. David, I deeply regret that we were unable to collaborate further.

Thanks, everyone else, for letting me do my thing.

Chapter 1

Introduction

Speech communication is rather magical: it is a process that takes words[1] from one brain and invisibly transfers them into another brain. This is typically achieved via extremely complex neurophysiological mechanisms involving the human vocal apparatus and auditory system. A precise understanding of these phenomena is still far beyond the realm of current scientific knowledge. In this increasingly advanced technological age, speech and language remain largely unexplained as mysteries of human evolution.

Nonetheless, scientists and engineers have created a flourishing field of research by establishing a well-defined and addressable problem. *Automatic speech recognition* (ASR) is presented as follows: determine the words that were uttered by the human who produced a captured audio signal. Or conversely, determine the words that would be understood in the brain of a human who would have heard this same audio signal.

Curiously, one does not need to understand anything about the human brain in order to make progress with this problem.[2] A tractable approach has evolved over more than a century's worth of insight and discovery. The conveyed speech is a message that is sent over the air, which is a wholly physical medium despite its otherwise difficult-to-grasp invisibility. These patterns of air pressure variation (sound waves) are therefore susceptible to interception by electromechanical means (microphones), and subsequent conversion into digital signals. At this point, the problem is reduced to one of mathematics – in the broadest sense, including: statistics, linear algebra, logic, etc. – and can be addressed by engineering efforts that leverage a variety of modern tools.

It is in this context that this thesis is presented. It is primarily an engineering effort: the work builds upon well established theoretical foundations but also exploits a large number of practical shortcuts that may accelerate progress toward the ultimate research goal of ASR: to reduce the word error rate (WER). However, while some new ideas will be introduced, and validated with experimental results, the overall purpose is improve understanding of factors

[1] Words, not ideas. Language understanding is rather different; it is a far more difficult problem.

[2] On the other hand, psychoacoustic phenomena such as critical band auditory filters have certainly been helpful in some engineering designs. However, a deeper understanding of higher-level brain functions may be required to *solve* the speech recognition problem.

that can advance understanding of the state of art. We aim to accelerate the ultimate deployment of ASR technologies for everyday use. The work therefore has a strong emphasis toward discovering solutions that are simple, fast, and *effective enough*. The thesis itself is that in many situations, these are desirable goals.

We consider the predominant paradigm of statistical ASR systems that are trained from data. The particular focus and contribution is in exploring discriminative acoustic features produced with the use of a multi-layer perceptron (MLP). This work fits in a broader research agenda that is currently popular: *deep learning* using *deep neural networks* (DNN), a term that has gained nearly mainstream adoption. While acknowledging the many recent advances in improved techniques reported with DNNs, this work presents an overview of a relatively simplified approach. Although it is not traditionally considered "deep", the MLP is a generalization of a DNN, and most of the experimental results should apply to both cases. From a personal perspective, here the term *DNN* is considered less appropriate than *MLP*.[3].

In addition to targeting reduced WER with systematic improvements, this work considers problems of a practical nature: how can we effectively use MLPs under various types of resource constraints? Also, how can we exploit massive amounts of "real-world" data to train MLPs? The techniques found in these explorations could advance speech technologies such as ASR and spoken keyword search, but more importantly will enable the state of the art to become more accessible beyond traditional research settings.

Chapter 2 will introduce Tandem acoustic features produced with an MLP classifier. This chapter will present a novel hypothesis: knowledge of classification errors should be exploited to produce better features during the system training phase. This idea is first validated using *idealized* Tandem features, and then experiments show improvement in accuracy using *corrected* Tandem features. Results will be presented on Mandarin broadcast news speech collected for the DARPA GALE Program. These systems contributed a feature extraction component used in some of the best results ever reported on that data set, built together with the University of Washington and SRI. Most interestingly, the experiments with idealized Tandem features demonstrated that near-perfect ASR performance may be possible by improving MLP classifiers.

Chapter 3 will describe techniques for improving the speed of MLP computations, mostly in the context of run-time (i.e. test-time) feature extraction but also for system training. In addition, Chapter 3 explores the audio signal processing that is required prior to MLP evaluation. These benchmarks establish the minimum resource overhead requirements for real-time MLP processing. Modest improvements are also discussed, principally in terms of using fast pitch extraction as an effective substitute for more complex techniques that may be impractical or impossible to deploy for real-time streaming operation.

In Chapter 4, some of the earlier performance results will be revisited to consider frame batching in the context of system training. Although this speeds up large matrix multiplications, this work emphasizes that such efficiency gains must be traded off against the

[3]If the reader prefers, please interpret *MLP* as "feedforward neural network, possibly shallow"

additional run-time latency and the negative effect on real-time stream processing. We also show that large batch sizes can also affect overall system training time in unexpected ways, by slowing the algorithmic convergence of MLP training.

A novel CPU-based fast approximation of transcendental functions is also presented, although it is primarily described in Appendix A. Although these functions are commonly single-instruction primitives on a GPU, we argue that such hardware is not typically available for applications on modern consumer portable devices. Also, these functions also dramatically speed up Baum-Welch HMM training, which requires many log-sum operations that are not easily parallelized.

Chapter 4 will also introduce transcript alignment, motivating this exploration from both a research and application perspective. Automatic phoneme alignment of carefully curated research datasets can be considered a solved problem; on the other hand, "real-world" speech presents many unexpected complications. Several solutions will be proposed to address problems such as: approximate versus optimal alignment algorithms, computational complexity and scalability of alignment for lengthy inputs, robustness to inaccurate transcriptions and non-speech audio, as well as portability to new languages. These considerations were explored by an alignment service that has been successfully deployed to customers in the video captioning industry for the past several years, and is presented as a case study.

This thesis presents a guide to help speech technology practitioners to build state-of-the-art systems that can perform under realistic conditions. Discriminative acoustic features are very effective, and thus highly recommended for inclusion in ASR systems. Although they are rather more complicated and computationally intensive than standard acoustic features, this work provides a set of design considerations that can guide development toward successful deployment. The ultimate goal is to enable discriminative acoustic feature extraction on mobile computing devices – using systems that can be rapidly trained and deployed with any kind of speech data.

Chapter 2

Discriminative Acoustic Features

This chapter introduces speech feature representations produced by discriminatively trained multi-layer perceptrons. Previous research has demonstrated that such a *tandem* approach can be successfully exploited for large-vocabulary automatic speech recognition systems. The principal aim of this work is to empirically evaluate some variants of these features. While experimental results validate some of the design choices of the standard implementation, other evidence suggests alternatives that may improve performance. From this exploratory investigation, we hypothesize which of the various modifications are most promising; applied to a Mandarin broadcast news task, the new configuration demonstrates significant improvement. Along with the novel presentation of a "best-case scenario" and other "cheating" experiments, an interpretation of these results is discussed with the hope of guiding future directions of research.

Much of this work predates the evolution of so-called "deep neural networks" used in more modern systems, in which multi-layer perceptrons are utilized for "hybrid" decoding or to produce "bottleneck" features. Nonetheless, many of the earlier results are still valid, as demonstrated in some more recent experiments conducted with Cantonese and Vietnamese conversational telephone speech.

2.1 Background: Automatic Speech Recognition

The traditional statistical formulation of automatic speech recognition (ASR) presents a generative model in which training and decoding operate under the maximum likelihood criterion. Several approaches have introduced more discriminative techniques: model parameter estimation based on alternative criteria (MMI [1], MPE [2], [3]); linear feature transforms (fMPE [4], MPE-HLDA [5]); and posterior-based features (Tandem [6], TRAP [7], Tandem/HAT [8]). This work is only concerned with the last of these, particularly Tandem features produced with a multi-layer perceptron (MLP), although it has been demonstrated that these three discriminative approaches can be combined to provide complementary improvements [9].

In the predominant Hidden Markov Model paradigm for large-vocabulary ASR, acoustic observation likelihoods are computed from a Gaussian Mixture Model. Due to the assumptions of the HMM-GMM framework, distributions are most accurately modeled for acoustic features that are relatively low-dimensional, somewhat decorrelated, and fairly localized to a single centisecond frame. By contrast, neural network structures have been used to classify inputs which are high-dimensional and temporally correlated by the inclusion of neighboring frames as context. Hybrid connectionist HMM-ANN systems [10], [11] were developed as an alternative to the HMM-GMM, taking advantage of the MLP's model accuracy, context sensitivity, and parsimonious use of parameters. However, incremental enhancements such as speaker adaptation and discriminative parameter estimation were more easily implemented in conventional systems [12], which exhibited much better performance in an evaluation of large-vocabulary broadcast news recognition [13].

A subsequent evaluation, the ETSI Aurora task [14], compared performance of feature extraction front-ends in noisy background conditions. Since all participants were required to use the same HMM-GMM architecture for acoustic modeling, groups with a connectionist background were forced to compromise. Tandem acoustic modeling [15], [16] was thus developed to conveniently enable conventional systems to exploit the advantages of the MLP, by simply treating a MLP-derived phone posterior probability distribution as if it were an ordinary acoustic feature vector. In noisy conditions, these Tandem features were shown to be more robust than standard ASR features. Tandem features and other MLP-derived variants have since been successfully employed in large-vocabulary ASR systems [6], [8], [17]. A prevailing explanation for the positive effect of these features is that the MLP performs a nonlinear feature transformation into a space which is explicitly oriented for discriminability of phones, the underlying structural units of the HMM architecture. Such an interpretation has been investigated in [16], [18], and in similar context by [19].

Much of the work in this chapter was performed in 2007, immediately prior to an explosion of interest in this field. Before this time, researchers had experienced considerable difficulty training multilayer perceptrons that comprised more than one or two hidden layers. A revolution started with Hinton's discovery of effective training algorithms for *deep neural networks* [20], originally demonstrating superior results in the field of computer vision. Soon after, researchers at Microsoft showed that a hybrid speech recognition system using such a deep architecture was capable of highly dramatic performance improvement [21]; this relatively simple speech recognizer essentially matched the improvement seen in a far more sophisticated GMM discriminative training and multi-pass speaker adaptation steps that had typically been characteristic of all state-of-the-art ASR system until that time. Meanwhile, researchers also continued to explore acoustic features derived from discriminative classifiers; however, most have since focused on *bottleneck* features [22]–[25], rather than traditional Tandem features based on phone posteriors or MLP output layers. A particular advantage of bottleneck features is that the MLP architecture defines the ultimate dimensionality of the features, rather than a language-specific peculiarity such as the number of units in a phonemic inventory. This is especially convenient when building multilingual ASR systems, or conducting research with systems applied across multiple languages, because it does not

require the experimenter to recalibrate certain parameters that are very sensitive to the dimensionality of features.

This chapter first describes Tandem acoustic modeling and other related MLP features. The experimental objective is to investigate the factors which influence performance on a Mandarin broadcast news task. In controlled experiments designed to isolate the variables involved, none of the modifications has a particularly large effect although some general trends are noticeable. Guided by these results, the most promising modifications are then selected and applied together to synthesize a significantly improved system.

Additionally, a series of cheating experiments simulate what is perhaps the "best-case scenario" for features based on phone posteriors – and which might also be considered an upper-bound on acoustic modeling techniques for ASR. Observations and insights from these experiments are discussed later in this chapter, considering future research possibilities for MLP feature extraction.

2.2 The Tandem approach for producing MLP features

Unfortunately, a confusing nomenclature has previously been used to describe the Tandem acoustic modeling approach, its associated features, and their variants. For clarity of exposition, we will therefore observe the following distinctions:

The Tandem approach to feature extraction is a data processing paradigm in which a speech signal is transformed into a sequence of acoustic observation features with the use of a classifier trained to discriminate speech units.

MLP features are the product of the Tandem approach when the classifier is implemented as a multi-layer perceptron. This term can often be used to describe what are also known as *posterior* or *probabilistic* features, but also includes other types of non-linear transformations – such as *bottleneck* features.

Tandem features are specifically MLP features produced from a fairly standard configuration, described in Sec. 2.3. Most experiments in this chapter explore variations on this *Tandem baseline*.

Bottleneck features are specifically MLP features produced as linear activations from a hidden layer of a multi-layer perceptron; they are typically used in *the Tandem approach*. Only the final experiments in this chapter explore such features.

The central component of Tandem acoustic modeling is a mechanism which is able to classify pre-processed acoustic inputs according to a finite set of categorical units. The outputs of this classifier are then post-processed and presented to a conventional ASR system as if they were acoustic observation features. Figure 2.1 depicts the general procedure.

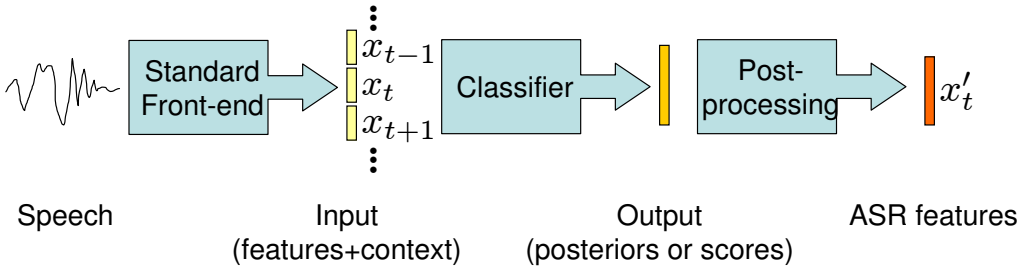


Figure 2.1: The Tandem acoustic modeling approach.

From the vantage point of a system builder, the process can be viewed as a separate feature extraction module that is independent of the modeling architecture, typically a HMM-GMM. Such black-box modularity is generally a desirable characteristic in complex ASR systems. For example, vocal tract length normalization can be implemented simply and efficiently using a generic speech reference model, as described by [26]: this allows the procedure to be entirely contained within the feature extraction module, as opposed to more complex approaches which integrate warp factor estimation into acoustic model training and multi-stage decoding. It is not uncommon to build ASR systems that use a completely different software toolkit for feature extraction, using acoustic features as the compatible interface to the acoustic modeling and decoding modules.

Tandem features

Traditionally, the most common choice for the classifier has been a three-layer MLP, a feed-forward neural network structure. (More recently, there has been an explosion of interest in networks with "deeper" structure, often comprising five or more layers. Despite the new nomenclature, these "deep neural networks" may still be considered as specific instances of multilayer perceptrons, or more generally as feed-forward neural networks.) The acoustic signal is usually processed by a standard ASR front-end to produce a centisecond stream of PLP features; however, the input to the network is often defined as a 9-frame context comprising the current frame plus four consecutive frames preceding and following. Output units of the MLP are determined by the phonetic inventory of a given language; training targets used to learn the network structure are derived from data which has been manually labeled or automatically aligned to reference transcriptions. The network's weight parameters are learned from training data using the back-propagation algorithm with a cross-entropy error function.

Distributions produced by the MLP are vectors in a probability space which is not easily modeled by a GMM; thus, probabilities are approximately Gaussianized by conversion to the logarithmic domain. The features are further processed by applying a Karhunen-Loeve

Transform (i.e., Principal Components Analysis), which orthogonalizes the features to satisfy the typical diagonal covariance GMM assumption; additionally, this procedure enables a dimensionality reduction by retaining only the feature components which contribute most to the overall variance of the data. Finally, the resulting vector is concatenated with a standard ASR feature vector, typically MFCC, to serve as higher-dimensional observations for the acoustic models.

Other MLP features

Whereas Tandem features are derived from a relatively medium-term context of cepstral feature inputs, another common class of MLP features considers an input representation of logarithmic critical band energies (via PLP analysis [27]) over a much longer temporal context, up to a full second. Influenced by Fletcher's experiments suggesting that speech information is independently conveyed in separate critical bands, TRAP processing [7] first attempts to estimate phone posterior probabilities using each critical band in isolation. Outputs from these Neural TRAP classifiers are then presented as inputs to a merger network, which is able to provide more accurate estimates. A similar hierarchical approach is used for HATs features [28], where the input to the merger network is composed from the hidden layer activations of the independent critical band classifiers.

A practical alternative to HATs is the Tonotopic MLP [29], a four-layer network with a partially connected input structure, allowing joint learning of the critical band and merger network weights. Four-layer network structures and other HATs alternatives were investigated in [30], which also discussed a method for reducing the input's large dimensionality due to the long temporal context. Another possible remedy is explored as the Split Temporal Context approach of [31].

The MLP does not necessarily have to be trained to classify phones. Indeed most modern systems now train on some form of clustered triphone states instead; in many instances, the clustered units are called *senones* [21]. Hierarchical expert networks have been used in [32] to distinguish speech versus non-speech sounds, as well as voicing states. Researchers have also investigated the use of articulatory features, using broad phonological classes as MLP targets [33]. It is hoped that a trained network might be language-independent, and could be applied to resource-poor languages. Portability across languages and domains is discussed in [34], and a solution is proposed in the form of a multi-task MLP by [35].

Due to the variety of MLP features, there has also been considerable interest in feature combination strategies. Simple addition of feature vectors can work well, but an inverse entropy-weighted combination [10], [36] is often preferred because of the probabilistic interpretation of the phone posteriors. Other experiments have also demonstrated that a better combination rule might be possible [37], inspired by the Dempster-Shafer theory of evidence and also appealing to Fletcher's observation of the product-of-errors phenomenon in human speech perception.

The application of the Tandem approach for generating MLP features is not restricted to processing of posterior probability distributions. Bottleneck features [17] can be derived from

a very narrow middle layer of a five-layer MLP; interestingly, the dataflow from the inputs through such a constricted layer does not greatly deteriorate phone discrimination at the output layer. It is hypothesized that the hidden activations of the middle layer project the input to a low-dimensional discriminable space, obviating the need for further dimensionality reduction techniques.

Although most MLP classifiers consider inputs from a temporal context of multiple frames, a speech signal carries a much greater amount of information at the utterance level. An alternative and possibly complementary method for estimating posteriors is to perform the top-down inference afforded by a HMM structure – e.g., using the forward-backward algorithm. A goal of such a research agenda [38], [39] is to allow the feature extraction module to exploit all possible acoustic evidence, including high-level word and phonological constraints encoded in the HMM's topology.

One of the most interesting recent advances in discriminative acoustic feature extraction using deep neural networks has been the development of *auto-encoder* bottleneck features [23]. This was inspired by results from computer vision that demonstrated a network's ability to produce compact low-dimensional abstract representations of images with the first layers of the network; layers after the bottleneck would then reproduce the original input. This is similar to image compression.

2.3 Experiments

At the International Computer Science Institute, research efforts have focused on Tandem features, HATs features, articulatory features, and their combination for speech recognition of Mandarin and Arabic broadcast news. For the experiments in this chapter we consider only Tandem features, primarily for practical reasons: the hierarchical structures of HATs and articulatory systems are implemented relatively inefficiently, making it extremely cumbersome to perform multiple experiments in a short period of time. Moreover, many of the experimental modifications are generally applicable to the Tandem approach, so it is presumed that they will also benefit other MLP features. Experiments are performed on Mandarin broadcast news because we have access to resources for developing a state-of-the-art baseline system for this task, though similar results may reasonably be expected for other domains.

A baseline system for Mandarin broadcast news

The baseline ASR system is based on the Mandarin broadcast news recognizers developed by the Univ. of Washington, using SRI's DECIPHER system. Thanks to a close collaboration between ICSI, SRI, and UW, it was possible to replicate a system presented in [40], [41].

Systems were trained on a relatively small data set of 29 hours of Hub-4 broadcast news shows, accurately transcribed along with speaker labels. The acoustic features were based on standard 39-dimensional MFCC plus the first two derivatives, prepared with fast GMM-based

maximum-likelihood vocal tract length normalization [26] and mean-and-variance normalization applied on a per-speaker basis. Because Mandarin is a tonal language, it has been found that better recognition can be attained using a smoothed log-pitch estimate [40] and its two temporal derivatives, which were appended to the MFCC features to result in a 42-dimensional acoustic feature vector. Acoustic models were based on a 72-unit phoneset comprising consonants and tonal vowels at four levels, with tone 5 mapped to tone 3. Elementary HMM units with a 3-state no-skip Bakis topology were used to represent within-word triphones. Model parameters were tied across 2000 states, clustered using a phonetic decision tree [42]. Each state's observation distribution was modeled by a diagonal covariance GMM with 32 mixture components. Maximum-likelihood parameter estimation was used to train on data which were iteratively re-aligned with the Viterbi algorithm.

The test set considered in this chapter is the hour-long RT04 evaluation set (eval04), although many experiments were conducted on a one-third subset of this data (CCTV) which was more similar in style to the training data. The recognition decoder first applied an automatic segmentation of the test data into 5-10s utterances, which were then assigned pseudo-speaker cluster labels. The recognition network was compiled from a word-level bigram language model trained on 121M words, with a lexicon of 49K words. Two heavily pruned forward passes of the decoder were separated by a round of 3-class MLLR speaker adaptation.

There were a few aspects of the language which required special attention, such as the specially processed pitch features described in [40]. Additionally, an automatic maximum-likelihood segmentation procedure [41] was devised to pre-process the text of the training transcriptions, since Chinese is written without whitespace to delimit word boundaries. While the pronunciation of a string of Chinese characters is to some extent independent of the segmentation, it is nonetheless important to have an accurate word transcription because the ASR system's vocabulary and language modeling operate on a consistent decomposition of word units. It should be noted, however, that the output of Mandarin ASR systems is typically scored against a reference transcription using the metric of character error rate (CER), rather than the usual word error rate.

Table 2.1 shows the performance of the baseline system, detailing the relative improvements due to modeling pitch features (15%) and performing a second recognition pass after speaker adaptation (10%). Although the baseline system created for these experiments was intended to be identical to [40], [41], there were slight differences – for reasons which are unknown but likely trivial.

A baseline system for Tandem features

The baseline system used for Tandem features is based on a setup used at ICSI for the 2007 GALE evaluation. The input to the MLP comprises 378 units, representing 9 frames of PLP and pitch features (and two derivatives). The outputs are 71 phone units – identical to the phoneset used for the baseline HMM-GMM system described previously, minus the "reject" phone. The labeled data used to train the network were derived from a forced-

Feature type	CCTV 1st-pass	CCTV Spkr-adapt	eval04 Spkr-adapt
MFCC	15.5	13.9	24.2
MFCC (UW)	15.5	13.9	24.1
MFCC+F0	13.0	11.7	21.0
MFCC+F0 (UW)	13.0	11.7	21.4
Tandem	11.1	10.6	19.7

Table 2.1: Performance (CER) of the baseline ASR systems for a Mandarin broadcast news task. The goal of this work was to replicate a system built at the University of Washington (UW).

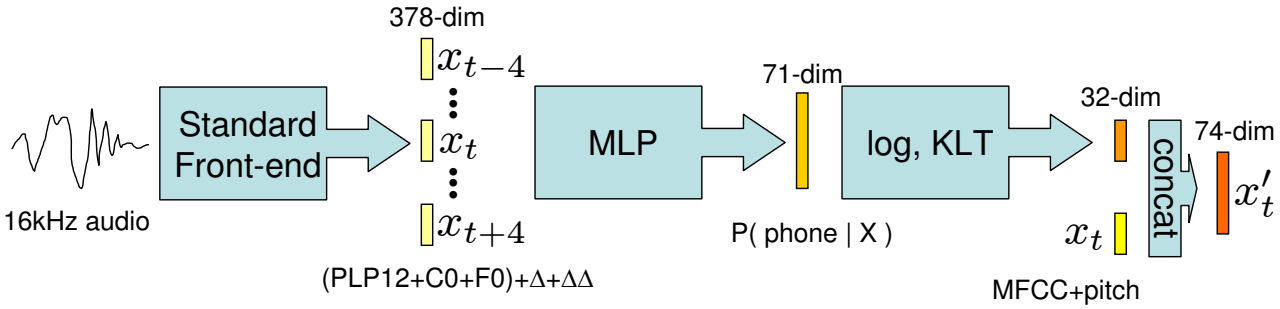


Figure 2.2: The baseline configuration for producing Tandem features.

alignment of the reference transcriptions using the baseline MFCC+F0 acoustic models, and mapping triphone states to monophone targets. The network was constructed with 1150 hidden units, or about 500K learnable weight parameters. This setting was determined in reference to [43], with a 20:1 ratio of training data frames to weights. After applying a softmax non-linearity to the activations at the output layer, a logarithm was applied, followed by a KLT orthogonalization and dimensionality reduction to 32 feature dimensions. The resulting vector was then concatenated with the 42-dimensional MFCC+F0 features used in the baseline system described above, resulting in a 74-dimensional Tandem feature. This process is summarized by Figure 2.2.

Table 2.1 shows the performance of this baseline Tandem system. While the Tandem features provide an improvement over the standard features, the effect is more dramatic in the first pass than after speaker adaptation, perhaps because the Tandem features are more speaker-independent. The rows labeled as "UW" indicate the performance of what should have been an identical system developed at the University of Washington; the pitch features used in this work's baseline were implemented based on the approach described in [40], with minor differences due primarily to the difference in the choice of spline interpolation: a simple

bicubic versus PCHIP (piecewise cubic Hermite interpolating polynomial). The virtually identical results suggest that the re-implemented pitch features suitably replicated the earlier work, and that the specific choice of interpolation function was not terribly important.

In the following experiments, we will attempt to modify this Tandem baseline system along one configurable parameter at a time. This will serve to guide intuitions about how to improve the system.

Modifying the input representation

The first set of experiments considers the nature of the input to the MLP. While there are some potentially confounding factors, the experiments can be justified from a practical perspective.

For optimal performance, it would be reasonable to make the input representation as rich as possible, in order to allow the MLP to learn any necessary patterns while ignoring less informative or redundant aspects. The expressiveness of the input is usually reflected in the size of the input layer to the MLP, which in turn affects the number of weight parameters which must be updated during training. Because training time is an overriding practical constraint, for a reasonably fair comparison we can decide to hold the training time constant. When varying the input layer size, the number of hidden units was also adjusted so that the network always maintained about 500K weight parameters, creating a trade-off between the expressiveness of the input versus the discriminative power of the network.

Table 2.2 displays results from several systems using different input representations. Character error rate is shown for both the eval04 testset and its CCTV subset. For the CCTV data, it was possible to evaluate the frame-level phone accuracy of the MLP classifier on the test data, using labels from the alignment procedure described in Sec. 2.4. The eval04 data was only used in cases where we wanted to further investigate a promising result from the CCTV data. To guide experimentation, p-values from a two-tailed MAPSSWE significance test [44], [45] were computed relative to the Tandem baseline system; assuming independence of errorful segments separated by two or more correctly recognized characters (rather than words, as usually applied), the MAPSSWE test typically generated sample sizes of over one hundred segments. The two-tailed test was used because it was unclear how to predict the direction of system differences. In most cases, the significance levels were above the usual threshold of $p < 0.05$, but were nonetheless useful signals to consider along with other experimental evidence and intuitions.

It is clear that pitch features are very helpful, since their removal substantially decreases frame-level phone classification accuracy and increases CER. This may appear obvious given the contributions seen in the baseline HMM-GMM system, but it was a helpful innovation for Tandem systems; it was also explored in a related context by [46]. In current systems used for the IARPA Babel Program, nearly all competing teams have discovered that pitch features are helpful, even for non-tonal languages. One possible explanation is that the voicing information (pitch features are often joined with a probability-of-voicing estimate) may help to distinguish or represent vowel sounds more accurately. Another interpretation

Feature type	CCTV CER	CCTV Acc.	eval04 CER
Tandem baseline	10.6	70.4	19.7
... w/o F0 inputs	11.2	67.7	
... w/ MFCC inputs	10.9	69.9	
... w/o context	11.2	60.9	
... ±1 frame	10.7	67.6	
... ±2 frames	10.2	69.4	19.6
... ±3 frames	10.4	70.2	19.6
... ±5 frames	11.0	70.6	
... ±6 frames	10.7	70.6	
... w/o Δ or ΔΔ	10.3	66.0	19.9
... w/o ΔΔ	10.1	69.1	19.7

Table 2.2: Effects of varying the representation of inputs to the MLP. Character error rate is presented in the first and third columns; test set frame-level phone classification accuracy is given in the second column.

could be argued for the case of intrinsic pitch (F0) of vowels [47]; for example, vowels with an articulation described as "low central" tend to have lower pitch than vowels described as having a "high" place of articulation. Another perspective could consider that acoustic features derived from a spectral envelope may be "contaminated" by the harmonics; by including pitch information, it may be possible for a classifier to remove those effects.

It also helps to use PLP features rather than MFCC features for the inputs. In addition to giving providing better frame-level phone classification accuracy, there may also be complementarity provided in the concatenation with the MFCC features at the final step of the Tandem process. Such crossing of MFCC and PLP feature streams is a common characteristic of the SRI system.

A further investigation considered the temporal context of the inputs. Varying the amount of context had a surprisingly minor effect, perhaps because the dependence of the hidden layer size and its discriminative power created an approximately equal and opposite tradeoff. A visual inspection of the input-to-hidden layer connections showed that weights had a greater magnitude for inputs that were centered on the current frame and were closer to zero for farther contexts.

Instead of varying the amount of temporal context, we could also remove the temporal derivative components of the input, since in theory this information can be learned by the network using only the static components. The removal of this redundant information did not seem to greatly affect the overall CER performance much, but it did have a considerable negative effect on the frame-level phone classification accuracy. This phenomenon is not very well understood, but underscores the importance of basing design decisions upon the CER metric – rather than simply observing phone accuracies on cross-validation or development

Feature type	CCTV	eval04
Tandem baseline	10.6	19.7
... w/ middle-state-only outputs	11.3	
... w/ monophone state outputs	10.5	
... w/ 50 triphone state outputs	11.4	
... w/ 100 triphone state outputs	11.5	
... w/ 200 triphone state outputs	11.0	
... w/ 400 triphone state outputs	10.7	
... w/ phone posterior soft targets	10.7	19.7
... w/ state posterior soft targets	10.2	19.6
... w/ monophone state targets (1150 HU)		19.5
... w/ state posterior soft targets (1150 HU)		19.3

Table 2.3: Varying the output representation of the MLP. CER is presented for the CCTV and eval04 testsets.

data and extrapolating ASR performance.

Yet another approach to decrease the size of the input is to apply a DCT along the temporal dimension of the input [30]. Due to difficulty of implementation, this was not attempted. (Such a practice is now standard in the configuration of bottleneck features, which typically reduce a 31-frame input context.)

None of these modifications seemed to have significant effect, leading us to conclude that it is probably best to keep the Tandem baseline's 9-frame context and inclusion of temporal derivatives. Modifying the input representation should most likely be viewed as a task-specific design decision in which one must consider the tradeoffs between training time and performance. If training time presents a significant cost, it does not seem unreasonable to use a smaller input representation by decreasing the temporal context and/or removing derivative components.

Modifying the output representation

We can also vary the representation of the output of the MLP, as summarized in Table 2.3.

Rather than phone targets, some researchers have experimented using HMM states. An articulatory feature system is presented in [48], where the articulatory target labels were derived only from the middle state of three-state phone HMM unit. While this allows better characterization of a phone-based target, it has the disadvantage of discarding about two-thirds of the available data; and in terms of implementation, this did not result in any significant decrease in training time.

HMM states themselves can be used as MLP targets [49], [50], in which case the non-stationary dynamic structure of phone may serve as usefully discriminative information. Using monophone states as targets has the unfortunate side-effect of expanding the output

layer by a factor of three, so the the hidden layer should be decreased to maintain a constant number of total weight parameters. Despite the reduced discriminative power of the network, the performance does not seem to be negatively affected.

A similar situation arises when using triphones or triphone states as targets, for which the output dimensionality is cubically enlarged. As a tractable approximation, we thus used aggressively clustered triphone states. It was not possible to achieve a promising result with these outputs, suggesting perhaps a better solution is needed for working with such large output layers. It is worth noting that such an approach is now standard in "deep neural networks", which most significantly demonstrated the practicality of training much larger multilayer perceptrons than was previously thought possible. Much of this benefit also comes from the possibility of leveraging large amounts of training data.

It is not clear that Viterbi-aligned one-hot unary "hard" targets are the best way to represent training labels for the data. Alternatively, state or phone posteriors from forward-backward HMM alignments can serve as "soft" targets. This allows for a representation of the uncertainty of labels near unit boundaries, although in practice most of the posterior distributions have very low entropy. Soft targets seem to be more helpful for discrimination of outputs represented as HMM states rather than phones, perhaps because there is more uncertainty in the labels.

In the results presented thus far, we have kept the total number of parameter weights constant, as we did when varying the input layer. However, the situation is subtly different from before since one cannot argue that more output units should provide better performance given an unlimited number of hidden units. Because large output layers are not inherently favored, it may not be fair to decrease the MLP's discriminative power by reducing the number of hidden units. An argument can be made that the number of hidden units should be constant, so that each output unit performs an integration over the same number of incoming connections. Assuming such a stance, where the hidden layer is fixed at 1150 units, we see that larger output layers indeed perform better.

The conclusion drawn from this set of experiments is that HMM states seem to be a most promising representation for the outputs of the MLP. Additionally, using state posteriors as soft targets seems to allow the network to be more accurately trained – although the results were not shown to be statistically significant.

Post-processing options

Table 2.4 shows the results of varying the post-processing of the output of the MLP, in order to generate suitable Tandem features for a HMM-GMM system.

Early implementations of Tandem features did not apply a logarithm to Gaussianize the softmax posterior; instead, linear output layer activations were were used for feature preparation. Some have claimed that the two approaches provide equivalent performance because the activations are simply logarithmic posteriors plus a scalar quantity; vice-versa, the logarithmic posterior is a normalization of the linear activations. However, the scaling factor might convey useful information, perhaps relating to confidence of the classifier, which is lost

Feature type	CCTV	eval04
Tandem baseline	10.6	19.7
... w/ linear output layer	10.2	19.7
... w/o MFCC concat	11.6	
... w/ LDA	10.9	
... w/ Δ before KLT	13.5	
... w/ Δ after KLT (no concat)	12.5	

Table 2.4: Evaluation of several post-processing options for Tandem features: with a linear output layer instead of a softmax normalization; without concatenated MFCC features; with a Linear Discriminant Analysis transformation and dimensionality reduction, instead of the usual Karhunen-Loeve Transform (KLT, a.k.a. PCA); with delta components appended prior to the transform, plus MFCC concatenation; and with delta components appended after the transform, without MFCC concatenation.

during the softmax normalization. Although the overall performance of the linear outputs appears to be identical to the baseline for the eval04 set, the MAPSSWE test uncovered a large number of differing segments on which the system outperformed the Tandem baseline.

Several other post-processing options were tried, without success. Omitting the final concatenation with MFCC features, the 32-dimensional MLP-derived features were insufficient by themselves. Applying a Linear Discriminant Analysis as dimensionality reduction did not help, perhaps because the resulting features were not properly orthogonalized for the GMM. Other researchers have demonstrated success in computing deltas before or after the KLT dimensionality reduction [49], [51], although the investigations in this work failed to replicate those results.

It would have been interesting to investigate the effect of retaining a different number of KLT-reduced components, instead of fixing this dimensionality to 32. However, in our experience, this usually covers at least 95% of the data's variance. Furthermore, because a GMM's likelihoods can depend on the dimensionality of the observation vectors, our systems have been tuned to features of a certain size; adjusting the Gaussian weighting factor would not be feasible for a large number of experiments.

Putting it all together

The results of the previous experiments should be interpreted as giving insight to guide a designer in combining the various modifications. It was determined that modifying the input representation is a tricky issue, so no change was made to the Tandem baseline's 9-frame context and use of derivatives. Using HMM states for the outputs seemed to be good, especially when trained using soft posteriors as targets, so this change was accepted. Finally, there is reason to believe that the post-processing should replace the softmax-logarithm step with linear activations – even though the result was not statistically significant by standard

criteria (p=0.15, two-tailed MAPSSWE test). One possible rationale for the use of linear output activations is that bottleneck features have since become more common, replacing the use of posterior-based features in some modern ASR systems. In this view, it is possible to view linear output activations as a special case of bottleneck features, one in which the bottleneck layer is at the output of the MLP. (Nonetheless, there is still a scenario in which posterior-based MLP features are very useful: when combining multiple feature streams with a relative weighting determined using criteria such as inverse entropy, or perhaps the Dempster-Shafer theory of evidence.)

The result of these combined modifications: the new system achieves 19.3% CER on eval04. At this point it should be noted that a hypothesis was made prior to running this final experiment: that these modifications would improve the Tandem baseline system (which achieved 19.7% CER). Therefore it is perhaps acceptable to use a one-tailed test in this situation, which demonstrates significance at the 0.01 level ($p = 0.006$, one-tailed MAPSSWE test, N=123).

It was also argued that it is perhaps acceptable to keep the number of hidden units constant rather than total number of weight parameters. In many ways, this is a tradeoff for practitioners who build systems; in this case we are more interested in raw performance improvement rather than a fair comparison. Alternatively, one could note that we aim to keep the dimensionality of the feature transformations constant, rather than the modeling power of the whole network. Using 1150 hidden units did not affect training time much, but improved performance to 19.0% CER on eval04.

2.4 More data and a best-case scenario

A maxim of the machine learning research community has long been that "there's no data like more data". While complicated system modifications can provide small improvements, sometimes a better use of time and resources is to simply obtain more training data and allow the learning mechanism to have more degrees of freedom to model that data. Such an attitude has been adopted in which large MLPs have been trained for weeks on enormous data sets.

To investigate the effect of using a better-trained classifier, we used a MLP that was trained for the 2007 GALE evaluation. It is similar to the MLP described for the Tandem baseline system with the following exceptions: there were 15000 hidden units, rather than 1150; the training data comprised 870 hours of broadcast news, rather than 29 hours; the target alignments were derived with a procedure for flexible alignment of quickly transcribed closed-caption annotations [52], rather than careful annotation with speaker labels. Even though the larger MLP was trained on more data, we trained the HMM-GMM acoustic models using the same 29hr training set.

Table 2.5 displays the results of this experiment using a larger MLP. The frame-level phone classification accuracy of the test set is dramatically improved, which leads to a sizable gain in ASR performance as well. This highlights the importance of having a high-

Feature type	CCTV CER	CCTV acc.
Tandem baseline (29hr MLP training)	10.6	70.4
... w/ 870hr MLP training	9.0	78.2
... w/ 870hr MLP training w/o concat	9.1	78.2
... w/ gold phone posteriors	5.9	99.2
... w/ gold phone posteriors w/o concat	4.8	99.2
Viterbi-align automatic segmentation to reference	0.9	

Table 2.5: Character error rate and frame-level phone classification accuracy on the CCTV test set. Experiments using a MLP trained on more data, and using nearly perfect phone posteriors.

Table 2.6: Comparison of Tandem features from two phone classifiers: an idealized simulation and a trained MLP. Character error rate reported on the CCTV subset of eval04.

Feature	Train	Test	CCTV CER
MFCC	–	–	11.7
Tandem	MLP	MLP	9.1
Tandem	idealized	MLP	8.6
Tandem	idealized	idealized	4.7

quality MLP classifier, and the desirability of networks which can be ported or adapted to domains which lack a suitable quantity of training data. It is interesting to note that omitting the final concatenation with MFCC features did not have such a drastic effect when the posterior-based features were of higher quality (cf. Table 2.5).

Given the results of using a better classifier, it is worthwhile to ask what would happen if we had access to a perfect phone classifier. To set up this cheating experiment, we use the phone posteriors from forward-backward alignment as if they were the output of such a classifier.

2.5 Idealized Tandem features

Our first experiment sought to determine a lower error bound on ASR performance using Tandem features. Idealized Tandem features were prepared by replacing the MLP outputs (after softmax) with forward-backward phone posterior probabilities to simulate a classifier with "perfect" accuracy.

To simulate this ideal phone classification, we defined the outputs of the hypothetical classifier to be $P_{\text{fb}}(Q_t|X,\hat{W})$: the posterior probability distribution over phones Q_t given the entire acoustic utterance X and its corresponding word transcription $\hat{W}$. This was com-

puted with forward-backward HMM inference, where the model structure was defined by composition of elementary phone models as specified by the word sequence and a pronunciation dictionary. To avoid numerical complications due to artificial zeros in the pruned forward-backward distributions, we lightly interpolated with the MLP-derived probability distribution:

$$P_{\text{ideal}}(Q_t|X,\hat{W}) \doteq P_{\text{fb}}(Q_t|X,\hat{W}) + 0.01P_{\text{mlp}}(Q_t|X_{t\pm4})$$

The MLP-derived distribution was chosen for interpolation to introduce realistic errors rather than arbitrary noise.

The simulation of $P_{ideal}(Q_t|X,\hat{W})$ for idealized Tandem features required forced alignment to the reference word transcriptions. Due to difficulty in obtaining proper alignments for all of the test data, in this section results are reported only on the relatively easy CCTV subset of eval04. The MLP classifier was able to achieve 79.7% frame-level phone accuracy on this data, scored relative to labels from aligned reference transcriptions. The simulated "perfect" classifier achieved 99.2% accuracy, a less than perfect score due to slight discrepancies between maxima of its forward-backward distributions and the Viterbi-aligned reference labels.

Table 2.6 summarizes the results of our exploratory experiment. Tandem features provided a gain in ASR performance relative to standard MFCC features. The simulation of an idealized classifier provided a very good result, albeit cheating on the test data. Interestingly, the non-cheating scenario in which idealized features were only used for training data was a bit better than the standard Tandem procedure, despite the expected negative effect due to mismatch of conditions.

In further experiments with idealized Tandem features, we note that it was possible to achieve even better results for the cheating scenario by slightly modifying the Tandem feature extraction process. We eliminated the concatenation step, removing the MFCC components from the Tandem feature vector. Then instead of a dimensionality reduction, we applied a full-rank KLT orthogonalization. Table 2.7 shows that this greatly decreased the ASR error for idealized Tandem features derived from a simulated perfect classifier; however, performance worsened when using a real MLP classifier.

2.6 Corrected Tandem features

The experiments of the previous section demonstrated the potential benefit of training acoustic models with idealized Tandem features, for which phone posteriors from an MLP (via softmax at the output layer) were replaced by forward-backward probabilities. However, in our experience we often find it best to use linear activations for the MLP outputs, so it would be desirable to apply an analogous technique in this situation. Yet it is not trivial to convert a forward-backward distribution into a vector of simulated linear activations.

We resolved this with a simple technique for correcting the linear activation outputs of an MLP. Using the Viterbi-aligned reference labels for the training data, we determined for each frame whether the MLP's classification was correct. If the MLP's maximal output

Table 2.7: Eliminating the MFCC concatenation and applying a full-rank KLT orthogonalization improved idealized Tandem features in a cheating scenario; however, the opposite effect was observed for MLP-derived features.

Train & Test	+MFCC	KLT	CCTV CER
MLP	yes	reduced	9.1
	no	reduced	9.2
	no	full-rank	9.7
idealized	yes	reduced	4.7
	no	reduced	3.4
	no	full-rank	1.8

Table 2.8: Comparison of standard and corrected Tandem features derived from an MLP's linear output activations.

Feature	Train	Test	eval04	eval06
MFCC	–	–	19.2	30.6
Tandem	MLP	MLP	15.5	24.2
Tandem	corrected	MLP	15.1	23.9

correctly related to the aligned phone, we left all the outputs unmodified for that frame. If the MLP's classification was incorrect, we changed the value at the output unit that should have had the maximal activation; we increased it to have the same value as the maximal activation over the other output units. Unlike the preparation for idealized Tandem features, this correction was a relatively minimal modification to the MLP outputs: it was applied only to frames which were incorrectly classified – about 20% of the training set – and affected just one of the MLP output units.

Table 2.8 shows the experimental results using corrected Tandem features for acoustic model training. For both the eval04 and eval06 test sets, the corrected training features provided modest improvement over the features from unmodified MLP outputs. Over the two sets, the statistical significance of the systems' difference was verified by a two-tailed MAPSSWE test [44]: $p = 0.015$ (179 vs. 135 unique errors).

Training with corrected Tandem features

The most important result in this work is the observation that an ASR system using Tandem features was significantly improved by applying a small correction to the training features.

The correction procedure is very simple to implement and relies only on having Viterbi-aligned reference labels for the training data; this information is often readily available as it is typically used to prepare the one-hot encoded targets for MLP training. By contrast,

an alternative procedure using forward-backward alignments – e.g. for preparing idealized Tandem features – might require a considerable amount of extra computation and storage space. This practicality of the approach provides an easy way to improve existing systems using MLP-derived features.

The method might be refined with a principled approach to determine the magnitude of correction. Rather than arbitrarily increasing the correct activation to equal the maximal activation, perhaps a larger increment would be better. (Otherwise, applying the softmax over these units gives a probabilistic interpretation that the correct phone is declared to have probability no greater than 0.5.) However, it is also possible that large corrections could exaggerate the mismatch between the train and test features.

Towards perfect feature extraction

Our cheating experiments with idealized Tandem features demonstrated that such an ASR front-end could reduce the error rate to as low as 1.8%. Analyzing this small amount of remaining error, we determined that in half of the cases the automatic utterance segmentation was directly responsible for deletion errors – this problem in our system has since been addressed [53]. We have therefore demonstrated that virtually perfect ASR performance can be achieved with little more than a front-end modification.

To claim that perfect features lead to perfect performance may at first seem obvious, and some researchers have commented that "if you put in the answer at the beginning, of course you'll get it back at the end". However, this is precisely the objective of Tandem feature extraction: a framework for easily exploiting a rich phonetic information stream within the constraints of a very complicated system. That the various manipulations of Tandem processing do not corrupt the idealized input is a validation of the approach.

It is also telling that the standard Tandem procedure had to be modified slightly in order to greatly reduce the error from 4.7% to 1.8% CER. In practice, the MFCC concatenation sometimes helps and the KLT reduction might remove noise. With idealized Tandem features, however, the added MFCC components were noisy and the truncated KLT dimensions were informative. Though not currently applicable, this suggests that special considerations might need to be examined when designing Tandem systems with extremely accurate classifiers.

Lastly, these experiments might suggest alternative approaches for efficient ASR decoding, considering that the MLP forward pass can be much faster than real-time. With more accurate classifiers, it may be possible to utilize less sophisticated back-end architectures for ASR; in experiments with idealized features, we observed that performance did not degrade even when the GMM models contained fewer mixtures and were trained on less data. Reviewers have suggested another interesting experiment: to decode directly from the idealized posteriors with a hybrid HMM/ANN system [10].

2.7 Discussion

This work has explored Tandem features and demonstrated that it is possible to achieve modest gains by modifying the standard configuration along certain dimensions, and significant gains by combining these modifications. However, it is shown that much better features can be achieved simply by obtaining more training data to improve the classifier. This improvement can progress up to a certain point – which we are still far from reaching, though we can simulate it to observe some intriguing implications. This investigation of MLP features has provided insights from which we can hypothesize future directions which are likely to lead us closer to our goal of better speech discrimination for improved ASR performance.

Although this work explored a limited set of input representations, it suggests that certain parts of the input may be more relevant than others. However, the fully-connected MLP initially gives equal consideration to all inputs, and may eventually be wasting resources by setting a large quantity of irrelevant weights near-zero values. It would be desirable for a network to achieve the same performance with fewer weight parameters to allow faster training. Another solution could be to perform weight pruning; however, it is more likely that efficiency could be implemented with a pre-defined sparsely-connected structure. In the style of the Tonotopic MLP [29], structured input-to-hidden layer connections could allow a system designer to allocate more hidden units to regions that need them, such as the current context frame. It might be possible to to restrict hidden units to operate on very localized input regions, such as spectro-temporal tiles. Also, it might help to have certain hidden units emulate delta computations by striping their inputs across one feature component only, similar in style to TRAP/HATs processing.

The experiments with output representations suggest that using more expressive outputs also can help, although we need a way to deal with the larger output layer induced by units such as triphones. Intuitively, it also seems odd that so much input context is used to classify an output unit with no context. One possibility is to use a variant of the multi-task MLP [35], in which multiple targets might be used to specify the output at different times.

The benefit of posterior soft targets could become more significant in situations where the distributions are of higher entropy. For example, it could be useful to use soft targets for speaker-adaptive networks in which state posteriors for a test utterance are derived from a first-pass N-best list or recognition lattice.

The post-processing stage of the Tandem approach is necessary to massage posterior distributions into a form that is more easily modeled by a GMM. An alternative might be to view the MLP as performing a sort of vector quantization into a discrete phonetic space; taking the single maximum value of the posterior distribution is akin to finding the nearest codebook value. It may be worthwhile to revisit HMM systems in which discrete observations are modeled, especially considering the potential computational speed advantages of such architectures.

Lastly, the "best-case scenario" experiment presents a novel way of looking at the ASR problem which may be of interest to researchers who wish to decouple the acoustic model

and focus on the other parts of the recognition system. We demonstrated a hypothetical system using *idealized* Tandem features to determine a bound on ASR performance within this framework, indicating that further front-end improvements have the potential to greatly benefit the overall system. This work has also described a method to improve a large vocabulary speech recognition system using *corrected* Tandem features for acoustic model training.

2.8 Conclusion

Two major themes can be extracted from the work described in this chapter.

First, there are a number of modifications to the standard process that together provide significant improvements, even though the individual modifications mostly yield statistically insignificant gains. This is not unusual, but still is worth noting. A standard experimental methodology is to derive inspiration from such small gains, even though they are not statistically significant by themselves. For example, even though using linear activations for features may not appear to offer much improvement, this researcher is at least encouraged by the fact that in isolation it does not seem to be detrimental to performance. When combined with other modifications and across larger test sets, the overall significance of the joint improvements seems to provide more support for that hypothesis – even if it is still not conclusive.

Secondly, when the MLP-based discriminative features perform well enough at phonetic classification, concatenating spectral-based MFCC features does not appear to provide much improvement for an ASR system. This may apply in situations where there is enough training data, and the MLP classifier has enough trainable parameters to learn the patterns, and if the train and test sets are reasonably well matched. However, having the spectral-based features is still useful under conditions in which the classifier performs considerably worse.

Indeed, this matches our research group's general observations during participation in the IARPA Babel Program from 2012 until 2014. Spectral features help a lot when the multilayer perceptron's accuracy on the cross-validation set is in a relatively low range (i.e. about 50% error). When the classifier is very accurate and robust (as is the ideal), the spectral features might more often than not simply introduce noise. This observation might also explain why the currently popular hybrid systems and bottleneck features (which do not directly model the spectral-based MFCC or PLP features) could be seen to outperform traditional Tandem-style features for ASR tasks that operate in a relatively low error rate regime, such as broadcast news or even conversational telephone speech for native accents (e.g Switchboard and Fisher corpora).

Chapter 3

Real-time processing

Whereas the previous chapter explored techniques for improving discriminative features for ASR in terms of recognition accuracy, we now consider the alternative performance metric of computational efficiency. Speeding up these relatively basic feature extraction computations is not only essential for real-time speech processing applications, but is also beneficial for research purposes, since faster overall processing allows more rapid iteration of experiments.

When developing a system that is targeted for practical deployment, it is helpful to consider performance tradeoffs that may not typically be encountered in a research setting. For example, it is not only important that a real-time speech processing system be able to operate in streaming mode – i.e., generating feature output faster than the rate at which audio input is received – but also that it perform with relatively low latency. Research systems that are evaluated on pre-recorded test data generally do not simulate such a condition. Our goal in this chapter is therefore to develop effective techniques for accelerating computations that enable real-time evaluation without incurring an unacceptable delay.

We begin by benchmarking a broad variety of computing platforms. The purpose is to determine the characteristics and minimum constraints that would bound real-time speech processing on portable devices. In addition to audio signal processing to extract spectral and pitch features, we consider the computations encountered when using multilayer perceptrons. This is generally dominated by linear operations (matrix-matrix multiplication), especially when considering the case of low-latency processing for which it may be infeasible to use large batches of frames. We also discuss accelerated approximation algorithms for computing sigmoid functions.

This chapter introduces a model to describe the runtime processing requirements and expected latency of MLP-based feature extraction, considering variables such as the computing platform, software alternatives, network structure, and frame batch size. We include case studies that illustrate some of the interesting and unexpected considerations, and discuss the the significant mismatch between techniques established for state-of-the-art research systems and the limitations for deployment of real-time speech processing on modern consumer devices.

MLP processing steps

The two primary tasks performed with an MLP are *forward pass* evaluation and *training* via an error-backpropagation algorithm [54]. The specific steps of these computations are enumerated below.

Forward pass, from input layer to output layer

1. Prepare the input layer from an audio signal stream
2. Propagate the (linear) activations from one layer to the next
3. Compute non-linear activations at each hidden unit
4. Repeat steps 2 and 3 for multiple hidden layers
5. Optionally, compute a *softmax* normalization at the output layer

Error back-propagation

Training is similar to the forward pass, with several additional steps of processing:

6. Compute an error signal (e.g. cross-entropy) at the output layer
7. Back-propagate the error to the hidden units
8. Update the MLP weights and biases, according to the error gradient

Although a large body of other research has investigated ways to improve and accelerate algorithms for MLP training, this chapter primarily focuses on accelerating the first three steps of the forward pass. In the context of *deep neural networks* (DNNs), we will discuss the adverse implications that multiple hidden layers (Step 4) can have on real-time performance. In addition, we will consider *hybrid speech recognition* [10] scenarios, in which the output of a MLP or DNN is directly integrated into HMM decoding, rather than used to produce acoustic features modeled by a GMM.

3.1 Computing platforms

The hardware devices chosen for benchmarking experiments in this chapter cover a wide range of capabilities and intended application settings. At the lower end, we consider mobile computing on ARM processors such as are found on the majority of tablets, smartphones, recently introduced "wearable" devices such as smartwatches, and many general appliances considered as part of the "Internet of Things". These represent a class of practical use cases in which speech technology is likely to be encountered, particularly due to the devices' small physical form factors and intended hands-free operation.

- *Raspberry Pi*: "hobbyist" (inexpensive, with many peripherals)

CPU	Broadcom BCM2835 ARM11 (1-core @ 700Mhz)
RAM	512MB (LPDDR2 SDRAM)
Disk	Micro SD T-Flash (10MB/s)
Physical	45g; 86x56x21mm w/o case; 5V DC via Micro-USB
OS	Raspbian 7 (Debian-based)
Price	$35

- *Odroid-U3*: "smartphone" (based on Samsung S3, currently the most popular Android)

CPU	Samsung Exynos4412 ARM Cortex-A9 (2-core @ 1.7Ghz)
RAM	2GB (LP-DDR2 SDRAM @ 880MB/s)
Disk	eMMC 4.41 (117MB/s sequential read; 1144 IOPS random writes)
Physical	48g; 87x52x29mm case; 5v 2A DC input
OS	Ubuntu 14.04.1 (Debian-based)
Price	$65

- *Odroid-XU3*: "tablet" (next-generation, such as the Samsung S5 Octa-core)

CPU	Exynos5422 ARM Cortex-A15/A7 (4-core @ 2.0GHz, 4-core @ 1.4GHz)
RAM	2GB (LPDDR3 RAM PoP; 933Mhz, 14.9GB/s bandwidth, 2x32bit bus)
GPU	Mali-T628 MP6 (6-core @ 600 MHz) w/ OpenCL 1.1 Full profile
Disk	eMMC 5.0 (98MB/s sequential read; 5749 IOPS random writes)
Physical	72g; 98x74x29mm case; 5V 4A DC input
OS	Ubuntu 14.04.1 (Debian-based)
Price	$179

All of the above devices are development boards, which are typically somewhat larger than the form factors encountered in consumer-grade products. For contrast, the list that follows below first includes a standard laptop computer.

As a personal and portable computing device, it shares some usage characteristics with the low-powered devices above. However, the laptop computer also straddles a middle ground in the spectrum of computing devices. Its performance characteristics make it much more similar to the high-powered Intel-based servers on the list below.

We consider professional-grade server equipment, both physically installed at a university research datacenter as well as virtualized offerings from Amazon Web Services, a leading utility computing provider. These higher-end platforms represent the conditions more traditionally associated with a research or enterprise setting. Note that the *C3-class* virtualized servers may be provisioned with variable resource quantities, and that some portion of the physical system's overall CPU and memory are reserved for the hypervisor (possibly 4 cores and 2GB, inferring from a typical server configuration).

Figure 3.1 displays some of these hardware devices. We will generally be interested in determining performance capacity of these platforms measured across variable numbers of computing cores. For the portable consumer devices, the operating system would likely need

to allocate resources to other multi-tasked applications. For the servers, the optimal use case would generally allocate one hyperthread per independent process – or if data bandwidth and memory resources were to be limiting factors, then it might be reasonable to allocate several threads per process. We will therefore attempt to evaluate the performance of each device at up to three points: using a single thread, four threads, and the maximum number of threads supported.

- *Macbook Air*: "laptop" (11-inch, mid-2012)

CPU	Intel Core i5 (2-core @ 1.7GHz)
RAM	4GB (DDR3 @ 1600MHz)
Disk	128GB SSD (consumer-grade)
Physical	1080g; 300x192x17mm case; 45W adapter
OS	OSX 10.10 (BSD-based)
Price	$899 for latest model (slightly upgraded)

- *C3-class*: "Amazon Web Services" (best price/compute performance)

CPU	2x Xeon E5-2680v2 (up to 32 virtualized hyperthreads @ 2.8GHz)
RAM	up to 30 GB
Disk	up to 640 GB SSD
Physical	unknown, proprietary installation at Amazon datacenter
OS	Ubuntu 14.04.1 (Debian-based)
Price	$0.0525 per hour per core (or 3-year reservation: $543 per core)

- *tetra.ist.berkeley.edu*: "single-node supercomputer" (custom-built workstation)

CPU	2x Xeon E5-2660v2 (40 non-virtualized hyperthreads @ 2.2GHz)
GPU	4x Tesla K40 (each: 2880 cores @ 745MHz, 288 GB/s transfer)
RAM	64GB (DDR3 @ 1333MHz; includes ECC)
Disk	60GB SSD (enterprise-grade)
Physical	4U extra-long chassis; 1620W redundant PSU
OS	Ubuntu 14.04.1 (Debian-based)
Price	$26,000 retail ($5,000 w/o the GPUs) + $60/mo datacenter colocation

Note that several of these devices include graphical processing units (GPUs). Wherever possible, software that can exploit these coprocessors will be measured for reference. In general, GPUs will be more relevant to system training scenarios, as opposed to runtime evaluation. Although general-purpose GPU computing is not yet common on low-end devices, we will explore this option further and discuss its increasing relevance.

Figure 3.1: Some of the hardware devices benchmarked in this chapter are depicted above. Numbers superimposed on image correspond to: (1) Power meter indicating that three devices draw an aggregate of 12W at 5V. (2) Odroid U3, with case. (USB WiFi and Bluetooth dongles at left) (3) Odroid X-U3. (4) Odroid W (similar to Raspberry Pi). (5, 6 & 7) Assorted peripherals such as displays, various I/O ports, auxiliary fan, and battery pack.

CPU benchmarks

Table 3.1 presents the NIST Scimark benchmark [55], comprising a composite test of five computational kernels. Results for the FFT and LU factorization kernels are displayed separately. The FFT kernel was computed on 4,096 complex numbers and exercises complex arithmetic, shuffling, non-constant memory references and trigonometric functions. The LU factorization was applied to a 100x100 matrix, exercising dense linear algebra operations. Both of those benchmarks are particularly relevant for speech processing applications, although as will be seen the FFT is usually significantly smaller and the matrix operations are typically a bit larger.

As expected, these results indicate a wide rift in performance between the three inexpensive, low-power devices with ARM-based (32-bit) processors and the higher-end devices with x86-based (64-bit) processors. The x86-based machines seem to perform particularly well on the dense linear algebra task. This may be due to exploiting streaming (SIMD) extensions of CPU instruction sets, which are not generally as available for ARM processors.

Table 3.1: Performance on the NIST Scimark benchmark [55], measuring single-threaded computation in terms of Mflops (million floating point operations per second)

Device	Composite Test	Fast Fourier Transform	Dense LU Factorization
Raspberry Pi	25	22	26
Odroid U3	157	150	195
Odroid XU3	352	325	593
Macbook Air	1,240	741	3,180
C3-class	1,410	991	2,670
tetra	1,310	843	2,850

Note also that the laptop computer performs nearly as well (or better) compared to the server machines for single-threaded tasks. This may appear somewhat surprising given the price differential and nominally lower clockrate; note, however, that consumer devices are often tuned for faster performance while server equipment may be tuned for long-running multi-tasking under continuous load. This might also be explained by operating system differences; as will be shown in the next set of benchmarks, the Apple OSX installation comes packaged with a particularly fast BLAS library.

BLAS benchmarks

Table 3.2 compares performance of several Basic Linear Algebra Subprograms (BLAS) libraries [56] compiled for the different target platforms. The computations performed were `s`gemm subroutines, $N \times N \times N$ single-precision matrix multiplication. The reported matrix dimensions N were chosen by increasing powers-of-two until the performance gain was less than 5%. This can be considered a problem size at which the hardware begins to perform optimally; larger sizes generally result in similar asymptotic performance or even degradation due to memory limitations, while smaller problem sizes tend to not fully exploit the capabilities of larger systems to distribute computations across resources.

CBLAS is considered a reference implementation; it is a set of C-language wrappers around the original BLAS code written in Fortran. Automatically Tuned Linear Algebra Software (ATLAS) [57] includes a widely used free and open-source implementation of the BLAS libraries, built in an extensive build process which involves empirical iterative optimization of code parameters. It is typically installed by default on many operating systems, although it is not generally tuned to the particular hardware (e.g. Ubuntu distributes Linux operating systems for an enormous variety of hardware, whereas Apple's OSX operating system can be specifically targeted to their own hardware). OpenBLAS [58] is a very new and increasingly popular free and open-source implementation; it is based on the manually optimized work of Kazushige Goto (GotoBLAS [59], which was traditionally difficult to install and is no longer maintained). OpenBLAS is intended to support multi-threaded processing

Table 3.2: Performance of BLAS libraries performing single-precision dense matrix multiplication (`sgemm` subroutines, $N \times N \times N$). Speed measured in Gflops (billion floating point operations per second). Optimal matrix sizes are determined as smallest of increasing powers-of-two before the successive performance gain is less than 5%.

Device	BLAS library	Threads	Gflops	Optimal N
Raspberry Pi	CBLAS	1	0.0613	32
	ATLAS	1	0.373	32
	OpenBLAS	1	0.347	32
Odroid U3	CBLAS	1	0.303	32
	ATLAS	1	2.41	256
	OpenBLAS	1	2.54	128
		2	5.02	128
Odroid XU3	CBLAS	1	0.944	128
	ATLAS	1	3.48	256
	OpenBLAS	1	3.73	128
		4	9.44	128
		4 fast + 4 slow	9.30	512
	clBLAS	6 (GPU shaders)	2.66	512
Macbook Air	CBLAS	1	5.00	128
	ATLAS	1	34.7	512
		2	61.5	512
	OpenBLAS	1	31.1	256
		2	56.4	256
C3-class	CBLAS	1	6.26	128
	ATLAS	1	19.3	256
	OpenBLAS	1	39.5	512
		4	159	2,048
		32	629	16,384
tetra	CBLAS	1	5.26	128
	ATLAS	1	14.5	256
	OpenBLAS	1	31.9	512
		4	129	2,048
		40	626	16,384
	cuBLAS	2,880 (1x GPU)	3,070	4,096
	cuBLAS-XT	5,760 (2x GPU)	5,770	16,384
	cuBLAS-XT	11,520 (4x GPU)	7,090	16,384

by default, using the OpenMP libraries.

Some x86-based CPU manufacturers have built optimized libraries for numerical computation (e.g. Intel's MKL and AMD's ACML), often exploiting specialized instructions or characteristics of their products. These have been known to provide superior BLAS performance. These are not considered in this research due to their proprietary nature. No vendor-supplied BLAS libraries are currently known to be supported for the ARM platform.

Exploiting graphical processing unit (GPU) co-processors can enable much faster computation. NVIDIA produces "general-purpose" GPGPUs such as the Tesla K40 specifically for scientific computing, and offers cuBLAS as part of the CUDA library [60]. A recent extension called cuBLAS-XT enables the use of multiple GPUs installed on the same node. CUDA is proprietary software designed solely for NVIDIA products, which are currently predominant in high-end server installations; consumer-grade mobile devices generally utilize a more heterogenous selection of GPU vendors. Thus, a recent alternative to CUDA has emerged in clBLAS, which originated in AMD's ACML, and aims to provide "a performance-portable solution for multi-platform GPU programming" [61].

Several observations can be made from the BLAS benchmarks presented in this section. First, note that the peak performance and optimal matrix sizes span several orders of magnitude between the lower-end portable devices and the high-end servers. As will be discussed in the next sections, most speech processing MLP applications operate in the middle of this range, where $512 \leq N \leq 2048$. (We will also later consider $M \times N \times N$ multiplications, where M is a relatively smaller frame batch size corresponding to real-time latency.)

In addition, note that the OpenBLAS implementation compares favorably against ATLAS. Across the consumer devices, OpenBLAS achieves peak performance at smaller matrix sizes. For server platforms it offers far superior single-threaded performance in addition to multi-threaded scaling capability. Therefore, in the subsequent experiments OpenBLAS will always be used for CPU BLAS implementation.

Performance scales almost linearly from 1 to 4 threads on all devices. However, further linear performance scaling is not fully achieved when using all CPU threads on the servers. Similarly, increasing the numbers of GPUs from 1 to 2 nearly doubles performance, but the gain for 4 GPUs is considerably less. NVIDIA has claimed somewhat better performance, up to 10 Tflops on 4 GPUS [62]; this may have been achieved by exploiting "memory pinning" to preload and register matrices on particular GPUs.

The Odroid XU3 is a particularly interesting device to consider. In addition to heterogenous CPUs operating at two different speeds, it features a general purpose GPU as well. Note that the BLAS performance is best using the 4 faster CPUs, while adding the 4 slower CPUS degrades performance slightly. This is possibly due to suboptimal scheduling, synchronization, and communication. Presumably, the OpenBLAS implementation was not designed to optimize performance on heterogenous CPU architectures, for which equal-sized task distribution might lead to some threads starving while others are still busy. The Odroid XU3's GPU offers BLAS performance that is not as good as a single CPU; however, it is possible that future generations of mobile GPUs will provide more significant computing power. A case study later in this chapter will highlight the Odroid XU3's various BLAS

computing options with respect to power consumption and other practical considerations.

Note that none of the GPU benchmarks in this section accounted for the overhead of transferring data between the host CPUs and the GPU devices. The effect of data transfer was not a major factor in these benchmark tests, due to the small size of the matrices on the Odroid XU3 and the extremely fast bus speed for the GPUs on the servers. This could be taken into consideration in the following sections which discuss BLAS computations in the context of MLP-based speech processing applications. However, there the focus will be primarily on CPU-based implementations on the consumer devices.

3.2 Audio signal processing

Several computing platforms are evaluated in this section, to determine the speed of basic audio signal processing for real-time scenarios. In a typical MLP used for speech processing, the input layer comprises high-dimensional continuous feature vectors sampled at a rate of 100 Hz. The audio signal is typically a monaural recording that is digitally sampled at 8,000 or 16,000 Hz, and analyzed on Hamming windows of approximately 25 milliseconds (or corresponding to the nearest power-of-two number of discrete samples).

These results could also apply for multi-stream signal processing – in a context that is not to be confused with alternative front-ends, such as those auditory-inspired features [7], but rather for applications in which multiple distinct audio channels are collected and processed. Examples include: two-channel telephone conversations, multi-channel radio signal acquisition (such as for intelligence or military applications), and multi-party conversations in a meeting room scenario (whether a multi-microphone array or a collection of individual close-talking microphones). In many relevant application settings, a significant proportion of the device's processing capabilities may already be allocated to other compute-intensive tasks, such as real-time video decoding.

Table 3.3 summarizes the results of benchmarking various acoustic feature extraction software tools of different computing platforms. This table report the percentage of CPU load utilized when processing real-time audio input (16kHZ waveform, 25ms window, 10ms step); such a measure is equivalent to a real-time processing factor when operating on a single CPU. All implementations are single-threaded; thus no speedup is possible on multi-processor architectures, except for the case of simultaneously processing separate audio channels.

We consider MFCC features that represent a short-term spectral envelope and are widely used for many speech processing tasks. These are computationally very similar to PLP and filterbank features, and can be computed by the popular HTK and Kaldi speech software toolkits. A custom implementation developed by Mod9 Technologies is also provided for comparison; it produces HTK-compatible features and matches HTK's output to within the range of floating-point rounding error. The performance difference is mostly due to FFT implementation and is discussed in the next subsection.

In addition, we consider pitch features computed by three different tools. A noise-robust pitch tracker implemented with Sub-band autocorrelation classification (SAcC) [63] was used

Table 3.3: MFCC and pitch extraction benchmarks. Speed reported as a percentage of CPU load when processing real-time audio input (16kHZ waveform, 25ms window, 10ms step)

Device	Feature	Software	% CPU load
Raspberry Pi	MFCC	Mod9	2.88
		HTK	6.97
		Kaldi	23.75
	pitch	aubiopitch	5.71
		Kaldi	25.04
Odroid U3	MFCC	Mod9	0.52
		HTK	1.57
		Kaldi	4.64
	pitch	aubiopitch	0.55
		Kaldi	5.65
Odroid XU3	MFCC	Mod9	0.32
		HTK	1.26
		Kaldi	5.05
	pitch	aubiopitch	0.29
		Kaldi	3.33
Macbook Air	MFCC	Mod9	0.14
		HTK	0.34
		Kaldi	1.30
	pitch	aubiopitch	0.01
		Kaldi	1.43
		SAcC	84.00
C3-class	MFCC	Mod9	0.10
		HTK	0.40
		Kaldi	0.81
	pitch	aubiopitch	0.01
		Kaldi	1.34
		SAcC	58.09
tetra	MFCC	Mod9	0.12
		HTK	0.45
		Kaldi	0.97
	pitch	aubiopitch	0.01
		Kaldi	1.50
		SAcC	68.76

Table 3.4: Percentage of CPU load when processing real-time MFCC+pitch features

	Mod9 + aubiopitch	Kaldi
Raspberry Pi	8.59%	48.79%
Odroid U3	1.07%	10.29%
Odroid XU3	0.61%	8.38%
Macbook Air	0.15%	2.73%

in our research group's ASR systems built for the IARPA Babel Program; this MATLAB software was not designed to run in streaming mode, and could not be easily ported to ARM architectures. A more widely available alternative is aubiopitch [64], provided within an open-source library of portable and highly optimized audio signal processing tools (primarily designed for real-time music applications). It implements a particularly fast FFT-based version [65] of the YIN pitch tracker [66], which we use in these experiments.

Both SAcC and aubiopitch can be used as preliminary pitch extractors that are further processed to produce pitch features for ASR. In our group's research, we have used a Python-based re-implementation of a method developed at the University of Washington [40], which was originally programmed in Matlab and Octave. The computational overhead of this post-processing can be easily optimized and is not be considered significant; it is disregarded in the present study. An alternative method of pitch feature extraction [67] was specifically designed for ASR and is provided by the Kaldi toolkit [68].

It is not clear why Kaldi is so slow, especially for the MFCC techniques which should be fairly comparable across all three implementations. One hypothesis is that the FFT implementation in Kaldi is not very well optimized. However, it is worth recognizing that Kaldi is typically run only in server settings, for which the speed of feature extraction relative to the overall processing can be considered trivial – and thus easily overlooked or disregarded for optimization purposes. Nonetheless, the speed of these computations are clearly signficant for lower-powered devices.

Real-time implications

Table 3.4 summarizes the implications of two software alternatives for the purpose of computing real-time MFCC and pitch features on portable consumer devices. This table should be referenced in subsequent calculations in our real-time processing model, as it describes the minimum CPU overhead necessary.

In terms of latency, we can assume that the audio signal processing introduces a delay of at least 12.5 ms, half the size of a standard analysis window. However, when forming the input layer to an MLP, spectral-based features are typically considered over a context window of several frames. A common context includes 4 frames before and after the center frame; frames are traditionally computed at 10ms intervals.

Moreover, as discussed in Chapter 2, it is often advantageous to include first and second-order derivatives; depending on the software tool, these may be computed on up to 9 frames of context (e.g. ICSI's speech tools, including *feacat, feacalc, and QuickNet*, all use a 9-frame FIR filter). This means that the formation of the input layer for a traditional MLP (esp. as used in Tandem-style acoustic feature extraction) introduces a real-time processing latency of at least 132.5ms. Using static features without derivatives would reduce this minimum latency to 52.5ms.

In the scheme used for *bottleneck* feature extraction in our group's work on the IARPA Babel Program, the input layer comprised a 31-frame context of filterbank features; thus the latency for such a network is 162.5ms.

An interesting alternative can be considered in which the network's input layer consists only of a left context, thereby reducing the latency to just 12.5ms for the latest frame's analysis window. Such an approach is reportedly used in some commercial ASR systems.

Use of the Fast Fourier Transform

The majority of computation during input feature preparation involves the Fast Fourier Transform, which is generally well-optimized. The post-processing steps used to compute MFCC [69] or PLP [27] acoustic features are relatively fast – indeed, these digital signal processing techniques were developed for computers of the 1980's... today, they can still be implemented on even the lowest-end processors to achieve performance that is hundreds of times faster than real-time.

Table 3.5 compares the performance of various implementations of this algorithm. These benchmarks were unfortunately produced on a compute server that is no longer accessible, nor are its technical specifications able to be recalled.

The "custom iterative" version computes a decimation-in-time radix-2 complex FFT, in which the result is computed in-place (replacing the inputs) using precomputed twiddles; this is a fairly "textbook" FFT implementation in which half the code performs the butterfly computation, and the other half descrambles the bit-reversed indices.

For contrast, the "custom recursive" version computes a slightly slower implementation that operates on real-valued inputs only; it resembles a high-level (nearly pseudo-code) abstraction of the FFT divide-and-conquer algorithm. This reinforces the fact that the FFT algorithm enables a vital speedup over the extremely slow "naive DFT", which is implemented to directly mirror the definition of the Discrete Fourier Transform as follows:

```
for (int k = 0; k < N/2 + 1; k++) {
  for (int n = 0; n < N; n++) {
    X_real[k] += x[n] * cos((-2 * M_PI * k * n) / N);
    X_imag[k] += x[n] * sin((-2 * M_PI * k * n) / N);
  }
}
```

Table 3.5: Comparison of Fast Fourier Transform implementations. Benchmarked for processing a 60s audio input, sampled at 16kHZ, windowed at 25ms (512-point FFT)

Implementation	Lines of Code (C)	Runtime (seconds)
FFTW [71]	100,000	0.11
KISS FFT [72]	500	0.11
HTK [73]	120	0.20
Custom iterative	30	0.22
Custom recursive	15	0.24
Naive DFT	10	50.0

Table 3.6: Comparison of MFCC feature extraction tools with various FFT implementations. Benchmarked for processing a 60s audio input.

FFT library	MFCC tool	Runtime (seconds)
FFTW	Mod9	0.116
custom iterative	Mod9	0.233
HTK	HTK	0.380

As an additional optimization, a power spectrum from the real-valued DFT can be computed from a half-size complex FFT in which the original real-valued inputs are reconsidered as complex values: even indices are real, while odd indices become imaginary. This exploits the Hermitian redundancy and applies the Danielson-Lanczos Lemma via techniques that are described in [70]; it can be implemented in an additional dozen lines of code using the precached twiddles from the above-mentioned "iterative" FFT implementation. Consequently, the entire MFCC feature extraction can be accelerated such that a custom implementation (*Mod9*) using a slower FFT algorithm becomes significantly faster than an existing (*HTK*) implementation that uses a faster FFT algorithm, as indicated in Table 3.6.

Note that the results provided in Tables 3.3 and 3.4 implement the Mod9 MFCC feature extraction using the FFTW library, which is generally easy to obtain on all modern operating systems. For portability to specialized platforms, such as embedded devices, it may however be practically advantageous to use an FFT library that is easier to build (such as KISS) or to simply include 15-30 lines of code for a self-contained function.

The main inferences that are intended to be seen from Tables 3.5 and 3.6 are that (1) implementation of the FFT algorithm need not be overly complex in order to achieve reasonable performance, and (2) it is still possible to improve upon a standard tool's MFCC calculation, an effort which may prove helpful in the case of enabling real-time processing for low-resource computational devices.

Use of pitch in ASR systems

Many modern automatic speech recognition sytems now include pitch features as part of the feature extraction processing. There is linguistic evidence of intrinsic F0 for English vowels, even though the language is non-tonal [74]. In our research group's work for the IARPA Babel Program, we found that including pitch-based features improved ASR results for all languages, tonal as well as non-tonal. Even if a target language does not have tonal phoneme distinctions, pitch – and its related measure of probability of voicing – can be utilized for various purposes in speech recognition.

In addition to acoustic modeling, pitch and PoV determinations can be useful as a proxy for speech/nonspeech discrimination as a pre-processing step in acoustic feature extraction. For segmentation of data, voiced frames can be a reasonable proxy for speech frames. In particular, with a slight collar (e.g. 200ms) padded around voiced frames, performance of a pitch-based speech/nonspeech system is comparable to using a more sophisticated MLP trained on aligned examples of speech and nonspeech frames. This shortcut can drastically reduce the system building time by removing the need to build an initial set of HMM alignment models, saving many hours of acoustic model training for faster experimentation.

Moreover, speech / nonspeech determinations are useful for feature normalization. Most modern automatic speech recognition systems perform some form of cepstral mean subtraction, and possibly also variance normalization (dividing by the standard deviation to result in unity variance), on the base acoustic features. Such normalizations are intended to compensate for channel convolution effects, and are typically applied to all frames in a single recording or belonging to a particular speaker. However, the normalization basis is best estimated over speech frames, and so the pitch-based proxy is helpful in this respect.

As with cepstral mean and variance normalization, VTLN warping factors are also best estimated on the basis of speech frames. Again, the pitch-based proxy is helpful. Moreover, it is even better to compute the VTLN reference acoustic model over *voiced* frames, as opposed to speech frames that may include non-vowel sounds. This could perhaps be because the rationale for VTLN is to shift the resonant frequencies (formants) corresponding to speaker-specific differences in vocal tract length; however, such a rationale does not apply for non-vowel phonemes such as fricatives, for which the noise frequency range should not vary according to speaker-specific differences in vocal tract length.

In addition, pitch extraction can be useful for direct pitch-based VTLN warp factor estimation [75]. The previously reported VTLN warp factor estimation methods were based on a maximum-likelihood approach using a generic speech Gaussian Mixture Model [76]. However, using the pitch-based VTLN warp factor estimation described in [75], it is possible to very succinctly define warp factors for this data set as:

$$w_{\mathrm{F}_0} = \mathrm{mean}(log(\mathrm{F}_0)) * a + b$$

When using pitch extraction as a proxy for speech / non-speech segmentation, or even as a substitute for VTLN warp factor estimation, the benefits of the simplified one-pass computation can be dramatic. The overall system runtimes needed to prepare input features

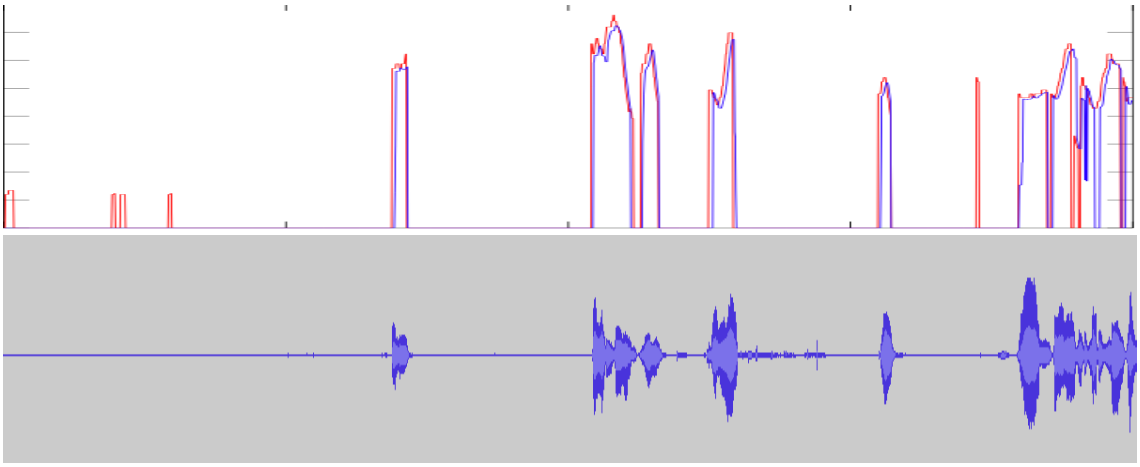


Figure 3.2: Sample of pitch extraction with SAcC and aubiopitch. The output of SAcC is red and shifted by 9 frames relative to the aubiopitch output in blue. The corresponding waveform is depicted below.

to an MLP using various pitch-based shortcuts can be reduced to a matter of hours instead of days. Corresponding performance tradeoffs should of course be considered, but in our research group's experience it was generally worthwhile to use pitch extraction.

A final consideration with respect to using pitch features or pitch-based shortcuts in the base acoustic feature preparation phase is that the pitch extraction procedure should not be too computationally expensive itself. In our research group, a very high quality pitch extractor (SAcC) [63]) was compared to the simpler Yin algorithm [66] (which is used as one of the several ground truth candidates in SAcC) as implemented by the free and open source aubiopitch tool. The resulting ASR accuracy results were nearly identical and the differences were far from being statistically significant.

The differences between the two pitch extractors are relatively minor, as depicted in Figure 3.2. Looking at this specific sample shows that the SAcC system was particularly vulnerable to very short-duration voicing detection "false alarms". Although such errors have a minor effect on normalization bases (since they do not significantly change the averages), they can have a very adverse effect on speech / non-speech segmentation.

Using the Kaldi-based pitch features was significantly better than using pitch features derived by either the SAcC or aubiopitch. Unfortunately, this tool is not a direct pitch-tracker, *per se*, and so it does not reflect many of the voicing determinations that are important for other purposes in ASR system building. In the context of this chapter's emphasis on real-time feature extraction, a reasonable recommendation would therefore be to pursue pitch extraction with aubiopitch instead of SAcC, for the purpose of proxying speech/nonspeech determinations, normalization bases, and VTLN warp factors. Pitch features could also be used to improve ASR accuracy, and could be derived either from aubiopitch F0 values, or directly via the Kaldi tool.

3.3 Matrix multiplication

In most scenarios, MLP computations are largely dominated by the linear algebra operations such as vector dot products or matrix-matrix multiplications. This type of computation is typically accelerated by batching data across a sequence of inputs for parallel processing. We are interested in exploring the implications of such a strategy on low-latency real-time systems, as well as system training.

After the first step of preparing base acoustic features to be used as the input layer to the MLP, the next major processing step that can be accelerated is the propagation of activations from one MLP layer to the next.

In a typical fully-connected network architecture, each hidden unit (and also units of the output layer) can be considered to have a (pre-sigmoidal, linear) activation that is determined by a dot product: the sum of the previous layer's unit activations (or input base acoustic feature values, for the first hidden layer) multiplied by the corresponding network weights that connect the previous layer to the hidden unit.

Instead of considering the hidden unit activations as isolated dot products, it is also possible to consider an entire hidden layer's activations at once. In this case, the multiple dot products can be represented by a single vector-matrix multiplication. That is, the I-dimensional input layer (or previous hidden layer) vector corresponding to each frame of input is multiplied by the $I \times H$-dimensional matrix corresponding to the weights of connections to the next hidden layer. For simplicity, we can assume that the networks biases are included in the weights, and that the input layer (or previous hidden layer) thus includes one extra "bias" unit that is fixed to always have the value of 1.0.

Dot products and matrix-vector multiplications are basic linear algebra operations that are well optimized by the BLAS libraries, as benchmarked earlier in this chapter. Dot products are considered BLAS Level 1, while matrix-vector subroutines are considered BLAS Level 2. To achieve the greatest gains from BLAS, we seek to use the Level 3 optimizations for matrix-matrix operations. This can be done by *batching* input layer observations (or hidden layer activations) across multiple frames.

Figure 3.3 illustrates these three approaches for accelerating the propagation of activations between MLP layers using different BLAS (Level 1, 2 & 3) subroutines.

Effect of frame batching on real-time processing load

An important consideration for real-time processing of streaming audio is whether the computing device is able to continuously process incoming samples without creating a backlog of unprocessed data. In this section we consider the effect of frame batch size on matrix multiplication, particularly with respect to real-time operation.

In Tables 3.7 and 3.8, we denote by M the frame batch size, assuming a frame rate of 100Hz – which is nearly universal and hard-coded into some ASR software. The value of M corresponds to the delay for the first frame in a batch to be buffered before the matrix multiplication can begin; this is the worst-case latency relative to that first frame. We

Table 3.7: Real-time processing delays due to frame batch size (M) effect on matrix multiplication for hidden layers of size N and connected layers of size O. For each device, three use cases are delineated: combined input-to-hidden and hidden-to-output multiplications for an MLP with one hidden layer ($O = 512$), hidden-to-hidden for a DNN ($N = O$), and hybrid decoding with a large output layer ($O = 8192$)

Device	M	N	O	Real-time?	CPU Load	Delay (ms)
Raspberry Pi	1	1,024	512	No		
	4	1,024	512	Yes	62%	65
	16	1,024	512	Yes	37%	219
	16	2,048	512	Yes	75%	280
	64	1,024	512	Yes	35%	864
	64	2,048	512	Yes	70%	1088
	64	4,096	512	No		
	1	512	512	No		
	4	512	512	Yes	31%	52
	16	512	512	Yes	18%	189
	4	1,204	1,024	No		
	16	1,024	1,024	Yes	73%	277
	64	512	8,192	No		
Odroid U3	4	16,384	512	No		
	16	16,384	512	Yes	87%	299
	1	1,024	1,024	Yes	60%	16
	4	1,024	1,024	Yes	18%	47
	4	2,048	2,048	Yes	71%	68
	16	2,048	2,048	Yes	41%	226
	4	1,024	8,192	No		
	16	1,024	8,192	Yes	83%	293
	64	2,048	8,192	No		

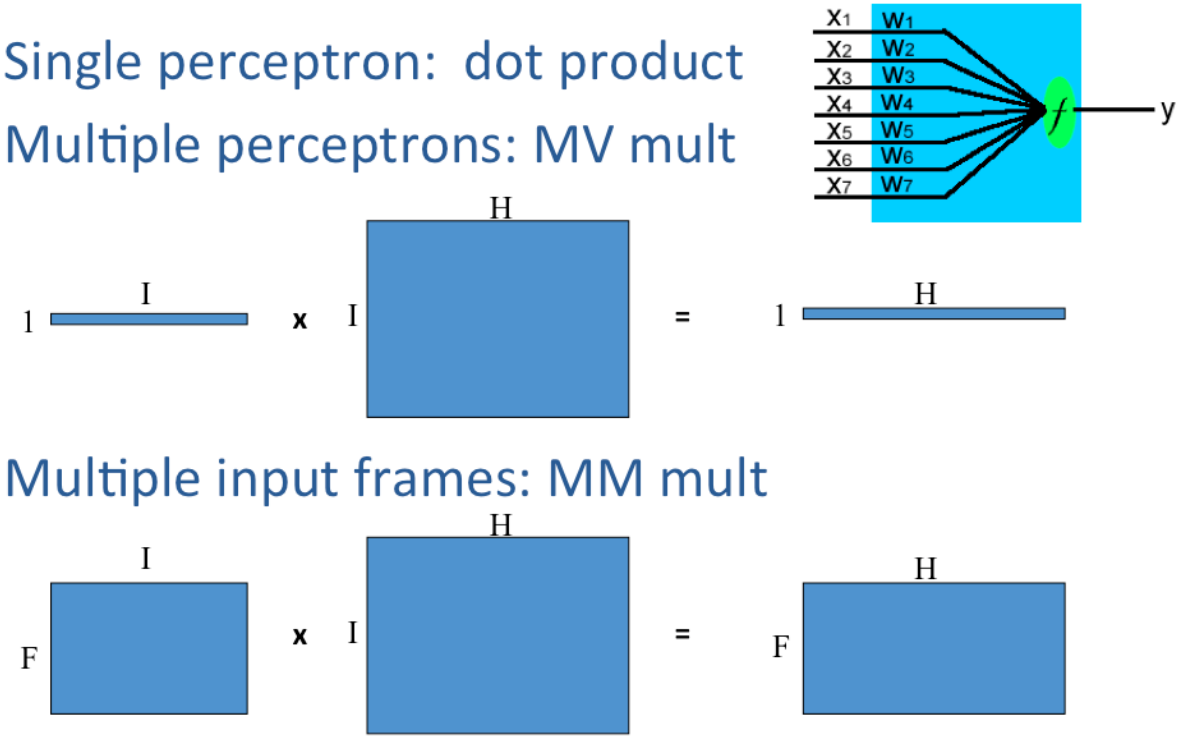


Figure 3.3: Three approaches to using BLAS (Levels 1, 2 & 3) for accelerating propagation of activations between MLP layers.

will consider batch sizes of $M \in \{1, 4, 16, 64\}$, which correspond to delays of 10-640ms in duration.

The size of a network hidden layer is given by N, while O represents the third dimensionality of the $M \times N \times O$ matrix multiplication. We consider values that relate three distinct use cases:

- For a traditional three-layer MLP with only one hidden layer, there are two matrix multiplications involved: input-to-hidden, and hidden-to-output. If we approximate $O = 512$ as the total size of the input and output layers together, we can consider a $M \times N \times 512$ multiplication to represent the cost of evaluating this simple MLP. The value of 512 is reasonable for Tandem-style or bottleneck feature extraction, since input layers may range from 465 to 240 units (15 critical bands over 31 frames, or reduced to 16 DCT components for each band) and output layers are typically determined by a given language's phonetic inventory (usually under 100) or specified as a fixed bottleneck dimensionality (often about 30).

- For DNNs with multiple hidden layers connected to one another, one or more $M \times N \times N$ multiplications are involved. The values considered here are $N \in \{512, 1024, 2048\}$,

Table 3.8: Real-time processing delays for the Odroid XU3, with heterogenous CPU and GPU processing.

BLAS	M	N	O	Real-time?	CPU	GPU	Delay (ms)
OpenBLAS	1	4,096	512	Yes	87%		19
	1	8,192	512	No			
	4	8,192	512	Yes	64%		66
	4	16,384	512	No			
	16	16,384	512	Yes	66%		265
	1	1,024	1,024	Yes	28%		13
	4	2,048	2,048	Yes	43%		57
	1	1,024	8,192	No	53%		
	4	1,024	8,192	Yes	81%		72
	16	1,024	8,192	Yes	53%		245
	16	2,048	8,192	No			
	64	2,048	8,192	Yes	87%		1197
clBLAS	64	16,384	512	Yes	14%	88%	1203
	4	1,024	1,024	Yes	2%	19%	48
	4	2,048	2,048	Yes	4%	69%	68
	16	1,024	8,192	No			
	64	1,024	8,192	Yes	2%	10%	704

representing hidden layers that can be considered approximately "small", "medium", and "large" in the context of modern DNNs used for ASR.

- For DNNs used in modern hybrid speech recognition decoder, the final layer may be considerably larger than the input or hidden layers. It typically corresponds to some form of tied triphone HMM state targets, and the value of $O = 8192$ is chosen as a representative size of such an output layer as used in large-vocabulary ASR systems.

In Tables 3.7 and 3.8, each of the three above use cases is delineated for the devices reported. In addition to the frame buffering, an additional delay is required while waiting for multiplication to be completed on the CPU or GPU. If this takes longer than the frame buffering delay, then real-time processing is not possible; this is noted with a "Yes" or "No" in the tables. Using the processor load percentages, as reported in the tables, the resulting total delay may be computed, as shown in the rightmost columns.

All of these experiments use a single-threaded OpenBLAS implementation for the CPU, or the clBLAS implementation for the GPU on the Odroid XU3. We report the expected delay based on an optimistic assumption that full processor utilization will be scheduled as soon as the frames have been fully buffered. In the worst case, if the processor scheduling

must allocate resources to other tasks, the total delay must be at most $2 \times M \times 10ms$ in order to support real-time processing.

Several observations can be made from these tables. The underpowered Raspberry Pi is capable of handling a modest-sized hidden layer for traditional MLPs used Tandem or bottleneck processing, albeit with rather large delays. For example, it may be reasonable to process a single-layer MLP with 1,024 hidden units using a frame batch size of 16; this would result in a delay of at least 219ms and utilize 37% CPU load. Since the Raspberry Pi has only a single CPU, keeping the overall CPU utilization below 50% seems advisable to achieve real-time MLP feature extraction, especially since additional resources will still be needed to extract the spectral features and compute sigmoids – as well as some overhead for the operating system.

Alternatively, using two hidden layers of size 512 in a DNN would utilize a similar amount of CPU ($36\% = 18\% \times 2$) but entails a latency of 378ms (189×2). These two network configurations are identical in terms of the number of total connection weights, with about 500,000 parameters each ($512 \times 512 \times 2 = 1024 \times 512$). The difference between these two networks is that one is "wide" while the other is "deep". While some research has investigated the performance differences between these two structures in terms of phone classification and ASR accuracy, it is also worth noting the performance difference in terms of real-time processing delays.

Note that it would be difficult to support hybrid decoding on the Raspberry Pi; even with a large batch size and small hidden layer, the $64 \times 512 \times 8,192$ multiplication cannot be achieved in real-time. By contrast, the Odroid U3 can support this with a batch size of 16 and hidden layer size of 1,024. Although this would occupy 83% of the CPU, this device features two processors. Thus it would be possible to use several additional hidden layers in a DNN configuration; using a frame batch size of 4, each DNN hidden layer incurs an additional 18% CPU utilization and adds 47ms of delay.

Most real-time processing scenarios can be comfortably achieved on the Odroid-XU3, representing a next-generation mobile device. Even a rather large DNN with hidden layers of size 2,048 would only add 43% CPU utilization and 57ms delay for each hidden layer. Hybrid decoding with such a large hidden layer is possible, but the final layer's $64 \times 2,048 \times 2,048$ computation incurs a rather significant delay of over one second. Thus, for this device it may be advisable to use smaller hidden layers of size 1,024 for hybrid decoding, or use large hidden layers of size 2,048 for posterior-based or bottleneck feature extraction in which the output layer is much smaller.

3.4 A model for real-time feature extraction

This chapter's discussions of real-time processing implications can be summarized with a formalized model to describe the effect of various design choices on the operational characteristics of MLP-based feature extraction on various devices.

Let $\text{Load}_{\text{DSP}}(\text{Device}, \text{Tool}_{\text{MFCC}}, \text{Tool}_{\text{pitch}})$ denote the CPU load for performing the digital signal processing required to produce the spectral-based features that form the input layer of a network. These values can be looked up in Table 3.3. The associated processing delays can be denoted by $\text{Delay}_{\text{DSP}}(\text{Device}, \text{Window}, \text{Deltas}, \text{Context}, \text{Centering})$, and the derivation of these values is discussed in Section 3.2.

Let $\text{Load}_{\text{BLAS}}(\text{Device}, M, N, O)$ denote the CPU load and $\text{Delay}_{\text{BLAS}}(\text{Device}, M, N, O)$ denote the associated processing delays for matrix multiplication. These values can be looked up in Tables 3.7 and 3.8. Recall that $\text{Load}_{\text{BLAS}}(\text{Device}, M, N, 512)$ is assumed to represent the combined input-to-hidden and hidden-to-output layers for a typical MLP used for Tandem or bottleneck features.

Finally, let $\text{Load}_{\text{sigmoid}}(\text{Device}, \text{Tool}_{\text{EXP}})$ and $\text{Delay}_{\text{sigmoid}}(\text{Device}, \text{Tool}_{\text{EXP}})$ denote the CPU loads and delays associated with computing a sigmoid function. In the context of this work, these values can be considered insignificant factors on the overall performance. Sigmoids can be computed for each layer of $N = 1024$ hidden units in just 6.8% real-time CPU load on the Raspberry Pi. On devices with multiple processors, it is trivial to parallelize the independent computations with nearly perfect performance scaling, so the batched processing latency ($M \times N \times 6.8\%$) can be divided by the number of CPU threads; the result for typical values of M and N will be less than 20ms. The reason these values are considered relatively insignificant is because it is not feasible to use very large values of M and N on the Raspberry Pi, and on other computing devices the speed of exponentiation is much faster. Also, in many modern DNN implementations, the nonlinearity activation is not performed with a sigmoid, but rather a faster-to-compute piecewise linear function. The next chapter will describe a scenario in which accelerating such computations provides a more salient performance gain in the context of system training.

If we let H denote the number of hidden layers in the network, then the overall CPU load can be summed as follows, for the case of an MLP producing bottleneck features or Tandem-style features with linear activations at the output layer:

$$\begin{aligned}
&\text{Load}_{\text{DSP}}(\text{Device}, \text{Tool}_{\text{MFCC}}, \text{Tool}_{\text{pitch}}) && (3.1)\\
&+ \text{Load}_{\text{BLAS}}(\text{Device}, M, N, 512) && (3.2)\\
&+ \text{Load}_{\text{BLAS}}(\text{Device}, M, N, N) \times (H - 1) && (3.3)\\
&+ \text{Load}_{\text{sigmoid}}(\text{Device}, \text{Tool}_{\text{EXP}}) \times H \times M \times N && (3.4)
\end{aligned}$$

For a Tandem-style MLP in which a softmax is computed at the output layer, an additional term ($\text{Load}_{\text{sigmoid}}(\text{Device}, \text{Tool}_{\text{EXP}}) \times M \times O$) must be added. For a DNN that is used for hybrid speech recognition decoding (where generally speaking, $O \approx 8,192$), the following terms must be added:

$$\begin{aligned}
&+ \text{Load}_{\text{BLAS}}(\text{Device}, M, N, 8192) && (3.5)\\
&+ \text{Load}_{\text{sigmoid}}(\text{Device}, \text{Tool}_{\text{EXP}}) \times M \times 8192 && (3.6)
\end{aligned}$$

The overall latency of these feature extraction pipelines may be summed as follows, for a MLP used in bottleneck or Tandem-style feature extraction:

$$\text{Delay}_{\text{DSP}}(\text{Device}, \text{Window}, \text{Deltas}, \text{Context}, \text{Centering}) \tag{3.7}$$

$$+ \text{Delay}_{\text{BLAS}}(\text{Device}, M, N, 512) \times 2 \; + \; \text{Delay}_{\text{BLAS}}(\text{Device}, M, N, N) \times (H - 1) \tag{3.8}$$

$$+ \text{Delay}_{\text{sigmoid}}(\text{Device}, \text{Tool}_{\text{EXP}}) \times H \times M \times N / \text{NumThreads} \tag{3.9}$$

3.5 Conclusion

This chapter described real-time processing of discriminative acoustic features, as well as DNNs that can be more generally applied for hybrid speech recognition. Performance was benchmarked on a variety of computing platforms, ranging from consumer-grade mobile devices to professional-grade servers.

As the minimum overhead required to support real-time operation, it must be possible to compute the basic signal processing for spectral-based acoustic features (e.g. MFCC) that form the input to a neural network. Surprisingly, this MFCC computation was not trivial on the most low-powered devices. Moreover, it is often helpful to include pitch-based features, which effectively doubles such computational overhead. Alternatives to standard research software (e.g Kaldi and SAcC) may be required for some devices.

After the basic audio signal processing, the computation is then dominated by the matrix multiplication corresponding to propagation of activations between successive layers of the MLP. Such dense linear algebra operations are easily optimized with BLAS subroutines. These implementations generally achieve optimal performance with rather large matrix sizes; unfortunately, batching input frames translates to an imposition of a delay on the real-time processing pipeline. Moreover, in multilayer DNNs, the effect of such latency is multiplied. Such factors should perhaps be considered when debating whether to choose a neural network architecture that is "deep"; in some cases, it may be more practical to use an architecture that is "shallow and wide", since it results in less latency during a forward pass.

Finally, a model was presented that should allow a system designer to predict the real-time CPU load and operational latency of discriminative acoustic feature extraction or hybrid speech recognition. Because many state-of-the-art techniques are first developed in research settings in which real-time processing is never encountered or considered, such a model may prove useful as a "sanity check". In certain cases, it may not be worthwhile to develop overly large and complex systems if they will never be capable of performing under realistic conditions such as mobile devices that must support real-time stream processing.

Chapter 4

System training and deployment

We now consider some of the performance improvements discussed in the previous chapter about real-time feature extraction, but in the context of system training. Here, fast processing is still desirable – but only to the extent that it does not significantly degrade performance in terms of ASR system accuracy – and latency is not a consideration since training is almost always performed offline.

When building ASR systems, the training data are generally pre-recorded (instead of arriving as live streams) and are passed over many times by iterated algorithms, often variants of expectation-maximization for training HMM-GMM models or error back-propagation for neural networks. The spectral base acoustic features used as observations or inputs are generally only computed once and cached to disk storage in order to optimize computation and network communication. The cost of such MFCC and pitch feature extraction is insignificant relative to overall system training, and often amortized over many successive experiments.

However, there are significant MLP training considerations that involve frame batch sizes. Especially on large servers with dozens of CPU threads or GPU co-processors, fast processing can be achieved using very large batch sizes. However, this is not necessarily better, if the larger batches result in fewer network parameter updates; such a scenario typically requires more training epochs to reach convergence. The performance tradeoff must therefore account for overall MLP training time.

HMM acoustic model training can also be accelerated using fast transcendental function approximations. In this case, it is sufficient to show that the approximation error does not adversely affect the accuracy achieved with the trained models. By calibrating the parameters of the approximation function, it is possible to control the error characteristics and bound it to a reasonable range of numerical precision for this task.

Discriminative feature extraction and hybrid speech recognition systems have demonstrated that significant improvements can be gained with the use of multilayer perceptron or deep neural networks. However, these systems typically rely on supervised training; more specifically, they require that the training data be associated with frame-level classification targets against which error gradients may be derived for the purpose of backpropagation training algorithms. Whether these targets are considered “hard” one-hot encodings or

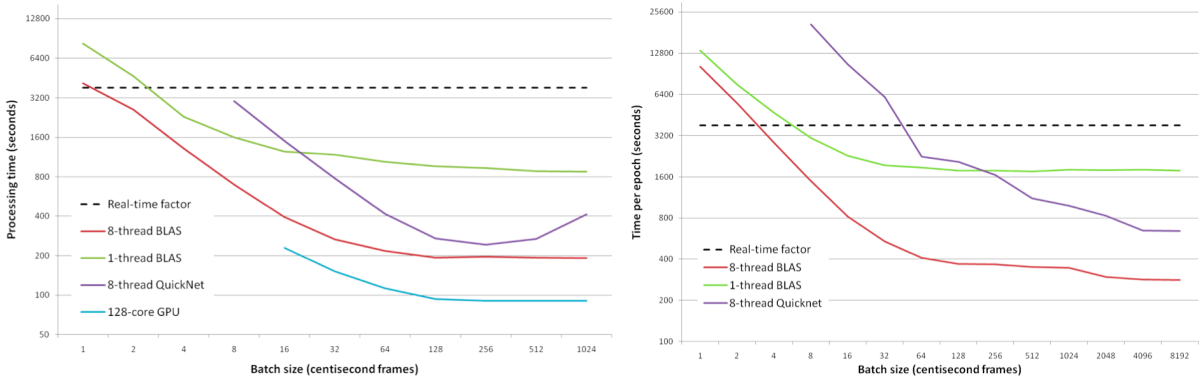


Figure 4.1: Comparison of MLP forward pass (left) and MLP training (right) using various BLAS implementations and batch sizes. The input data used for this benchmark comprised one hour of audio and was processed by a MLP comprising: 351 input units, 15,000 units in a single hidden layer, and 135 output units.

"soft" posterior distributions, or whether they represent monophones or states of clustered triphones – the common factor is that training such classifiers will always require an *alignment* of acoustic observation frames to labeled targets.

This chapter thus concludes by showing the importance of speech transcript alignment for the purpose of MLP training. In particular, the alignment problem becomes especially critical in situations where the training data is drawn from "real-world" sources that may not be as easily aligned as traditional speech corpora used in academic research settings. A case study is presented in which a robust alignment service has been deployed for commercial use, addressing several complications that are not usually seen in research settings.

4.1 Effect of batch size on MLP training

Real-time latency is no longer a relevant factor in a training context. Thus, it would be logical to increase frame batch sizes used in the matrix multiplications used in the back-propagation training of MLPs and DNNs. As noted in Chapter 3, the matrix sizes at which the BLAS functions perform optimally (Table 3.2) are generally far larger than the frame sizes recommended for low-latency feature extraction (Table 3.7). Moreover, MLP training is typically performed on professional server hardware rather than consumer devices. Such platforms generally provide dozens of CPUs and are often augmented with GPU co-processors. In order to exploit such parallelism most effectively, large frame batch sizes are often required so that the matrices can be distributed across many processing units.

Figure 4.1 demonstrates the effect of frame batch size on the processing speed for both the MLP forward pass, as well as a MLP training – which includes the forward pass, plus computation of an cross-entropy error signal and back-propagation of this error (mainly via additional matrix multiplications). These experiments were performed with a MLP having

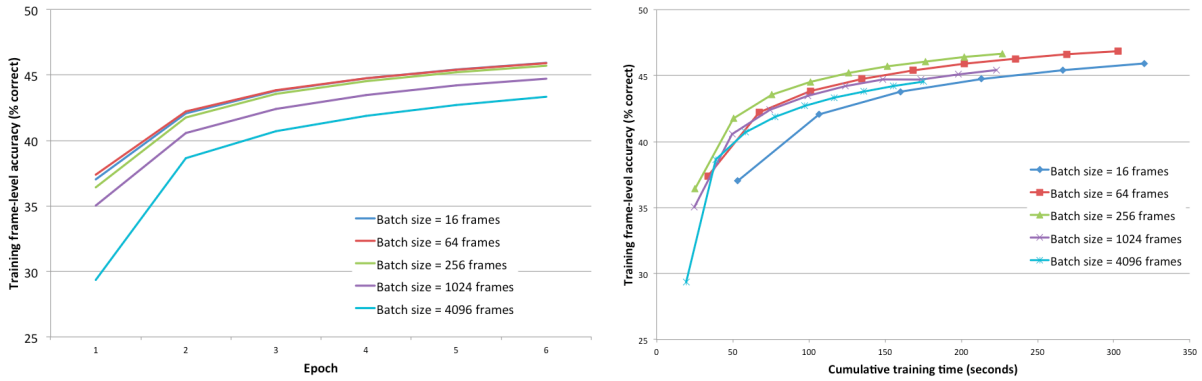


Figure 4.2: Analysis of the tradeoff between processing speed and algorithmic convergence, for a MLP trained on 29 hours of Mandarin broadcast news. Smaller batch sizes lead to better frame-level cross-validation accuracies at each epoch; however, larger batch sizes compute each epoch more rapidly. For this particular scenario, a batch size of 256 led to the fastest convergence overall, as noted at right.

a single hidden layer of 15,000 units, using hardware that is approximately ten years out of date: AMD Opteron-875 processors with dual-core 2.2GHz processors installed in four sockets (up to 8 threads), and an NVIDIA GPU with 128 cores. These figures also compare the performance between a custom MLP implementation using multi-threaded BLAS and the popular *ICSI QuickNet* software. This older version of QuickNet used single-threaded BLAS implementations that were distributed by independently splitting across the batched input frames; this is less efficient than simply considering the inputs as a single matrix and distributing the entire matrix multiplication.

However, large frame batches in training equate to fewer network updates in the error backpropagation algorithm, which leads to slower overall convergence of the training process. This is shown on the left side of Figure 4.2, which shows the cross-validation accuracies at each successive training epoch. On the other hand, using larger frame batches makes each pass through the data faster, and thus enables more training epochs to complete within a given amount of time. This tradeoff can be tuned for particular applications and computing platforms: for this particular setup (Tandem features for Mandarin ASR, trained on 29 hours of Mandarin broadcast news), the fastest overall training time was achieved using 256 frames per batch, as seen in the right side of Figure 4.2. Note that at any given time, the MLP trained with a batch size of 256 frames has a higher cross-validation accuracy than any of the other networks trained with different batch sizes.

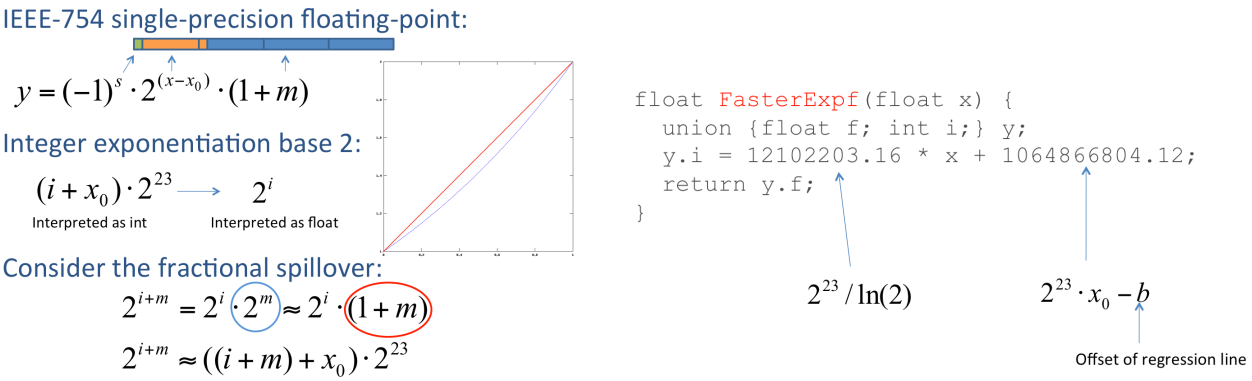


Figure 4.3: The exponential function can be approximated by exploiting IEEE-754 floating-point representations. Left: the intuition is depicted graphically, where the mantissa bits provide a linear approximation ($1 + m$, in red) to the true function (2^m, in blue). Right: the approximation function's implementation in C, with constants that provide a single-precision variant of Schraudolph's original EXP macro calibrated for optimal RMS error. (Note that the left figure illustrates 2^x instead of e^x, with an unadjusted linear approximation where $b = 0$, which is exact for integer arguments and an upper bound elsewhere.)

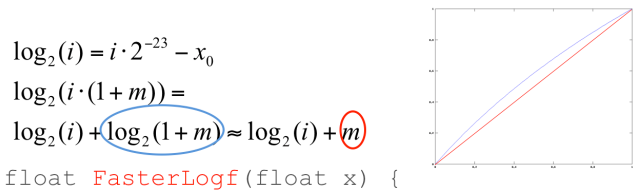
4.2 Fast transcendental functions

Appendix A describes a set of novel algorithms for computing fast and reasonably accurate approximations of the logarithm and exponential functions. The basic "trick" is to exploit *type-punning* with IEEE-754 floating point representations. The method was inspired by a report first published by Nicol Schraudolph [77], in which a C-language macro was provided for the double-precision EXP function. Improvements described in Appendix A were added to make this a thread-safe inline function, and the idea was also extended to the logarithm and single-precision variants; in addition, function input arguments are initially checked to ensure that they are in the functions' properly bounded domains, such that the output range does not produce improper results or *NaN* values.

Figure 4.3 presents the intuition behind Schraudolph's discovery, illustrated for single-precision, as well as the corresponding code to implement the approximation in C.

Independently exploiting similar type-punning, researchers at ICSI developed *ICSILog*, an adjustable table-lookup approximation for the logarithm function [78]. These researchers referenced the informally published *FastLog* function written by Laurent DeSoras [1]. DeSoras' function included an interesting insight: unlike Schraudolph, he used a quadratic function to adjust the mantissa's linear interpolation between integer values of the approximation at $f(1)$ and $f(2)$, greatly reducing the error of the approximation. DeSoras notes that he chose the values for his quadratic approximation, $f(x) = -\frac{1}{3}x^2 + 2x - \frac{2}{3}$ because they preserve

[1] DeSoras originally posted his work on a website forum that seems to be no longer accessible, but it is now easily found at other code-sharing sites by searching for "fastlog"




```
float FasterLogf(float x) {
  typedef union {float f; int i;} pun_t;
  pun_t* a = (pun_t*) &x;
  pun_t b, m;
  b.f = (a->i >> 23) - 128;
  m.i = (a->i & 0x007fffff) | 0x3f800000;
  return (b.f + m.f) * M_LN2;
}
```

Figure 4.4: The logarithm function can be approximated similarly. The type-punning variable must here be bit-masked to separate the mantissa and add it to the integral portion of the logarithm. (Note that the upper graphics depict the $\log_2$ case, whereas the code below it implements the natural logarithm function by scaling the result by $\ln 2$.)

continuity in the first derivative at the integer boundaries. By contrast, the linear and other approximations may be discontinuous, where $f(1) + 1 \neq f(2)$.

Continuing this approach, Appendix A considers more generalized approaches using rational function approximation, applied to both the exponential and logarithm functions, in both double-precision and single-precision variants. Figure 4.4 illustrates the logarithm.

It is possible to specify various error criteria and analytically or empirically derive the polynomial coefficients that optimize them. This was accomplished using the *Mathematica* software to implement the Minimax (Remez) algorithm. A fourth-order polynomial was found to provide very good approximation error under all conditions; quadratic approximation was not much faster, and higher-order polynomials were not much more accurate – especially for single-precision. Appendix A lists a detailed set of the derived constants applicable for the various error criteria, including: root mean squared error, mean unsigned error, mean biased error, maximum absolute error, and maximum positive/negative error (tight upper/lower bounds on the true function).

A novel contribution, also described in Appendix A, is to combine the logarithm and exponential type-punning approximations in a log-domain addition function, *LogAdd*. This function is typically implemented as follows, to compute $\log(x + y)$ given $\log(x)$ and $\log(y)$:

$$\log(x + y) = \log(e^{\log(x)} + e^{\log(y)}) \tag{4.1}$$

This is commonly reformulated to use just two transcendental operations instead of three:

$$\log(x + y) = \log(x) + \log(1 + e^{\log(y) - \log(x)}) \tag{4.2}$$

where to retain numerical precision, one should ensure that $x > y$.

Rather than simply applying the fast approximations for the logarithm and exponential functions to the equation above, this reformulation already creates the conditions under

Table 4.1: Comparison of single-precision log-domain addition, using the optimized type-punned approximations and the more traditional *LogAdd* function (Eq. 4.2). Processing times reported are the number of seconds needed to compute one billion iterations. See Appendix A for more detailed results.

Device	*LogAdd*	*FastLogAdd*	*FasterLogAdd*
Raspberry Pi	1,150	245	85.4
Odroid U3	165	78.9	14.7
Odroid XU3	232	63.9	24.6
Macbook Air	45.0	38.9	12.4
C3-class	42.3	29.3	4.2
tetra	50.3	34.8	4.3

which the fast log approximation can be applied to an input in the domain $[1, 2)$. In this special case where the range is known a priori to be restricted, the log approximation can be simplified further. For example, the optimized *FastLogAdd* and *FasterLogAdd* functions described in Appendix A each eliminate four unnecessary comparisons, one addition, and one float-to-integer conversion. Consequently, the speedup relative to *LogAdd* is even greater than would be expected from naively combining the exponential and logarithm approximations independently.

Table 4.1 summarizes some results reported in Appendix A, demonstrating the speedup of the *FastLogAdd* and *FasterLogAdd* functions over a traditional *LogAdd* function (using one logarithm and one exponential, as in Equation 4.2). The *FastLogAdd* function uses a 4th-order polynomial approximation, while the *FasterLogAdd* uses a linear approximation with more error. These results consider the single-precision variants of the functions. As noted in the detailed benchmark section of Appendix A, the ARM-based processors were generally faster for single-precision while the Intel-based processors were faster for double-precision.

The *LogAdd* function as described here is implemented in the HTK software toolkit [73], which has been widely used for training HMM-GMM acoustic models for ASR. The log-domain summation of probabilities is a common operation used in HMM-related algorithms, such as Baum-Welch training (implemented by HTK in the *HERest* tool for *embedded re-estimation*). In a test using read speech from the Wall Street Journal, an ASR system was built using a version of the HTK software that was modified to use the *FasterLogAdd* function. At first, the software crashed because internal checks reported an error if particular summations of multiple probabilities resulted in a value outside the range $(0.99, 1.01)$. However, the training process completed successfully by slightly modifying those error-checking bounds to $(0.96, 1.04)$. The resulting ASR system performed almost identically to a system built with the unmodified HTK software; there were no statistically significant differences. However, acoustic models were trained 30% faster, compared to unmodified HTK software.

Obtaining or training HMM-based acoustic models for alignment purposes is a prerequi-

site for MLP training, as will be further discussed in the following section. The fast transcendental functions described here, especially variants of the *LogAdd* function, can provide dramatic speedups for CPU implementations. Unfortunately, such acceleration of the exponential function is generally less noticeable when applied to the sigmoid computations used in traditional neural networks. This is mostly because the overall computation is dominated by matrix multiplications. However, it is also becoming somewhat less relevant because many modern techniques use different non-sigmoidal activation functions, such as rectified linear units. Also, the NVIDIA GPU hardware on which such algorithms are implemented typically are able to exploit transcendental functions that are natively implemented in the instruction set as a single clock cycle.

4.3 Speech Transcript Alignment

Given an audio recording and an associated transcript indicating the spoken words, the objective of speech transcript alignment is to produce a temporal segmentation – or equivalently, to determine timestamp boundaries – of *events* that typically represent words, phonemes, and/or subphonetic model states. This can be considered a relaxation of the ASR problem: rather than recognizing *what* is spoken, we only seek to determine *when* it happens.

Speech transcript alignment is not generally considered a difficult research problem, as it is widely assumed that the standard approaches seem to work. The data sets used in such scenarios are typically of very high quality, consisting of accurate and detailed transcriptions of not only spoken words but also their subword phonemes – as well as other audible sounds such as noises and speech disfluencies. These transcripts are usually manually segmented or timestamped in the form of relatively short sentences, which can simplify the computation and ensure that alignments cannot erroneously cross sentence boundaries. Moreover, when training ASR systems or building TTS voices, the examples that do not align well are simply discarded to prevent the overall quality of the training data from being degraded.

Unfortunately, traditional speech transcript alignment is not adequate when dealing with "real-world" data and applications. An example of real-world data would be consumer-generated online videos with non-verbatim transcripts. This data is extremely plentiful and it would be desirable to be able to use it as a more affordably obtained source of ASR training data. Real-world transcripts do not often include inter-sentence timestamps, so alignment must be performed on very lengthy segments for which the traditional algorithms scale poorly. Speech transcript alignment is especially useful for the application of video captioning, assigning timestamps to words so that they may be displayed at the appropriate times; in this use case, it is not possible to disregard the results of poor alignment.

4.4 Literature review and research motivation

Historical perspective

The earliest approaches to speech transcript alignment considered segmentation and labeling of subword units. While most have been based on phonetic units, some researchers initially explored syllable segmentation using a loudness function [79]. Phonetic segmentation could be achieved with dynamic time warping (DTW) to align waveforms – whether from previously labeled utterances, concatenated templates, or synthesized speech – for the purpose of either ASR or text-to-speech (TTS) [80]. DTW was sometimes combined with rule-based hierarchical classification of broad phonetic classes [81], [82]. A review and comparison of such early approaches (through the mid-1980's) is given in [83].

Progress in automatic alignment benefited from the development of standardized datasets used for ASR research, including TIMIT [84] and SWITCHBOARD [85]. The TIMIT corpus comprised carefully elicited and cleanly recorded speech waveforms that were manually labeled with very precise phonetic alignments [86]. Human labelers were first presented with the automatic alignment produced by the MIT CASPAR system [82], and then asked to adjust the boundaries [87]; more than a third were left unmodified, resulting in an abnormally high distribution of alignment boundaries at 5ms intervals [88].

By the early 1990's, some alignment approaches continued to be developed using DTW [89] as well as neural networks [90], [91]. However, the ASR community had rapidly converged on the predominant HMM framework. As noted by [92]: "[because] alignment can be considered as simplified speech recognition, it is natural to adopt a successful paradigm of ASR, namely HMMs, for alignment". HMM-based alignment systems trained and evaluated on the TIMIT corpus achieving satisfactory performance: generally about 90% accurate within 20ms relative to hand-corrected reference boundaries [88], [93], [94].

An excellent journal article published in 2009 by Hosom [95] can be considered the most comprehensive review of prior work on phoneme alignment. In a survey of 33 automatic alignment systems: 14 were HMM-based (e.g. [96], [97], [98], [99], and [100]); 8 used DTW (e.g [101] and [102]), primarily for speaker-dependent alignment; and the remaining 11 used various alternative approaches (e.g. [103], [104], and [105]). Alignment systems have also been developed for languages as diverse as Basque [106] and Filipino [107], and specifically for foreign-accented dialects of Australian English [108]. Although usually based on phonemes, some systems have used surface-level graphemes as well [109].

Manual alignment: difficult and inconsistent

The most commonly used metric for phoneme alignment quality calculates the proportion of boundaries placed within 20ms of a human-labeled reference. However, due to the inherent difficulty and subjectivity of phoneme boundary placement, alignments manually labeled by experts have been found to be considerably inconsistent [90], [93], [95], [110]–[113]. Table 4.2 summarizes inter-labeler agreement from several independent measurements. These results

Reference	Corpus	(% w/in 20ms)
Hosom [95]	English (TIMIT)	93.49
Leung & Zue [82]	English	∼90
Cosi *et al.* [112]	Italian (isolated)	93.5
Cosi *et al.* [112]	Italian (continuous)	89.1
Ljolje *et al.* [113]	Italian	92.9
Wesenick & Kipp [114]	German	96

Table 4.2: Human-human agreement in phoneme boundary placement.

suggest that even in the best case (cleanly recorded, carefully produced read speech), human judgments of phoneme alignment are far from perfect. Thus, as noted by [95], "the goal of automatic phoneme alignment is not to achieve 100% accuracy, but to achieve agreement in boundary placement that is always as good as the best human-human agreement."

Evaluation is considerably more difficult for other task domains. As noted by the creators of the Buckeye corpus [115], a carefully transcribed collection of conversational speech, the task of phoneme labeling and segmentation is complicated by the extreme variation in phonetic realizations of word forms [116]. Using four redundant human transcribers, only 80.3% pairwise labeling agreement was achieved; a unanimous label was assigned only 62% of the time; for those instances in agreement the mean deviation in boundary placement was 16ms [115]. A similar study was prepared by [117], comparing inter-labeler consistency for the English, German, Mandarin, and Spanish telephone speech.

Not only are such human-labeled phoneme alignments very inconsistent and perhaps unsuitable for use as scoring references, they are also extremely expensive to produce. An effort undertaken to phonetically transcribe a subset of the SWITCHBOARD corpus, employed highly trained linguistics students who reportedly performed the alignment and labeling tasks at nearly 400x real-time [118]. This corpus was useful for gaining insights into spoken language [118], such as the prevalent reduction of function words [119].

State-of-the-art phoneme alignment

The most commonly used dataset for comparing phoneme alignment quality is TIMIT. Unfortunately, the results are not entirely comparable, since researchers map the TIMIT phone sets differently and use slightly different test subsets. Nonetheless, Table 4.3 presents several results compiled from phoneme alignment experiments on the TIMIT corpus.

Hosom [95] reports the best performance for this task. His "baseline" is a fairly standard HMM/ANN hybrid system where the Viterbi decoder is modified to enforce HMM durations with an approximated Gamma distribution. Hosom's "proposed" system is rather unique: it uses several energy-based acoustic features, a specialized HMM that predicts the probability of a transition given an observation; and multiple classifiers of distinctive phonetic features. Such a system is difficult to replicate – although the software has been freely shared [121].

Reference	(% w/in 20ms)	Description
Hosom [95]	93.36	"proposed" HMM/ANN hybrid
	91.48	"baseline" HMM/ANN hybrid
Keshet *et al.* [105]	92.3	Discriminative alignment algorithm
	88.8	... using single best base function
Sjölander [120]	89.9	HMM/GMM
Brugnara *et al.* [93]	88.8	HMM/GMM
Pellom & Hansen [110]	85.9	HMM/GMM; 8-kHz resampled
	56.0	... noises added at 10dB SNR
	34.6	... cellphone retransmission

Table 4.3: Machine-human agreement in TIMIT phoneme boundary placement.

The system produced by Keshet *et al.* [105] is noteworthy for its exceptional performance without the use of the HMM framework. Instead, a discriminative learning approach is used to predict the start times of events in a given signal. An alignment prediction vector is formed by solving a constrained optimization problem using an iterative algorithm. The overall approach is quite different from mainstream ASR techniques, yet it is a general-purpose alignment algorithm that has also been applied to music-to-score alignment. This work is related to max-margin Markov networks [122], a framework which has also been applied to ASR [123] but has not been very widely adopted.

The best alignment results on TIMIT are close to the limit of human-human agreement, suggesting that it may be a solved problem. However, as found by Pellom & Hansen [110], the performance degrades very dramatically when the TIMIT corpus is transformed by additive noise or retransmitted through a cellular telephone channel. A more recent study considered concatenated TIMIT utterances with babble noise, and reported nearly 80% word-level alignment agreement within 50ms [124].

Recent phoneme alignment results have been reported in other languages, and seem very comparable to prior work on TIMIT. For French: 84.6% of alignments were in agreement under 20ms [125]. For Italian: 94% were in agreement within 20ms or less [126]. For Mandarin Chinese: 87.7% were in agreement within 20ms or less [127]. Using an English-trained forced alignment system, 62.9% agreement within 20ms was achieved when tested on the (endangered) Mixtecan language of Mexico [128].

As an excellent step toward standardized evaluation, the Italian EVALITA 2011 Forced Alignment task was established [129] and received submissions from three participants [130]–[132]. Utilizing a time-mediated error rate metric, the submissions achieved 1-4% error on word-level alignment and 12-21% error on phone-level alignment [129].

Toward "real-world" data

Some researchers have explored more difficult alignment problems. An effort to automatically align the entire SWITCHBOARD corpus revealed the novel challenges posed by conversational telephone speech [133]. Using a constrained Baum-Welch algorithm [99], the phonetically transcribed subset of SWITCHBOARD [118] was aligned to within an average of 30.9ms of the manual references; the comparable metric for TIMIT was 20.3ms [99].

The SWITCHBOARD recordings aligned by [133] were considerably longer than TIMIT recordings: an average of 7 minutes. Although neither phone-level nor word-level alignment quality was evaluated, a 24% alignment failure rate was noted for instances characterized with poor signal quality [133]. A similar study [134] examined a system designed for aligning long video transcripts presegmented paragraph level, achieving 100% word-level alignment accuracy within a 3-second window – excluding 8.7% of paragraphs which failed to align.

It is possible to align 2-hour long spoken book recordings in a single step [135], presumably with tight beam pruning that succeeds thanks to the high fidelity of the audio and transcriptions of professionally produced audio-books. Another approach for aligning audiobooks first runs ASR with a language model biased to the book text, and then performs text matching to align sentences and utterances [136].

However, the quality of commercially produced transcription for other media can vary from 5-10% word error rate [137]. In these cases, the problem of long audio alignment has been addressed by recursive [124], [138] or multi-pass [137] strategies that attempt to recognize speech and then perform text-level alignment [139], [140]. An alternative single-pass approach proceeds with the alignment of incremental chunks of audio or text data [141], and may include a modified Viterbi search algorithm [142]; a similar greedy variant of DTW is considered in [143]. Presegmentation at phrase boundaries has also been used for long audio alignment [144].

A simple and efficient method for long audio alignment is presented in [145]: first an unconstrained phoneme decoding is performed, and this is then aligned to a phone sequence derived from the transcript. The authors note that despite phone error rate in the range of 40-60%, approximately 96% of word boundaries are within 0.5s of the manual reference. Another method that produced can produce accurate word-level alignment uses TTS and an incrementally checkpointed DTW algorithm operating on silence-filtered audio [146].

As an alternative to n-gram language models, researchers from Google have also used a decoding graph comprising factor automata, which accept only subsequences of a transcript [147]. Tested on a set of diverse YouTube videos, this system used optimized WFST graphs that were orders of magnitude smaller than those used for large-vocabulary decoding (577KB, as opposed to approximately 1GB) and provided a 65% speedup. While noting "that the timing information produced by the system is usually accurate when there is a match in transcripts and hypotheses", these authors also observed that most errors were "due to the segmenter rejecting noisy or speech under music, low quality speech (strong noise), and accented speech."

Some interesting experiments have been presented in which transcriptions are method-

ically corrupted, to measure the robustness of alignment under adverse conditions [148]. Since human-labeled references are difficult to obtain, consensus agreements among multiple systems have been used as an alternative measure [149]. When phone-level references are available, some authors have proposed measuring alignment quality in terms of overlap rate [150], or a combination of phone edit distance and boundary timing differences [151]. Other authors have instead reported primarily word-level alignment metrics [152], and explored factors of ASR systems that contribute to word alignment quality [153].

Diverse applications

Speech transcript alignment is useful for the purpose of preparing high quality corpora for acoustic model training [154], which in the case of imperfect or unreliable transcriptions is considered *lightly supervised* [155]. Approaches vary from performing a constrained recognition [156], to flexible alignment using garbage/filler models [157], and phonetic decoding followed by phone-level alignment [158]. This approach can be used to detect precisely transcribed portions of corpora [159], [160], including "found data" such as that harvested from the Internet [161] or collections of corporate videos [162]. It can also be used to bias lattice generation for discriminative acoustic model training [163], or for the purpose of text-to-speech [164].

Much of the work in phone alignment was motivated by the application of concatenative text-to-speech, for which high accuracy alignments were required [113], [165]. HMM-based alignment was typically employed; its quality and effect on synthesis were evaluated by [166], finding speaker-dependent crossword triphone HMM models to outperform even manual alignments. Nonetheless, some alignment post-processing was often found to be useful [167]–[169], and some researchers found other automatic alignment methods to outperform HMM approaches [170]. An area of interest has been the automatic detection of alignment errors [171]. It is worth remarking that corpus-based TTS systems usually operate on data that is prepared with extreme care and does not exhibit the characteristics of "real-world" data presented for recognition by ASR systems. A detailed review of high-accuracy phonetic alignment for the purpose of TTS is beyond the scope and aim of this thesis.

Because of the impracticality of manually producing phonemic alignment – and perhaps due to its inherent inconsistency – automatic alignment has begun to be exploited by linguists (principally at the University of Pennsylvania). Automatic processing provides an effective way to quickly search a large speech corpus to detect regions of interest that may later be more closely examined or statistically analyzed. Such alignment techniques have been used to detect approximate stop burst regions for the measurement of voice onset time [172]; and previously in several similar investigations of other phonetic phenomena [173]–[175].

One of the most direct commercial applications of transcript alignment is for the production of captions or subtitles that are displayed along with video. Captioning of television broadcasts has been explored by [176] and [177]; in the case of live broadcasts, the alignment algorithm must be specialized to respect certain real-time latency requirements [178]. In

contrast to television shows, the alignment of feature-length films to a screenplay or script [179], [180] must address problems with long audio and filtering of extraneous unspoken text.

Perhaps the most challenging application of automatic alignment is to align sung music to lyrics, for which various diverse strategies have been attempted with modest success [181]–[183]. For this task, it is not a trivial problem to obtain and appropriately pre-process the lyrics text, which is often not-verbatim and includes inconsistent transcription conventions such as for the repetition of chorus lines. One approach gathers multiple sources of lyrics and combines them for alignment [184]. Research that is very related to music-lyrics alignment considers the alignment of speech in the presence of background music [185].

A compelling case has been been made for *same-language subtitling* of television broadcasts in India [186], [187], as an effort to increase literacy and reading rates. Karaoke-style same-language subtitling of Bollywood film songs [188] especially requires accurate alignment to highlight syllables of words as they are sung. television captions. Alignment is also a useful technology for computer-assisted language learning [189], particularly to help in assessing pronunciation quality of foreign languages; it can also be used as an aid in the acquisition of elementary reading ability.

For a highly comprehensive review of educational applications for caption and subtitle technologies, see [190]. Reflecting a trend toward human computation and game-based incentives, an amusing project demonstrated a game in which players effectively performed word-level alignment of audio – producing results that exceeded the accuracy of automatic methods [191].

Discussion

Speech transcript alignment is a problem that has been studied primarily from the perspective of phoneme alignment of carefully prepared corpora such as TIMIT. However, given the already exceptional performance of existing systems on this task – approaching human levels of annotation (in)consistency – it would be useful to consider more difficult problems for speech transcript alignment. In general, such problems would be characterized by natural speech, noisy audio, inaccurate transcriptions, and computational limits. Unfortunately, it is very difficult to objectively evaluate alignment solutions that address such data. Therefore, this thesis contributes a discussion of the factors that need to be considered in such scenarios, along with a qualitative assessment of a system that has been successfully deployed.

The following aspects of alignment quality can be addressed:

- **Accuracy**: the standard Viterbi algorithm determines a maximum-likelihood path. This is an approximation that is suboptimal in relation to algorithms that perform alignment using a maximum a posteriori criterion or full observation likelihoods.

- **Speed**: especially in the case of very long inputs, the standard quadratic performance scaling could be improved. A typical approach is to apply beam pruning, but performance must be considered with speed-accuracy tradeoffs.

- **Robustness**: the alignment should gracefully handle common issues with suboptimal transcriptions that have incorrect or missing labels. In particular, the alignment should be robust to the significant amounts of untranscribed music and non-speech audio that characterize "real-world" data.
- **Portability**: it would be desirable for transcript alignment to be easily applied to any task domain and language, without requiring expert knowledge or configuration.

Speech transcript alignment is an essential technique employed in the training of ASR models. For example, phone-level alignments can be used to bootstrap HMM-GMM models prior to refinement with Baum-Welch training (expectation maximization); this may be preferable to a "flat-start" initialization setting identical parameters for each phone model. In some cases, alignments are iteratively recomputed for Viterbi-style training ("hard EM"); the Kaldi software toolkit follows such an approach. In addition, alignments are often used to generate target labels for neural network training, and may also be used to approximate likelihoods for VTLN warp factor estimation. Although manually-produced alignments have previously been demonstrated to be superior for the purpose of training ASR models [192], for most speech corpora it is generally only feasible to produce automatic alignments.

For the evaluation of keyword search systems, alignment is also used to prepare "ground-truth" references used for time-based scoring of keywords. For concatenative TTS systems, phone-level alignments are often used to segment diphone units. It may also be useful to explore the effect of automatic speech transcript alignment quality for these technologies.

4.5 Forced alignment algorithms

A speech transcript alignment could be formally defined as a sequence of triples, where each alignment unit is associated with a start time and end time. However, for our current purposes it will suffice to consider an alignment as:

$$\mathbf{t}_{1:N} = (t_1, \ldots, t_N) \quad : \quad t_i \in [1, T], t_i \leq t_{i+1} \tag{4.3}$$

a sequence of monotonically increasing start times for each of N events (words, phonemes, nonspeech sounds) in a transcript corresponding to a sequence of T acoustic observations.

A set of HMM acoustic models may be used for *forced alignment*, in which a long HMM chain is concatenated according to the sequence of events in the transcript. For word alignment, each word may be looked up in a pronunciation dictionary, copying the models for constituent pronunciation units and inserting transitions from the end state of each model to the start state of the next. Words with multiple pronunciations may be represented with parallel branches in the chain. For phoneme alignment, the model sequence is typically known without requiring the use of a pronunciation dictionary.

In this section we present several algorithms that compute HMM-based alignments. For now, this exposition will disregard silence models which are typically inserted before, between, and after all word models. To a rough approximation, these can be considered as

optionally skippable events for which the HMM chain includes a non-emitting transition from the start to the end state of each silence model. Also, due to windowing assumptions used in the temporal discretization of acoustic observation sequences, it may be necessary to adjust the alignment times by 12.5 milliseconds. Further theoretical and implementation details of HMM-based ASR and forced alignment may be provided by excellent tutorial overviews [193], [194] and the documentation for HTK software [195].

Maximum likelihood path (Viterbi; approximate)

HMMs are probabilistic graphical models [196]–[198] defining conditional independence assumptions that factor the joint probability of observation and state sequences as:

$$P(\mathbf{o}_{1:T}, \mathbf{s}_{1:T}) = \prod_{t=1}^{T} P(o_t|s_t)P(s_t|s_{t-1}) \tag{4.4}$$

The best-path state sequence $\tilde{\mathbf{s}}_{1:T}$ that maximizes this quantity may be expressed as:

$$\tilde{\mathbf{s}}_{1:T} = \underset{(s_1,\ldots,s_T)}{\arg\max} \prod_{t=1}^{T} P(o_t|s_t)P(s_t|s_{t-1}) \tag{4.5}$$

While the number of possible paths $(s_1, \ldots, s_T)$ grows exponentially with the length of the sequence, the search for the maximum-likelihood path may be tractably computed by the *Viterbi* algorithm [199], [200]. This dynamic programming solution exploits a recurrence relation between partial sequences:

$$\left[\max_{\mathbf{s}_{1:t-1}} P(\mathbf{o}_{1:t}, \mathbf{s}_{1:t})\right] = P(o_t|s_t) \max_{s_{t-1}} \left(P(s_t|s_{t-1}) \left[\max_{\mathbf{s}_{1:t-2}} P(\mathbf{o}_{1:t-1}, \mathbf{s}_{1:t-1})\right]\right) \tag{4.6}$$

By incrementally caching the quantities denoted between square brackets above, a dynamic programming implementation can compute the maximum joint likelihood of $P(\mathbf{o}_{1:T}, \mathbf{s}_{1:T})$ in $O(NT^2)$ time – or $O(T^2)$ assuming that the number of concatenated HMM states is proportional to the number of acoustic observation frames ($N \propto T$) and that the number of transitions from each state has a small constant bound (typically left-to-right models for which each state has only a self-loop and a transition to the next state). By retaining a data structure of backtraces, the best-path state sequence may be recovered by an algorithm using $O(T^2)$ space. In practice, *beam pruning* optimizations [201] could typically be used to improve the best-path search to use linear time and space.

The Viterbi alignment start times $\tilde{\mathbf{t}}_{1:N}$ can then easily be derived from this best-path state sequence $\tilde{\mathbf{s}}_{1:T}$ by associating with the first occurring state of each event:

$$\tilde{\mathbf{t}}_{1:N} = \left(\tilde{t}_1, \ldots, \tilde{t}_N\right) \quad : \quad \tilde{t}_i = \min(\{t : \tilde{s}_t \in \mathcal{M}_i\}) \tag{4.7}$$

where $\mathcal{M}_i$ is the set of states in the model for the i-th event in the transcript.

The Viterbi algorithm is mostly commonly used as an *approximation* of quantities that are otherwise difficult to optimize. For example, in ASR, the goal is to find an optimal sequence of words $\hat{\mathbf{w}}$ among the set of all strings $\mathcal{V}^*$ (a formal language defined as the Kleene closure on the vocabulary $\mathcal{V}$), maximizing the following criterion:

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}\in\mathcal{V}^*} P(\mathbf{o}_{1:T}|\mathbf{w})P(\mathbf{w}) \tag{4.8}$$

In practice, this quantity is reformulated by compiling an HMM based on several structural assumptions (such as word sequences being characterized by an *ngram* language model, a low-order Markov process), allowing a maximum-likelihood state sequence $\tilde{\mathbf{s}}_{\mathbf{1}:\mathbf{T}}$ to be determined with the Viterbi algorithm. The optimal word sequence $\hat{\mathbf{w}}$ is assumed to be mapped to this state sequence – but this is not guaranteed. A similar situation holds in the case of alignment: the state sequence determined by the Viterbi algorithm is not necessarily "optimal" in terms of alignment or segmentation. For this reason, we denote the approximate nature of a *Viterbi alignment* using the *tilde* ($\tilde{}$) notation.

Least-squares/MAP boundaries (forward-backward; pointwise)

A variant of the Viterbi algorithm is the *forward* algorithm for HMM inference, which substitutes summation in place of maximization, in order to compute total joint probability:

$$\left[\sum_{\mathbf{s}_{1:t-1}} P(\mathbf{o}_{1:t}, \mathbf{s}_{1:t})\right] = P(o_t|s_t) \sum_{s_{t-1}} \left(P(s_t|s_{t-1}) \left[\sum_{\mathbf{s}_{1:t-2}} P(\mathbf{o}_{1:t-1}, \mathbf{s}_{1:t-1})\right]\right) \tag{4.9}$$

In conjunction with a closely related *backward* recursion formula, the *forward-backward* algorithm (introduced with the Baum-Welch HMM parameter re-estimation algorithm [202]) can be used to determine $P(s_t|\mathbf{o}_{1:T})$, the posterior probability distribution of state occupations at any time given only the observation sequence.

Demuynck [100], [203] presents an interesting alternative to Viterbi alignment, proposing that posterior probabilities derived by the forward-backward algorithm could be used to define a segmentation. This requires the computation of the probability of a boundary – as opposed to a state – occurring at a given time. For many ASR implementations, the HMMs may include non-emitting entry and exit states that can be assumed to occur at times $t \pm \epsilon$, i.e. just before or after an observation emitted at time t [195]. In this case, an alignment maybe computed in terms of the temporally integrated posterior probabilities of such states:

$$\bar{\mathbf{t}}_{1:N} = (\bar{t}_1, \ldots, \bar{t}_N) \quad : \quad \bar{t}_i = \sum_{t=1}^{T} tP(s_{t-\epsilon} = \mathcal{M}_i^0 \mid \mathbf{o}_{1:T}) \tag{4.10}$$

where $\mathcal{M}_i^0$ indicates the (zero-th) entry state of the model for the i-th event in the transcript. In Demuynck's comparison, the Viterbi alignment is viewed as merely a maximum likelihood

approximation, while the forward-backward segmentation provides "the best possible estimate of the boundary in a least squares sense" [100]. He reports that these are slightly more accurate than Viterbi alignments for phoneme segmentation of a Dutch corpus. We will refer to this method as *least-squares alignment*, and denote it with the *bar* ($\bar{}$) notation.

Under this view, another interpretation of alignment based on such posterior probabilities could be considered as:

$$\dot{\mathbf{t}}_{1:N} = \left(\dot{t}_1, \ldots, \dot{t}_N\right) \quad : \quad \dot{t}_i = \arg\max_t P(s_{t-\epsilon} = \mathcal{M}_i^0 \mid \mathbf{o}_{1:T}) \tag{4.11}$$

Such an alignment might be considered pointwise optimal in the sense that each boundary is estimated at its *maximum a posteriori* (mode of the posterior distribution). We will refer to this method as *MAP alignment* and denote it with the *dot* ($\dot{}$) notation.

Note that neither Equation (4.10) nor (4.11) enforce a global constraint on ordering of alignment times, as defined in (4.3). This could be problematic if the optimum for one time is greater than the optimum for any subsequent times – but presumably this scenario does not occur frequently in practice. Demuynck also notes that the factorization of the joint probability given in Equation (4.4) may be adjusted by a weighting factor to account for the inaccurate HMM assumption that observations are independent, but found that this parameter had little effect. In addition, Demuynck found it useful to post-process the boundaries by utilizing a confidence interval computed with respect to the variance:

$$\sigma_i^2 = \sum_{t=1}^{T} P(s_{t-\epsilon} = \mathcal{M}_i^0 | \mathbf{o}_{1:T})(\bar{t}_i - t)^2 \tag{4.12}$$

Such a quantity may be useful in certain applications in which a segmentation or alignment may not be required between all events; for example, to prepare video captions it may be desirable to segment only after every few words at the boundaries with highest certainty.

Total probability segmentation (forward; optimal)

It would be interesting to consider alignment in terms of a segmentation that maximizes the total probability of observation subsequences corresponding to each event:

$$\hat{\mathbf{t}}_{1:N} = \arg\max_{(t_1,\ldots,t_N)} \sum_{i=0}^{N} P_{\mathcal{M}_i}(\mathbf{o}_{t_i:t_{i+1}-1}) \tag{4.13}$$

where $P_{\mathcal{M}_i}(\mathbf{o}_{t_i:t_{i+1}-1})$ is computed with a model corresponding to the i-th event, determining the sum of all paths from event model's (non-emitting) entry state at time $t_i - \epsilon$ to the exit state at time $t_{i+1} - 1 + \epsilon$. For completeness: $t_0 = 1$ and $t_{N+1} - 1 = T$, while $t_i \leq t_{i+1}\ \forall i \in [0, N]$. An equivalent representation of Equation (4.13):

$$\hat{\mathbf{t}}_{1:N} = \arg\max_{(t_1,\ldots,t_N)} \sum_{i=0}^{N} P(\mathbf{o}_{t_i:t_{i+1}-1} | s_{t_i-\epsilon} = \mathcal{M}_i^0, s_{t_{i+1}-1+\epsilon} = \mathcal{M}_i^{-1}) \tag{4.14}$$

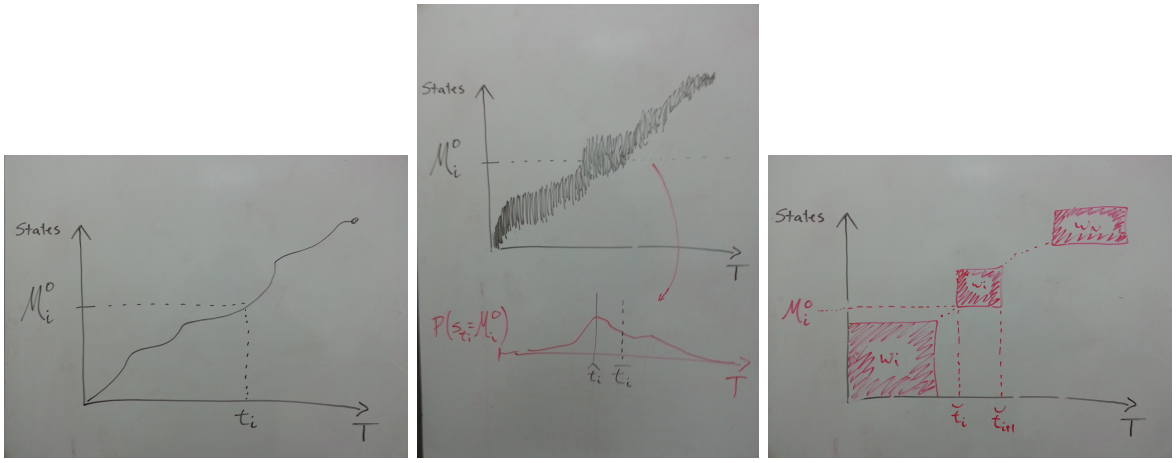


Figure 4.5: Illustrative comparison of alignment algorithms. Left: Viterbi alignment ($\tilde{\mathbf{t}}$). Center: least-squares ($\bar{\mathbf{t}}$) and MAP alignment ($\dot{\mathbf{t}}$). Right: optimal alignment ($\hat{\mathbf{t}}$). (Note that the labels in these drawings do not precisely match the variable names used in the text.)

where $\mathcal{M}_i^{-1}$ indicates the (last-indexed) exit state of the model for the i-th event. Because this formulation specifies alignment explicitly and directly as the optimization of a sequence of boundary times (unlike the Viterbi algorithm which maximizes over state sequences) and results in a globally constrained solution (rather than multiple independent decisions), we will refer to it as the theoretically *optimal alignment* and denote it with the *hat* ($\hat{}$) notation.

The value of $P_{\mathcal{M}_i}(\mathbf{o}_{t_i:t_{i+1}-1})$ for all subsequences of $\mathbf{o}_{1:T}$ may be computed with a variant of the forward algorithm related to segmental DTW [204]. The traditional trellis data structure should include an additional time index to cache intermediate quantities such as:

$$m_i[t'][t][s] = P(\mathbf{o}_{t':t} | s_{t'-\epsilon} = \mathcal{M}_i^0, s_{t(+\epsilon)} = s) \tag{4.15}$$

Ultimately, the cached quantities of interest would be:

$$m_i[t'][t] = P(\mathbf{o}_{t':t} | s_{t'-\epsilon} = \mathcal{M}_i^0, s_{t+\epsilon} = \mathcal{M}_i^{-1}) \tag{4.16}$$

In other words: the total probability of the observation subsequence from time t' to t generated by the i-th event model. For each event model $\mathcal{M}_i$, the computation of Eq. (4.16) has complexity $O(T^2)$, assuming the number of states in an event model is a relatively small constant. It may also be reasonable to assume a maximum duration for many events (such as spoken words or phonemes), limiting the range of $t - t'$ and enabling linear-time computation. These quantities must be computed once for each unique event in the alignment vocabulary $\mathcal{V}$, resulting in asymptotic time and space complexity of $O(|\mathcal{V}|T)$.

Provided the quantities in Eq. (4.16), the optimization of Eq. (4.13) can be solved with a dynamic programming formulation. Let:

$$d_n[t] = \max_{(t_1,\ldots,t_n)} \sum_{i=0}^{n} P_{\mathcal{M}_i}(\mathbf{o}_{t_i:t_{i+1}-1}) = \max_{(t_1,\ldots,t_n)} \sum_{i=0}^{n} m_i[t_i][t_{i+1}-1] \tag{4.17}$$

(subject to the constraints that $t_0 = 1$ and $t_{n+1} - 1 = t$, while $t_i \leq t_{i+1}\ \forall i \in [0, n]$). In other words, the cached quantities $d_n[t]$ represent the partial optimal solutions for the subproblems considering $n \leq N$ events and $t \leq T$ observations. A recurrence relation holds:

$$d_n[t] = \max_{t'} d_{n-1}[t'] + m_n[t'][t] \tag{4.18}$$

To compute the ultimate solution at $d_N[T]$ thus requires $O(NT^2)$ time and $O(NT)$ space to store the backtraces needed to recover the optimal alignment.

Figure 4.5 illustrates the variants of alignment algorithms described thus far. Whereas the Viterbi algorithm results in a single path through the trellis, the forward and forward-backward algorithms consider multiple paths.

4.6 Case study: the Mod9 Alignment API Service

To demonstrate some of the issues encountered in "real-world" deployment of speech-transcript alignment, we consider a case study in which this technology was provided to serve diverse customer needs. The Mod9 Alignment API [205] implements a RESTful web service [206], delivered as a customized virtual machine image [207] that runs on the Amazon Elastic Compute Cloud (EC2) infrastructure. The software primarily comprises components of the HTK toolkit, primarily using the *HVite* tool to perform Viterbi alignment.

The service began operating in early 2011, as a technology used by various transcription providers (manual, human-powered – not using ASR). In addition to specific use cases such as transcribing interviews for journalism activities and for insurance claims, the bulk of transcription was intended for online video captioning. This demand was partly generated by compliance requirements due to recently enacted legislation: the Twenty-first Century Communications and Video Accessibility Act has been imposing increasingly stricter requirements for online programming, similar to broadcast television coverage.

Speech transcript alignment in this situation serves at least two purposes:

- **Time-stamping**: Many humans are able to skillfully and efficiently provide transcriptions for audio; in some cases, depending on the subject matter, it may even be considered reasonably enjoyable. However, it is almost universally considered incredibly tedious to manually associate timestamps with such transcriptions. These are necessary, however, for video captioning purposes: the words must be displayed on screen as they are spoken. Alignment algorithms can serve this purpose extremely well, especially since the desired boundaries are often at phrase or sentence breaks associated with easy-to-align silences. Moreover, caption viewers are rather tolerant of minor mistakes; whereas research systems typically evaluate alignments with a 20ms tolerance, television viewers are accustomed to seeing captions that may appear with several seconds of time skew (generally lagging behind, especially for live broadcasts).

- **Quality Assurance**: Particularly with *crowd-sourced* workforces, there is tremendous variance in the quality of work. Much of the value provided by management platforms for these human resources involves identifying work and workers of trusted quality. A typical approach involves redundantly distributing identical samples to different workers. In this case, two workers may provide differing transcripts for the same speech example. Running alignment on both cases can provide two likelihood scores; it can be assumed that the higher-scoring transcript is more likely to be more correct. Although this is based on a Viterbi best-path approximation, in practice these likelihood scores have proved to be useful indicators of comparative transcription reliability.

Since its release as version number 0.1, this *software-as-a-service* has undergone several iterations of improvement based on customer feedback and feature requests. After processing tens of thousands of hours of audio and video over the past four years, the service has reached a level of relative maturity at version 0.8. The following sections describe some of these adjustments that were made in response to sometimes unexpected problems or feedback.

Speed improvement with pruning and simplified models

As originally implemented, the Viterbi alignment algorithm used by the service did not employ any beam pruning. This guaranteed that the resulting maximum-likelihood path would be optimal, but also suffered from very poor scaling due to the quadratic complexity of the search.

In most research settings, this is never a problem because alignment inputs are considered to be pre-segmented utterances: generally speaking, the duration of utterances in the ASR context is measured in seconds and would ideally represent an amount of speech corresponding to a grammatical sentence. In practice, however, the inputs presented to the alignment service were sometimes many orders of magnitude longer. It was not uncommon for customers to pass in a feature-length film and try to align it to a movie script (which also suffers from text-formatting problems described in the next section).

A change was made to provide customers with an option to utilize beam pruning; the problem then became what threshold would be appropriate. If the pruning threshold were too low, many alignments could fail if no surviving path would be found to occupy the last HMM state in the concatenated model at the last time step. Fortunately, such catastrophic failures would generally return quickly due to the acceleration of a tight pruning threshold. Thus a strategy was suggested to customers in which the beam width would be successively increased until the alignment did not fail. This strategy is common in ASR system training, as recommended in The HTK Book [195] and supported by the *HVite* software.

Additional speedup was attained by simplifying the acoustic models. Although triphones are known to provide superior speech recognition performance, we found that monophones provide nearly identical alignment performance. Monophones require many fewer distinct models to be evaluated. Reducing the number of Gaussian mixtures also provided a significant acceleration, without exhibiting any noticeable deterioration of alignment quality.

Robustness to inaccurate transcripts and non-speech sounds

Another major complication was encountered in the quality of the "real-world" data. This was seen in the form of transcripts that did not accurately represent all of the sounds in an audio signal, as well as audio signals containing sounds that would not generally be representable with a transcript. Unlike a research setting, in which both data sources would be more carefully controlled, the alignment service had to contend with an extreme diversity and complexity of inputs.

In the case of non-verbatim or inaccurate word-level transcriptions, a first step was to allow *skip* transitions in the HMM alignment graph. By making any "word" token optional, the alignment could flexibly deal with insertion errors; these were commonly caused by text formatting problems, such as customers supplying transcripts that were not properly cleaned to remove metadata such as page numbers, HTML tags, etc... A common insertion in interviews would be the provision of participants' names at the beginning of each change-of-speaker conversational turn.

A second step was to train a "garbage" or "filler" model that could be used to align to generic speech. This model would be inserted between each pair of successive transcribed words, and allows the alignment to flexibly account for deletion errors in the transcript. In practice, however, it did not appear very common for transcripts to be missing spoken words. A greater complication arose from non-speech sounds that were not transcribed.

In many research-quality training datasets, some types of non-speech sounds – such as laughs, coughs, clicks, etc. – are included in the transcripts. In such cases, specialized phone-like models could be trained for these sounds, and then optionally inserted to occur between any pair of words in an alignment graph. Such "richer" alignment graphs were seen to help in many situations, albeit at the expense of slower computation due to the increased number of HMM states in the graph search.

Unfortunately, due to licensing issues, the alignment service could not operate with acoustic models trained from datasets intended for research purposes. Moreover, research datasets were not typically representative of the type of audio encountered by the alignment service. Thus the acoustic models had to be trained from earlier subsets of the data that had been presented to the service.

These rarely included transcriptions of non-speech sounds, especially in commercial-grade captions for television of online videos. A notable exception, however, was found in film subtitling. In contrast to captions, subtitles (specifically, *same-language* subtitles – not foreign language translations) are generally produced with far greater care and expense in order to capture both the dialog spoken by actors as well as other acoustic events that contribute to the theater experience.

Portability to new languages, without pronunciation resources

In late 2013, a customer commissioned an expansion of the alignment service to address several non-English languages: French, Spanish, German, and Mandarin Chinese. Unlike in

a research setting, no resources were provided other than a small initial set of transcribed speech; in particular, no pronunciation dictionaries were available.

Rather than purchase pronunciation lexica, or attempt to leverage publicly available pronunciations on the Internet [208], we chose to explore the use of graphemic rather than phonemic acoustic models. Our hypothesis, based on expectations derived from recognition results, was that graphemic models should underperform relative to phonemic models, especially when trained on very limited amounts of data and using monophone (i.e. monographeme) HMM models. This was expected to be more significant for languages such as Chinese, French, and English – which have less straightforward letter-to-sound orthographic/pronunciation rules than languages such as Spanish and German.

However, it was unknown how these expectations would translate to the task of speech-transcript alignment. Much to our surprise, we discovered the following:

- English models trained with graphemes performed indistinguishably similar to English models trained with phonemes and a pronunciation dictionary. The alignment service's default English models were thus updated to the graphemic models. A pleasant side-effect of using graphemes was that they immediately and automatically provided a facility for better alignment of out-of-vocabulary words – which had previously been replaced with garbage/filler models.

- French graphemic models trained with about 50 hours of data have performed very well, from a subjective perspective based on limited manual inspection and a lack of customer complaints.

- Mandarin Chinese graphemic models failed to train, due to complications with data sparseness. It was concluded that a pronunciation dictionary would be required.

As expected, the Spanish and German graphemic models performed well for the purpose of alignment. Notably, the German models were sufficiently trained despite having only five hours of supplied data.

4.7 Conclusion

This chapter has described some of the practical considerations for training deployable ASR systems and MLP classifiers.

Whereas the previous chapter described the effect of signal processing and frame batch sizes on MLP evaluation in a low-latency real-time setting, here it was seen that for training purposes spectral feature extraction and latency are not relevant concerns. While larger frame batch sizes can enable more efficient matrix multiplication, especially on high-end server platforms with greater processing resources, it is nonetheless important to consider the effect that large batch sizes have on the convergence of neural network training techniques. The tradeoff that large batch sizes exhibit is between faster processing of each training epoch

and slower convergence in terms of the number of epochs. For one particular experiment, it was found that training an MLP using a frame batch size of 256 was strictly better to larger or smaller batch sizes.

The previous chapter also had suggested that the computation of the exponential function (used for the traditional sigmoidal nonlinear activation function in neural networks, as well as the softmax output layer in some cases) is a relatively inconsequential consideration in MLP computations, particularly compared to the greater significance of the matrix multiplications. However, this chapter showed that accelerating transcendental functions can have a very large effect on other algorithms that involve probabilistic inference. In particular, it was found that implementing a fast approximation of log-domain addition resulted in 30% faster HMM-GMM acoustic model training. Although the focus of this thesis is on neural networks and discriminative acoustic features, it was explained that training HMM-GMM acoustic models are typically a prerequisite.

Similarly, the importance of speech-transcript alignment was also presented in the context of preparing discriminative acoustic features. Alignments serve as supervised training labels to train MLPs and DNNs, whether for acoustic feature extraction or hybrid speech recognition scenarios. In addition to a literature survey and comparison of alignment algorithms, a case study was presented describing some of practical problems (and attempted solutions) that were encountered when deploying alignment as a service to customers with "real-world" data.

Chapter 5

Conclusion

Discriminative acoustic feature extraction is a helpful technique that generally provides a significant improvement over standard spectral-based acoustic features such as MFCC plus pitch features. Chapter 2 presented many variants of MLP-based *Tandem*-style acoustic features, demonstrating a set of configurations that performed well for particular tasks. In an extreme case, it was shown that *idealized* Tandem features could potentially solve the entire ASR problem; a more practical demonstration proved that *corrected* Tandem features could at least exploit this same intuition to provide modest improvements.

In the greater context of this entire thesis, however, let us revisit a few observations from that early chapter:

- Table 2.2 showed that varying the representation of inputs to the MLP did not seem to have much effect on accuracy. Using a 5-frame context achieved 10.2% CER; using no delta components achieved 10.3%. These were not considered significant improvements over the Tandem baseline result of 10.6%. It was therefore deemed reasonable to retain the baseline configuration: a 9-frame context with double-deltas.

- Appending pitch features was considered significantly helpful, especially since these experiments were conducted on a tonal language (Mandarin).

- The KLT (i.e. PCA) transform applied as post-processing of Tandem features was demonstrated to be very helpful.

As would later be discussed in Chapter 3, these results could have been further examined with respect to real-time processing. Rather than simply comparing performance in terms of recognition error rate or classification accuracy, we can now consider much richer set of implications:

- The baseline Tandem system's 9-frame input context using double-deltas implies real-time processing latency of 132.5ms. An alternative MLP input using 5 frames of input context and no deltas would only contribute a latency of 32.5ms. Although it is not

clear whether such a system would perform significantly better or worse in terms of accuracy, a difference of 100ms latency certainly could be seen as significant – for example, in an interactive spoken dialog system.

- Computing pitch features may not be feasible for real-time processing, depending on the software implementation (e.g. SAcC) and the computing device. In such a scenario, a system designer may have to select a more efficient pitch tracker (e.g. aubiopitch or Kaldi), or perhaps upgrade the hardware resources. It may be reasonable to forgo pitch altogether for non-tonal languages; although, as discussed in Chapter 3, pitch information can serve many other useful purposes such as proxying speech/nonspeech determinations and estimating VTLN warp factors.

- What is the computational cost of the KLT operation? Is this feasible to perform under real-time CPU load constraints on very low-powered devices? If not, it may be worthwhile for a system designer to consider alternatives such as bottleneck structures, for which the MLP activations can be used directly as acoustic features without requiring any post-processing.

Considerations such as these were brought to our attention in Chapter 3, which also presented a formal model for estimating the real-time CPU load and minimum latency of systems based on a number of parameters such as: frame batch size, hidden layer size, number of hidden layers, computing platform, software variants, and more. Such a perspective is not often addressed in ASR research, which tends to focus more strongly on results as defined in terms of word error rate.

With respect to the current popularity of DNNs, it may thus be worth noting that "deep" can translate as "more latency" in the context of real-time processing. In addition, it may be infeasible to perform hybrid speech recognition in real-time on some some mobile devices, due to the rather large matrix multiplication at the output layer; in these cases, it may be worthwhile to consider more aggressive triphone state clustering to reduce the size of the output layer. By contrast, discriminative acoustic feature extraction (Tandem or bottleneck) could serve as viable alternatives when considering such performance constraints.

Chapter 4 showed that frame batch size has a very different effect when training an MLP, as opposed to performing a forward pass evaluation. Because latency is not a relevant concern in this case, large batch sizes can be exploited to speed up the matrix multiplications. However, such efficiency must be balanced with the reduced number of updates performed in each epoch of error back-propagation training; thus more epochs will be needed before the convergence is achieved.

Similarly, whereas computing the exponential function did not significantly impact feature extraction CPU load or latency, it was shown to have a very different effect when training an ASR system. Because the expectation-maximization algorithms used in HMM-GMM training involve many log-domain probability summations, a specialized set of transcendental function approximations could be used to dramatically reduce acoustic model training time. This is not unrelated to the overall thesis of discriminative acoustic features: HMM-GMM

acoustic models are required for speech-transcript alignment, which is needed for preparing targets used in supervised training of neural networks.

Chapter 4 thus concluded with a literature review of speech-transcript alignment, compared the standard Viterbi approximation with two alternative alignment algorithms, and presented a case study of a deployed system. The alignment service described has been used for several years and has processed an enormous quantity and variety of speech data; along the way, its users have introduced "real-world" problems that were addressed.

Appendix A

Fast type-punned transcendental function approximation

A.1 Implementation in Go

The following code written in the Go language is considered the reference implementation.

exp.go

```
package punmath

import "math"

// Exponentiate via multiply-add, integer conversion, and a type pun.
//
// Reference: Schraudolph, Nicol. "A Fast, Compact Approximation of
// the Exponential Function", Neural Computation, 1999
//
// "FasterExp" functions are based on Schraudolph's EXP macro, except:
//  - use 64-bit integers rather than a struct with two 32-bit ints.
//    -> no staircase effect; now monotonically increasing, continuous
//    -> machine byte order doesn't matter
//    -> reformulated approximation constants for 32 and 64 bit
//  - use a thread-safe function; not a macro with a global variable
//  - overflow/error handling
//  - new approximation constant for unbiased error criterion
//  - implemented in Go (as well as C)
//
// To his credit, Schraudolph notes that his implementation is a bit
// of a hack because 64-bit integer operations were "unacceptably slow
```

```
// on the typical workstation platforms [circa late 1990's]".  He also
// dutifully notes the shortcomings of using a C preprocessor macro.
//
// The basic idea of the algorithm, is to exploit the structure of
// IEEE-754 floating point and realize that multiplying a float is
// similar to left-shifting an int. So first multiply the float
// argument by 2^m (m = mantissa len); then add in the exponent bias
// (e.g. 1023) multiplied by 2^m; then convert it to an integer, and
// return the result as if it were a float (i.e. type punning).  In
// addition to placing the integer portion of the argument into the
// correctly interpreted exponent bits of the result, the spillover
// from the multiplication in the mantissa bis is very convenient: it
// will linearly interpolate values for non-integer arguments.
//
// The approximation error can be improved by subtracting a constant
// to shift the curve.  See Schraudolph's Appendix A.3 for analytical
// derivations of constants that optimize several error criteria; a
// comparison is given in his Table 2.  See constants.go for an
// alternative constant that results in unbiased error.  This is the
// default value for ExpC, and is very similar to the constant that
// minimizes RMS relative error (Schraudolph's recommendation).
//
// A novel "FastExp" function is provided as well.  The idea here is
// to separate out the mantissa bits, interpret it as a floating point
// value in [1,2), and then apply a rational function approximation to
// the desired exponential curve: 2^(x-1).  This is inspired by
// Laurent DeSoras' FastLog, but requires some trickier bit wrangling.
// It can achieve more accuracy with 4th order polynomial
// coefficients determined using a MiniMax (Remez) algorithm.
//
// FasterExp: unbiased linear approximation:
//  RMS rel err:  1.7703%
//  Max rel err:  1.9978%
//  Min rel err: -0.3909%
//
// FastExp: minimax 4th-order polynomial approximation:
//  RMS rel err:  0.192896%
//  Max rel err:  0.253504%
//  Min rel err: -0.285750%
//
// An alternative approach was presented by Robin Green, of Sony, at
// http://www.research.scea.com/gdc2003/fast-math-functions_p2.pdf,
```

```
// but it does not seem to exploit the linearly interpolated spillover
// in the mantissa.  Instead, the technique proposes mitigating the
// staircase effect by subtracting 0.7 and increasing resolution by
// shifting some fractional bits into the integral input power; this
// would appear to be more complicated, and the resulting error graphs
// do not seem to indicate that it is not particularly accurate.
//
// It's worth noting that the best approach might be to use a lookup
// table on the linearly interpolated mantissa bits, similar in spirit
// to the ICSILog by Oriol Vinyals.  However, it can be a bit
// difficult to predict/guarantee cache performance using lookup
// tables.  For modern tasks with concurrent multiprocessing, it may
// be generally desirable to minimize memory access.

// Exponential approximation (4th-order Minimax)
// This is slightly faster for inputs x < ln(2)
func FastExp(x float64) float64 {
bits := uint64((1<<52/math.Ln2)*x + 1023<<52)
if x < math.Ln2 {
if x >= ExpLo {
m := math.Float64frombits(bits | 0x3ff0000000000000)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
//m = (ExpN0 + m*(ExpN1 + m*ExpN2))/(ExpD0 + m*(ExpD1 + m*ExpD2))
return math.Float64frombits((bits | 0x000fffffffffffff) & math.Float64bits(m))
} else if x < ExpLo {
return 0
}
} else {
if x <= ExpHi {
m_bits := (bits & 0x000fffffffffffff) | 0x3ff0000000000000
m := math.Float64frombits(m_bits)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
//m = (ExpN0 + m*(ExpN1 + m*ExpN2))/(ExpD0 + m*(ExpD1 + m*ExpD2))
m_bits = math.Float64bits(m) & 0x000fffffffffffff
return math.Float64frombits((bits & 0xfff0000000000000) | m_bits)
} else if x > ExpHi {
return math.Inf(1)
}
}
return x // NaN
}
```

```
// Exponential approximation (unbiased linear interpolation)
func FasterExp(x float64) float64 {
switch {
case x >= ExpLo+ExpC && x <= ExpHi:
return math.Float64frombits(uint64(ExpA*x + ExpB))
case x < ExpLo+ExpC:
return 0
case x > ExpHi+ExpC:
return math.Inf(1)
default: // NaN
return x
}
}

// Base-2 exponential approximation (4th-order Minimax)
// This is slightly faster for inputs x < 1.0
func FastExp2(x float64) float64 {
bits := uint64((1<<52)*x + 1023<<52)
if x < 1.0 {
if x >= Exp2Lo {
m := math.Float64frombits(bits | 0x3ff0000000000000)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
//m = (ExpN0 + m*(ExpN1 + m*ExpN2))/(ExpD0 + m*(ExpD1 + m*ExpD2))
return math.Float64frombits((bits | 0x000fffffffffffff) & math.Float64bits(m))
} else if x < Exp2Lo {
return 0
}
} else {
if x <= Exp2Hi {
m_bits := (bits & 0x000fffffffffffff) | 0x3ff0000000000000
m := math.Float64frombits(m_bits)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
//m = (ExpN0 + m*(ExpN1 + m*ExpN2))/(ExpD0 + m*(ExpD1 + m*ExpD2))
m_bits = math.Float64bits(m) & 0x000fffffffffffff
return math.Float64frombits((bits & 0xfff0000000000000) | m_bits)
} else if x > Exp2Hi {
return math.Inf(1)
}
}
return x // NaN
}
```

```
// Base-2 exponential approximation (unbiased linear interpolation)
func FasterExp2(x float64) float64 {
switch {
case x >= Exp2Lo+ExpC && x <= Exp2Hi:
return math.Float64frombits(uint64(Exp2A*x + Exp2B))
case x < Exp2Lo+ExpC:
return 0
case x > Exp2Hi+ExpC:
return math.Inf(1)
default: // NaN
return x
}
}

// Exponential approximation (4th-order Minimax)
// This is slightly faster for inputs x < ln(2)
// This version operates on 32-bit floating point representations.
func FastExpf(x float32) float32 {
bits := uint32((1<<23/math.Ln2)*x + 127<<23)
if x < math.Ln2 {
if x >= ExpfLo {
m := math.Float32frombits(bits | 0x3f800000)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
//m = (ExpN0 + m*(ExpN1 + m*ExpN2))/(ExpD0 + m*(ExpD1 + m*ExpD2))
return math.Float32frombits((bits | 0x007fffff) & math.Float32bits(m))
} else if x < ExpfLo {
return 0
}
} else {
if x <= ExpfHi {
m_bits := (bits & 0x007fffff) | 0x3f800000
m := math.Float32frombits(m_bits)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
//m = (ExpN0 + m*(ExpN1 + m*ExpN2))/(ExpD0 + m*(ExpD1 + m*ExpD2))
m_bits = math.Float32bits(m) & 0x007fffff
return math.Float32frombits((bits & 0xff800000) | m_bits)
} else if x > ExpfHi {
return float32(math.Inf(1))
}
}
return x // NaN
}
```

```
// Exponential approximation (unbiased linear interpolation)
// This version operates on 32-bit floating point representations.
func FasterExpf(x float32) float32 {
switch {
case x >= ExpfLo+ExpC && x <= ExpfHi+ExpC:
return math.Float32frombits(uint32(ExpfA*x + ExpfB))
case x < ExpfLo+ExpC:
return 0
case x > ExpfHi+ExpC:
return float32(math.Inf(1))
default: // NaN
return x
}
}

// Base-2 exponential approximation (4th-order Minimax)
// This is slightly faster for inputs x < 1.0
// This version operates on 32-bit floating point representations.
func FastExp2f(x float32) float32 {
bits := uint32((1<<23)*x + 127<<23)
if x < 1.0 {
if x >= Exp2fLo {
m := math.Float32frombits(bits | 0x3f800000)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
//m = (ExpN0 + m*(ExpN1 + m*ExpN2))/(ExpD0 + m*(ExpD1 + m*ExpD2))
return math.Float32frombits((bits | 0x007fffff) & math.Float32bits(m))
} else if x < Exp2fLo {
return 0
}
} else {
if x <= Exp2fHi {
m_bits := (bits & 0x007fffff) | 0x3f800000
m := math.Float32frombits(m_bits)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
//m = (ExpN0 + m*(ExpN1 + m*ExpN2))/(ExpD0 + m*(ExpD1 + m*ExpD2))
m_bits = math.Float32bits(m) & 0x007fffff
return math.Float32frombits((bits & 0xff800000) | m_bits)
} else if x > Exp2fHi {
return float32(math.Inf(1))
}
}
```

```
return x // NaN
}

// Base-2 exponential approximation (unbiased linear interpolation)
// This version operates on 32-bit floating point representations.
func FasterExp2f(x float32) float32 {
switch {
case x >= Exp2fLo+ExpC && x <= Exp2fHi:
return math.Float32frombits(uint32(Exp2fA*x + Exp2fB))
case x < Exp2fLo+ExpC:
return 0
case x > Exp2fHi+ExpC:
return float32(math.Inf(1))
default: // NaN
return x
}
}
```

log.go

```
package punmath

import "math"

// These logarithm approximations use the same trick as the fast
// exponential, but in reverse order: a type pun, a float conversion,
// and a multiply-add.  Consider the floating point's integer
// bit-level representation, and convert that back to a float; right
// shift it by multiplication by a very small number; subtract the
// exponent bias.  That lower bounds the function, so add an offset to
// minimize the error.  Alternatively, a far more accurate and only
// slightly slower approximation adjusts the mantissa with a
// polynomial approximation of 1 + ln(x)/ln(2) over the domain [1,2).
// Credit to Laurent de Soras' FastLog (FOSS on the Web?) which used a
// quadratic approximation, and inspiration from Nicol Schraudolph's
// fast exponential approximation.  See also the ICSILog for a table
// lookup on the mantissa, exploiting L1 cache.

// Logarithm approximation (4th-order Minimax)
func FastLog(x float64) float64 {
bits := math.Float64bits(x)
k := float64(int64(bits)>>52 - 1024) // floor(ln(x)/ln(2)) - 1
```

```
var m float64      // fraction in [1,2)
if x < 2 {
if x >= LogLo {
m = math.Float64frombits(bits | 0x3ff0000000000000)
} else if x >= 0 {
return math.Inf(-1)
} else { // x < 0
return math.NaN()
}
} else {
if x <= LogHi {
m = math.Float64frombits((bits & 0x000fffffffffffff) | 0x3ff0000000000000)
} else if x > LogHi {
return math.Inf(1)
} else { // NaN
return x
}
}
return math.Ln2 * k + LogP0 + m*(LogP1 + m*(LogP2 + m*(LogP3 + m*LogP4)))
// return math.Ln2 * k + (LogN0 + m*(LogN1 + m*LogN2)) / (LogD0 + m*(LogD1 + m*LogD2))
}

// Logarithm approximation (unbiased linear interpolation)
func FasterLog(x float64) float64 {
switch {
case x >= LogLo && x <= LogHi:
return float64(math.Float64bits(x))*LogA + LogB
case x > LogHi:
return math.Inf(1)
case x < LogLo && x >= 0:
return math.Inf(-1)
case x < 0:
return math.NaN()
default: // NaN
return x
}
}

// Base-2 logarithm approximation (4th-order Minimax)
func FastLog2(x float64) float64 {
bits := math.Float64bits(x)
k := float64(int64(bits)>>52 - 1024) // floor(ln(x)/ln(2)) - 1
```

```
var m float64      // fraction in [1,2)
if x < 2 {
if x >= Log2Lo {
m = math.Float64frombits(bits | 0x3ff0000000000000)
} else if x >= 0 {
return math.Inf(-1)
} else { // x < 0
return math.NaN()
}
} else {
if x <= Log2Hi {
m = math.Float64frombits((bits & 0x000fffffffffffff) | 0x3ff0000000000000)
} else if x > Log2Hi {
return math.Inf(1)
} else { // NaN
return x
}
}
return k + Log2P0 + m*(Log2P1 + m*(Log2P2 + m*(Log2P3 + m*Log2P4)))
// return k + (Log2N0 + m*(Log2N1 + m*Log2N2)) / (Log2D0 + m*(Log2D1 + m*Log2D2))
}

// Base-2 logarithm approximation (unbiased linear interpolation)
func FasterLog2(x float64) float64 {
switch {
case x >= Log2Lo && x <= Log2Hi:
return float64(math.Float64bits(x))*Log2A + Log2B
case x > Log2Hi:
return math.Inf(1)
case x < Log2Lo && x >= 0:
return math.Inf(-1)
case x < 0:
return math.NaN()
default: // NaN
return x
}
}

// Logarithm approximation (4th-order Minimax)
// 32-bit version
func FastLogf(x float32) float32 {
bits := math.Float32bits(x)
```

```
k := float32(int32(bits)>>23 - 128) // floor(ln(x)/ln(2)) - 1
var m float32     // fraction in [1,2)
if x < 2 {
if x >= LogfLo {
m = math.Float32frombits(bits | 0x3f800000)
} else if x >= 0 {
return float32(math.Inf(-1))
} else { // x < 0
return float32(math.NaN())
}
} else {
if x <= LogfHi {
m = math.Float32frombits((bits & 0x007fffff) | 0x3f800000)
} else if x > LogfHi {
return float32(math.Inf(1))
} else { // NaN
return x
}
}
return math.Ln2 * k + LogP0 + m*(LogP1 + m*(LogP2 + m*(LogP3 + m*LogP4)))
// return math.Ln2 * k + (LogN0 + m*(LogN1 + m*LogN2)) / (LogD0 + m*(LogD1 + m*LogD2))
}

// Logarithm approximation (unbiased linear interpolation)
// 32-bit version
func FasterLogf(x float32) float32 {
switch {
case x >= LogfLo && x <= LogfHi:
return float32(math.Float32bits(x))*LogfA + LogfB
case x > LogfHi:
return float32(math.Inf(1))
case x < LogfLo && x >= 0:
return float32(math.Inf(-1))
case x < 0:
return float32(math.NaN())
default: // NaN
return x
}
}

// Base-2 logarithm approximation (4th-order Minimax)
// 32-bit version
```

```
func FastLog2f(x float32) float32 {
bits := math.Float32bits(x)
k := float32(int32(bits)>>23 - 128) // floor(ln(x)/ln(2)) - 1
var m float32     // fraction in [1,2)
if x < 2 {
if x >= Log2fLo {
m = math.Float32frombits(bits | 0x3f800000)
} else if x >= 0 {
return float32(math.Inf(-1))
} else { // x < 0
return float32(math.NaN())
}
} else {
if x <= Log2fHi {
m = math.Float32frombits((bits & 0x007fffff) | 0x3f800000)
} else if x > Log2fHi {
return float32(math.Inf(1))
} else { // NaN
return x
}
}
return k + Log2P0 + m*(Log2P1 + m*(Log2P2 + m*(Log2P3 + m*Log2P4)))
// return k + (Log2N0 + m*(Log2N1 + m*Log2N2)) / (Log2D0 + m*(Log2D1 + m*Log2D2))
}

// Base-2 logarithm approximation (unbiased linear interpolation)
// 32-bit version
func FasterLog2f(x float32) float32 {
switch {
case x >= Log2fLo && x <= Log2fHi:
return float32(math.Float32bits(x))*Log2fA + Log2fB
case x > Log2fHi:
return float32(math.Inf(1))
case x < Log2fLo && x >= 0:
return float32(math.Inf(-1))
case x < 0:
return float32(math.NaN())
default: // NaN
return x
}
}
```

logadd.go

```
package punmath

import "math"

// Use a well-known trick to add numbers in log domain, to
// compute Log(x+y) given Log(x) and Log(y):
//
// Log(x+y) = Log(Exp(Log(x)) + Exp(Log(y)))
//          = Log(x) + Log(1 + Exp(Log(y) - Log(x)))
//
// This does just one Log and one Exp, instead of requiring two Exp
// calls.  For numerical precision, it is best to ensure x > y.
//
// Moreover, this creates some nice conditions under which the fast
// log approximation can be applied to an input in the domain [1,2).
// In this case where the range is restricted, the log approximation
// can be simplified further. For example: comparing to the more
// straightforward Log(x) + FasterLog(1 + FasterExp(Log(y) - Log(x))),
// the optimized FasterLogAdd below eliminates four unnecessary
// comparisons, one addition, and one float-to-integer conversion.

// Log-domain addition with polynomial approximations
func FastLogAdd(logx, logy float64) float64 {
var diff, max float64
if logx > logy {
diff, max = logy - logx, logx
} else {
diff, max = logx - logy, logy
}
switch {
case diff >= ExpLo:
bits := uint64((1<<52/math.Ln2)*diff + 1023<<52)
m := math.Float64frombits(bits | 0x3ff0000000000000)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
m = 1 + math.Float64frombits((bits | 0x000fffffffffffff) & math.Float64bits(m))
return max + (- math.Ln2 + LogP0) + m*(LogP1 + m*(LogP2 + m*(LogP3 + m*LogP4)))
case diff < ExpLo:
return max
default: // NaN
return diff
```

```
}
}

// Log-domain addition with linear interpolation
func FasterLogAdd(logx, logy float64) float64 {
var diff, max float64
if logx > logy {
diff, max = logy - logx, logx
} else {
diff, max = logx - logy, logy
}
switch {
case diff >= ExpLo+ExpC:
return max + (math.Ln2*math.Float64frombits(uint64(diff*ExpA+ExpB)) + LogC*math.Ln2)
case diff < ExpLo+ExpC:
return max
default: // NaN
return diff
}
}

// Base-2 log-domain addition with polynomial approximations
func FastLog2Add(logx, logy float64) float64 {
var diff, max float64
if logx > logy {
diff, max = logy - logx, logx
} else {
diff, max = logx - logy, logy
}
switch {
case diff >= Exp2Lo:
bits := uint64((1<<52)*diff + 1023<<52)
m := math.Float64frombits(bits | 0x3ff0000000000000)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
m = 1 + math.Float64frombits((bits | 0x000fffffffffffff) & math.Float64bits(m))
return max + (-1 + Log2P0) + m*(Log2P1 + m*(Log2P2 + m*(Log2P3 + m*Log2P4)))
case diff < Exp2Lo:
return max
default: // NaN
return diff
}
}
```

```
// Base-2 log-domain addition with linear interpolation
func FasterLog2Add(logx, logy float64) float64 {
var diff, max float64
if logx > logy {
diff, max = logy - logx, logx
} else {
diff, max = logx - logy, logy
}
switch {
case diff >= Exp2Lo+ExpC:
return max + (math.Float64frombits(uint64(diff*Exp2A+Exp2B)) + LogC)
case diff < Exp2Lo+ExpC:
return max
default: // NaN
return diff
}
}

// Log-domain addition with polynomial approximations
// 32-bit version
func FastLogfAdd(logx, logy float32) float32 {
var diff, max float32
if logx > logy {
diff, max = logy - logx, logx
} else {
diff, max = logx - logy, logy
}
switch {
case diff >= ExpfLo:
bits := uint32((1<<23/math.Ln2)*diff + 127<<23)
m := math.Float32frombits(bits | 0x3f800000)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
m = 1 + math.Float32frombits((bits | 0x007fffff) & math.Float32bits(m))
return max + (- math.Ln2 + LogP0) + m*(LogP1 + m*(LogP2 + m*(LogP3 + m*LogP4)))
case diff < ExpfLo:
return max
default: // NaN
return diff
}
}
```

```
// Log-domain addition with linear interpolation
func FasterLogfAdd(logx, logy float32) float32 {
var diff, max float32
if logx > logy {
diff, max = logy - logx, logx
} else {
diff, max = logx - logy, logy
}
switch {
case diff >= ExpfLo+ExpC:
return max + (math.Ln2*math.Float32frombits(uint32(diff*ExpfA+ExpfB)) + LogC*math.Ln2)
case diff < ExpfLo+ExpC:
return max
default: // NaN
return diff
}
}

// Base-2 log-domain addition with polynomial approximations
func FastLog2fAdd(logx, logy float32) float32 {
var diff, max float32
if logx > logy {
diff, max = logy - logx, logx
} else {
diff, max = logx - logy, logy
}
switch {
case diff >= Exp2fLo:
bits := uint32((1<<23)*diff + 127<<23)
m := math.Float32frombits(bits | 0x3f800000)
m = ExpP0 + m*(ExpP1 + m*(ExpP2 + m*(ExpP3 + m*ExpP4)))
m = 1 + math.Float32frombits((bits | 0x007fffff) & math.Float32bits(m))
return max + (-1 + Log2P0) + m*(Log2P1 + m*(Log2P2 + m*(Log2P3 + m*Log2P4)))
case diff < Exp2fLo:
return max
default: // NaN
return diff
}
}

// Base-2 log-domain addition with linear interpolation
func FasterLog2fAdd(logx, logy float32) float32 {
```

```
var diff, max float32
if logx > logy {
diff, max = logy - logx, logx
} else {
diff, max = logx - logy, logy
}
switch {
case diff >= Exp2fLo+ExpC:
return max + (math.Float32frombits(uint32(diff*Exp2fA+Exp2fB)) + LogC)
case diff < Exp2fLo+ExpC:
return max
default: // NaN
return diff
}
}
```

constants.go

```
package punmath

import "math"

// Linear interpolation coefficients
// Exported for convenience of writing manually inlined functions
const (
ExpA   = 1 << 52 / math.Ln2
Exp2A  = 1 << 52
ExpfA  = 1 << 23 / math.Ln2
Exp2fA = 1 << 23
ExpB   = 1 << 52 * (1023 - ExpC)
Exp2B  = 1 << 52 * (1023 - ExpC)
ExpfB  = 1 << 23 * (127 - ExpC)
Exp2fB = 1 << 23 * (127 - ExpC)
)

// Various adjustment constants can be used for the linear
// approximation, depending on what error criteria are to be
// satisfied.  I recommend the unbiased approximation.
const ExpC = ExpCUnbiased

// See Schraudolph's Appendix A for analytical derivations of:
const (
```

```
ExpCUpper    = 0
ExpCLower    = 0.086071332055934207
ExpCMean     = 0.045111411 / math.Ln2 // mean *unsigned* rel err
ExpCRms      = 0.057984814725439755
ExpCMinimax  = 0.043677448903601848
)

// Unbiased approximation constant. This was not given by Schraudolph,
// but can be derived by integrating the (signed, unsquared) relative
// error as a function of c, and solving for c when this quantity
// equals zero.  If you forgot how to do calculus, use Mathematica:
//
//  domain = {x, 1, 2};
//  approx = 2 x;
//  target = 2^(x + c);
//  relerr = approx/target - 1;
//  Solve[Integrate[relerr, domain] == 0, c]
//    >> {{c -> -2Log[Log[2]]/Log[2] - 1}}
//
// Some characteristics of the relative error:
//  RMS: 0.17703~ = 1/2 Sqrt[-4 + Log[4]^2 + Log[8]]
//  Mean (unsigned): 0.015268~
//  Bias (mean signed): 0
//  Max: 0.019978~ = (4 Log[2])/E - 1
//  Min: -0.03909~ = 2 Log[2]^2 - 1
//
// This constant is very similar to ExpCRms, which can be derived as:
//
//  ArgMin[Integrate[relerr^2, domain], c]
//    >> (Log[3+4Log[2]] - Log[Log[2]])/Log[2] - 3
//
// The difference of the RMS relative errors (which also have
// closed-form expression) is insignificant: 0.017703~ vs. 0.017700~
const ExpCUnbiased = 0.057532745889795228

// Warp the mantissa fraction with a polynomial function to
// approximate the curve 2^(x-1) over the domain [1,2].  These
// constants were derived using Mathematica's implmentation of the
// Minimax (Remez) algorithm.  Performance seems more predictable
// using a numerator polynomial rather than a rational function with a
// denominator requiring floating point division (e.g. slower in
// gccgo).  A 4th-order approximation is desirable because the error
```

```
// is positive at the low boundary and negative at the high boundary;
// this is required for correctness of the type-punned approximation
// algorithm.  Also, any higher order approximation will be limited by
// rounding artifacts in the case of 32-bit working precision. The
// relative error bound is theoretically +/- 2.5934e-06, which is
// achieved for the 64-bit computations.  The relative error has been
// empirically measured as high as -1.175e-05 for FastExpf and
// +8.031e-06 for FastExp2f; these might be machine-dependent results.
// Although it might be possible to compute the 32-bit approximations
// using 64-bit working precision, the speed/cost of conversion would
// probably not offset the slightly better accuracy.
const (
ExpP0 = 0.50996422309793632713
ExpP1 = 0.31201603085368849693
ExpP2 = 0.16661338255701787737
ExpP3 = -0.0021252110540508578569
ExpP4 = 0.013534167916102130292
ExpN0 = 0.50077448375996435198
ExpN1 = 0.19921856787676976035
ExpN2 = 0.035087041664743134084
ExpD0 = 1.0000000000000000000
ExpD1 = -0.28973051084145248813
ExpD2 = 0.024810285092917136451
)

// These approximations are valid for the inclusive ranges defined
// below.  At these extremes, the result should always be finite and
// error should be reasonably bounded -- i.e. not more than elsewhere.
// Note that the standard math library's exponential functions have a
// somewhat broader domain, extending into the subnormals.
const (
Exp2Lo  = -1022              // below this value, error grows rapidly
Exp2Hi  = 1023.9999999999998 // two values less than 1024
ExpLo   = Exp2Lo * math.Ln2
ExpHi   = Exp2Hi * math.Ln2
Exp2fLo = -126       // below this value, error grows rapidly
Exp2fHi = 127.999985 // two values less than 128
ExpfLo  = Exp2fLo * math.Ln2
ExpfHi  = Exp2fHi * math.Ln2
)

// Constants for linear logarithmic function approximation
```

```
const (
Log2A  = 1.0 / (1 << 52)
Log2B  = -1023 + LogC
LogA   = Log2A * math.Ln2
LogB   = Log2B * math.Ln2
Log2fA = 1.0 / (1 << 23)
Log2fB = -127 + LogC
LogfA  = Log2fA * math.Ln2
LogfB  = Log2fB * math.Ln2
)

// Adjustments constants for linear logarithmic function approximation
const LogC = LogCUnbiased

// Various constants, with error characteristics: (Max neg, Max pos,
// RMS, Mean unsigned, Mean biased).  Note that the error will be
// larger by a factor of 1/ln(2) for the case of approximating the
// base-2 logarithm.  The integrated errors (RMS, mean, bias) are with
// respect to the input (e.g. mantissa fraction from 1 to 2).
const (
LogCLower    = 0                    // (-0.05966, 0.0000, 0.04351, 0.03970, -0.03970)
LogCUpper    = 0.086071332055934207 // (0.0000, 0.05966, 0.02673, 0.1993, 0.1993)
LogCRms      = 1.5 - 1.0/math.Ln2   // (-0.01994, 0.03972, 0.01780, 0.01532, 0.0000)
LogCUnbiased = LogCRms              // nice coincidence
// TODO: LogCMean (unsigned mean)
LogCMinimax  = 0.043035665977967103 // numerically optimized
)

// Valid input ranges for logarithmic function approximations
const (
Log2Lo  = 2.2250738585072014e-308 // 2 ^ -1022
Log2Hi  = math.MaxFloat64
LogLo   = 2.2250738585072014e-308 // Actually works for subnormals, but is very slow to
LogHi   = math.MaxFloat64
Log2fLo = 1.1754943508222875e-38 // 2 ^ -126
Log2fHi = math.MaxFloat32
LogfLo  = 1.1754943508222875e-38
LogfHi  = math.MaxFloat32
)

// Coefficients for unbiased quadratic fit.  Error characteristics:
//  RMS:  0.376675%
```

```
//  Max:  0.495193%
//  Min: -0.557789%
const (
LogQuadA = LogQuadAUnbiased
LogQuadB = LogQuadBUnbiased
LogQuadC = LogQuadCUnbiased
)

// Laurent Desoras recommends these coefficients, which are very close
// to optimal.  He notes that they preserve continuity in the first
// derivative at the boundaries; it is also constrained at the
// boundaries f(1)=2 and f(2)=2.
const (
LogQuadADesoras = -1.0 / 3
LogQuadBDesoras = 2
LogQuadCDesoras = -2.0 / 3
)

// These coefficients are far less elegant, but slightly better in
// terms of minimizing the error criteria.  There don't appear to be
// analytical solutions for the values with less precision below...
const (
LogQuadAUnbiased         = 6/math.Ln2 - 9
LogQuadBUnbiased         = 28 - 18/math.Ln2
LogQuadCUnbiased         = (12 - 18*math.Ln2) / math.Ln2
LogQuadARms              = -0.342671
LogQuadBRms              = 2.02801
LogQuadCRms              = -0.685343
LogQuadARmsUnconstrained = -0.33688
LogQuadBRmsUnconstrained = 1.9949
LogQuadCRmsUnconstrained = -0.648985
)
```

A.2 Implementation in C

The implementation in C is somewhat faster than the Go implementation, likely because the compiler is better optimized.

punmath.h

```
#include <float.h>
```

```
#ifndef __PUNMATH_H__
#define __PUNMATH_H__

/* Constants for optimizing error characteristics */
#define ExpA 6497320848556798.09173979462855473657
#define Exp2A 4503599627370496.0
#define ExpfA 12102203.16156148555081299436f
#define Exp2fA 8388608.0f
#define ExpB 4606923314347066524.775851634471206912
#define Exp2B 4606923314347066524.775851634471206912
#define ExpfB 1064870596.347566896632037376f
#define Exp2fB 1064870596.347566896632037376f
#define ExpC 0.057532745889795228
#define Exp2C 0.057532745889795228
#define ExpfC 0.057532745889795228f
#define Exp2fC 0.057532745889795228f
#define ExpP0 0.50996422309793632713
#define ExpP1 0.31201603085368849693
#define ExpP2 0.16661338255701787737
#define ExpP3 -0.0021252110540508578569
#define ExpP4 0.013534167916102130292
#define ExpN0 0.50077448375996435198
#define ExpN1 0.19921856787676976035
#define ExpN2 0.035087041664743134084
#define ExpD0 1.0000000000000000000
#define ExpD1 -0.28973051084145248813
#define ExpD2 0.024810285092917136451
#define Log2A 2.220446049250313e-16
#define Log2B -1022.9426950408889
#define LogA 1.539095918623324e-16
#define LogB -709.049844941984
#define Log2fA 1.1920928955078125e-07f
#define Log2fB -126.94269504088896f
#define LogfA 8.262958294867817e-08f
#define LogfB -87.98997116027313f
#define LogC 0.05730495911103661
#define Log2P0 -1.5128546239024574368
#define Log2P1 4.0700907918495929955
#define Log2P2 -2.1206751311114850627
#define Log2P3 0.64514236358644357902
#define Log2P4 -0.081615808497907706511
```

```
#define Log2N0 -2.8496767587086202979
#define Log2N1 4.2745474177776235540
#define Log2N2 2.9248383794977732165
#define Log2D0 1.0000000000000000000
#define Log2D1 2.8496982786074985681
#define Log2D2 0.50000000003262791862
#define LogP0 -1.0486309171550647
#define LogP1 2.821171956993541
#define LogP2 -1.4699399880135182
#define LogP3 0.4471786103797225
#define LogP4 -0.05657176754944515
#define LogN0 -1.9752454108060837
#define LogN1 2.962890490802354
#define LogN2 2.0273434763424008
#define LogD0 1.0000000000000000000
#define LogD1 2.8496982786074985681
#define LogD2 0.50000000003262791862

/* Other constants that could have been precomputed by a good compiler */
#define LogAddA 6497320848556798.09164605810288196293 /* 1 << 52 / M_LN2 */
#define LogAddB 4607182418800017408.0 /* 1023 << 52 */
#define LogAddfA 12102203.16156148555063839648 /* 1 << 23 / M_LN2 */
#define LogAddfB 1065353216.0 /* 127 << 23 */
#define LogAdd2A 4503599627370496.0 /* 1 << 52 */
#define LogAdd2fA 8388608.0 /* 1 << 23 */
#define LogAddP0 -1.74177809771501000942 /* LogP0 - M_LN2 */
#define LogAdd2P0 -2.5128546239024574368 /* Log2P0 - 1 */

/* Limits */
#define Exp2Lo -1022.0
#define Exp2Hi 1023.9999999999998
#define ExpLo -708.39641853226410621702
#define ExpHi 709.78271289338385820640
#define Exp2fLo -126.0f
#define Exp2fHi 127.999985f
#define ExpfLo -87.336544755055310898566f
#define ExpfHi 88.722828714465291205306f
#define Log2Lo 2.2250738585072014e-308
#define Log2Hi DBL_MAX
#define LogLo 2.2250738585072014e-308
#define LogHi DBL_MAX
#define Log2fLo 1.1754943508222875e-38f
```

```
#define Log2fHi FLT_MAX
#define LogfLo 1.1754943508222875e-38f
#define LogfHi FLT_MAX

/* Linear interpolation between values of integer arguments, adjusted
 * with an additive constant to optimize error characteristics.
 */
double FastExp2(double x);
float FastExp2f(float x);
double FastExp(double x);
float FastExpf(float x);
double FastLog2(double x);
float FastLog2f(float x);
double FastLog(double x);
float FastLogf(float x);

/* Rational function approximation between values of integer
 * arguments.  This is somewhat slower but far more accurate.
 */
double FasterExp2(double x);
float FasterExp2f(float x);
double FasterExp(double x);
float FasterExpf(float x);
double FasterLog2(double x);
float FasterLog2f(float x);
double FasterLog(double x);
float FasterLogf(float x);

/* Log-domain addition: return log(x+y) given log(x) and log(y). */
double FastLogAdd(double logx, double logy);
float FastLogfAdd(float logx, float logy);
double FastLog2Add(double logx, double logy);
float FastLog2fAdd(float logx, float logy);
double FasterLogAdd(double logx, double logy);
float FasterLogfAdd(float logx, float logy);
double FasterLog2Add(double logx, double logy);
float FasterLog2fAdd(float logx, float logy);

/* For comparison */
double NaiveLogAdd(double logx, double logy);
float NaiveLogfAdd(float logx, float logy);
double NaiveLog2Add(double logx, double logy);
```

```
float NaiveLog2fAdd(float logx, float logy);

#endif
```

punmath.c

```
#include <math.h>
#include "punmath.h"

double FastExp2(double x) {
  union {double d; long long ll;} a, m, y;
  a.ll = Exp2A*x + 4607182418800017408;
  if (x < 1.0) {
    if (x >= Exp2Lo) {
      m.ll = a.ll | 0x3ff0000000000000;
      m.d = ExpP0 + m.d*(ExpP1 + m.d*(ExpP2 + m.d*(ExpP3 + m.d*ExpP4)));
      y.ll = (a.ll | 0x000fffffffffffff) & m.ll;
      return y.d;
    } else if (x < Exp2Lo) {
      return 0.0;
    }
  } else {
    if (x <= Exp2Hi) {
      m.ll = (a.ll & 0x000fffffffffffff) | 0x3ff0000000000000;
      m.d = ExpP0 + m.d*(ExpP1 + m.d*(ExpP2 + m.d*(ExpP3 + m.d*ExpP4)));
      y.ll = (a.ll & 0xfff0000000000000) | (m.ll & 0x000fffffffffffff);
      return y.d;
    } else if (x > Exp2Hi) {
      return INFINITY;
    }
  }
  return x;
}
float FastExp2f(float x) {
  union {float f; int i;} a, m, y;
  a.i = Exp2fA*x + 1065353216;
  if (x < 1.0f) {
    if (x >= Exp2fLo) {
      m.i = a.i | 0x3f800000;
      m.f = ExpP0 + m.f*(ExpP1 + m.f*(ExpP2 + m.f*(ExpP3 + m.f*ExpP4)));
      y.i = (a.i | 0x007fffff) & m.i;
      return y.f;
```

```
    } else if (x < Exp2fLo) {
      return 0.0f;
    }
  } else {
    if (x <= Exp2fHi) {
      m.i = (a.i & 0x007fffff) | 0x3f800000;
      m.f = ExpP0 + m.f*(ExpP1 + m.f*(ExpP2 + m.f*(ExpP3 + m.f*ExpP4)));
      y.i = (a.i & 0xff800000) | (m.i & 0x007fffff);
      return y.f;
    } else if (x > Exp2fHi) {
      return INFINITY;
    }
  }
  return x;
}
double FastExp(double x) {
  union {double d; long long ll;} a, m, y;
  a.ll = ExpA*x + 4607182418800017408;
  if (x < M_LN2) {
    if (x >= ExpLo) {
      m.ll = a.ll | 0x3ff0000000000000;
      m.d = ExpP0 + m.d*(ExpP1 + m.d*(ExpP2 + m.d*(ExpP3 + m.d*ExpP4)));
      y.ll = (a.ll | 0x000fffffffffffff) & m.ll;
      return y.d;
    } else if (x < ExpLo) {
      return 0.0;
    }
  } else {
    if (x <= ExpHi) {
      m.ll = (a.ll & 0x000fffffffffffff) | 0x3ff0000000000000;
      m.d = ExpP0 + m.d*(ExpP1 + m.d*(ExpP2 + m.d*(ExpP3 + m.d*ExpP4)));
      y.ll = (a.ll & 0xfff0000000000000) | (m.ll & 0x000fffffffffffff);
      return y.d;
    } else if (x > ExpHi) {
      return INFINITY;
    }
  }
  return x;
}
float FastExpf(float x) {
  union {float f; int i;} a, m, y;
  a.i = ExpfA*x + 1065353216;
```

```
  if (x < M_LN2) {
    if (x >= ExpfLo) {
      m.i = a.i | 0x3f800000;
      m.f = ExpP0 + m.f*(ExpP1 + m.f*(ExpP2 + m.f*(ExpP3 + m.f*ExpP4)));
      y.i = (a.i | 0x007fffff) & m.i;
      return y.f;
    } else if (x < ExpfLo) {
      return 0.0f;
    }
  } else {
    if (x <= ExpfHi) {
      m.i = (a.i & 0x007fffff) | 0x3f800000;
      m.f = ExpP0 + m.f*(ExpP1 + m.f*(ExpP2 + m.f*(ExpP3 + m.f*ExpP4)));
      y.i = (a.i & 0xff800000) | (m.i & 0x007fffff);
      return y.f;
    } else if (x > ExpfHi) {
      return INFINITY;
    }
  }
  return x;
}
double FastLog2(double x) {
  union {double d; long long ll;} a, m;
  a.d = x;
  double k = (a.ll >> 52) - 1024;
  if (x < 2) {
    if (x >= Log2Lo) {
      m.ll = a.ll | 0x3ff0000000000000;
    } else if (x >= 0) {
      return -INFINITY;
    } else {
      return NAN;
    }
  } else {
    if (x <= Log2Hi) {
      m.ll = (a.ll & 0x000fffffffffffff) | 0x3ff0000000000000;
    } else if (x > Log2Hi) {
      return INFINITY;
    } else {
      return x;
    }
  }
```

```
  return k + Log2P0 + m.d*(Log2P1 + m.d*(Log2P2 + m.d*(Log2P3 + m.d*Log2P4)));
}
float FastLog2f(float x) {
  union {float f; int i;} a, m;
  a.f = x;
  float k = (a.i >> 23) - 128;
  if (x < 2) {
    if (x >= Log2fLo) {
      m.i = a.i | 0x3f800000;
    } else if (x >= 0) {
      return -INFINITY;
    } else {
      return NAN;
    }
  } else {
    if (x <= Log2fHi) {
      m.i = (a.i & 0x007fffff) | 0x3f800000;
    } else if (x > Log2fHi) {
      return INFINITY;
    } else {
      return x;
    }
  }
  return k + Log2P0 + m.f*(Log2P1 + m.f*(Log2P2 + m.f*(Log2P3 + m.f*Log2P4)));
}
double FastLog(double x) {
  union {double d; long long ll;} a, m;
  a.d = x;
  double k = (a.ll >> 52) - 1024;
  if (x < 2) {
    if (x >= LogLo) {
      m.ll = a.ll | 0x3ff0000000000000;
    } else if (x >= 0) {
      return -INFINITY;
    } else {
      return NAN;
    }
  } else {
    if (x <= LogHi) {
      m.ll = (a.ll & 0x000fffffffffffff) | 0x3ff0000000000000;
    } else if (x > LogHi) {
      return INFINITY;
```

```
    } else {
      return x;
    }
  }
  return M_LN2 * k + LogP0 + m.d*(LogP1 + m.d*(LogP2 + m.d*(LogP3 + m.d*LogP4)));
}
float FastLogf(float x) {
  union {float f; int i;} a, m;
  a.f = x;
  float k = (a.i >> 23) - 128;
  if (x < 2) {
    if (x >= LogfLo) {
      m.i = a.i | 0x3f800000;
    } else if (x >= 0) {
      return -INFINITY;
    } else {
      return NAN;
    }
  } else {
    if (x <= LogfHi) {
      m.i = (a.i & 0x007fffff) | 0x3f800000;
    } else if (x > LogfHi) {
      return INFINITY;
    } else {
      return x;
    }
  }
  return M_LN2 * k + LogP0 + m.f*(LogP1 + m.f*(LogP2 + m.f*(LogP3 + m.f*LogP4)));
}

double FasterExp2(double x) {
  if (x >= Exp2Lo+ExpC && x <= Exp2Hi) {
    union {double d; long long ll;} a;
    a.ll = Exp2A * x + Exp2B;
    return a.d;
  } else if (x < Exp2Lo+ExpC) {
    return 0.0;
  } else if (x > Exp2Hi) {
    return INFINITY;
  } else {
    return x;
  }
```

```
}
float FasterExp2f(float x) {
  if (x >= Exp2fLo+ExpfC && x <= Exp2fHi) {
    union {float f; int i;} a;
    a.i = Exp2fA * x + Exp2fB;
    return a.f;
  } else if (x < Exp2fLo+ExpfC) {
    return 0.0f;
  } else if (x > Exp2fHi) {
    return INFINITY;
  } else {
    return x;
  }
}
double FasterExp(double x) {
  if (x >= ExpLo+ExpC && x <= ExpHi) {
    union {double d; long long ll;} a;
    a.ll = ExpA * x + ExpB;
    return a.d;
  } else if (x < ExpLo+ExpC) {
    return 0.0;
  } else if (x > ExpHi) {
    return INFINITY;
  } else {
    return x;
  }
}
float FasterExpf(float x) {
  if (x >= ExpfLo+ExpfC && x <= ExpfHi) {
    union {float f; int i;} a;
    a.i = ExpfA * x + ExpfB;
    return a.f;
  } else if (x < ExpfLo+ExpfC) {
    return 0.0f;
  } else if (x > ExpfHi) {
    return INFINITY;
  } else {
    return x;
  }
}
double FasterLog2(double x) {
  if (x >= Log2Lo && x <= Log2Hi) {
```

```
    union {double d; long long ll;} a;
    a.d = x;
    return a.ll * Log2A + Log2B;
  } else if (x > Log2Hi) {
    return INFINITY;
  } else if (x < Log2Lo && x >= 0) {
    return -INFINITY;
  } else if (x < 0) {
    return NAN;
  } else {
    return x;
  }
}
float FasterLog2f(float x) {
  if (x >= Log2fLo && x <= Log2fHi) {
    union {float f; int i;} a;
    a.f = x;
    return a.i * Log2fA + Log2fB;
  } else if (x > Log2fHi) {
    return INFINITY;
  } else if (x < Log2fLo && x >= 0) {
    return -INFINITY;
  } else if (x < 0) {
    return NAN;
  } else {
    return x;
  }
}
double FasterLog(double x) {
  if (x >= LogLo && x <= LogHi) {
    union {double d; long long ll;} a;
    a.d = x;
    return a.ll * LogA + LogB;
  } else if (x > LogHi) {
    return INFINITY;
  } else if (x < LogLo && x >= 0) {
    return -INFINITY;
  } else if (x < 0) {
    return NAN;
  } else {
    return x;
  }
```

```
}
float FasterLogf(float x) {
  if (x >= LogfLo && x <= LogfHi) {
    union {float f; int i;} a;
    a.f = x;
    return a.i * LogfA + LogfB;
  } else if (x > LogfHi) {
    return INFINITY;
  } else if (x < LogfLo && x >= 0) {
    return -INFINITY;
  } else if (x < 0) {
    return NAN;
  } else {
    return x;
  }
}

double FastLogAdd(double logx, double logy) {
  double diff, max;
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  if (diff >= ExpLo) {
    union {double d; long long ll;} a, m;
    a.ll = LogAddA*diff + LogAddB;
    m.ll = a.ll | 0x3ff0000000000000;
    m.d = ExpP0 + m.d*(ExpP1 + m.d*(ExpP2 + m.d*(ExpP3 + m.d*ExpP4)));
    m.ll &= a.ll | 0x000fffffffffffff;
    m.d += 1.0;
    return max + LogAddP0 + m.d*(LogP1 + m.d*(LogP2 + m.d*(LogP3 + m.d*LogP4)));
  } else if (diff < ExpLo) {
    return max;
  } else {
    return diff;
  }
}
float FastLogfAdd(float logx, float logy) {
  float diff, max;
```

```
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  if (diff >= ExpfLo) {
    union {float f; int i;} a, m;
    a.i = LogAddfA*diff + LogAddfB;
    m.i = a.i | 0x3f800000;
    m.f = ExpP0 + m.f*(ExpP1 + m.f*(ExpP2 + m.f*(ExpP3 + m.f*ExpP4)));
    m.i &= a.i | 0x007fffff;
    m.f += 1.0;
    return max + LogAddP0 + m.f*(LogP1 + m.f*(LogP2 + m.f*(LogP3 + m.f*LogP4)));
  } else if (diff < ExpfLo) {
    return max;
  } else {
    return diff;
  }
}
double FastLog2Add(double logx, double logy) {
  double diff, max;
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  if (diff >= Exp2Lo) {
    union {double d; long long ll;} a, m;
    a.ll = LogAdd2A*diff + LogAddB;
    m.ll = a.ll | 0x3ff0000000000000;
    m.d = ExpP0 + m.d*(ExpP1 + m.d*(ExpP2 + m.d*(ExpP3 + m.d*ExpP4)));
    m.ll &= a.ll | 0x000fffffffffffff;
    m.d += 1.0;
    return max + LogAdd2P0 + m.d*(Log2P1 + m.d*(Log2P2 + m.d*(Log2P3 + m.d*Log2P4)));
  } else if (diff < Exp2Lo) {
    return max;
  } else {
    return diff;
```

```
  }
}
float FastLog2fAdd(float logx, float logy) {
  float diff, max;
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  if (diff >= Exp2fLo) {
    union {float f; int i;} a, m;
    a.i = LogAdd2fA*diff + LogAddfB;
    m.i = a.i | 0x3f800000;
    m.f = ExpP0 + m.f*(ExpP1 + m.f*(ExpP2 + m.f*(ExpP3 + m.f*ExpP4)));
    m.i &= a.i | 0x007fffff;
    m.f += 1.0;
    return max + LogAdd2P0 + m.f*(Log2P1 + m.f*(Log2P2 + m.f*(Log2P3 + m.f*Log2P4)));
  } else if (diff < Exp2fLo) {
    return max;
  } else {
    return diff;
  }
}
double FasterLogAdd(double logx, double logy) {
  double diff, max;
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  if (diff >= ExpLo+ExpC) {
    union {double d; long long ll;} a;
    a.ll = diff*ExpA + ExpB;
    return max + (M_LN2 * a.d + LogC*M_LN2);
  } else if (diff < ExpLo+ExpC) {
    return max;
  } else {
    return diff;
```

```
  }
}
float FasterLogfAdd(float logx, float logy) {
  float diff, max;
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  if (diff >= ExpfLo+ExpC) {
    union {float f; int i;} a;
    a.i = diff*ExpfA + ExpfB;
    return max + (M_LN2 * a.f + LogC*M_LN2);
  } else if (diff < ExpfLo+ExpC) {
    return max;
  } else {
    return diff;
  }
}
double FasterLog2Add(double logx, double logy) {
  double diff, max;
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  if (diff >= Exp2Lo+ExpC) {
    union {double d; long long ll;} a;
    a.ll = diff*Exp2A + Exp2B;
    return max + a.d + LogC;
  } else if (diff < Exp2Lo+ExpC) {
    return max;
  } else {
    return diff;
  }
}
float FasterLog2fAdd(float logx, float logy) {
  float diff, max;
```

```
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  if (diff >= Exp2fLo+ExpC) {
    union {float f; int i;} a;
    a.i = diff*Exp2fA + Exp2fB;
    return max + a.f + LogC;
  } else if (diff < Exp2fLo+ExpC) {
    return max;
  } else {
    return diff;
  }
}

double NaiveLogAdd(double logx, double logy) {
  double diff, max;
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  //return max + FastLog(1.0 + FastExp(diff));
  return max + log(1.0 + exp(diff));
  //return log(exp(logx) + exp(logy));
}
float NaiveLogfAdd(float logx, float logy) {
  float diff, max;
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  return max + logf(1.0 + expf(diff));
}
```

```
double NaiveLog2Add(double logx, double logy) {
  double diff, max;
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  return max + log2(1.0 + exp2(diff));
}
float NaiveLog2fAdd(float logx, float logy) {
  float diff, max;
  if (logx > logy) {
    diff = logy - logx;
    max = logx;
  } else {
    diff = logx - logy;
    max = logy;
  }
  return max + log2f(1.0 + exp2f(diff));
}
```

A.3 Benchmark

Below are the timing results from running one billion computations on various devices. This used the C-language implementation. The summations are included to prevent compiler over-optimization and to verify the correctness of results.

test.c

```
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <punmath.h>

#define arg(i, n) (2.0 * (i + 1.0) / (N + 1.0))
#define argf(i, n) (2.0f * (i + 1.0f) / (N + 1.0f))

#define timing(a) start=clock(); sum = 0.0; for (i = 0; i < N; i++) sum += a(arg(i,N));
#define timingf(a) start=clock(); sumf = 0.0f; for (i = 0; i < N; i++) sumf += a(argf(i,
```

```
#define timingla(a) start=clock(); sum = 0.0; for (i = 0; i < N; i++) sum += a(arg(i,N),
#define timinglaf(a) start=clock(); sumf = 0.0f; for (i = 0; i < N; i++) sumf += a(argf(

double Baseline(double x) { return x; }
float Baselinef(float x) { return x; }
double Baselinela(double logx, double logy) { return logx + logy; }
float Baselinelaf(float logx, double logy) { return logx + logy; }

int main() {
  int i, N, round;
  int baseline, baselinef, msec;
  clock_t start, diff;
  double sum;
  float sumf;

  for (int round = 0; round < 2; round++) {
    if (round == 0) {
      printf("Burn-in round...\n");
      N = 1000000;
    } else {
      N = 10000000;
      printf("Testing round %d: %d trials\n", round, N);
    }

    /* Set baselines */
    baseline = 0; timing(Baseline); baseline = msec;
    baselinef = 0; timingf(Baselinef); baselinef = msec;

    timing(FasterExp);
    timing(FasterExp2);
    timingf(FasterExpf);
    timingf(FasterExp2f);
    timing(FasterLog);
    timing(FasterLog2);
    timingf(FasterLogf);
    timingf(FasterLog2f);

    timing(FastExp);
    timing(FastExp2);
    timingf(FastExpf);
    timingf(FastExp2f);
    timing(FastLog);
```

```
    timing(FastLog2);
    timingf(FastLogf);
    timingf(FastLog2f);

    timing(exp);
    timing(exp2);
    timingf(expf);
    timingf(exp2f);
    timing(log);
    timing(log2);
    timingf(logf);
    timingf(log2f);

    /* Reset baselines */
    baseline = 0; timingla(Baselinela); baseline = msec;
    baselinef = 0; timinglaf(Baselinelaf); baselinef = msec;

    timingla(FastLogAdd);
    timingla(FastLog2Add);
    timinglaf(FastLogfAdd);
    timinglaf(FastLog2fAdd);
    timingla(FasterLogAdd);
    timingla(FasterLog2Add);
    timinglaf(FasterLogfAdd);
    timinglaf(FasterLog2fAdd);
    timingla(NaiveLogAdd);
    timingla(NaiveLog2Add);
    timinglaf(NaiveLogfAdd);
    timinglaf(NaiveLog2fAdd);
  }
  return(0);
}
```

Timing results

```
tetra:
Baseline: 15.59 (sum = 9.995e+08)
Baselinef: 12.73 (sum = 3.35544e+07)
FasterExp: 2.50 (sum = 2.81598e+09)
FasterExpf: 2.66 (sum = 1.34218e+08)
FasterLog: 2.11 (sum = -4.74425e+07)
FasterLogf: 2.27 (sum = -8.38851e+06)
```

```
FastExp: 20.45 (sum = 2.83155e+09)
FastExpf: 23.67 (sum = 1.34218e+08)
FastLog: 12.51 (sum = -4.57729e+07)
FastLogf: 15.75 (sum = -8.3885e+06)
exp: 22.96 (sum = 2.83155e+09)
expf: 21.65 (sum = 1.34218e+08)
log: 38.00 (sum = -4.57782e+07)
logf: 28.73 (sum = -8.3885e+06)
Baselinela: 28.09 (sum = 1e+09)
Baselinelaf: 26.56 (sum = 1.67772e+07)
FasterLogAdd: 5.00 (sum = 1.33942e+09)
FasterLogfAdd: 4.25 (sum = 3.35544e+07)
FastLogAdd: 31.53 (sum = 1.34552e+09)
FastLogfAdd: 34.83 (sum = 3.35544e+07)
NativeLogAdd: 69.48 (sum = 1.34551e+09)
NativeLogfAdd: 50.33 (sum = 3.35544e+07)
FasterSigmoid: 8.12
FasterSigmoidf: 7.64
FastSigmoid: 14.96
FastSigmoidf: 16.03
NativeSigmoid: 30.06
NativeSigmoidf: 32.03

c3-class:
Baseline: 13.11 (sum = 9.995e+08)
Baselinef: 10.71 (sum = 3.35544e+07)
FasterExp: 2.10 (sum = 2.81598e+09)
FasterExpf: 2.24 (sum = 1.34218e+08)
FasterLog: 1.78 (sum = -4.74425e+07)
FasterLogf: 1.91 (sum = -8.38851e+06)
FastExp: 17.18 (sum = 2.83155e+09)
FastExpf: 19.88 (sum = 1.34218e+08)
FastLog: 10.49 (sum = -4.57729e+07)
FastLogf: 13.23 (sum = -8.3885e+06)
exp: 19.33 (sum = 2.83155e+09)
expf: 18.21 (sum = 1.34218e+08)
log: 32.02 (sum = -4.57782e+07)
logf: 24.15 (sum = -8.3885e+06)
Baselinela: 23.60 (sum = 1e+09)
Baselinelaf: 22.33 (sum = 1.67772e+07)
FasterLogAdd: 4.21 (sum = 1.33942e+09)
FasterLogfAdd: 3.57 (sum = 3.35544e+07)
```

```
FastLogAdd: 26.52 (sum = 1.34552e+09)
FastLogfAdd: 29.28 (sum = 3.35544e+07)
NativeLogAdd: 58.67 (sum = 1.34551e+09)
NativeLogfAdd: 42.31 (sum = 3.35544e+07)
FasterSigmoid: 7.48
FasterSigmoidf: 6.44
FastSigmoid: 13.72
FastSigmoidf: 13.65
NativeSigmoid: 25.30
NativeSigmoidf: 25.69

MacBook Air:
Baseline: 14.87 (sum = 9.995e+08)
Baselinef: 11.51 (sum = 3.35544e+07)
FasterExp: 3.27 (sum = 2.81598e+09)
FasterExpf: 2.53 (sum = 1.34218e+08)
FasterLog: 3.25 (sum = -4.74425e+07)
FasterLogf: 3.02 (sum = -8.38851e+06)
FastExp: 18.86 (sum = 2.83155e+09)
FastExpf: 21.08 (sum = 1.34218e+08)
FastLog: 14.12 (sum = -4.57729e+07)
FastLogf: 18.15 (sum = -8.3885e+06)
exp: 24.66 (sum = 2.83155e+09)
expf: 20.27 (sum = 1.34218e+08)
log: 22.42 (sum = -4.57782e+07)
logf: 19.89 (sum = -8.3885e+06)
Baselinela: 29.74 (sum = 1e+09)
Baselinelaf: 22.66 (sum = 1.67772e+07)
FasterLogAdd: 7.37 (sum = 1.33942e+09)
FasterLogfAdd: 12.24 (sum = 3.35544e+07)
FastLogAdd: 32.73 (sum = 1.34552e+09)
FastLogfAdd: 38.91 (sum = 3.35544e+07)
NativeLogAdd: 50.89 (sum = 1.34551e+09)
NativeLogfAdd: 45.03 (sum = 3.35544e+07)
FasterSigmoid: 8.40
FasterSigmoidf: 8.23
FastSigmoid: 13.44
FastSigmoidf: 15.43
NativeSigmoid: 16.28
NativeSigmoidf: 17.28

Odroid XU3:
```

```
Baseline: 19.08 (sum = 9.995e+08)
Baselinef: 16.94 (sum = 3.35544e+07)
FasterExp: 28.45 (sum = 2.81598e+09)
FasterExpf: 2.93 (sum = 1.34218e+08)
FasterLog: 32.16 (sum = -4.74425e+07)
FasterLogf: 2.47 (sum = -8.38851e+06)
FastExp: 44.38 (sum = 2.83155e+09)
FastExpf: 23.06 (sum = 1.34218e+08)
FastLog: 42.97 (sum = -4.57729e+07)
FastLogf: 25.30 (sum = -8.3885e+06)
exp: 220.93 (sum = 2.83155e+09)
expf: 177.80 (sum = 1.34218e+08)
log: 98.44 (sum = -4.57782e+07)
logf: 55.10 (sum = -8.3885e+06)
Baselinela: 30.18 (sum = 1e+09)
Baselinelaf: 26.17 (sum = 1.67772e+07)
FasterLogAdd: 45.13 (sum = 1.33942e+09)
FasterLogfAdd: 24.60 (sum = 3.35544e+07)
FastLogAdd: 74.35 (sum = 1.34552e+09)
FastLogfAdd: 63.92 (sum = 3.35544e+07)
NativeLogAdd: 320.69 (sum = 1.34551e+09)
NativeLogfAdd: 232.04 (sum = 3.35544e+07)
FasterSigmoid: 30.20
FasterSigmoidf: 13.23
FastSigmoid: 53.94
FastSigmoidf: 32.47
NativeSigmoid: 223.96
NativeSigmoidf: 223.46

Odroid U3:
Baseline: 47.08 (sum = 9.995e+08)
Baselinef: 41.19 (sum = 3.35544e+07)
FasterExp: 31.23 (sum = 2.81598e+09)
FasterExpf: 1.80 (sum = 1.34218e+08)
FasterLog: 14.19 (sum = -4.74425e+07)
FasterLogf: -2.33 (sum = -8.38851e+06)
FastExp: 63.04 (sum = 2.83155e+09)
FastExpf: 37.59 (sum = 1.34218e+08)
FastLog: 38.29 (sum = -4.57729e+07)
FastLogf: 25.93 (sum = -8.3885e+06)
exp: 128.38 (sum = 2.83155e+09)
expf: 81.50 (sum = 1.34218e+08)
```

```
log: 138.70 (sum = -4.57782e+07)
logf: 54.37 (sum = -8.3885e+06)
Baselinela: 65.94 (sum = 1e+09)
Baselinelaf: 63.59 (sum = 1.67772e+07)
FasterLogAdd: 38.86 (sum = 1.33942e+09)
FasterLogfAdd: 14.71 (sum = 3.35544e+07)
FastLogAdd: 83.60 (sum = 1.34552e+09)
FastLogfAdd: 78.88 (sum = 3.35544e+07)
NativeLogAdd: 290.47 (sum = 1.34551e+09)
NativeLogfAdd: 165.34 (sum = 3.35544e+07)
FasterSigmoid: 70.07
FasterSigmoidf: 38.86
FastSigmoid: 101.53
FastSigmoidf: 70.65
NativeSigmoid: 168.25
NativeSigmoidf: 175.90

Raspberry Pi:
Baseline: 198.5 (sum = 9.995e+07)
Baselinef: 174.9 (sum = 3.35544e+07)
FasterExp: 205.6 (sum = 2.81598e+08)
FasterExpf: 62.7 (sum = 1.12202e+08)
FasterLog: 123.7 (sum = -4.74425e+06)
FasterLogf: 52.7 (sum = -4.95048e+06)
FastExp: 307.3 (sum = 2.83155e+08)
FastExpf: 172.9 (sum = 1.34218e+08)
FastLog: 201.2 (sum = -4.57729e+06)
FastLogf: 136.2 (sum = -4.54458e+06)
exp: 492.1 (sum = 2.83155e+08)
expf: 586.9 (sum = 1.34218e+08)
log: 924.7 (sum = -4.57782e+06)
logf: 337.5 (sum = -4.54992e+06)
Baselinela: 324.5 (sum = 1e+08)
Baselinelaf: 315.9 (sum = 1.67772e+07)
FasterLogAdd: 245.0 (sum = 1.33942e+08)
FasterLogfAdd: 85.4 (sum = 3.35544e+07)
FastLogAdd: 388.7 (sum = 1.34552e+08)
FastLogfAdd: 244.7 (sum = 3.35544e+07)
NativeLogAdd: 1705.4 (sum = 1.34551e+08)
NativeLogfAdd: 1144.7 (sum = 3.35544e+07)
FasterSigmoid: 374.3
FasterSigmoidf: 213.6
```

```
FastSigmoid: 463.7
FastSigmoidf: 320.3
NativeSigmoid: 646.8
NativeSigmoidf: 661.6
```

Bibliography

[1] P. Woodland and D. Povey, "Large scale MMIE training for conversational telephone speech recognition," *Proc. Speech Transcription Workshop*, 2000.

[2] D. Povey and P. Woodland, "Minimum phone error and I-smoothing for improved discriminative training," *Acoustics, Speech, and Signal Processing, 2002. Proceedings.(ICASSP'02). IEEE International Conference on*, vol. 1, 2002.

[3] J. Zheng and A. Stolcke, "Improved discriminative training using phone lattices," *EUROSPEECH*, 2005.

[4] D. Povey, B. Kingsbury, L. Mangu, G. Saon, H. Soltau, and G. Zweig, "fMPE: Discriminatively Trained Features for Speech Recognition," *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, vol. 1, 2005.

[5] B. Zhang and S. Matsoukas, "Minimum Phoneme Error Based Heteroscedastic Linear Discriminant Analysis for Speech Recognition," *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, vol. 1, 2005.

[6] D. Ellis, R. Singh, and S. Sivadas, "Tandem acoustic modeling in large-vocabulary recognition," *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, vol. 1, 2001.

[7] H. Hermansky and S. Sharma, "Temporal patterns (TRAPs) in ASR of noisy speech," *Acoustics, Speech, and Signal Processing, 1999. ICASSP'99. Proceedings., 1999 IEEE International Conference on*, vol. 1, 1999.

[8] Q. Zhu, A. Stolcke, B. Chen, and N. Morgan, "Using MLP features in SRI's conversational speech recognition system," *Proc. Interspeech, Lisbon*, 2005.

[9] J. Zheng, O. Cetin, M.-Y. Hwang, X. Lei, A. Stolcke, and N. Morgan, "Combining Discriminative Feature, Transform, and Model Training for Large Vocabulary Speech Recognition," *Acoustics, Speech, and Signal Processing, 2007. Proceedings.(ICASSP'01). 2007 IEEE International Conference on*, vol. 1, 2007.

[10] H. Bourlard and N. Morgan, *Connectionist speech recognition: a hybrid approach*. Kluwer Academic Publishers Boston, 1994.

[11] A. Robinson, G. Cook, D. Ellis, E. Fosler-Lussier, S. Renals, and D. Williams, "Connectionist speech recognition of broadcast news," *Speech Communication*, vol. 37, no. 1-2, pp. 27–45, 2002.

[12] M. Gales, D. Kim, P. Woodland, H. Chan, D. Mrva, R. Sinha, and S. Tranter, "Progress in the CU-HTK Broadcast News Transcription System," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, 2006.

[13] D. Pallett, J. Fiscuss, J. Garofolo, A. Martin, and M. Przybocki, "1998 Broadcast News benchmark test results: English and non-English word error rate performance measures," *Proc. DARPA Broadcast News Workshop*, pp. 5–12, 1999.

[14] D. Pearce, "Aurora Project: Experimental framework for the performance evaluation of distributed speech recognition front-ends," *ISCA ITRW ASR2000*, 2000.

[15] H. Hermansky, D. Ellis, and S. Sharma, "Tandem connectionist feature extraction for conventional HMMsystems," *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, vol. 3, 2000.

[16] S. Sharma, D. Ellis, S. Kajarekar, P. Jain, and H. Hermansky, "Feature extraction using non-linear transformation for robustspeech recognition on the Aurora database," *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, vol. 2, 2000.

[17] F. Grezl, M. Karafiat, K. Stanislav, and J. Cernocky, "Probabilistic and Bottle-neck Features for LVCSR of Meetings," *Acoustics, Speech, and Signal Processing, 2007. Proceedings.(ICASSP'01). 2007 IEEE International Conference on*, vol. 1, 2007.

[18] S. Sivadas and H. Hermansky, "Generalized tandem feature extraction," *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, vol. 1, 2003.

[19] V. Fontaine, C. Ris, and J. Boite, "Nonlinear discriminant analysis for improved speech recognition," *Proc. Eurospeech*, vol. 97, pp. 2071–2074, 1997.

[20] G. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[21] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks.," in *Interspeech*, 2011, pp. 437–440.

[22] D. Yu and M. L. Seltzer, "Improved bottleneck features using pretrained deep neural networks.," in *INTERSPEECH*, 2011, pp. 237–240.

[23] T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Auto-encoder bottleneck features using deep belief networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, IEEE, 2012, pp. 4153–4156.

[24] K. Vesely, M. Karafiát, and F. Grezl, "Convolutive bottleneck network features for lvcsr," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, IEEE, 2011, pp. 42–47.

[25] J. Gehring, Y. Miao, F. Metze, and A. Waibel, "Extracting deep bottleneck features using stacked auto-encoders," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, IEEE, 2013, pp. 3377–3381.

[26] S. Wegmann, D. McAllaster, J. Orloff, B. Peskin, D. Inc, and M. Newton, "Speaker normalization on conversational telephone speech," *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, vol. 1, 1996.

[27] H. Hermansky, "Perceptual linear predictive (PLP) analysis of speech," *The Journal of the Acoustical Society of America*, vol. 87, p. 1738, 1990.

[28] B. Chen, S. Chang, and S. Sivadas, "Learning discriminative temporal patterns in speech: Development of novel TRAPS-like classifiers," *Proc. EUROSPEECH*, 2001.

[29] B. Chen, Q. Zhu, and N. Morgan, "Tonotopic Multi-Layered Perceptron: A Neural Network for Learning Long-Term Temporal Features for Speech Recognition," *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, vol. 1, 2005.

[30] Q. Zhu, B. Chen, F. Grezl, and N. Morgan, "Improved MLP Structures for Data-Driven Feature Extraction for ASR," *Proceedings of European Conference on Speech Communication and Technology*, 2005.

[31] P. Schwarz, P. Matejka, and J. Cernocky, "Hierarchical Structures of Neural Networks for Phoneme Recognition," *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 1, 2006.

[32] S. Sivadas and H. Hermansky, "Hierarchical tandem feature extraction," *Acoustics, Speech, and Signal Processing, 2002. Proceedings.(ICASSP'02). IEEE International Conference on*, vol. 1, 2002.

[33] K. Livescu, O. Çetin, M. Hasegawa-Johnson, S. King, C. Bartels, N. Borges, A. Kantor, P. Lal, L. Yung, A. Bezman, *et al.*, "Articulatory Feature-based Methods for Acoustic and Audio-visual Speech Recognition: Summary from the 2006 JHU Summer Workshop," *ICASSP 2007*, 2007.

[34] A. Stolcke, F. Grezl, M. Hwang, X. Lei, N. Morgan, and D. Vergyri, "Cross-domain and Cross-language Portability of Acoustic Features Estimated by Multilayer Perceptrons," *ICASSP 2006*, 2006.

[35] J. Frankel, O. Cetin, and N. Morgan, "Transfer Learning for MLP-derived feature transforms," *NOLISP: ISCA Tutorial and Workshop on NOnLInear Speech Processing*, 2007.

[36] H. Misra, H. Bourlard, and V. Tyagi, "New entropy based combination rules in HMM/ANN multi-stream ASR," *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, vol. 2, 2003.

[37] F. Valente and H. Hermansky, "Combination of Acoustic Classifiers based on Dempster-Shafer Theory of evidence," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 2007.

[38] J. Hennebert, C. Ris, H. Bourlard, S. Renals, and N. Morgan, "Estimation of global posteriors and forward-backward training of hybrid hmm/ann systems.," 1997.

[39] H. Bourlard, S. Bengio, M. Doss, Q. Zhu, B. Mesot, and N. Morgan, "Towards using hierarchical posteriors for flexible automatic speech recognition systems," *Proc. of the DARPA EARS RT04 Workshop*, 2004.

[40] X. Lei, M. Siu, M.-Y. Hwang, M. Ostendorf, and T. Lee, "Improved Tone Modeling for Mandarin Broadcast News Speech Recognition," *Interspeech*, 2006.

[41] M. Hwang, X. Lei, W. Wang, and T. Shinozaki, "Investigation on Mandarin Broadcast News Speech Recognition," *ICSLP*, 2006.

[42] S. Young, J. Odell, and P. Woodland, "Tree-based state tying for high accuracy acoustic modelling," *Proceedings of the workshop on Human Language Technology*, pp. 307–312, 1994.

[43] D. Ellis and N. Morgan, "Size matters: an empirical study of neural network training forlarge vocabulary continuous speech recognition," *Acoustics, Speech, and Signal Processing, 1999. ICASSP'99. Proceedings., 1999 IEEE International Conference on*, vol. 2, 1999.

[44] L. Gillick, S. Cox, D. Inc, and M. Newton, "Some statistical issues in the comparison of speech recognition algorithms," *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on*, pp. 532–535, 1989.

[45] D. Pallet, W. Fisher, and J. Fiscus, "Tools for the analysis of benchmark speech recognition tests," *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*, pp. 97–100, 1990.

[46] X. Lei, M. Hwang, and M. Ostendorf, "Incorporating Tone-related MLP Posteriors in the Feature Representation for Mandarin ASR," *Proc. Interspeech*, pp. 2981–2984, 2005.

[47] D. Whalen, A. G. Levitt, P.-L. Hsiao, and I. Smorodinsky, "Intrinsic f0 of vowels in the babbling of 6-, 9-, and 12-month-old french-and english-learning infants," *The Journal of the Acoustical Society of America*, vol. 97, no. 4, pp. 2533–2539, 1995.

[48] F. Metze and A. Waibel, "A flexible stream architecture for ASR using articulatory features," *Proc. ICSLP*, 2002.

[49] D. Ellis and M. Gomez, "Investigations into tandem acoustic modeling for the Aurora task," *Proc. Eurospeech-2001, Special Event on Noise Robust Recognition*, 2001.

[50] P. SCHWARZ, P. MATEJKA, and J. CERNOCKY, "Towards lower error rates in phoneme recognition," *Lecture notes in computer science*, pp. 465–472, 2004.

[51] M. Reyes-Gomez and D. Ellis, "Error visualization for tandem acoustic modeling on the Aurora task," *ICASSP 2002*, 2002.

[52] A. Venkataraman, A. Stolcke, W. Wang, D. Vergyri, V. Gadde, and J. Zheng, "An Efficient Repair Procedure For Quick Transcriptions," *Proc. ICSLP*, pp. 2002–2005, 2004.

[53] G. Peng, M.-Y. Hwang, and M. Ostendorf, "Automatic acoustic segmentation for speech recognition on broadcast recordings," *Proc. Interspeech*, 2007.

[54] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, 1988.

[55] R. F. Boisvert, J. Moreira, M. Philippsen, and R. Pozo, "Java and numerical computing," *Computing in Science & Engineering*, vol. 3, no. 2, pp. 18–24, 2001.

[56] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, "A set of level 3 basic linear algebra subprograms," *ACM Transactions on Mathematical Software (TOMS)*, vol. 16, no. 1, pp. 1–17, 1990.

[57] R. C. Whaley and J. J. Dongarra, "Automatically tuned linear algebra software," in *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, IEEE Computer Society, 1998, pp. 1–27.

[58] Z. Xianyi, W. Qian, and Z. Chothia, "Openblas," *URL: http://xianyi. github. io/OpenBLAS*, 2012.

[59] K. Goto, "Gotoblas," *Texas Advanced Computing Center, University of Texas at Austin, USA. URL¡ http://www. otc. utexas. edu/ATdisplay. jsp*, 2007.

[60] C. Nvidia, "Compute unified device architecture programming guide," 2007.

[61] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra, "From cuda to opencl: towards a performance-portable solution for multi-platform gpu programming," *Parallel Computing*, vol. 38, no. 8, pp. 391–407, 2012.

[62] *Cublas-xt: accelerate blas calls with multiple gpus*, `https://developer.nvidia.com/cublasxt`, Accessed: 2014-10-31.

[63] D. P. Ellis and B. S. Lee, "Noise robust pitch tracking by subband autocorrelation classification," in *13th Annual Conference of the International Speech Communication Association*, 2012.

[64] *Aubio, a tool for audio labeling*, `http://aubio.org/`, Accessed: 2014-11-10.

[65] P. Brossier, "Automatic annotation of musical audio for interactive systems," PhD thesis, Queen Mary University of London, 2006.

[66] 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.

[67] P. Ghahremani, B. BabaAli, D. Povey, K. Riedhammer, J. Trmal, and S. Khudanpur, "A pitch extraction algorithm tuned for automatic speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, IEEE, 2014.

[68] *KALDI: 'Kaldi'*, `http://kaldi.sourceforge.net/`, Accessed: 2013-07-12.

[69] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, no. 4, pp. 357–366, 1980.

[70] X. Huang, A. Acero, H.-W. Hon, and R. Foreword By-Reddy, *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice Hall PTR, 2001.

[71] M. Frigo and S. G. Johnson, "Fftw: fastest fourier transform in the west," *Astrophysics Source Code Library*, vol. 1, p. 01 015, 2012.

[72] M. Borgerding, *KISS FFT*, 2009.

[73] *HTK Speech Recognition Toolkit*, `http://htk.eng.cam.ac.uk/`, Accessed: 2013-07-12.

[74] D. H. Whalen and A. G. Levitt, "The universality of intrinsic f0 of vowels," *Journal of Phonetics*, vol. 23, no. 3, pp. 349–366, 1995.

[75] A. Faria and D. Gelbart, "Efficient pitch-based estimation of vtlnwarp factors," *Proc. Interspeech*, 2005.

[76] S. Wegmann, D. McAllaster, J. Orloff, and B. Peskin, "Speaker normalization on conversational telephone speech," in *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, IEEE, vol. 1, 1996, pp. 339–341.

[77] N. N. Schraudolph, "A fast, compact approximation of the exponential function," *Neural Computation*, vol. 11, no. 4, pp. 853–862, 1999.

[78] O. Vinyals, G. Friedland, and N. Mirghafori, "Revisiting a basic function on current cpus: a fast logarithm implementation with adjustable accuracy," *International Computer Science Institute Technical Report TR-07-002*, 2007.

[79] P. Mermelstein, "Automatic segmentation of speech into syllabic units," *The Journal of the Acoustical Society of America*, vol. 58, p. 880, 1975.

[80] H. Hohne, C. Coker, S. Levinson, and L. Rabiner, "On temporal alignment of sentences of natural and synthetic speech," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 31, no. 4, pp. 807–813, 1983.

[81] M. Wagner, "Automatic labelling of continuous speech with a given phonetic transcription using dynamic programming algorithms," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'81.*, IEEE, vol. 6, 1981, pp. 1156–1159.

[82] H. Leung and V. Zue, "A procedure for automatic alignment of phonetic transcriptions with continuous speech," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'84.*, IEEE, vol. 9, 1984, pp. 73–76.

[83] T. Svendsen and F. Soong, "On the automatic segmentation of speech signals," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'87.*, IEEE, vol. 12, 1987, pp. 77–80.

[84] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1," *NASA STI/Recon Technical Report N*, vol. 93, p. 27 403, 1993.

[85] J. J. Godfrey, E. C. Holliman, and J. McDaniel, "Switchboard: telephone speech corpus for research and development," in *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, IEEE, vol. 1, 1992, pp. 517–520.

[86] L. F. Lamel, R. H. Kassel, and S. Seneff, "Speech database development: design and analysis of the acoustic-phonetic corpus," in *Speech Input/Output Assessment and Speech Databases*, 1989.

[87] V. Zue, S. Seneff, and J. Glass, "Speech database development at mit: timit and beyond," *Speech Communication*, vol. 9, no. 4, pp. 351–356, 1990.

[88] A. Ljolje and M. Riley, "Automatic segmentation and labeling of speech," in *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, IEEE, 1991, pp. 473–476.

[89] Y. Gong, O. Siohan, and J. P. Haton, "Minimization of speech alignment error by iterative transformation for speaker adaptation," in *ICSLP*, 1992.

[90] K. Torkkola, "Automatic alignment of speech with phonetic transcriptions in real time," in *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, IEEE, 1988, pp. 611–614.

[91] P. Dalsgaard, "Semi-automatic phonemic labelling of speech data using a self-organising neural network," in *First European Conference on Speech Communication and Technology*, 1989.

[92] S. Rapp, "Automatic phonemic transcription and linguistic annotation from known text with Hidden Markov Models. An Aligner for German," *Proc. ELSNET Goes EAST and IMACS Workshop*, 1995.

[93] F. Brugnara, D. Falavigna, and M. Omologo, "Automatic segmentation and labeling of speech based on hidden markov models," *Speech Communication*, vol. 12, no. 4, pp. 357–370, 1993.

[94] D. Talkin and C. W. Wightman, "The aligner: text to speech alignment using markov models and a pronunciation dictionary," in *The Second ESCA/IEEE Workshop on Speech Synthesis*, 1994.

[95] J.-P. Hosom, "Speaker-independent phoneme alignment using transition-dependent states," *Speech Communication*, vol. 51, no. 4, pp. 352–368, 2009.

[96] K. Sjölander, "An hmm-based system for automatic segmentation and alignment of speech," in *Proceedings of Fonetik*, Citeseer, vol. 2003, 2003, pp. 93–96.

[97] D. T. Toledano, L. A. H. Gómez, and L. V. Grande, "Automatic phonetic segmentation," *Speech and Audio Processing, IEEE Transactions on*, vol. 11, no. 6, pp. 617–625, 2003.

[98] J.-P. Hosom, "Automatic time alignment of phonemes using acoustic-phonetic information," PhD thesis, Oregon Graduate Institute of Science and Technology, 2000.

[99] D. Huggins-Daines and A. I. Rudnicky, "A constrained baum-welch algorithm for improved phoneme segmentation and efficient training," in *Ninth International Conference on Spoken Language Processing*, 2006.

[100] K. Demuynck and T. Laureys, "A comparison of different approaches to automatic speech segmentation," in *Text, Speech and Dialogue*, Springer, 2006, pp. 277–284.

[101] S. Paulo and L. C. Oliveira, "Dtw-based phonetic alignment using multiple acoustic features.," in *INTERSPEECH*, Citeseer, 2003.

[102] F. Malfrère, O. Deroo, T. Dutoit, and C. Ris, "Phonetic alignment: speech synthesis-based vs. viterbi-based," *Speech Communication*, vol. 40, no. 4, pp. 503–515, 2003.

[103] I. Gholampour and K. Nayebi, "A new fast algorithm for automatic segmentation of continuous speech.," in *ICSLP*, 1998.

[104] A. Sarkar and T. Sreenivas, "Automatic speech segmentation using average level crossing rate information," in *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, IEEE, 2005.

[105] J. Keshet, S. Shalev-Shwartz, Y. Singer, and D. Chazan, "A large margin algorithm for speech-to-phoneme and music-to-score alignment," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 15, no. 8, pp. 2373–2382, 2007.

[106] G. Bordel, S. Nieto, M. Penagarikano, L. J. Rodríguez, and A. Varona, "Automatic subtitling of the basque parliament plenary sessions videos.," in *INTERSPEECH*, 2011, pp. 1613–1616.

[107] R. G. Sagum, R. A. Ensomo, E. M. Tan, and R. C. L. Guevara, "Phoneme alignment of filipino speech corpus," in *TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region*, IEEE, vol. 3, 2003, pp. 964–968.

[108] J. Vonwiller, C. Cleirigh, H. Garsden, K. Kumpf, R. Mountstephens, and I. Rogers, "Development and application of an accurate and flexible automatic aligner," *International Journal of Speech Technology*, vol. 1, no. 2, pp. 151–160, 1997.

[109] A. Stan, P. Bell, and S. King, "A grapheme-based method for automatic alignment of speech and text data," in *Spoken Language Technology Workshop (SLT), 2012 IEEE*, IEEE, 2012, pp. 286–290.

[110] B. L. Pellom and J. H. Hansen, "Automatic segmentation of speech recorded in unknown noisy channel characteristics," *Speech Communication*, vol. 25, no. 1, pp. 97–116, 1998.

[111] K. Kvale and A. K. Foldvik, "Manual segmentation and labelling of continuous speech," in *Phonetics and Phonology of Speaking Styles*, 1991.

[112] P. Cosi, D. Falavigna, and M. Omologo, "A preliminary statistical evaluation of manual and automatic segmentation discrepancies.," in *EuroSpeech*, 1991.

[113] A. Ljolje, J. Hirschberg, and J. P. v. Santen, "Automatic speech segmentation for concatenative inventory selection," in *The Second ESCA/IEEE Workshop on Speech Synthesis*, 1994.

[114] M.-B. Wesenick and A. Kipp, "Estimating the quality of phonetic transcriptions and segmentations of speech signals," in *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, IEEE, vol. 1, 1996, pp. 129–132.

[115] M. A. Pitt, K. Johnson, E. Hume, S. Kiesling, and W. Raymond, "The buckeye corpus of conversational speech: labeling conventions and a test of transcriber reliability," *Speech Communication*, vol. 45, no. 1, pp. 89–95, 2005.

[116] K. Johnson, "Massive reduction in conversational american english," in *Spontaneous speech: Data and analysis. Proceedings of the 1st session of the 10th international symposium*, 2004, pp. 29–54.

[117] R. A. Cole, B. T. Oshika, M. Noel, T. Lander, and M. A. Fanty, "Labeler agreement in phonetic labeling of continuous speech.," in *ICSLP*, 1994.

[118] S. Greenberg, J. Hollenback, and D. Ellis, "The Switchboard transcription project," in *1996 LVCSR Summer Workshop Technical Reports*, 1996.

[119] D. Jurafsky, A. Bell, E. Fosler-Lussier, C. Girand, and W. Raymond, "Reduction of english function words in switchboard.," in *ICSLP*, Citeseer, 1998.

[120] K. Sjölander, "Automatic alignment of phonetic segments," *Lund Working Papers in Linguistics*, vol. 49, pp. 140–143, 2001.

[121] *CSLU Toolkit*, `http://www.cslu.ogi.edu/toolkit/`, Accessed: 2013-07-12.

[122] B. Taskar, C. Guestrin, and D. Koller, "Max-margin markov networks," 2003.

[123] F. Sha and L. K. Saul, "Large margin hidden markov models for automatic speech recognition," in *Advances in neural information processing systems*, 2006, pp. 1249–1256.

[124] A. Katsamanis, M. Black, P. G. Georgiou, L. Goldstein, and S. Narayanan, "Sailalign: robust long speech-text alignment," in *Proc. of Workshop on New Tools and Methods for Very-Large Scale Phonetics Research*, 2011.

[125] S. Brognaux, S. Roekhaut, T. Drugman, and R. Beaufort, "Automatic phone alignment," in *Advances in Natural Language Processing*, Springer, 2012, pp. 300–311.

[126] F. Cangemi, F. Cutugno, B. Ludusan, D. Seppi, and D. Van Compernolle, "Automatic speech segmentation for italian (assi): tools, models, evaluation, and applications," *Proc. of AISV, Lecce, Italy*, pp. 337–344, 2011.

[127] C.-Y. Lin, J.-S. R. Jang, and K.-T. Chen, "Automatic segmentation and labeling for mandarin chinese speech corpora for concatenation-based tts," *Computational Linguistics and Chinese Language Processing*, vol. 10, no. 2, pp. 145–166, 2005.

[128] C. DiCanio, H. Nam, D. H. Whalen, H. T. Bunnell, J. D. Amith, and R. C. García, "Assessing agreement level between forced alignment models with data from endangered language documentation corpora.," in *INTERSPEECH*, 2012.

[129] F. Cutugno, A. Origlia, and D. Seppi, "Evalita 2011: forced alignment task," in *Evaluation of Natural Language and Speech Tools for Italian*, Springer, 2013, pp. 305–311.

[130] B. Bigi, "The SPPAS participation to Evalita 2011," in *EVALITA 2011: Workshop on Evaluation of NLP and Speech Tools for Italian*, 2012.

[131] B. Ludusan, "UNINA System for the EVALITA 2011 Forced Alignment Task," in *Evaluation of Natural Language and Speech Tools for Italian*, Springer, 2013, pp. 330–337.

[132] G. Paci, G. Sommavilla, and P. Cosi, "Sad-based italian forced alignment strategies," in *Evaluation of Natural Language and Speech Tools for Italian*, Springer, 2013, pp. 322–329.

[133] B. Wheatley, G. Doddington, C. Hemphill, J. Godfrey, E. Holliman, J. McDaniel, and D. Fisher, "Robust automatic time alignment of orthographic transcriptions with unconstrained speech," in *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, IEEE, vol. 1, 1992, pp. 533–536.

[134] J. Robert-Ribes, R. G. Mukhtar, *et al.*, "Automatic generation of hyperlinks between audio and transcript.," in *EUROSPEECH*, Citeseer, 1997.

[135] I. Trancoso, A. J. Serralheiro, C. Viana, and D. Caseiro, "Aligning and recognizing spoken books in different varieties of portuguese.," in *INTERSPEECH*, 2005, pp. 2825–2828.

[136] N. Braunschweiler, M. J. Gales, and S. Buchholz, "Lightly supervised recognition for automatic alignment of large coherent speech recordings.," in *INTERSPEECH*, 2010, pp. 2222–2225.

[137] T. J. Hazen, "Automatic alignment and error correction of human generated transcripts for long speech recordings.," in *INTERSPEECH*, 2006.

[138] P. J. Moreno, C. F. Joerg, J.-M. Van Thong, and O. Glickman, "A recursive algorithm for the forced alignment of very long audio segments.," in *ICSLP*, 1998.

[139] A. F. Martone, C. M. Taskiran, and E. J. Delp, "Automated closed-captioning using text alignment," in *Electronic Imaging 2004*, International Society for Optics and Photonics, 2003, pp. 108–116.

[140] K. Biatov, "Large text and audio data alignment for multimedia applications," in *Text, Speech and Dialogue*, Springer, 2003, pp. 349–356.

[141] K. Prahallad, A. R. Toth, and A. W. Black, "Automatic building of synthetic voices from large multi-paragraph speech databases.," in *INTERSPEECH*, 2007, pp. 2901–2904.

[142] K. Prahallad and A. W. Black, "Segmentation of monologues in audio books for building synthetic voices," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 19, no. 5, pp. 1444–1449, 2011.

[143] D. Damm, H. Grohganz, F. Kurth, S. Ewert, and M. Clausen, "Syncts: automatic synchronization of speech and text documents," in *Audio Engineering Society Conference: 42nd International Conference: Semantic Audio*, Audio Engineering Society, 2011.

[144] A. R. Toth, "Forced alignment for speech synthesis databases using duration and prosodic phrase breaks," in *Fifth ISCA Workshop on Speech Synthesis*, 2004.

[145] G. Bordel, M. Peñagarikano, L. J. Rodríguez-Fuentes, and A. Varona, "A simple and efficient method to align very long speech signals to acoustically imperfect transcriptions.," in *INTERSPEECH*, 2012.

[146] X. Anguera, N. Perez, A. Urruela, and N. Oliver, "Automatic synchronization of electronic and audio books via tts alignment and silence filtering," in *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, IEEE, 2011, pp. 1–6.

[147] P. J. Moreno and C. Alberti, "A factor automaton approach for the forced alignment of long speech recordings," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, IEEE, 2009, pp. 4869–4872.

[148] R. Das, J. Izak, J. Yuan, and M. Liberman, "Forced alignment under adverse conditions," *University of Pennsylvania, CIS Dept. Senior Design Project Report*, 2010.

[149] L. Baghai-Ravary, G. Kochanski, and J. Coleman, "Data-driven approaches to objective evaluation of phoneme alignment systems," in *Human Language Technology. Challenges for Computer Science and Linguistics*, Springer, 2011, pp. 1–11.

[150] S. Paulo and L. C. Oliveira, "Automatic phonetic alignment and its confidence measures," in *Advances in Natural Language Processing*, Springer, 2004, pp. 36–44.

[151] S. Cox, R. Brady, and P. Jackson, "Techniques for accurate automatic annotation of speech waveforms.," in *ICSLP*, vol. 98, 1998, pp. 1947–1950.

[152] K. Sjölander and M. Heldner, "Word level precision of the nalign automatic segmentation algorithm," in *Proc. of Fonetik*, 2004, pp. 116–119.

[153] L. Chen, Y. Liu, M. P. Harper, E. Maia, and S. McRoy, "Evaluating factors impacting the accuracy of forced alignments in a multimodal corpus.," in *LREC*, 2004.

[154] B. Lecouteux, G. Linarès, and S. Oger, "Integrating imperfect transcripts into speech recognition systems for building high-quality corpora," *Computer Speech & Language*, vol. 26, no. 2, pp. 67–89, 2012.

[155] L. Lamel, J.-L. Gauvain, and G. Adda, "Lightly supervised and unsupervised acoustic model training," *Computer Speech & Language*, vol. 16, no. 1, pp. 115–129, 2002.

[156] P. Placeway and J. Lafferty, "Cheating with imperfect transcripts," in *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, IEEE, vol. 4, 1996, pp. 2115–2118.

[157] A. Venkataraman, A. Stolcke, W. Wang, D. Vergyri, J. Zheng, V. R. R. Gadde, and R. Ramana, "An efficient repair procedure for quick transcriptions.," in *INTERSPEECH*, 2004.

[158] A. Haubold and J. R. Kender, "Alignment of speech to highly imperfect text transcriptions," in *Multimedia and Expo, 2007 IEEE International Conference on*, IEEE, 2007, pp. 224–227.

[159] K. Ohta, M. Tsuchiya, and S. Nakagawa, "Detection of precisely transcribed parts from inexact transcribed corpus," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, IEEE, 2011, pp. 541–546.

[160] L. Baghai-Ravary, S. Grau, and G. Kochanski, "Detecting gross alignment errors in the spoken british national corpus," *arXiv preprint arXiv:1101.1682*, 2011.

[161] M. H. Davel, C. Van Heerden, N. Kleynhans, and E. Barnard, "Efficient harvesting of internet audio for resource-scarce asr," 2011.

[162] M. Paulik and P. Panchapagesan, "Leveraging large amounts of loosely transcribed corporate videos for acoustic model training," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, IEEE, 2011, pp. 95–100.

[163] H. Chan and P. Woodland, "Improving broadcast news transcription by lightly supervised discriminative training," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, IEEE, vol. 1, 2004, pp. I–737.

[164] D. R. Van Niekerk and E. Barnard, "Phonetic alignment for speech synthesis in under-resourced languages," in *Proceedings of Interspeech*, International Speech Communication Association (ISCA), 2009.

[165] J. P. van Santen and R. Sproat, "High-accuracy automatic segmentation.," in *Proceedings of EUROSPEECH*, International Speech Communication Association, 1999.

[166] M. J. Makashay, C. W. Wightman, A. K. Syrdal, and A. Conkie, "Perceptual evaluation of automatic segmentation in text-to-speech synthesis," in *Proc. ICSLP*, vol. 2, 2000, pp. 431–434.

[167] A. Sethy and S. S. Narayanan, "Refined speech segmentation for concatenative speech synthesis.," in *Proceedings of INTERSPEECH*, Citeseer, International Speech Communication Association, 2002.

[168] Y.-J. Kim and A. Conkie, "Automatic segmentation combining an hmm-based approach and spectral boundary correction.," in *Proceedings of INTERSPEECH*, International Speech Communication Association, 2002.

[169] L. Wang, Y. Zhao, M. Chu, J. Zhou, and Z. Cao, "Refining segmental boundaries for tts database using fine contextual-dependent boundary models," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, IEEE, vol. 1, 2004, pp. I–641.

[170] J. Adell, A. Bonafonte, J. A. Gómez, and M. J. Castro, "Comparative study of automatic phone segmentation methods for tts," in *Proceedings of ICASSP*, 2005, pp. 309–312.

[171] E. Barnard and M. Davel, "Automatic error detection in alignments for speech synthesis," in *Proceedings of PRASA*, 2006.

[172] N. Ryant, J. Yuan, and M. Liberman, "Automating phonetic measurement: the case of voice onset time," in *Proceedings of Meetings on Acoustics*, vol. 19, 2013, p. 060 277.

[173] J. Yuan and M. Liberman, "Investigating/l/variation in english through forced alignment.," in *Interspeech*, vol. 10, 2009, pp. 2215–18.

[174] ——, "Automatic detection of "g-dropping" in american english using forced alignment," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, IEEE, 2011, pp. 490–493.

[175] ——, "Automatic measurement and comparison and vowel nasalization across languages," in *Proceedings of ICPHS*, 2011.

[176] A. Álvarez, A. del Pozo, and A. Arruti, "APyCA: Towards the automatic subtitling of television content in Spanish," in *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, IEEE, 2010, pp. 567–574.

[177] J. E. Garcia, A. Ortega, E. Lleida, T. Lozano, E. Bernues, and D. Sanchez, "Audio and text synchronization for tv news subtitling based on automatic speech recognition," in *Broadband Multimedia Systems and Broadcasting, 2009. BMSB'09. IEEE International Symposium on*, IEEE, 2009, pp. 1–6.

[178] J. Gao, Q. Zhao, and Y. Yan, "Towards precise and robust automatic synchronization of live speech and its transcripts," *Speech Communication*, vol. 53, no. 4, pp. 508–523, 2011.

[179] R. Ronfard and T. T. Thuong, "A framework for aligning and indexing movies with their script," in *Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference on*, IEEE, vol. 1, 2003, pp. I–21.

[180] R. Turetsky and N. Dimitrova, "Screenplay alignment for closed-system speaker identification and analysis of feature films," in *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on*, IEEE, vol. 3, 2004, pp. 1659–1662.

[181] Y. Wang, M.-Y. Kan, T. L. Nwe, A. Shenoy, and J. Yin, "Lyrically: automatic synchronization of acoustic musical signals and textual lyrics," in *Proceedings of the 12th annual ACM international conference on Multimedia*, ACM, 2004, pp. 212–219.

[182] H. Fujihara, M. Goto, J. Ogata, K. Komatani, T. Ogata, and H. G. Okuno, "Automatic synchronization between lyrics and music CD recordings based on Viterbi alignment of segregated vocal signals," in *Multimedia, 2006. ISM'06. Eighth IEEE International Symposium on*, IEEE, 2006, pp. 257–264.

[183] C. H. Wong, W. M. Szeto, and K. H. Wong, "Automatic lyrics alignment for cantonese popular music," *Multimedia Systems*, vol. 12, no. 4-5, pp. 307–323, 2007.

[184] P. Knees, M. Schedl, and G. Widmer, "Multiple lyrics alignment: automatic retrieval of song lyrics.," in *ISMIR*, 2005, pp. 564–569.

[185] A. Pedone, J. J. Burred, S. Maller, and P. Leveau, "Phoneme-level text to audio synchronization on speech signals with background music.," in *INTERSPEECH*, 2011, pp. 433–436.

[186] B. Kothari, "Let a billion readers bloom: same language subtitling (sls) on television for mass literacy," *International review of education*, vol. 54, no. 5-6, pp. 773–780, 2008.

[187] B. Kothari, A. Pandey, and A. R. Chudgar, "Reading out of the idiot box: same-language subtitling on television in india," *Information Technologies and International Development*, vol. 2, no. 1, pp. 23–44, 2004.

[188] P. Arora, "The folksong jukebox: singing along for social change in rural india," *Asian Journal of Communication*, vol. 22, no. 4, pp. 337–352, 2012.

[189] M. Eskenazi, "Using automatic speech processing for foreign language pronunciation tutoring: some issues and a prototype," *Language learning & technology*, vol. 2, no. 2, pp. 62–76, 1999.

[190] R. Vanderplank, "Déjà vu? a decade of research on language laboratories, television and video in language learning," *Language teaching*, vol. 43, no. 01, pp. 1–37, 2010.

[191] N. Diakopoulos, K. Luther, and I. Essa, "Audio puzzler: piecing together time-stamped speech transcripts with a puzzle game," in *Proceedings of the 16th ACM international conference on Multimedia*, ACM, 2008, pp. 865–868.

[192] J.-P. Hosom, "A comparison of speech recognizers created using manually-aligned and automatically-aligned training data," Technical Report CSE-00-002, Oregon Graduate Institute of Science and Technology, Center for Spoken Language Understanding, Beaverton, OR, Tech. Rep., 2000.

[193] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[194] M. Gales and S. Young, "The application of hidden Markov models in speech recognition," *Foundations and Trends in Signal Processing*, vol. 1, no. 3, pp. 195–304, 2008.

[195] S. J. Young, G. Evermann, M. Gales, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, *The HTK book version 3.4*. Cambridge University Engineering Department, 2006.

[196] S. L. Lauritzen, *Graphical models*. Oxford University Press, 1996.

[197] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1-2, pp. 1–305, 2008.

[198] D. Kollar and N. Friedman, *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.

[199] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *Information Theory, IEEE Transactions on*, vol. 13, no. 2, pp. 260–269, 1967.

[200] G. D. Forney Jr, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[201] B. T. Lowerre, "The HARPY speech recognition system," PhD thesis, Carnegie-Mellon University, 1976.

[202] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *The annals of mathematical statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[203] K. Demuynck, D. Van Compernolle, and P. Wambacq, "Doing away with the viterbi approximation," in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, IEEE, vol. 1, 2002, pp. I–717.

[204] Y. Zhang and J. R. Glass, "Unsupervised spoken keyword spotting via segmental dtw on gaussian posteriorgrams," in *Automatic Speech Recognition & Understanding, 2009. ASRU 2009. IEEE Workshop on*, IEEE, 2009, pp. 398–403.

[205] *Mod9 alignment api version 0.8*, `http://mod9.com/doc/Mod9_Alignment_API_v0.8.pdf`, Accessed: 2014-12-03.

[206] R. T. Fielding, "Architectural styles and the design of network-based software architectures," PhD thesis, University of California, Irvine, 2000.

[207] *Mod9 alignment server version 0.8-i386-20140123**, `http://mod9.com/doc/Mod9_Alignment_Server_v0.8-i386-20140123.pdf`, Accessed: 2014-12-03.

[208] T. Schlippe, S. Ochs, and T. Schultz, "Web-based tools and methods for rapid pronunciation dictionary creation," *Speech Communication*, vol. 56, pp. 101–118, 2014.