1 Goals & Import of Dissertation

The main goal of this dissertation is to develop training methods for neural networks which lead to better classification of data from unseen conditions. In particular, this work focuses on small data sets, where traditional neural net training leads to overfitting.

When training a statistical classifier on a small data set, the researcher must take extra precautions to avoid overfitting. A powerful model (such as a deep neural network) will easily achieve 100% classification accuracy on a small training dataset, because the model learns "useful" noise in its parameters. However, that "useful" noise is specific to the training data, and will not generalize to a new dataset. For example, deep neural network training to classify images of cats and dogs may learn to associate teeth with dogs, if some of the dogs have their mouths open. This kind of random chance occurs more often with small datasets, which means the smaller the dataset, the worse the generalization of the model. If this cat vs dog classifier gets a new picture of a cat with its mouth open, it may be very well classified as a dog.

Likewise, in speech recognition we must be careful with small datasets, so that our acoustic models don't learn "useful" noise during training. The sources of this noise in speech recognition stem from background conditions (busy streets vs quiet room), the person doing the talking (old woman vs young boy), or an extreme example is the language itself. A small dataset of speech from an audiobook will produce an acoustic model which fails miserably on speech from a noisy car. The most common approach to combat this problem is to collect a new, bigger dataset from the target condition. However, if we want speech recognition that is truly human-like, we need to develop training techniques which lead to better generalization.

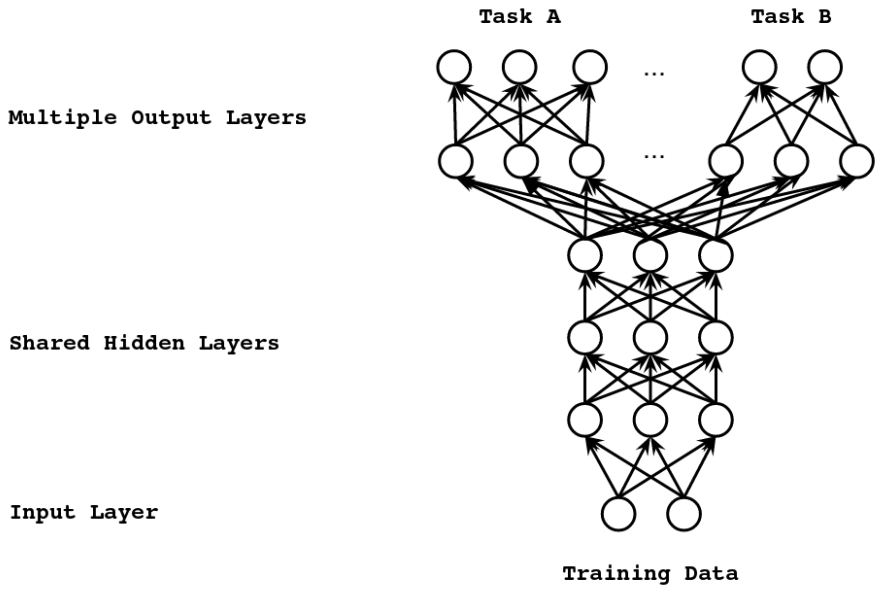
This dissertation investigates training techniques for acoustic modelling on small datasets. To determine the generalizability of the resulting models, the training and testing datasets are sampled from very different recording conditions. The unseen testing data conditions are (1) `new noise`, (2) a `new speaker`, or (3) a `new language`.

Using the framework of Multi-Task Learning, I train a single neural net to classify a small dataset with several tasks, encouraging the hidden layers to learn generic, useful represetations of the data. This is accomplished without any explicit adaptation of model parameters or data transformations.

1. For the `Noise` condition, I train on clean audio and test on noisy audio.
2. For the `Speaker` condition, I train on a set of speakers, and I test on an unseen speaker.
3. For the `Language` condition, I train on (mostly) English, and I test exclusively on Kyrgyz.

My approach is innovative because it doesn't require massive data sets, as some other popular reseach labs have relied on (Google, Firefox, Baidu, Microsoft, etc), and it doesn't require tuning the net to the new dataset (e.g. fMLLR). Further more, my approach is not specific to any one dataset, and it can be used to train any neural net (not just for speech recognition).

MTL training for neural nets works by training a set of hidden layers to perform multiple tasks (multiple output layers). Here is an example of MTL architecture for an acoustic model from Heigold 2013.



This is the architecture that will be used in my dissertation. The number of input nodes on my neural nets corresponds to the dimensionality of audio features, and the number of nodes on each output layer corresponds to the number of phonemes (i.e. monophones or triphones) I've defined for the language.

During training all `Tasks` are trained in unison, but during testing only one `Task` is used. The only benefit from these extra tasks comes from the training phase, and their influence on the shared hidden layers.

2 Overview of Speech Recognition

All research exists in some historical context, answering pressing questions of the times, making use of and reacting to existing technologies. ASR research is a fine example of a field which has grown in directions defined by either military, academic, or commercial incentives. Early work on ASR reflected the needs of telecommunications companies. Then came a wave of interest from the US Department of Defense, and most recently the four tech giants - Google, Amazon, Facebook, and Apple. While all these initiators pushed researchers in different directions, they all share one common goal: to make ASR more human-like.

This section contains the training and testing procedures for standard automatic speech recognition (ASR) pipelines. This short overview will provide the reader with a technical grounding in ASR.

2.1 1952: Isolated Word Systems: Bell Labs

Most of the early work on ASR in the 50's and 60's focused on isolated word, speaker-dependent speech recognition. This research was lead by R&D labs in telecommunications companies like Bell Labs and the NEC Corporation (a Japanese technology giant), and also academic labs like MIT Lincoln Labs. (XXX)

The typical use case was a single adult male carefully reading off single digits [0-9] into a microphone. One of the very first demonstrations reported accuracy rates of up to 99% on isolated digit recognition **?**. This system relied on approximations of the formant frequencies to recognize entire words. That is, there was no concept of syllables or consonants or vowels in these systems. The word was treated as a single unit, and during classification all words were compared to each other to find the best match.

This work, along with most others of this time period, relied on a template-matching framework to classify spoken words. An exemplar of each word was saved to disk (for each speaker ideally), and when a user spoke a new, unknown word into the microphone, the computer compared that audio with all the exemplars it had on file for that speaker. The closest match was returned back, and when recognizing a set of ten digits, this worked surprisingly well.

The Bell Labs system worked in the following way:

1. FEATURE EXTRACTION
    (a) two frequency ranges of the audio are extracted, which roughly correlate to the first two formants of speech
    (b) these two formants are plotted on an x-y axis in time
2. TEMPLATE MATCHING
    (a) the 2-D x-y plot from new audio is compared to each of 10 exemplars on file, and closest is returned

An example set of templates (from the original Bell Labs paper) is show below:

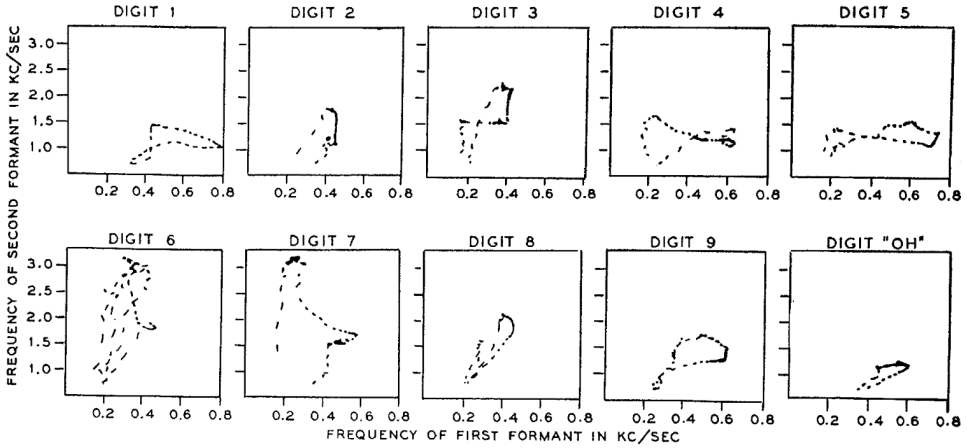


FIG. 2. Photographs of formant 1 *vs* formant 2 presentation of the digits. Trace interruption period = 10 ms. Recognition criteria depend upon significant differences in these shapes and upon their relative duration in the frequency space.

It may sound like speech recognition research was off to a great start with accuracy near perfect, but in truth the approach taken in much of this early work could not be extended further for a few reasons. These approaches relied on acoustic properties of vowels in a whole-word template-matching scheme which requires a representation of each word saved on disk. This paradigm, if extented to larger vocabularies would run into major issues in time and space complextiy (big O). If the system needed to recognize 1000 words instead of just 10, the time needed to compare new input to each of the 1000 exemplars would be prohibitive. Additionally, the space on disk would increase with every additional word. Therefore, both time and space complexity are proportional to the number of words in the lexicon. This does not bode well for scaling to all words in a language.

Aside from implementation issues with this approach, a more fundamental limitation is the use of whole words as templates, and extracting only formant frequencies as features. Two words with similar consonants and identical vowels (eg. 'dog' vs 'dock') would be nearly indistinguishable for this system.

As such, two of the major problems with these digit recognizers relate to the unit of comparison (ie. what does the template represent?) and the extraction of features (ie. how is the template stored?). In the following, the unit of comparison switches from the word to the phoneme, and the features extracted switch from formants to more fine-grained frequency bins (eg. MFCCs).

2.2 1971: Constrained Sentence Recognition: ARPA

Speech research soon was boosted into full gear in 1971, when the Advanced Research Program Agency (ARPA) of the US Department of Defense launched the 5-year Spoken Understanding Research (SUR) program. The goal of the program was to "obtain a breakthrough in speech understanding capability that could then be used toward the development of practical man-machine communication systems" (Klatt 1977). ARPA wanted something that Airforce pilots could control with their voice while their hands were busy steering. The SUR program spurred various papers from four main research groups (XXX). Probably the most significant result of the ARPA project was James Baker's 1975 dissertation at CMU, which firmly established the Hidden Markov Model in ASR (Baker 1975).

The contestants were given the task of creating a system which could recognize simple sentences from a vocabulary of 1000 words with a 10% WER in reasonable time. In order to make a recognizer that could handle sentences instead of isolated words, where the length of that string was unknown to the system, major overhauls of the Isolated Word system were needed.

First of all, it was clear that storing an exemplar of each word on disk is not an option with continuous speech systems. In addition to the nearly impossible task of discovering word boundaries from raw audio, the decoding speed would be horrendous (even if word boundaries

were found). The machine would have to compare each word to each of the 1,000 candidate words on disk. The space used on disk would be prohibitive, not to mention the time needed by the user to record every word during speaker enrollment. Modeling whole words became an obvious bottleneck to recognition of continuous speech. As such, the phoneme became an obvious candidate.

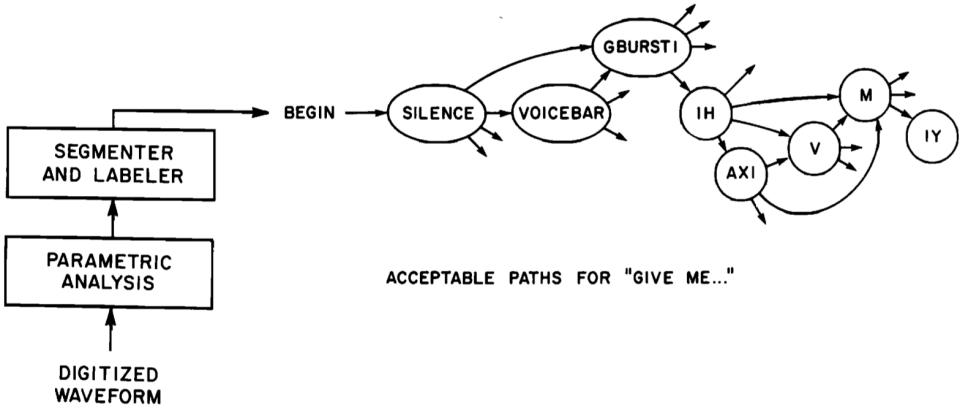
The phoneme is the smallest meaningful speech sound. Every language has a finite set of phonemes, and with this finite set of phonemes all words are created. Typically languages don't have more than 50 phonemes, and that number will not increase with vocabulary or grammar complexity. Where simple systems had hard limits of 100 or 1000 words, with only 50 discrete phonemes there is no upper limit to the number of words a system can recognize.

All of the teams in the ARPA project used the phoneme as the unit for modeling speech. At the end of the project, the team at Carnegie Mellon showed best performance with their 'Harpy' system (XXX, for review cf. Juang 2005). Like in Isolated Word Recognition, all teams used some kind of template matching, but with phoneme templates instead of word templates.

The Harpy system decoded a new utterance in the following way:

1. FEATURE EXTRACTION
   (a) process audio with 5kHz low-pass filter and sample to 10k samples per second
   (b) extract linear prediction coefficients in 10-ms frames with a 10-ms shift
   (c) group together adjacent acoustic segments if similar
2. GRAPH CONSTRUCTION
   (a) a set of 98 phonemes (and diphones) defined by experts
   (b) pronunciations for all words defined
   (c) pronunciations of all accepted sentences in the grammar compiled into one graph (15,000 states with self-loops)
3. DECODING
   (a) incoming audio segments compared against 98 templates
   (b) best path (with beam search) returned

An example of what a decoding graph in Harpy might look like (Klatt 1977):



Harpy is a speaker-specific system, and the 98 phoneme templates need to be tuned to each speaker. For a new speaker to be enrolled into the system, she must spend about 30 minutes recording example sentences, which are then force-aligned to the graph. This forced-alignment, however, assumes that at least one speaker has already been enrolled, and their 98 phoneme templates are used to align the next speaker's audio.

Given that command and control was the target application, a limited vocabulary and limited grammar was reasonable. The user said one short sentence (from a constrained, finite set), and the decoder compared that sentence to a graph of all possible sentences, and returned the closest match. This assumes the user actually said a sentence in the machine's vocabulary. Each user was trained to work with the machine (learn its grammar), and the machine was trained to work with each user (via enrollment).

What these recognizers lacked in flexibility, they gained in accuracy. The machine didn't have to consider more than 1,000 words and a simple grammar. Furthermore, there was no real issue of noise conditions, because recording and testing would be both in quiet conditions. There was no worry about microphone robustness or sampling-rate issues, because the creators knew exactly beforehand what hardware the recognizer ran on.

This approach worked just fine until users wanted something more human-like. First of all, training the recognizer to work for every new user was a hassle. We humans don't need to relearn all sounds in our language when we meet someone new, but these machines did. We humans can understand our friends when we're in an echoey factory or in a small room, but these machines couldn't.

Regardless of the successes of the SUR program, ARPA was dissappointed. The best system, Harpy, decoded in about 80x real time (WER of XXX). Harpy could not be used in practice, and speeding her up was not a simple task. In his review of ARPA's Speech Understanding Research Program, (Klatt 1977) concludes that "all [teams] failed to meet the ARPA goals."

In addition to problem of speed, grammar flexibility was a major concern for making systems that could recognize spontaneous speech. The kinds of sentences recognized by Harpy were determined by a BNF grammar. This consisted of a set of hand-crafted rules, and was not easily extensible. Even in the 1980's, researchers realized that such a grammar was not a viable option. A major shift was about to take place in the ASR world, moving away from template matching and strict grammars to statistical acoustic models and statistical grammars. Instead of hard assignments (grammatical or not), a better system would assign a kind of likelihood to the sentence, word, or sound in question.

2.3 1994: The GMM-HMM Approach: HTK

The 1970's introduced the HMM (Baker 1975), the 80's introduced the GMM (Juang 1986) and the statistical grammars (Katz 1987), and the 90's made it all trainable in one toolkit: the Hidden Markov Model Toolkit (HTK) (Young 1994). HTK was the first toolkit to incorporate all of the core components of Modern GMM-HMM speech recognition.

CMU's Sphinx was a leader in the ASR toolkit game (XXX), but the first version of CMU's Sphinx used Vector Quantized codebooks to estimate emission probabilities on HMM states, while HTK began with GMMs in 1994. Regardless of HTK and Sphinx's (minor) differences in performance, HTK's extensive documentation (The HTK Book) became the reference of choice for most speech recognition researchers. The first results of the HTK system were reported as a benchmark on DARPA's Resource Management task in 1992 (Woodland & Young 1992).

The standard GMM-HMM pipeline of HTK is outlined below.

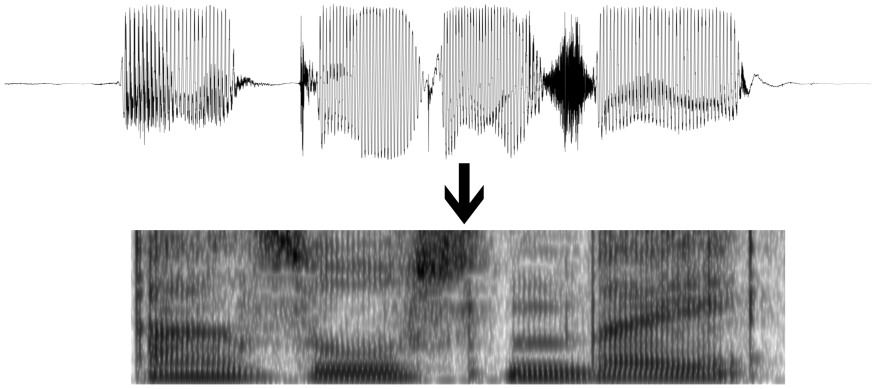
1. FEATURE EXTRACTION
    (a) sliding window feature extraction
2. GMM-HMM MONOPHONE TRAINING
    (a) Flat-start alignment
    (b) Baum-Welch re-estimation
3. GMM-HMM TRIPHONE TRAINING
    (a) Phonetic decision tree
    (b) Baum-Welch re-estimation

4. DECODING
   (a) Network compilation
   (b) Viterbi decoding

FEATURE EXTRACTION

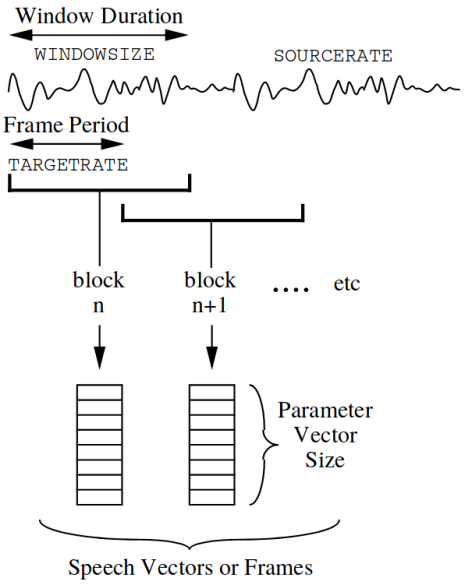
A common goal of all feature extraction in speech recognition is the following: the audio signal in its raw form is not condusive to modeling speech, and we need to make speech-information more available for statistical modeling. Feature extraction enhances the signal-to-noise ratio, but it is more than just a "cleaning" of audio.

Raw audio is simply air pressure measured in time. Speech is transmitted via change in pressure, and as such raw audio contains all the information relevant to human speech. However, speech sounds are differentiated by amplitudes in both the frequency and time domain. Feature extraction makes that frequency information available explicitly, instead of relying on the statistical model to infer on its own.



The two standard audio feature extraction techniques in HTK (and in ASR in general) are Mel-cepstral coefficients (MFCCs) or perceptual linear prediction (PLPs).

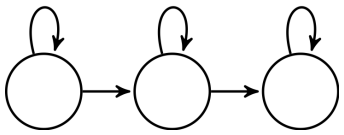
Overlapping windows are shifted across the audio, from beginning to end, to extract feature vectors. For each timestep, the audio falling inside the window is subjected to a series of transformations which reduce the dimensionality of the data and highlight speech-specific information. Typically the windows are 25 milliseconds long, and the the shift is 10 milliseconds. In this way, enough time is allowed to capture phonetic information inside one window.



Ideally, every feature vector should contain enough information to identify the phoneme which was uttered at that timestep.

MONOPHONE TRAINING

A monophone GMM-HMM acoustic model contains (as the name suggests) one model for one phoneme. For a language with 50 phonemes, a monophone system will have 50 separate GMM-HMMs. The number of states in each phoneme's HMM is not specified, and is a matter of researcher discretion. Given that human speech is a purely left-to-right signal, monophone HMMs are defined to be strictly left-to-right, with the addition of self-loops. Abstracting away from some of the implementation specifics, each monophone HMM will be designated as such:



The self-loops (as in the Harpy system) nicely model the differences in time which occur in speech. A long, drawn-out vowel will be modeled with the exact same HMM as a short, quick vowel. The longer vowel will just have to pass through more self-loops.

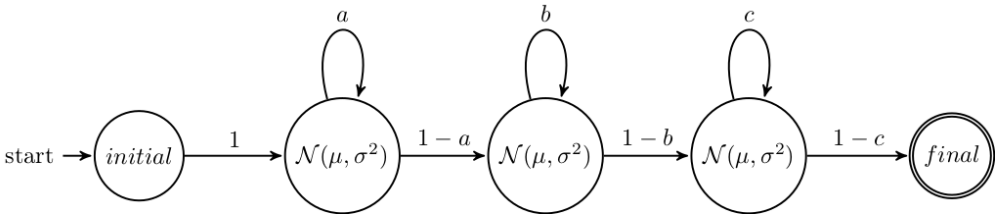
Most GMM-HMMs in speech recognition use three states because neighboring phonemes blend together. Acoustic features at phoneme boundaries differ significantly from the center of the phoneme. All else being equal, the central state of a monophone should be more consistent than its edges. In addition to modeling co-articulation, three states will allow for the modeling of more complex sounds like affricates or stop consonants, which are composed of multiple acoustic events. For instance, a stop consonant like [b] can be broken into three parts (closed voicing, plosive release, aspiration). Each of these three parts is a separate acoustic event, and should ideally be modeled separately. Multiple state HMMs allow for fine-grained modeling of a single phoneme into its component parts.

Defining a typology for these monophones is not enough to use them in decoding new audio. HMMs are defined by:

1. a set of states
2. a set of transitions between states

3. the probability of traversing each transition
4. a set of possible observations
5. a probability of each state generating each possible observation

So far we have defined how many states we want (three states per phoneme), we have defined the *kinds* of transitions between states (left-to-right or self-loop), and we have defined the shape of the emission probability of each state (a multivariate Gaussian mixture model). However, what this gets us is merely a set of skeleton GMM-HMMs. For these HMMs to be useful, we need to have accurate transition probabilities (s.t. the probability of leaving any non-final state == 1) and emissions probabilities (the μ, σ^2 for each Gaussian component). That is, we need to estimate the parameters of our HMM-GMMs.



One approach to get these numbers would be to sit down and think really hard about what these numbers should be, filling them in ourselves. We could make self-loops for vowels longer than stop consonants, and we could define the parameters of Gaussians based on some knowledge of acoustic phonetics. However, this is a pretty horrible idea. So instead of trying to guess these values, we use some concepts from machine learning to *learn* them from data.

So, to learn about the acoustics of human speech, we need to collect a lot of it. However, just having access to a huge spoken corpus isn't enough; we need to know which segments of the audio correspond to which states in our GMM-HMMs so that we can update parameters given the appropriate data. While it is certainly possible to align segments of audio to HMM states by hand, it is a very tedious job (examples of this kind of corpus include TIMIT and the Buckeye corpus).

A much better approach is to automatically align audio segments (i.e. feature vectors) to HMM states. The algorithms for this alignment have been known since the 1960's, and are guaranteed to acheive the best HMM for the given data (Rabiner, Lawrence. "First Hand: The Hidden Markov Model"). This parameter estimation technique is called the Baum-Welch Algorithm, and it falls into the category of Expectation Maximization (EM) algorithms. The technique involves an iterative process in which the model generates a new set of labels for the data, and trains itself on that new dataset.

Given (1) a speech corpus, (2) transcriptions of the speech, and (3) a pronunciation dictionary, we can estimate all the parameters of our GMM-HMMs.

Flat-Start Alignment

In order to successfully train our monophone GMM-HMMs from speech data, we eventually want to have the entire corpus split into segments which correspond to the states of the HMMs. Before we can begin iterations of Baum-Welch, we need a first guess as to what the parameters could be - ie. we need an initial alignment to the data. Flat-start training is a very simple solution to this first-alignment problem.

For Baum-Welch re-estimation to work, we use the parameters of our model to generate new alignments for the model itself. However, in the first step, we have no initialization for the model parameters. We could initialize all parameters randomly, and then start Baum-Welch. This in principle should work, but it would take a long time because the initialization would be truly horrendous. A better first guess would come not from random paramters, but from the audio itself.

In flat-start training, we assume a few things: (1) utterances, (2) transcriptions for the utterances, and (3) a phonetic transcription for every word in the transcripts. Given these three things, we can construct a phonetic transcription for each utterance.

Given a phonetic transcription of an utterance in training, starting from left to right, we assign an equal section of the audio to each phoneme in the transcription. This first left-to-right alignment is our initial guess (ie. our flat-start). In practice, this works faster than randomly initialized parameters.

In this way, Flat Start training allows us to make an initial estimate for every parameter of every monophone GMM-HMM in our acoustic model. This is a very, very crude first estimate, and its assumptions are absurd. This alignment assumes that each phoneme in a word is exactly the same length as every other phoneme in that word, and actually in the whole utterance. Human speech doesn't work like this, and as such it seems like there's no way that flat start training could work, but it does. One assumption is the saving grace of Flat start training. The assumption is that the linear order of phonemes in an utterance is left-to-right. With just that one assumption built into the typology of our HMMs, we can use the Baum-Welch algorithm to make increasing better alignments to the speech data. It works because Baum-Welch training is guaranteed to improve model-data fit, no matter how bad the original fit is.

Baum-Welch re-estimation

Now that we have an initial estimate of all the parameters in our monophone GMM-HMMs, we can use both the data and the model to make better and better alignments. The basic idea of Baum-Welch is that we can use the the current model parameters of any given utterance HMM to generate the most likely alignment of that HMM to its corresponding audio clip. Then we take that alignment as the ground-truth alignement, and update the model parameters accordingly.

By iterating over (1) parameter updating and then (2) alignment generation, we reach parameters that get closer and closer to true estimates. It is key to remeber that all the utterances are updated in unison, and as such, the sounds and speech rates and other peculiarities of individual audio utterances become averaged out. Given that GMM-HMMs are generative models, we are maximizing the likelihood of the data given the model. Our implementation of the EM algorithm (i.e. Baum-Welch training) is a special case of Maximum Likelihood Estimation (MLE), where we don't have full information about the relationship between our model and the data. If we had the golden truth for alignments between feature vectors and HMM states, then we would just update the parameters once, and there would be no re-alignment afterwards. However, since we don't have the alignments, we need to guess at them, and our guesses get better over time.

TRIPHONE TRAINING

Monophone systems work well, but they are too simple to model natural speech in its full complexity. In order to increase the complexity of monophone HMMs, there are a few routes we can take.

1. add more states to each HMM
2. add more GMMs to each state
3. create more HMMs

Monophone HMMs by themselves can become fairly complex (ie. more parameters) if we just keep adding Gaussian components to each state. We can also add more states to each HMM (up to five is not uncommon), and in this way capture more nuanced time information and co-articulation effects. However, no matter how many extra states we add and no matter how many extra Gaussians we thrown into a state, at the end of the day, all instances of a phoneme will be collapsed together. This is bad because phonemes vary acoustically depending on their context, and we can predict those variations. Since we can predict it, we should model it (if we have enough data).

Collapsing every realization of a phoneme into one HMM results in fuzziness in the model. For example, the vowel [ah] will display very different formant patterns if is pronounced between nasals [nahn] ('non') versus when it is found between two stops [taht] ('tot'). These co-articulation effects are predictable given the context, and as such, we can reliably model them if we give our model enough freedom. For the monophone [ah], both realizations [n+ah+n] and [t+ah+t] will be trained as one model [*+ah+*].

Ideally, we would like to model every phoneme in every phonetic context with its own HMM. In that way, we would be sure that all predictable variation would be built into the acoustic model. If we have N number of phonemes in our language, instead of defining a mere N monophones, we would ideally train $N * N * N$ monophones in context, to account for both left and right co-articulation effects. However, training N^3 models is much more difficult that training N models, because the chances of finding N^3 contexts in any speech corpus are none (no slim here). No language will allow for N^3 combinations of phonemes. Furthermore, given a finite data set, the more HMM states we define in the model the less data is available for each state.

Phonetic decision tree

Introduced by Young in 1994, the phonetic decision tree balances model complexity and overfitting by automatically defining acoustic models which increase in complextiy as the dataset increases in size. This effficiently avoids over-fitting on small datasets, and increase model power to better exploit big datasets. Also, phonetic decision trees allow for modeling of triphone contexts which were never observed in the training data.

Triphones are a natural extension of monophones. Monophones are the tree roots, and triphones are the tree leaves.

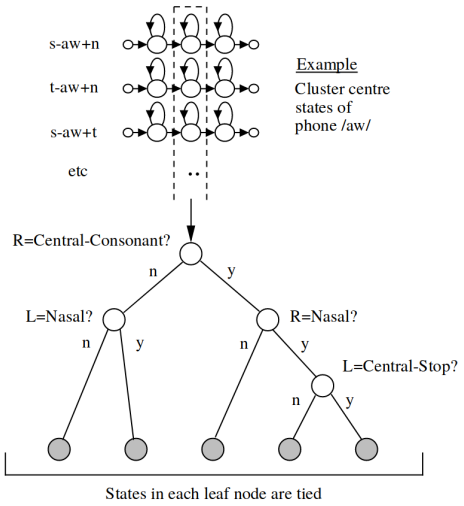
We begin with a pool of feature vectors and information about utterances. From the phonetic transcription of the utterance we can glean lots of good, predictive information about the acoustics found in those feature vectors. A powerful enough statistical model should account for all predictable variation in the data (think extra latent variables in regression analysis). Given the utterance transcriptions, there's a wealth of predictive information we could extract and codify into our models. For instance, we can code information not only about each phoneme, but also its 10 closest neighbors, what word the phoneme belongs to, what position in the word the phoneme occupies (beginning, middle, end) etc. The list is potentially infinite, and each extra variable we account for (if meaningful) will help our model correctly classify the acoustic features. In another application, statistical speech synthesis, this is exactly what happens, to be able to produce more detailed, accurate speech. For ASR, we want to be able to recognize the speech of any speaker, and as we add more complexity to the model we need more data to train without overfitting.

With monophone HMMs, we are using one variable only to define the HMM: the identity of the phoneme itself. To double the model complexity, we could split all training examples for each monophone into two categories: (a) phoneme occurs in middle of word or (b) phoneme occurs at word edge. This division between word-internal phoneme and word-edge phonemes will account for a good deal of predictable variation. The [l] in [lamp] and the [l] in [complex] would be assigned to different monophones.

Now, we can stop here, (doubling complexity) or we can try to model more predictable information about each phoneme. Triphone decision trees work by extending the tree downwards from monophones, by asking questions about the identity of the phonemes immediately neighboring the monophones (left and right context). Out of all the possible predictors we could add to the model, left and right context naturally catches a lot of variation, regardless of speaker, noise channel, or anything else. Because we account for both the left and right context of the phoneme, we call the resulting models 'triphone HMMs'. Each HMM defined in this model contains information about a sequence of three phonemes. The immediate problem with these models we mentioned above: our N parameters suddenly become N^3, and modeling each context is impossible. We run into the Curse of Dimensionality, and becuase finding more data isn't an option, Young (1994) came up with an approach to reduce the number of triphones to

a reasonable number, where the resulting triphones were sensical.

The solution comes from the intuition that groups of phonemes may extert similar co-articulation effects. As such, instead of a blind clustering algorithm that decreased the N^3 possible triphones to M physical triphones relying on just the data, Young came up with the Phonetic deicision tree, which makes splits in its branches based on some linguistically relevant questions about the left and right context. For example, a question in the tree could ask if the phoneme to the right is a nasal consonant (`R=Nasal?`). This way, triphones are grouped together by data and prior beliefs about natural classes of phonemes.



To make the phonetic decision tree (similar to CART), first a set of monophone GMM-HMMs is trained. Next, every monophone HMM is copied and re-labeled as a triphone (given left and right context). For this copy-paste first pass, all observed triphones are defined and given their parameters (just a copy of the original monophone). Next, given these new triphones, Baum-Welch re-estimation is performed just as before over all the training data. As such, the inidividual triphones will have a chance to update their parameters based on their own designated subset of the feature vectors. Next given these new triphone GMM-HMMs, we train the decision tree.

Here it is important to add a nuance of the standard phonetic decision tree. All splitting for triphones occurs at the state level. That is, each state can have its own root (see Kaldi docs). The above diagram would result in three separate decision trees for the phoneme [aw].

During this process, the researcher decides

1. what kinds of questions can split branches
2. how many leaves in the tree
3. how many Gaussians are allotted to the entire tree
4. how many Gaussians alloted to each leaf

Each leaf represents a single state of a 3-state GMM-HMM, and as such, the number of Gaussians in the entire model must be greater than or equal to the number of leaves.

Baum-Welch re-estimation

After the phonetic decision tree has been defined, the new triphones are retrained based on the alignments from the possible triphones.

DECODING

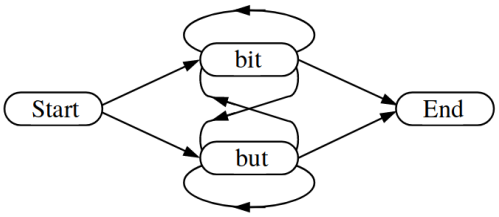
Decoding in HTK can be customized to a good extent, but there is a common thread in its approach that stems from Baker's work all the way back in the mid-1970's. A graph is compiled, and then the graph is searched. In HTK the graph is called a network.

Network compilation

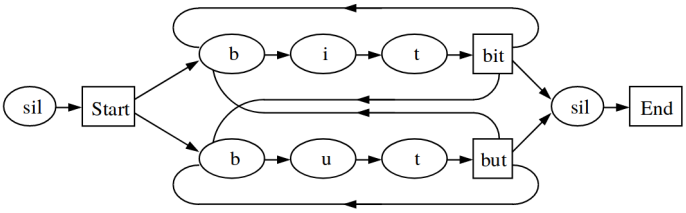
The compiled network contains heirarchichal information about (1) grammatical word sequences, (2) word pronunciations, and (3) phoneme contexts. HTK is very flexible about which kinds of grammars can be integrated into the network, but the standard approach is to use an n-gram backoff language model. HTK also implements dynamic graph construction, so that the entire graph may be larger than allotted RAM.

Taking an example from the HTK book itself, here is a toy grammar for the two words 'bit' and 'but'. The grammar will accept any sequence of these two words, in any order.

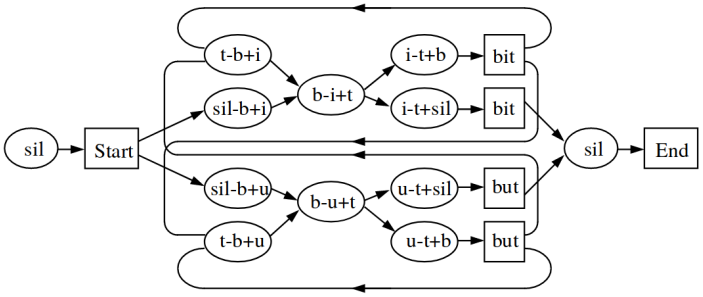
Looking at this HTK network from the level of words, we will find this kind of structure:



When we add information about pronunciations, we get the following (monophones):



Finally, the lowest level of detail relates to the contexts of the phonemes (triphones):

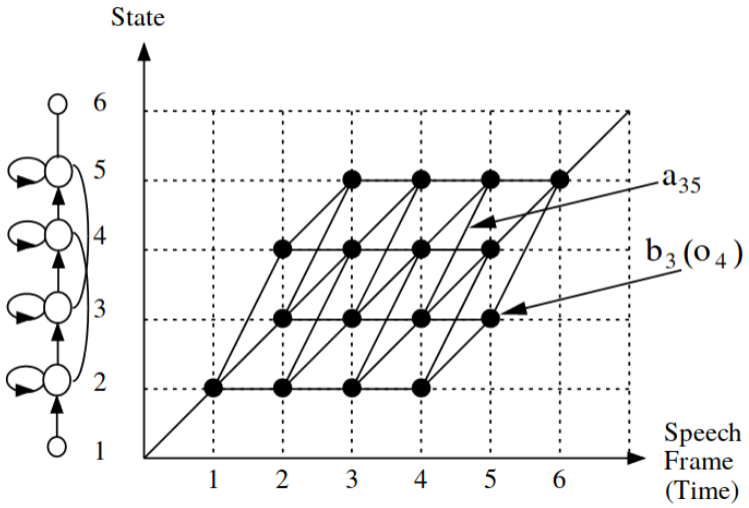


In the graph above, we can see that by modeling context-dependency, we've expanded our original monophone [b] into four possible [b]'s in context: `[t+b+i]`, `[sil+b+i]`, `[t+b+u]`, `[sil+b+u]`.

Viterbi decoding

In HTK, the Viterbi algorithm is re-framed into what is called the 'Token Passing Model. This approach keeps knowledge of paths trhough the graph by creating a Token that is passed from state to state, seeking the best possible path through the graph. As a Token is passed from state to state, it collects the words through which it passes. The decoding graph in HTK represents

words and phones as separate states, such that at the end of a string of phones, the Token must pass through some Word state.



Overall, HTK is a solid toolkit for speech recognition. More than anything, it became replaced because of its restrictive license.

2.4 2011: The Modern DNN-HMM Approach: Kaldi

The standard DNN-HMM approach is called the 'hybrid' approach, because it is a merger of the HMMs from traditional GMM-HMM ASR and neural net acoustic modelling.

The hybrid approach started gaining traction in the 2000's, when neural nets were rediscovered and GPUs were used to perform backprop operations. For ASR in particular, the publishing of Hinton et al's 2012 paper "Deep Neural Networks for Acoustic Modeling in Speech Recognition", solidified the staying power of DNNs in speech recognition.

Kaldi is currently the most popular ASR toolkit for research not only because it supported DNN acoustic modeling before other toolkits - Kaldi also takes a different implementation of the graph decoder, based on Weighted Finite State Transducer technology (Mohri 2001). Kaldi is written in C++ with a much more free license than HTK or Sphinx, making it nicer for both researchers and industry developers.

The DNN-HMM and GMM-HMM approaches are very similar in almost all respects, and the DNN is essentially just a drop-in replacement for the GMM acoustic model. Some minor adjustments have to be made for the Viterbi decoding math to work out: i.e. the standard equation requires the acoustic model to produce a *likelihood* of the audio given a phomeme. GMMs give *likelihoods*, but DNNs give *posteriors*. Aside from some minor math adjustments, at decoding time GMMs and DNNs serve the same purpose: give information about phoneme-audio relations. The rest of the decoder (phonetic dictionary, n-gram language model, phonetic decision tree) remains the same.

Even though the DNN-HMM and GMM-HMM follow nearly identical procedures during decoding, their training is significantly different. In fact, the standard DNN training procedure relies on the GMM training to produce labeled data. Feedforward deep neural nets require labeled data for supervised training (there are some exceptions as in CTC training).

GMM-HMM training uses Baum-Welch parameter estimation via flat-start monophone training to iterate steps of alignement and estimation in order to generate more and more accurate alignements (and more accurate GMMs). This procedure is mathimatically sound, guaranteed to lead to better and better parameter estimates for the given data.

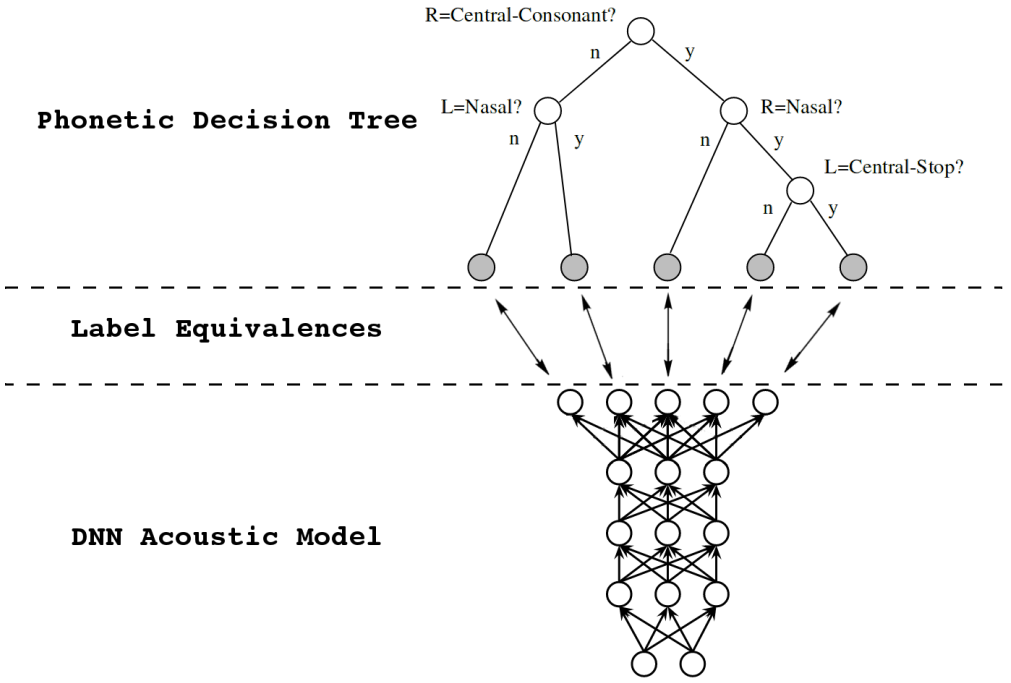
Piggy-backing off this procedure, DNN training requires Baum-Welch not because Baum-Welch train good GMMs, but because it leads to good alignments. These alignments are then used as training data for DNN training with gradient descent via backprop.

Typically, to integrate time information into DNN acoustic model training, the input features are not merely passed in one frame at a time, frame-splicing occurs. The net still makes predictions one frame at a time, but left and right context is added on to some central frame. The net then makes a prediction as to the identity of the phoneme which produced the central frame.

1. FEATURE EXTRACTION
    (a) sliding window feature extraction
2. GMM-HMM MONOPHONE TRAINING
    (a) Flat-start
    (b) Baum-Welch re-estimation
3. GMM-HMM TRIPHONE TRAINING
    (a) Phonetic decision tree
    (b) Baum-Welch re-estimation
4. DNN TRAINING
    (a) Frame-splicing
    (b) Gradient descent via backprop
5. DECODING
    (a) WFST Graph compilation
    (b) Viterbi decoding

DNN TRAINING

Recall that the phonetic decision tree from the GMM-HMM model was used to define which triphone states would be merged together, and as such, the number of leaves in the tree equals the number of GMMs to be trained. The identity of each GMM is therefore the label for the training data, and when we train a DNN in Kaldi, we are predicting these same labels. We can view the DNN-hybrid approach as a kind of model transfer, where we first create a generative classifier (GMMs defined by decision tree) and we train a DNN to perform the same task. The number of nodes on the output layer of the DNN, therefore, equals the number of leaves in the previously trained decision tree.



Even though the labels come directly from the GMM model, for the DNN, each training data instance (audio frame) will contain additional contextual information. Given the alignments from the triphone GMM-HMM system, first we splice features in order to create training examples. This step should result in the same number of `(label,data)` pairs as were provided by the GMM system. It is common to splice a large window of frames (up to 10 frames on each side), and then reduce the dimensionality of the data with a some transform (typically LDA).

DECODING GRAPH

Kaldi's approach to graph compilation sets it out from other toolkits. Building on work by Mohri (1997, 2002), Kaldi implements every knowledge source in the ASR pipeline as a Weighted Finite State Transducer (WFST). WFSTs are an elegant choice for speech decoding, because they allow for not only heirarchical combination of knowledge (sentences, words, phonemes), but also the combination of weights from each level of the heirarchy. That is, we can just as easily assign probabilities to sequences of words as to sequences of phonemes. All these sources of probabilities are encoded as weights on a WFST, and during compilation, all that information and bias is saved and used.

Following Mohri, Kaldi creates a single master decoding graph from four (4) WFSTs. The final graph is called HCLG, and it is the composition of subgraphs H, C, L, and G.

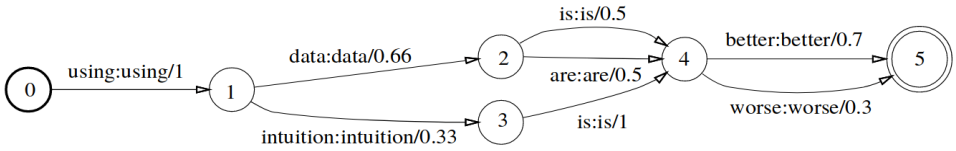
H: pdfs → triphones

C: triphones → monophones

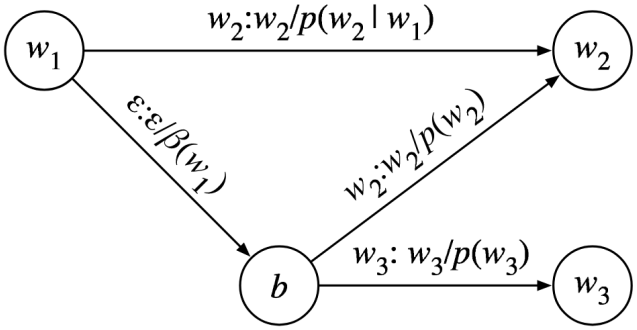
L: words → monophones

G: words → words

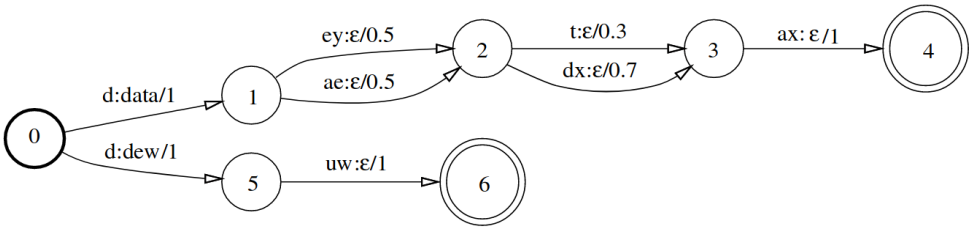
Starting from the highest level and working down, the WFST 'G' represents the grammar defined by the language model. It accepts valid sequences of words, and returns both a weight (pseudo-probability) and that same sequence of words.



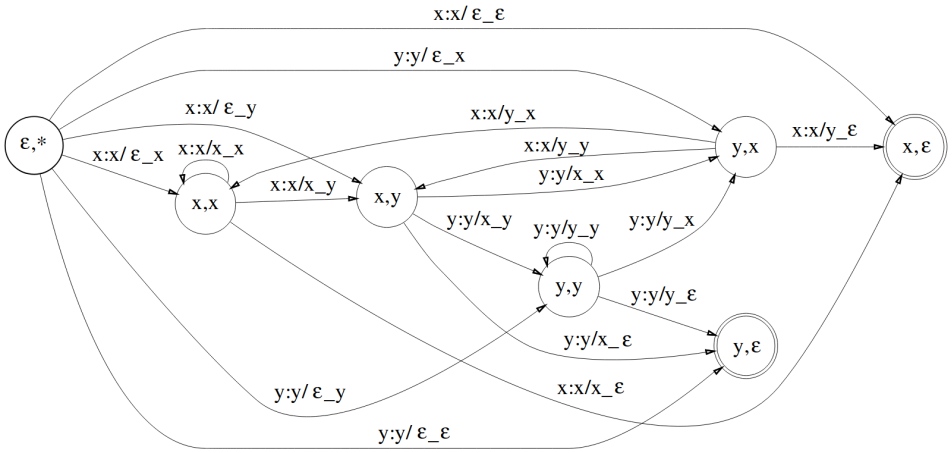
If we use a backoff n-gram language model to define our grammar, then we will have different possibilities depending on how much of the sequence was observed in training. A decoding graph cannot recognize a word it never saw. For example, given a sequence of words (eg. "She codes best in Python."), that sequence will be assigned a higher probability if it was seen contiguously, rather than if all the words were seen apart. If that sentence was never seen in training, but the two sentences: "She codes best in class." and "She prefers C++ to Python.", then the original sentence is grammatical, but the bigram "in Python" was never seen. As such, the backoff weights for "in" and "Python" would be combined. This is the idea behind a backoff language model, and WFSTs naturally encode this with a backoff state. In the following diagram, b is the backoff state, *w1* is "in" *w2* is "class" *w3* is "Python"



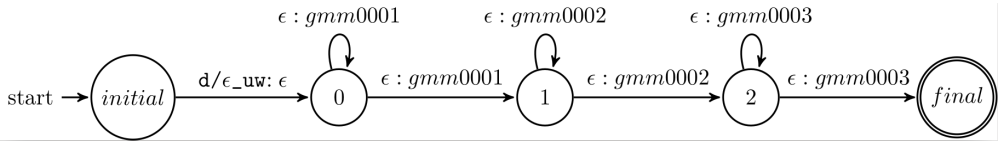
To encode information about word pronunciation from the phonetic dictionary (also known as a Lexicon), Kaldi uses a WFST called L.fst. This FST accepts a sequence of phonemes and returns a word and a weight (along with a string of epsilons, which are ignored).



The next FST encodes information about the triphone-monophone relationship. This FST is called `C.fst` because it contains phoneme context. It translates a sequence of phonemes into a sequence of triphones.



This last WFST encodes all the information from the phonetic decision tree. It accepts triphones and returns state identities. You will notice that all arcs leaving a state have the same label. This is because Kaldi encodes all information about states via arcs. Technically, HCLG does not contain states. They are inferred from the arcs. As such, the H.fst accepts a triphone and returns the IDs of the states of which the triphone is comprised. If we have 3-state models, then C.fst will return 3 states for every triphone. These state IDs are linked to either individual Gaussian mixtures (for GMM-HMM) or output nodes of a neural network (for DNN hybrid system).



3 Overview of Multi-Task Learning

The reason we spend so much time worrying about the number of hidden layers in our neural net and how many nodes are in each hidden layer is because we are afraid of our data. We are afraid that our data has too much noise, and if we give our neural net too much freedom, our data will lead our parameters astray, into a nice and cosy local minimum. We end up in a situation where we punish the neural net for merely following orders (minimize loss on data). We punish it by taking away parameters, even though it could use those extra parameters for good. We know though, that the more parameters we give the net, the more cozy misleading local minima become.

However, in reality, taking parameters away from the net is not fair. We provide the data, we provide the optimization function, and the net does extactly what we told it to do. It would be better if instead of trying to minimize the parameter space as a handicap to the net, we nurture it with more information, changing the role of a researcher from harsh school principal to guiding educator.

Mulitask learning is a way to do exactly this, giving the net more objectives on the same data. With more opjective functions to optimize, we can give the net more freedom via more parameters, not being afraid that the net will be lead astray into overfitting on a local minimum.

With MTL we better equip our nets to take on the world, and can be more secure they will use their resources well.

A task (in classification) is a set of (data,label) pairs.

Most machine learning training uses single-task learning (e.g. classifying an image as a digit [0-9].

In Mutli-Task learning, we learn multiple tasks which share useful information. An example of a non-useful auxillary task is classifying a number as greater or less than 7, when the main task is to identify the identity of a single digit.

4 Background Literature

Here I will cover the literature relevant to working with small (or completely new) datasets. There are two main approaches, (1) adapt a model from one training dataset to a new, smaller dataset; (2) create a model that is robust enough to handle data from multiple domains.

- **Model Adaptation: (e.g. Speaker; Language)**
- **Model Robustness: (e.g. Noise; Channel)**

5 Experiments

This section contains the main contributions of the dissertation research.

This dissertation investigates training methods for acoustic modeling in the Neural Net + HMM ASR pipeline.

I aim to produce acoustic models which perform better (i.e. lower Word Error Rates) on datasets which are not similar the original training dataset.

I investiage the effectiveness of different `Tasks` (eg. linguistic `Tasks` vs machine learning `Tasks`) in a Multi-task Learning framework.

5.1 Data

I am creating acoustic models which generalize well to new data. To measure how well the models generalize, I use a set of speech corpora which exhibit some interesting differences between training and testing data. These differences between corpora exemplify the typical challenges faced in speech recognition generalization.

The training and testing data will differ in either (1) the recording `noise` conditions, (2) who the `speaker` is, or (3) what `language` the speaker is using. The following table shows which data sets are used for each audio condition.

		Corpus	
		Train	**Test**
Audio Condition	**Noise**	TIDIGITS	Aurora 5
	Speaker	LibriSpeech-A	LibriSpeech-B
	Language	LibriSpeech	Kyrgyz Audiobook

Table 1: Speech Corpora

5.2 Model Training Procedure

This dissertation investigates the creation of new tasks for MTL, either using (1) linguist-expert knowledge, (2) ASR Engineer-expert knowledge, or (3) general Machine Learning knowledge.

The former two knowledge sources are useful for buidling acoustic models, but not much else. On the other hand, the final knowledge source (general machine learning concepts) can be applied to *any* classification problem.

The three knowledge sources will be abbreviated as such:

- (LING) **Linguistic Knowledge**
- (ASR) **Traditional Speech Recognition Pipeline**
- (ML) **General Machine Learning**

Each of these categories contains a wealth of ideas, but I will consolidate each into three experiments. With three experiments for each knowledge source, my dissertation will contain nine (9) experimental conditions (for each audio condition).

Specifically, I will use the following concepts to create new tasks to be used in MTL training:

	Knowledge Source		
	LING	**ASR**	**ML**
Experiments	voicing	monophones	k-means
	place	1/2 triphones	random forests
	manner	3/4 triphones	bootstrapped resamples

Table 2: Experimental Setup

Each of these tasks will be added to a Baseline model. More specifically, the Baseline model will be a Neural Net with a single output layer (Task A), and the tasks above will be added as a second task (Task B). You can think of the tasks as simply a new set of labels for the existing data set. For example, when the LING task of VOICING is used, any audio segment labeled `[b]` will be assigned the new label `voiced`.

When these experiments will be applied to each of the three audio conditions, we get the following 30 experiments:

Data Condition	Train Data	Test Data	MTL Training Tasks	Num. Exps
NOISE	TIDIGITS	AURORA 5	Basline	1
			Baseline + LING	3
			Baseline + ASR	3
			Baseline + ML	3
SPEAKER	LIBRISPEECH-A	LIBRISPEECH-B	Baseline	1
			Baseline + LING	3
			Baseline + ASR	3
			Baseline + ML	3
LANGUAGE	LIBRISPEECH + KYRGYZ-A	KYRGYZ-B	Baseline	1
			Baseline + LING	3
			Baseline + ASR	3
			Baseline + ML	3
				30

Table 3: Experimental Setup

5.3 Task Creation Specifics

1. **Baseline**

   All the following architectures will be compared to the performance of the following baseline.

   To account for any advantage mutliple output layers may bring about, the baseline also contains two output layers, where the `Tasks` are identical. In this way, random initializations in the weights and biases for each `Task` are accounted for.

   During testing, *only one* of the `Tasks` is used. The additional `Tasks` are dropped and the `Baseline Triphones` are used in decoding. This highlights the purpose of the extra `Tasks`: to force the learning of robust representations in the hidden layers during training. The `Tasks` may in fact not be the best option for final classification; they serve as "training wheels" which are then removed once the net is ready.

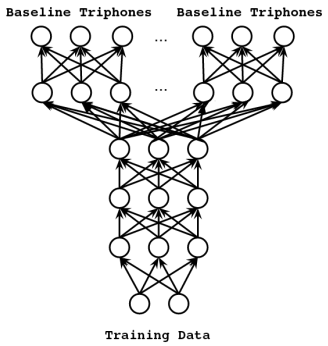


Figure 1: `Baseline`

2. **LING**

All of the linguistic knowledge tasks view the phoneme as a bundle of features.

Using standard features from articulatory phonetics (voicing, place, and manner), the following tasks generate labels for each data point by collapsing the given phoneme labels along one of these three dimensions.

All information from one dimension is removed from the labeled data. This forces the classifier to rely on audio signal features which do not relate to that dimension. The DNN must project the input data into a new space for classification, using only information from the other two dimensions.

(a) VOICING

Voicing information is removed from the data labels.

Speaker Robustness Experiments: The training data is a 4.5 hour subset of Librispeech, with mixed speakers, men and women. The testing data is 30 minutes of speech from two speakers (one man one woman).

First, two separate GMM-HMM models are trained on the training data. The first GMM-HMM model uses the standard CMUDict phoneset (39 phones + stress variants).

From this standard phoneset, the normal 3-state monophones are trained from a flat-start via EM training. A total of XXX states are trained with a total of XXX Gaussian components over XXX iterations of EM. These monophones are then expanded into context-dependent triphones via a phonetic decision tree, with a maximum of XXX leaves. The resulting leaves (state clusters) are then trained with XXX Gaussian components over XXX iterations of EM. The final model acheives a WER of XXX on the testing data.

The second GMM-HMM model trained differs from the first model in its set of initial phones. Instead of building monophones (and then triphones) from the standard CMUDict, this `-Voicing` model collapsed all voicing information from the phonetic dictionary (i.e. the lexicon file).

```
B P   --> P
CH JH --> CH
D T   --> T
DH TH --> TH
F V   --> F
G K   --> G
S Z   --> S
SH ZH --> SH
```

(b) PLACE

All place information is removed from the data labels.

```
F TH SH S HH --> F        voiceless fricatives
V DH Z ZH    --> V        voiced fricatives
P T K        --> P        voiceless plosives
B D G        --> B        voiced plosives
M N NG       --> N        voiced nasal
L R          --> R        voiced laterals
Y W          --> Y        voiced approximants
```

(c) MANNER

All manner information is removed from the data labels.

```
B M V W --> W          voiced labials
P F     --> P          voiceless labials
D Z     --> D          voiced alveolar
N L R   --> R          voiced alveolar2
```

```
T S     --> T          voiceless alveolar
ZH JH   --> JH         voiced postalveolar
SH CH   --> CH         voiceless postalveolar
NG G    --> G          voiced velar
```

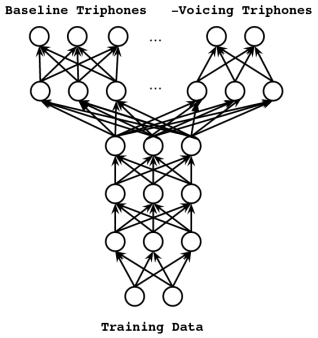


Figure 2: `-Voicing`

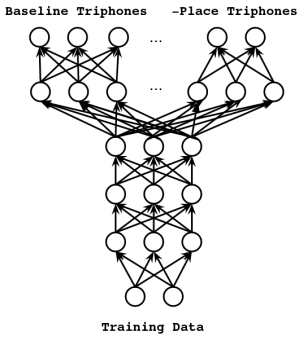


Figure 3: `-Place`

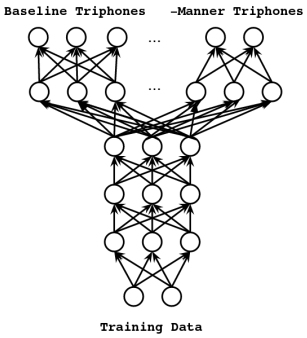


Figure 4: `-Manner`

3. **ASR**

   All the following tasks relate to the structure of the phonetic decision tree used in the traditional ASR pipeline to cluster context-dependent triphones. In GMM training the leaves of the decision tree are then assigned their own Gaussians, and in DNN training the same leaves are used as labels during training via backprop.

   The main intuition behind these experiments is that in using the decision tree labels as targets for the DNN classifier, we are performing model transfer. The decision tree and it's associated Gaussians perform classification, and we are merely training a DNN to perform that same task. So, the decision tree can be thought of as a single task for the DNN to learn.

   However, the DNN only sees the leaves of the decision tree. It doesn't see any of the branches, or any of its wonderful linguistic structure. So, in order to force the DNN to learn the information hidden in the decision tree, the following tasks are like cross-sections of the tree, slicing it from leaves up. The DNN then has to learn how to read these cross-sections, and how to map data onto each layer.

   If we slice the tree at the roots, we have the MONOPHONES. If we slice down half-way (1/2 TRIPHONES), we have more contextual information than monophones but less than full triphones. If go a little farther down (3/4 TRIPHONES), we get even more context, but less general information about the original phoneme.

   (a) monophones
   When we chop the tree at the roots.

   (b) 1/2 triphones
   Chop the tree half-way down.

   (c) 3/4 triphones
   Chopping a little further.

4. **ML**

   The following tasks do not make use of any linguistic knowledge or any part of the ASR pipeline. The only things needed to perform these tasks is labeled data.

   The two approaches above use linguistics or the ASR pipeline to force a DNN to learn structure about the data, because that information is useful for classification.

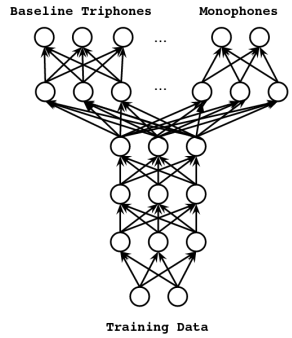


Figure 5: `Monophones`

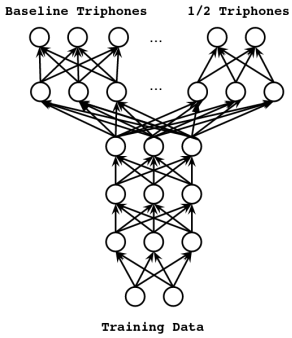


Figure 6: `1/2 Triphones`

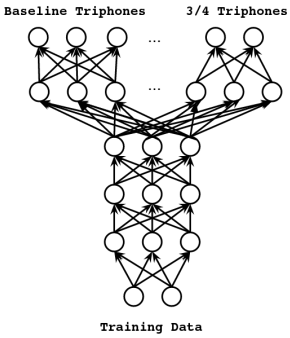


Figure 7: `3/4 Triphones`

We typically do not have this kind of *a priori* information about the datasets we use in Machine Learning. Therefore, an interesting problem is how to learn this structure in a data set when we don't have access to that expert knowledge.

The following tasks force the DNN to learn structure in the data without any knowledge about that structure. In order to do so, I make the assumption that the data does in fact have heirarchical relations. That is, I assume the `(data,label)` pairs were generated by something like a phonetic decision tree, and I try to recover the structure of that tree.

(a) k-means

Standard k-means on the data, with the caveat that labels cannot be split across clusters. A first round of clustering is performed, and then all data from the same original label are shifted to the cluster with the most data points from that label. Then, centroids are recalculated, and data is re-clustered. This adapated k-means should find related data points in the same clusters. If k-means is working, we would expect to be able to recover phonemes (monophones) from the labeled triphone data.

(b) random forest

In another attempt to cluster triphones along phonetic categories, the random forest procedure works as follows: (1) take a random subset of the labels, (2) train a random forest with all data points associated with those labels, (3) re-classify all the rest of the data with the new random forest. In this way, we will reduce the number of labels (eg. out of 2,000 triphone labels I choose 500), and classify unseen data as its closest seen data point.

(c) bootstrapped resamples

In this approach, new labels are not generated at all. The separate tasks for the DNN are just different samples of the data.

Some sub-samples may exhibit a more useful decision plane than others, and if we randomly subsample for multiple tasks, the different decision planes will all have something in common. The individual peculiarities of one sub-sample will have to be ignored for the DNN to perform well on all tasks.

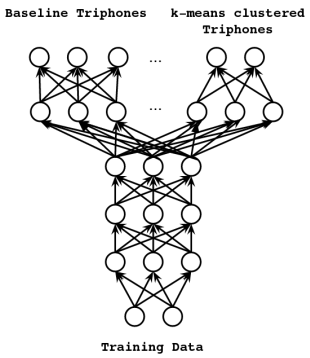


Figure 8: `k-means`

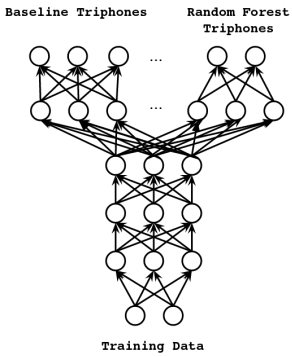


Figure 9: `1/2 Triphones`

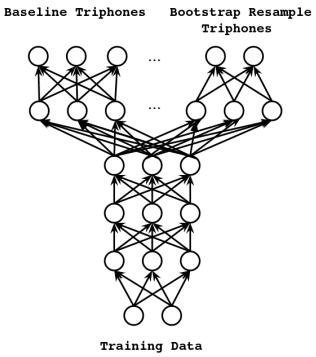


Figure 10: `3/4 Triphones`