Classical and Modern Approaches in Speech Recognition

Long Le
College of Information and Computer Sciences
University of Massachusetts Amherst
lnle@umass.edu

Abstract

Automatic speech recognition (ASR) is the process of transforming spoken language, in the form of acoustic signals, to written text. This field of study, dated back to 1952, has progressed rapidly in recent years due to advances in computation and the rise of deep learning. Speech recognition is a field of broad application but there are still some unsolved challenges. In this paper, we survey the techniques used in speech recognizers, from the classic Hidden Markov Models in the 1980s to Neural Networks today.

1 Introduction

Automatic Speech recognition (ASR) is an exciting subfield at the crossroad of Computer Science, leveraging tools from Natural Language Processing (NLP), Machine Learning, and signal processing. Speech recognition has become the face of Artificial Intelligence (AI) in recent years due to its crucial role in powering popular voice assistants such as Siri, Amazon Alexa, Google Home, and dialogue systems. In fact, if you are calling a reasonably sophisticated entity, such as a government agency or a business, chances are you will first converse with such a dialogue system before being routed to a human representative. ASR also finds application in situations where the user is hand or eye-busy (Jurafsky and Martin, 1999). For example, a car driver would not be able to type out his command, *e.g.* to play radio, but instead have to rely on voice command. The ability of voice-command ASR systems to function reliably is even more important in safety-critical and time-constrained scenarios such as in assisting fighter jet pilots. Another application is *dictation*. Nowadays, Youtube videos or Zoom meeting recordings are shipped with automatically generated subtitles, which can be useful to non-native language learners. In recent years, there are also some work in human-robot teaming using language command (Chevalier-Boisvert et al., 2018; Wandzel et al., 2019). These work currently use simple predefined language command but a more comprehensive ASR system that allows for further natural language processing could be important in the future.

This article aims to provide some background, history, and techniques in ASR. The organization is as follows. Section 2 gives some background on linguistic and the speech recognition task. Section 3 is the history portion on the field. That section is a synthesis of three different sources: Wikipedia, a Medium article and a BBC Future article. Section 4 introduces the concept of a language model, and two popular models: N-gram and neural language model. Section 5 is on Hidden Markov Models (HMMs), which had maintained a monopoly in ASR from the 1980s until the 2000s. Sections 6 and 7 go into details about speech recognizers based on feed-forward and recurrent neural nets respectively. Thanks to advances in hardware, learning algorithm, and the availability of data, these neural-network-based methods have recently become very successful and replaced HMMs as the de-facto standard. Section 8 is the concluding remark.

2 Background

In what follows, we summarize some important background knowledge from Chapter 4 and Chapter 7 of the excellent book (Jurafsky and Martin, 1999).

2.1 Acoustic Signal and Phonetics

Sound is generated by changes in air pressure. A human speaks by expelling air from the lungs through their mouth. To a speech detection system, the acoustic input is often represented as a continuous stream of sound waves in time. The horizontal axis of the diagram represents time, and the vertical represents the change in air pressure (compared to normal atmospheric pressure) at the time.

A high-level abstraction from sound waves is phonetics. Under the phonetics system, an atomic speech element is called a *phone*. A word pronunciation then is a sequence of phones. Two common phonetic standards for American-English are the IPA (International Phonetic Alphabet) and ARPAbet (D*ARPA*- alpha*bet*). This abstraction has the advantage over wave-form representation of eliminating slight variation in pronunciation between different speakers, and external noise to the speech receiver (*e.g.* environment noise to the microphone). However, even under the phonetic system, the mapping from phones to words is not necessarily one-to-one. For instance, the IPA phone [ni] can both refer to the words *knee* or *new* (as in the pronunciation of *New York*). As such, the problem of transcribing phones to text is non-trivial, and generations of computer scientists and linguists have remained employed.

2.2 Text-to-Speech: a learning problem

We will formalize the text-to-speech problem as a MAP (maximum a posteriori probability) estimation. From the continuous wave-form acoustic input, we need to obtain a discrete sequence of data. For example, we can discretize time into intervals of a fixed size (say 10 milliseconds), and compress each continuous interval to a single number (*e.g.* by taking the amplitude at the middle of the time interval). Section 5.1 describes this feature extraction process in more details. The input to a ASR system is a sequence of observations $O = o_1, o_2, ..., o_n$. The task is to find a sentence $W = w_1, w_2, ..., w_n$, composing of words w_i's, that maximizes the *posterior* $\Pr(W|O)$ or equivalently $\Pr(O|W)\Pr(W)$. The prior $\Pr(W)$ is called a *language model*. Note that $\Pr(W)$ can be estimated completely independent of the acoustic input, and represents the *prior* knowledge of the world of the ASR model. For example, ASR model might be able to recognize that the sentence "I love you" makes more sense and thus is more likely a priori than "You love I". Such language models are usually trained using an N-gram model or more recently a neural probabilistic model (Bengio et al., 2003), which we will explain in Section 4. The main task of ASR is to compute the likelihood acoustic model $\Pr(O|W)$.

2.3 Components of an Speech Recognizer

In a typical speech recognizer, the continuous sound waves are first digitized into a sequence of discrete data. The data might be transformed to obtain some useful features (*e.g.* through Fast Fourier Transform). The sequence is then inputted to some model to produce a sequence of words or a sequence of phones. A dominant model between the 1980s and 2000s was the Hidden Markov Model. Other models using recurrent neural networks or deep neural networks are possible, and their construction is explained in the subsequent sections of this article.

3 History of Speech Recognition

In 1953, Bell Lab developed the first ASR system *Audrey* , a huge machine 6 feet tall, that could recognize 10 spoken digits. The machine could only recognize one single digit at a time, with 90% accuracy, and only the voices of some designated speakers. A decade later, IBM came out with *Shoebox*, a much smaller machine, that could recognize 16 words. A breakthrough was made at Carnegie Mellon with the introduction of *Harpy* (Lowerre, 1976) in 1976. This system boasted a vocabulary of just more than 1000 words, and was able to recognize an entire sentence. In the 1980s, Hidden Markov Models (HMMs) became popular and were applied to speech recognition. IBM's Tangora, powered by HMMs, had a vocabulary size of around 20,000 words. HMMs was another turning point where a statical approach to language was taken. By then, speech recognition had come a long way from the primitive days of storing voice patterns from a specific speaker and using spectral distance to match the new voice with existing templates. In the 1990s, speech recognition technology had become sophisticated enough to be commercially feasible. AT&T used this technology to route incoming telephone calls without a human operator. In the 2000s, the spectre of deep learning had finally reached this field in the form of LSTM (Graves et al., 2006) that outperformed traditional methods. In the 2010s, advances in computing power and the availability of data allowed research groups (Hinton et al., 2012) to train deep neural networks speech recognizers. Nowadays, ASR systems have the property of *Large-Vocabulary Continuous Speech Recognition (LVCSR)*. That is they possess a respectable vocabulary size (much larger than that of the average human), and are able to *continuously* recognize natural sentences without pausing. State-of-the-art systems can also often recognize the voice of a new speaker without requiring a period of training to get accustomed.

4 Prior Language Models

The goal of a *language model* (LM), also known as a *grammar*, is to assign to each sentence a probability, indicating how likely that sentence would be encountered in the language. More formally, a sentence is a string of words. We will denote a word as w. Given a sentence $w_1, ..., w_L$ of L words, the LM will compute the joint probability $\Pr(w_1, ..., w_L)$. Often, it does this by computing $\Pr(w_{t+1}|w_1, ..., w_t)$, the probability of the next word given the previous words. For example, given the words "I love", the most likely next word is "you".

4.1 N-gram

To compute $\Pr(w_{t+1}|w_1, ..., w_t)$, the n-gram model simplifies the problem by assuming conditional independence: given the $N-1$ previous words $w_{t-N}, ..., w_t$, the next word w_{t+1} is conditionally independent of further away words $w_1, ..., w_{t-N-1}$. The idea is that only the immediate $N-1$ words should have a say on what the next word is likely to be. For example, for all one knows, I could go on a rant about quantum mechanics before saying the words "I love you". But only the immediately previous words "I love" should have bearing on the prediction of the next word, "you".

N-gram models are constructed by iterating through a large text corpus and counting up all the co-occurrences of groups of N words. Some smoothing technique such as Laplace (add-one) is usually applied at the end.

4.2 Neural probabilistic Language Model

First, a word can be represented by a vector. For example, in one-hot encoding, a very long vector of length equal to the size of the language's alphabet is constructed. The one-hot representation of a word in the alphabet will have a numerical value of 1 at exactly one index and 0 everywhere else. Given word representation, we can compute $\Pr(w_{t+1}|w_1, ..., w_t)$ by a feed-forward neural network (ANN) by feeding the vectors $w_1, ..., w_t$ as the input to the network. The output of the network is a probability distribution over the alphabet, corresponding to the prediction of w_{t+1}. (Bengio et al., 2003)'s idea is to use an ANN to replace the N-gram model, and more importantly to replace the one-hot encoding of the input by a much more expressive continuous embedding. (Bengio et al., 2003)'s main criticism of the N-gram model is that it is not able to group similar words and generalize well. A good LM, they argued, having seen the sentence "The cat is walking in the bedroom" in training, should be able to form similar sentences like "A dog was running in a room" in testing. However, an N-gram model would likely miss the fact that "dog" and "cat" are similar. To construct useful word embeddings, (Bengio et al., 2003) has a projection layer, mapping each word to a continuous vector representation. The projection and the neural network are learned jointly by backpropagation. The resulting embeddings were found to capture the similarity between words well.

5 Hidden Markov Models (HMMs)

We will first build up some background in speech processing, pronunciation models before introducing the Hidden Markov Models.

5.1 Spectral feature extraction

The process of transforming a continuous sound wave to a discrete sequence of feature vectors is called feature extraction. First, the continuous signal (*analog*) has to be *digitized*. The signal is *sampled* at regular time intervals with a sampling rate (*e.g.* 8,000 Hz for telephone speech). For a sampling rate of 8,000 Hz, we only see snapshots of the continuous signal every 0.125 milliseconds. Then, the real-valued signal amplitude is *quantized*, through rounding and truncation, to a finite-precision representation in a computer. The digital data can now be converted into vectors of features, for example through Fast Fourier Transform (FFT) or Linear Predictive Coding (LPC). These features are called "spectral" features.

Now, a vector of spectral features is often (probabilistically) mapped to a phone. (Zhang et al., 1994) provides a popular approach to learn this mapping using Mixture Gaussian Models (GMMs). The training data is a sequence of observations $o_1, ..., o_t$ where each o_i is a high-dimensional spectral vector (12-dimensional in (Zhang et al., 1994)). Analyzing human speech from 33 speakers, (M. Pijpers and Togneri, 1993) observed that the spectral vectors corresponding to a phone in the data were distributed like a Gaussian. (Zhang et al., 1994)'s idea is then to cluster $o_1, ..., o_t$ into N Gaussian clusters where N is the number of phones in the language ($N = 44$ for English). The clustering is done through the EM algorithm. We can later manually inspect each cluster to decide which cluster corresponds to which phone. In the paper, they had a dataset equipped with a phonetic transcript so they utilized the transcript to compute the mean μ_i and covariance Σ_i of each cluster i beforehand, and used those values as the initial starting point for the EM algorithm to boost the performance. The learned model through EM is $b_i(o)$, giving the probability of observing a spectral feature o given that the underlying phone is i.

Another approach to learning $b_i(o)$ is using feed-forward neural networks (usually referred to as multilayer perceptron or MLP or ANN in the literature). This approach is explained in Section 6.1.

5.2 Pronunciation Model

Recall that the main goal is to learn the acoustic model $\Pr(W|O)$. To do so, we first construct a model that predicts the pronunciation of a single word: $\Pr(\text{a word} \mid \text{a sequence of phones})$. For each word in a lexicon, we will construct a separate pronunciation model, $\Pr(w|O)$ where w is a single word and $O = o_1, ..., o_t$ is a sequence of phones. Note that there are possibly multiple ways to pronounce one word due to dialect ("tomato" in the sentence "I say tomato [tow m ey t ow], you say tomato [t ow m aa t ow]") or phonological context of the word ("the" in "the end" versus "the university"). A pronunciation model for a word is a weighted directed graph with start and end nodes. Figure 1 is the pronunciation model for the word "tomato". [t], [ow], [m], [ey], [aa], [t], [ow] are the phones. There are two ways to pronounce "tomato". The weight of each edge is the transition probability to the next phone given this word and the current phone.

The different pronunciations of the words are usually decided by manually consulting a pronunciation dictionary such as Pronlex. A pronunciation network such as one in Figure 1 can then be built. It remains to estimate the edge weights in the network. These weights can be obtained by training on a transcribed speech corpus such as the SWITCHBOARD (Godfrey et al., 1992), provided that orthographic and phonetic transcripts are available.

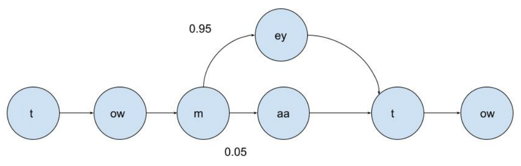

Figure 1: A pronunciation Model for the word "tomato". Following the work from (Jurafsky and Martin, 1999).

For example, in Figure 1 after [m] is pronounced, 95% of the times, when the word "tomato" is pronounced in the corpus, the next phone is [ey] (*i.e.* [tow m ey t ow] is a more common pronunciation), and hence the edge weight 0.95 shown.

When the corpus is shipped with an orthographic transcript (text of words *e.g.* "I need") and a phonetic transcript (text of phones *e.g.* [aa n iy]), constructing the pronunciation model is as simple as counting up the utterances in the corpus. (Wooters and Stolcke, 1994) introduced an automatic way to obtain pronunciation models from corpus where only an orthographic transcript is available. We will come back to explain their method in Section 5.4 once we have described the Viberti algorithm in the next section.

5.3 Bigram Viberti Algorithm

We adopt a mix of notations and terminology from (Jurafsky and Martin, 1999) and (Gales and Young, 2008). Recall that we would like to find the optimal sentence W^* that maximizes $\Pr(O|W)\Pr(W)$ where O is a sequence of spectral vectors. Note that for a sentence, there are possibly multiple ways of pronunciation (see Figure 1). Let $Q = q_1, ..., q_t$ be a sequence of phones, denoting a pronunciation of W. Then,

$$\Pr(O|w) = \sum_{Q} \Pr(O|Q)\Pr(Q|w). \tag{1}$$

Note that $\Pr(O|Q)$ is the emission or the observation probabilities that can be computed by a Mixture Gaussian or MVP in Section 5.1. $\Pr(Q|w)$ is the transition probability obtained from a pronunciation model as explained in Section 5.2. Thus, in principle, it is possible to obtain W^* by comput-

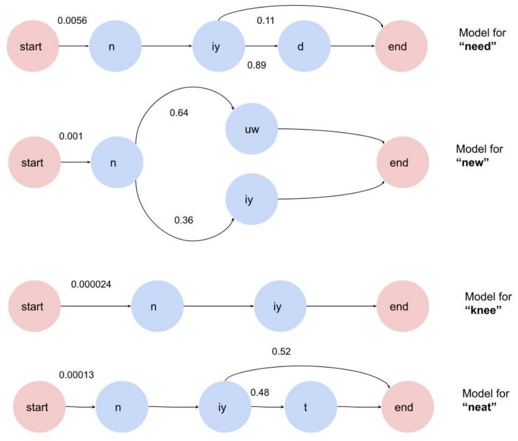

Figure 2: Single-word-from-phones recognition: individual pronunciation models. Following the work of (Jurafsky and Martin, 1999).

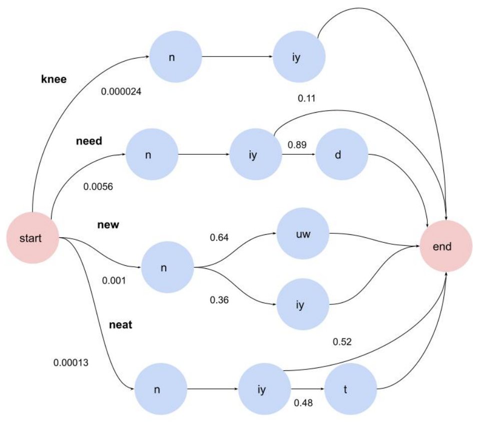

Figure 3: Single-word-from-phones recognition: composite model. Source:(Jurafsky and Martin, 1999)

ing $\Pr(O|W)$ using the expression above for every possible W. However, the number of possible sentences and pronunciations can be exponentially large so it is important to search for W^* efficiently. The search problem is called *decoding* the sound signal.

A popular method is the Viterbi algorithm (Viterbi, 1967), originally developed in the field of information theory. (Vintsyuk, 1968) first applied the algorithm to the problem of recognizing a single word. They achieved no error in an experimental test consisting of 600 pronunciations of 12 words from a single speaker. For simplicity, let us also start with the problem of recognizing a single word from a sequence of phones. Suppose that we'd like to match the phones [n iy] to a word. The possible candidates are "need", "knee", "new" and "neat". Figure 2 shows different pronunciation models for each of those words. Note that the first edge weight in each model has been augmented with the unigram probability from a prior language model. For example, 0.001 in the first edge of the word "new" indicates that this word appears in the corpus 0.1% of the time. We form a *composite model* by combining all the pronunciation networks into one big model (Figure 3). Note that in Figure 3, different states labeled [n] corresponding to different words are treated as distinct. As in a Hidden Markov Model, each node of the graph in Fig. 3 is called a (hidden) state. We assume that states are conditionally independent of other states given the current state. There are transition probabilities a_{ij}'s from one state to another. We also assume that each hidden state can emit an observation. The observation is conditionally independent of other observations given the state that generates it. The emission probability $b_i(o)$ gives the likelihood of emitting observation o in state i. Note that the graph in Figure 3 only contains a at the moment. When we are given a sequence of phone observations (*e.g.* [ni iy]), the composite model's edge weights can be modified further to include the emission probability b. For example, the edge to [d] would be zero-ed out since state [d] cannot emit any phone in the observation sequence [ni iy]. We usually denote the augmented graph as $\lambda = (a, b)$. The question becomes: given the graph λ, find the "best" path from the goal state to the end state. [1] The value of a path is $q_1, ..., q_t$ is $\prod_{i=1}^{t} a_{q_i,q_{i+1}} \cdot b_{q_i}(o_i)$. The Viterbi algorithm is an efficient dynamic programming approach to find such a best path (it is essentially an analog of Dijkstra's algorithm in the shortest path problem).

The problem of decoding a sequence of words is more difficult. One main problem is segmentation. For example, consider the phones [aa n iy]. They can either be segmented into *I need* [aa — n iy] or *on eat* [aa n — iy]. We need to find the word boundary between the phones. Also, it is not clear a priori whether [aa n iy] corresponds to one word, two words or three words. Recall that this problem does not exist in the single word model above: we are told that the entire phone sequence corresponds to only one word so there is no word boundary

[1]In single-word recognition, a word is spoken in isolation. We create two artificial states, "start" and "end", to denote the beginning and the end of a word utterance. In continuous word recognition, words are spoken naturally one after another so it is hard to determine the end state or the end of a word utterance.

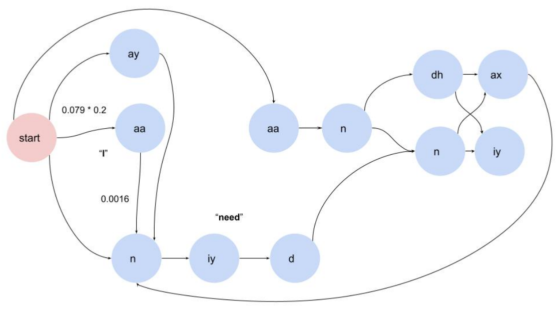

Figure 4: Multiple word recognition: composite model. Following the work of (Jurafsky and Martin, 1999). Only some edge weights are shown to avoid cluttering.

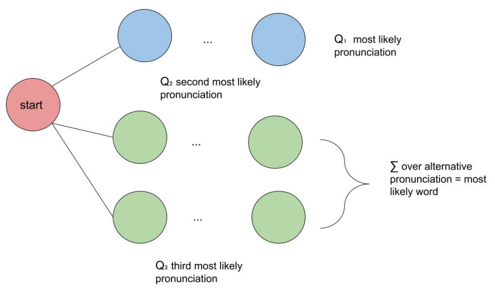
...
Q_1 most likely pronunciation
Q_2 second most likely pronunciation
start
...
$\sum$ over alternative pronunciation = most likely word
...
Q_3 third most likely pronunciation

Figure 5: Viterbi's limitation: only produce the most likely state sequence. In the picture, Q_2 and Q_3 refer to different pronunciations of the same word, w, while Q_1 is the pronunciation of a different word w'. Although Q_1 is the most likely pronunciation, w is the most likely word because the sum of likelihoods of Q_2 and Q_3 is greater than the likelihood of Q_1.

in between the phones. The solution is to use a bigram grammar in the composite model. From the pronunciation networks, we obtain the bigram composite model (Figure 4) by having edges from one word to another. For example, [aa] is a pronunciation of the word "I". The word "I" appears in the corpus 7.9% of the time, and the [aa] pronunciation appears 20% of the times within "I". [n iy d] is a pronunciation of the word "need". The (bigram) phrase "I need" appears in the corpus 0.16% of the times, and hence the edge weight 0.0016 from [aa] to [n]. We can run the Viterbi algorithm on this bigram composite model and find the most likely pronunciation path.

However, note that this path does not necessarily corresponds to the most likely word. Recall from Equation 1 that a word might have multiple pronunciations (multiple paths). This is the first limitation of the Viterbi algorithm. In Figure 5, we illustrate this idea that it might be the case that the most likely word does not contain the most likely path but the sum of its path probability exceeds that of any other word. The second problem is that the Viterbi algorithm can only be run on a bigram language model. Figure 6 illustrates why the Viterbi algorithm would fail for a trigram model. The crucial dynamic programming assumption is that if the best path contains a state i then the path from the start goal to i has to be optimal as well [2]. This assumption breaks down for trigram and higher N-grams. For example, conditional on "I love", the word "you" is the most likely next word. However, that does not necessarily mean that "I love" is the most likely bigram. Perhaps, "we love" is more common in the corpus but "we love you" is not as common as "I love you".

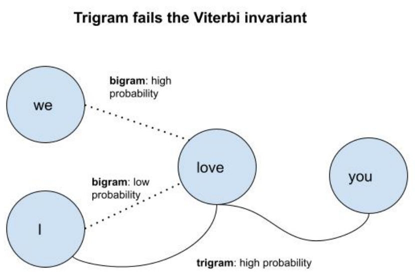
Trigram fails the Viterbi invariant
we
bigram: high probability
love
you
bigram: low probability
I
trigram: high probability

Figure 6: Viterbi's limitation: only applicable to bigram prior language model.

An alternative algorithm to Viterbi is A^*, also known as stack decoder, (Jelinek, 1969). This is a best-search on the lattice of words. A^* uses heuristics, called match match, to select the next most likely word to explore. The downside is that the run-time complexity of this search is hard to control (Gales and Young, 2008). Another fix is to use a Viterbi-like algorithm (Schwartz and Austin, 1990) that returns the N-best state sequences. The same sentence can appear multiple times in the N-best output due to different pronunciation. These sentences can then be filtered by another language

[2]The argument is that if there is an alternative P' path to i that is more optimal than the current path P, then we should replace P by P' to obtain a more globally optimal path through i.

model (*e.g.* a trigram). Different state sequences of the same sentence (different pronunciations) can be summed up to give a more accurate approximation of the sentence's score. More generally, the purpose of the N-best algorithm is to quickly winnow down the exponentially large space of all possible sequences to only N sequences. Then, another knowledge source (*e.g.* a trigram language model) can then be used to select the best candidate.

5.4 Estimating transition a and observation probability b

Now that we have understood the Viberti algorithm, it is time to return to the problem of automatically learning the pronunciation model from orthographic speech corpora. (Wooters and Stolcke, 1994) first obtained a graph of different ways of pronouncing a word such as the graph in Figure 1 by manually consulting a pronunciation dictionary. The transition probability model a is initialized to uniform. Then, a is iteratively updated through an EM-like procedure [3]. In the "adaption" stage, the Viterbi algorithm is applied to the speech data using the existing a to generate a synthetic phonetic transcript. The transcript is then used to update a in the "re-estimation" stage. Note that in section 5.1, we compute b from Gaussian Mixture Models in an independent phase. But from the discussion just now, it is clear that we can also re-estimate both a, b in this "re-estimation" stage. To sum up, we first initialize a, b uniformly. Then, we run the Viterbi algorithm using the current a, b to obtain the probability of each sentence. Using that output, we then re-estimate a, b. This alternating procedure is then repeated.

The parameters of the HMM model $\lambda = (a, b)$ can also be jointly estimated using the forward-backward (Baum-Welch) algorithm ((Jurafsky and Martin, 1999) Appendix D, (Gales and Young, 2008) 2.2). The algorithm is a dynamic programming EM approach. We are given a sequence of spectral vectors $O = o_1, ..., o_T$. In the E-step, we compute γ_t^i, the probability of being in state i of the composite model at time t given the observations O and given the current parameters λ_t. γ_t^i is computed by combining the *forward probability* $\alpha_t^i = \Pr(\text{in state i at t}|\lambda_t, O_{1:t})$, and the *backward probability* $\beta_t^i = \Pr(O_{t:T}|\lambda_t, \text{in state i at t})$. α and β are calculated by two passes of dynamic programming. In the M-step, λ is updated to maximizes the likelihood of the data O.

[3]EM stands for "expectation-maximization"

6 Feed-forward Neural Network

In this section, we explain two main ways that feed-forward neural networks have been used in speech recognition: (1) as shallow, supporting net in hybrid models with HHMs or (2) as deep, powerful nets to replace HMMs all together.

6.1 ANN-HMM Hybrid system

Feed-forward (non-recurrent) neural network, known in the literature as artificial neural network (ANNs), has been used in conjunction with HMMs from at least 1960s. These are known as ANN-HMM hybrid system. ANN can be used to (1) compute the emission probability in lieu of the Mixture Gaussian Model or (2) pre-process the observations or (3) post-process the sentence predictions (Bengio, 1996).

(Austin et al., 1991) introduced a way to post-process the predictions from an HMM using an ANN. Their idea is to use the N-best algorithm introduced before to generate the N-best sentence hypotheses using an HMM. Then, an ANN is used as an acoustic model to score each sentence. They constructed a segmental neural net (SNN), which is an ANN that takes as input a fixed number of spectral vectors (frames) and outputs a distribution over the phones. Note that in our previous discussion of HMM in section 5, we have largely assumed that each spectral frame is mapped to exactly one phone. However, in practice, depending on the speed of speech, it might take a speaker several frames to pronounce one single phone. Thus, this ANN has to take in as input a number of frames instead of a single frame. The fix of this issue of many-frames-to-one-phone in HMM is to include self-loop at each state. Here, the number of frames inputted to an ANN is fixed but the number of frames associated with a phone is variable. Thus, (Austin et al., 1991) used time-warping algorithm to reduce a variable-length frame sequence to fixed length (*e.g.* if the fixed size of the phonetic segment is 5 and the number of frames associated with the phone [ni] is 17, we will judiciously discard 12 frames). As such, the SNN can function as a likelihood acoustic model and assign a score to each sentence hypothesis based on how likely the spectral frames to produce such sentences. The second knowledge source usually consists of multiple scoring models. Thus, SNN score is usually added with the HMM score and a grammar score (*e.g.* generated from a trigram language model) to produce the final score for best sentence selection.

ANNs can be used to estimate the emission probability in the HMM. (Bourlard and Wellekens, 1989) trained an ANN to output a distribution over phones given several frames much like (Austin et al., 1991) did. There is a centered frame o_t and a context window is expanded to include $o_{t-L:t}$ and $o_{t:t+L}$ where L is the window size. Let q denote a state in the HMM. The ANN computes the state likelihood $\Pr(q|o_t)$ but what we want is the emission probability $\Pr(o_t|q)$. By Bayes' rule, $\Pr(o_t|q) = \frac{\Pr(q|o_t)\Pr(o_t)}{\Pr(q)}$. $\Pr(o_t)$ is a constant so for all intents and purposes of the HMM training, $\Pr(o_t)$ is not computed and ignored. $\Pr(q)$ is computed by looking at the training dataset of $o_1, ..., o_T$, and sum the ANN's outputs up *i.e.* $\Pr(q) = \sum_{i=1}^{T} \Pr(q|o_i)$, where $\Pr(q|o_i)$ is read off from the output of the ANN. ANN-emission has the same performance as mixture Gaussian but has fewer parameters and is longer to train (Jurafsky and Martin, 1999).

(Bengio et al., 1992) used ANNs as a preprocessor of inputs to the HMM that was used for a phone recognition task. They have three ANNs. These ANNs extract high-level features from digital signals. For example, ANN2 extracts "nonnasal sonorant, nasal, plosive, fricative, and silence" from the signal. The ANNs were trained (jointly with λ of the HMM) on labeled data by back-propagating the gradient from the HMM decoder. The authors showed that ANN+HMM architecture outperformed ANN + (some other postprocessor) like DP (Robinson and Fallside, 1991). What could have also been interesting to see to illustrate the power of ANN as a preprocessor is a comparison between ANN+HMM and (spectral features) + HMM.

6.2 Deep neural networks (DNNs)

(Mohamed et al., 2011) used a deep belief network (DBN) from restricted Boltzmann machines (RBMs) to beat traditional HHMs on the TIMIT corpus.

(Hinton et al., 2012) is a review paper from University of Toronton, Microsoft, Google and IBM about using deep neural networks (DNNs) to do acoustic modeling. Using ANN for acoustic modeling was not a new idea as already explained in the last section. But shallow ANNs were not able to outperform Gaussian Mixture HMM much. It was only due to the progress on hardware and new learning algorithms that made training DNNs possible. DNNs are often trained by initializing the weights randomly and then do discriminative backpropagation. But instead of random weight initialization, (Hinton et al., 2012) explains a method of initializing good weights by "generative pre-training". The idea is to stack (RBMs) to make a DBN. Each RBM is generatively trained. The generative weights of the DBN are then used to construct a DNN. That DNN is subsequently trained by backpropagation. Their DNN achieved a phone error rate (PER) of 20% against 25.6% of the Mixture Gaussian-HMM on the TIMIT corpus.

7 Recurrent Neural Networks (RNN)

7.1 Connectionist Temporal Classification (CTC)

The explanation in this section is largely based on (Jurafsky and Martin, 2020). (Graves et al., 2006) introduced a paradigm for training RNNs for speech recognition. The main challenge of using an RNN is that the input data (spectral vectors) are unsegmented. If the RNN output was a word or a character, it would not be clear what would be the word boundary between the spectral vectors for the input. Recall that in Section 5.3, we overcome this problem by using a bigram HMM. (Graves et al., 2006)'s insight is to use an RNN to output a character at each time frame anyway. The resulting sequence of characters is called an alignment. Due to the problem of many-frames-to-one-phone or many-frames-to-one-word we have discussed before, the alignment would likely contain a lot of contiguous duplicate characters. They then apply a collapsing function to the alignment to remove duplicate characters. More precisely, at each time frame, the output of the RNN is a probability distribution over the letter alphabet. The probability of an alignment is just the product of the member character at each time frame.

Note that the collapsing function is many-to-one. Thus, the probability of a sentence is the sum of all possible underlying alignments. Since the number of possible alignments is exponentially very large, a modification of the forward-backward algorithm is used instead to approximate the sentence probability. Using these ideas, CTC-RNN can be trained on labeled data by log-likelihood loss and backpropagation. (Graves et al., 2006)'s model outperformed the traditional HMM on the TIMIT corpus by around 8% on label error rate (LER). CTC has the assumption that the network's output is independent of previous outputs given the input. Architecturally, this means that the RNN has no connection from the output back to the network. Recall that HMMs have a very strong assumption that each observation is independent of each other given the state. CTC-RNN requires no such assumption: it takes in as input the previous hidden state, thus more effectively capturing long-range dependencies in the input time series.

Note that the output independence assumption means that CTC cannot implicitly learn or take advantage of a language model. (Graves, 2012) rectifies this by including a separate language model called the prediction network. The final model takes in as inputs the hidden states of the language model and the CTC network. Their RNN-transducer achieved the state-of-the-art phoneme error rate for RNN-based models on the TIMIT corpus.

7.2 Sequence-to-sequence problem

The ASR problem can be thought of as a problem of learning a map from a sequence to another sequence. Often, the input and the target sequences have different lengths. In ASR, the input sequence is spectral frames, and the output is a sequence of phones or words. Machine translation (MT) is another field in NLP where we have a sequence-to-sequence problem. In MT, we are trying to map a sentence in one language to another. The input and target might have different lengths (*e.g.* the word "apple" in English is translated to "trái táo" in Vietnamese). Most popular models in MT include encoder-decoder (Sutskever et al., 2014), which consists of one RNN-encoder and one RNN-decoder, and transformers (Vaswani et al., 2017), which uses self-attention mechanism. However, in ASR, the input spectral sequence is often much longer than the phonetic output. Thus, there is usually some *subsampling* procedure to reduce the input sequence length (Jurafsky and Martin, 2020). A simple procedure called *low frame rate* is to stack three consecutive spectral frames into one long vector, thereby cutting the sequence length by a factor of $1/3$. After sub-sampling, we can use those sequence-to-sequence architectures in MT for ASR.

A language model can also be incorporated into the sequence-to-sequence (encoder-decoder or transformer) scheme by the N-best paradigm discussed in Section 6.1. A variant of n-best search (Schwartz and Austin, 1990) is run to produce the n most likely sentences from the sequence-to-sequence model. Then, a language model is applied to the n-best candidate to choose the best sentence.

8 Discussion and Future Work

Automatic speech recognition has a long history from the 1950s. It has evolved from matching voice templates to using Hidden Markov Model (HMMs) to using deep neural networks in the current day. The main challenge in ASR is to have an accurate prediction in a noisy environment with multiple speakers (Jurafsky and Martin, 2020). It is observed that ASR models work much better in a quiet environment with deliberate pronunciation (*e.g.* audiobook) than a noisy environment with conversational speech (*e.g.* at a dinner table). (Jurafsky and Martin, 2020) shows that the performance of ASR models in a noisy dinner environment is much worse than clean audiobook. As discussed in the Introduction, one application of ASR technology is in machine assistants. A Techcrunch article claims that ASR models do not work very well for children since their pronunciation tends to be very erratic and they do not have proper grammar. This is a problem for voice assistants that work with kids. (Jurafsky and Martin, 2020) also observes that speech systems do not work well for speakers who possess accents and dialects that the systems were not trained on. These are important limitations in real-combat or rescue scenarios where human operators would speak in heightened voice, in short incomplete phrases without proper grammar, and so on. Thus, these problems can limit the use of speech recognition technology to its full potential. Then, perhaps a direction for future work is to develop better preprocessing and feature extraction methods to de-noise the sound waves.

References

S. Austin, G. Zavaliagkos, J. Makhoul, and R. Schwartz. 1991. A hybrid continuous speech recognition system using segmental neural nets with hidden markov models. In *Neural Networks for Signal Processing Proceedings of the 1991 IEEE Workshop*, pages 347–356.

Y. Bengio, R. De Mori, G. Flammia, and R. Kompe. 1992. Global optimization of a neural network-hidden markov model hybrid. *IEEE Transactions on Neural Networks*, 3(2):252–259.

Yoshua Bengio. 1996. Markovian models for sequential data. *Neural Computing Survey*, pages 129–162.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155.

H. Bourlard and C.J. Wellekens. 1989. Speech pattern discrimination and multilayer perceptrons. *Computer Speech & Language*, 3(1):1–19.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2018. Babyai: First steps towards grounded language learning with a human in the loop. *CoRR*, abs/1810.08272.

Mark Gales and Steve Young. 2008. The application of hidden markov models in speech recognition. *Foundations and Trends® in Signal Processing*, 1(3):195–304.

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

Alex Graves. 2012. Sequence transduction with recurrent neural networks. arXiv:1211.3711.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 369–376, New York, NY, USA. Association for Computing Machinery.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

F. Jelinek. 1969. Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, 13(6):675–685.

Daniel Jurafsky and James H. Martin. 1999. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (1st ed)*, 1st edition. Prentice Hall PTR, USA.

Daniel Jurafsky and James H. Martin. 2020. *Speech and Language Processing (3rd ed. draft)*, 3rd edition. online, USA.

Bruce T. Lowerre. 1976. *The Harpy Speech Recognition System.* Ph.D. thesis, CMU, USA. AAI7619331.

M.D. Alder M. Pijpers and R. Togneri. 1993. Finding structure in the vowel space. In *Proceedings of First Australian and New Zealand Conference on Intelligent Information Systems*.

A. Mohamed, T. N. Sainath, G. Dahl, B. Ramabhadran, G. E. Hinton, and M. A. Picheny. 2011. Deep belief networks using discriminative features for phone recognition. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5060–5063.

Tony Robinson and Frank Fallside. 1991. A recurrent error propagation network speech recognition system. *Computer Speech & Language*, 5(3):259–274.

Richard Schwartz and Steve Austin. 1990. Efficient, high-performance algorithms for n-best search. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '90, page 6–11, USA. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

T. K. Vintsyuk. 1968. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57. Russian Kibernetika 4(1):81-88 (1968).

A. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Arthur Wandzel, Yoonseon Oh, Michael Fishman, Nishanth Kumar, Lawson L.S. Wong, and Stefanie Tellex. 2019. Multi-object search using object-oriented pomdps. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7194–7200.

Chuck Wooters and A. Stolcke. 1994. Multiple-pronunciation lexical modeling in a speaker independent speech understanding system. In *ICSLP*.

Y. Zhang, M. Alder, and R. Togneri. 1994. Using gaussian mixture modeling in speech recognition. In *Proceedings of ICASSP '94. IEEE International Conference on Acoustics, Speech and Signal Processing*, volume i, pages I/613–I/616 vol.1.