Implementations of Automatic Speech Recognition

Garett Stage

April 2018

1 Abstract

Automatic speech recognition, or ASR, is a complex task in Computer Science with the goal of reading speech and interpreting the result into text. It has been studied in great lengths to help with common issues of society. The ability of a machine to interpret spoken language is a daunting task when faced with the constraints of a machine. This study is focused on exploring the possible solutions to ASR and how it has been developed over time. Specifically, the impact of hidden Markov models and neural networks on automatic speech recognition software.

2 Introduction

As the population grows, issues with deafened hearing and texting while driving incidences often increase. A person with deafened hearing, or is completely deaf, still needs a way to understand what a lecturer is saying, and often there may not be a transcript of the speech given readily available at the time of lecture. If a machine would be able to listen to speech and transcribe it into a written language, then it could be used to help people with hearing disabilities. In the same form, a person driving would no longer have to look at their phone to send messages, reducing the risk of using devices while driving. Interactivity in technology is crucial in a world with an ever expanding emphasis on smart tablets, phones, smart watches, and even smart shoes. The ability to expand how we interact with newer technologies and create solutions for these existing problems helps drive the emphasis on speech recognition software.

Automatic speech recognition is the concept of machines using a spoken language and interpreting it in many ways. Speech-to-text is the most commonly used aspect of speech recognition, and is used in many tasks today. While STT helps reduce risks and provide solutions to medical issues, AI with the ability to handle voice commands provide users with high interactivity and only negligible loss. Siri, Alexa, and Cortana are examples of this. Many call centers use automated voice systems to allow callers access to voice commands. Medical

centers use voice recognition software to help doctors relay information about patients to other doctors. The benefits of ASR manifest itself in many ways.

This paper is focused on the systems created through the research in ASR. While many systems exist, such as CMU Sphinx, HTK, Julius, Kaldi, as well as Google and Microsoft API, the objective is to understand how different methods of ASR can be applied to settings of ASR. Computers can not understand speech naturally, however. Computers must think mathematically, and so mathematical models must be created to assist in the machine recognize speech patterns. These models must incorporate large amounts of data and can not feasibly be created by hand, requiring the use of training software. Furthermore, a dictionary is required as a reference to what words these systems can use as a knowledge base. Because of the complexity of this problem, a way of deterministically recognizing speech is not currently possible. With enough training, probabilistic models of speech recognition can perform extremely well, but often suffer from errors in transcription. The most common method of speech recognition is in using hidden Markov models, or HMM. Research has found that HMMs are the most efficient way of probabilistically converting speech into text. While this is the most common, many other methods have been developed over time with varying results. Neural networks, Bayesian filters, and beam searches are all commonly seen as methods in developing ASR software with varying results for different settings. In this case, we will be discussing the use of HMMs and neural networks for use in speech recognition. The data being tested is from the speech corpus provided by Voxforge[19] using the open-source code provided by the CMU Sphinx team and Julius for experimentation.

3 Textual Analysis

3.1 Early History

Speech recognition is a category of software with the aim to allow for a computer to recognize a spoken language, and convert it into a written language. Many types of speech recognition exist, focusing on one or many: voice recognition, clarity, speech to text, and speech modification. Continuous speech recognition is the form of speech recognition in which speech is not broken into parts, but spoken as it would be naturally. The focus in this discussion will be on the history and background research into automatic speech recognition algorithms.

Dr. Raj Reddy was one of the pioneers in developing software for continuous speech recognition. While much of the research in ASR was held in Japan, its application relied on breaking speech into parts for translation[15]. In one of his first papers on the topic, Dr. Reddy and others collaborated in 1976[16] to create a machine capable of taking voice inputs to make moves in chess. This system was called Hearsay-I and was the precursor to the 1980s version Hearsay-II. Hearsay-II was designed to be applied to a wider range of problems, and consisted of multiple independent systems working together to create best guess solutions to speech recognition[7]. These models aimed to separate

the workings of the speech center in the brain and combine them cooperatively as "knowledge sources". As the field expanded, algorithms and methods became known that made speech recognition systems more effective. The HARPY speech recognition system, developed by researchers at Carnegie Melon University (CMU), was an attempt to understand the usefulness of different methods for speech recognition. While Hearsay-II used separate procedures for creating a knowledge base, another CMU development by the name of Dragon used Markov networks to create the knowledge base for speech recognition[12]. The Markov network utilized by Dragon consisted of multiple Markov models, each maintaining one part of the recognition process. Each Markov model would analyze the parts of data provided to it by various systems in the architecture. It would then send its output to the next level, in which James Baker described as a "hierarchical system"[1]. Dr. Lowerre's paper explains that "systematic performance analysis of various design choices of these two systems resulted in the HARPY system"[12].

Research in Mathematics led to the creation of theory behind Markov Models in 1966 by Leonard Baum[2]. Because of the time period, and the early stages of this theory, HMM was simply a resource for research and was not fully developed for practical use until the early 1980s. Lawrence Rabiner was one of the leaders in developing this technology for use, with his scientific paper bringing light to its use in speech recognition in 1989[14]. HMM used an extended Markov process to probabilistically determine finite state spaces. In speech recognition, each state is a subset of the spoken language with connections to the possible speech patterns. A hidden Markov Model allows for each state to probabilistically determine the outcome based on the entire speech and the context to which the machine already has. With this, the ability to automatically build large dictionaries led to an expansion in the development of speech recognition. Hidden Markov Models then became one of the main technologies for creating speech recognition software.

3.2 CMU Sphinx and Julius

Today, many applications of HMM are seen in commercial use. The CMU Sphinx-4 system is one of the more openly sourced programs for research and testing. It uses HMM-based models to score and decode speech given a set dictionary. The Sphinx system performs well partly because of the way it relaxes the constraints of other systems[9]. Some of these include: small vocabulary, speaker dependence, and specific word constraints[11]. Over 20 years, the research team at CMU developed Sphinx, and the result is a system with high correctness and efficiency with support for many different languages. Sphinx is also documented extremely well, and is maintained to be used as an open source API for speech recognition. The Java API from Sphinx is simple to incorporate into any Java build, but has a static acoustic model that can not be easily configured after deployment. Comparatively, Julius is another open sourced speech recognition platform that uses HMM with a separate focus from Sphinx. Julius is an open source project developed in Japan by Lee Akinobu

in 1997[10]. One difference is that Julius relies on a large vocabulary that tends to higher correctness than small dictionaries, and can expand more of the search space. These types of systems are denoted as large vocabulary continuous speech recognition systems, or LCVSR. Large dictionaries have issues with modularity, and necessitates the use of common platforms as explained in the paper by Lee Akinobu[10]. This is because training data used to maintain large vocabularies requires a much more specific training set. For example, Sphinx and Julius only allow .wav files for transcription, but CMU Sphinx allows for 16KB and 8KB sampling rate, compared to Julius requiring a 16KB sampling rate for most models. The C/C++ source code of Julius makes it simple to deploy executable programs including the Julius base. This makes its API well suited for highly deployable projects with the ability to reconfigure the acoustic model as needed.

3.3 Kaldi

Kaldi is another application designed for speech recognition. Unlike Sphinx and Julius, Kaldi is most efficient when trained through the use of recurrent neural networks (RNN). Neural networks began being studied for use in speech recognition in 1993 through research done by Herve Bourlard and Nelson Morgan[3]. Developments led to the paper by Bourlard and Morgan in 2012[4] to connect HMM and RNN methods for ASR. By using a neural network to train the system, the system can save time in modifying the HMM bias values. The trained network can then take input data and be used as a language model for the audio sample. This model can then be used to help the HMM probabilistically determine states more efficiently. Because of the efficiency in using HMM for creating solutions to this problem, methods implementing a neural network also use hidden Markov models. As technology continues to expand, neural networks alone may solve the issue of inconsistency in training and output of the system[8].

Sphinx, Kaldi, and Julius are the most popular examples of the systems out providing a solution to speech recognition. Modern approaches to speech recognition have developed to use many technical was to reduce search space: bidirectional searches, Gaussian pruning, and layered n-gram models. The problem of speech recognition is complicated in many ways and comes in many different forms. The varying technologies in research give evidence to the nuances of this problem. While Sphinx is great in recognizing speech with a small initial vocabulary and little use of filters, Julius becomes more efficient as the vocabulary increases. Kaldi shows how versatile software can be to cover the problem space and still provide efficient results. The history behind this problem shows how far research has come, and demonstrates what is still left to be developed.

4 Experimentation

4.1 Systems

The systems chosen for experimentation are the CMU Sphinx-4 and Julius. While many exist, these were chosen because of the specific difference in technical specifications. Sphinx-4 is comprised of a directed acyclic graph, a language-HMM, an internal grammar, a dictionary, and an acoustic model. The input is routed through a series of systems designed to break the speech into a series of independent words, expand the space of probabilities by using an HMM for each word, and then heuristically search each HMM with the use of pruning and beam searches. This search allows for each step to be weighted on its accuracy based on models given to it in the form of the acoustic model. These acoustic models can be "controlled directly or through any application-enforced learning algorithm"[9]. The function produces an output for each word it believes is correct as a hypothesis. This allows the dictionary to be smaller, with more emphasis on training the acoustic model. The focus in Sphinx-4 is to reduce the search space of the algorithm by Gaussian pruning and beam search with a decreasing beam size.

Julius is comprised of a two-pass system. This means that the search space is iterated once, reduced and then iterated a second time to be verified for accuracy. Instead of a graph, Julius uses a "tree-structured lexicon"[10]. Beginning by parsing the speech into words, much like Sphinx, Julius analyzes each word as a frame, using a beam search algorithm to find the 1-best values in the lexicon tree. In reference, Sphinx uses an HMM at each word where Julius uses a tree. 1-best in this case defines the amount of words in each frame to be searched. 2-best would allow for word pairs to be used, allowing for contextual interpretations of speech. Dr. Akinobu explains why this is unnecessary in his specifications "The degradation ... in the first pass is recovered by the tree-trellis search in the second pass"[10]. It then computes a heuristic score with help of the acoustic model. On the second pass, the procedure matches closely to Sphinx-4, with additional pruning and the help of Gaussian mixture selection.

4.2 Implementation

In order to properly use these systems, certain models and data had to be present. The first task was to find training and testing data in a form that could be used in both systems. Voxforge is a source of information on all open source speech systems and includes acoustic models and speech corpuses for testing[19]. 8KB and 16KB sampling frequencies were used for each test in Sphinx to be used in accuracy and speed comparisons. Due to the specifications of Julius, 8KB frequencies were not able to be tested, and many attempts to use 8KB speech were noticeably mangled. Acoustic models were then needed to help the algorithm recognize wave forms. Because of time constraints, training an acoustic model was not an option. Sphinx-4 and Julius both state that training acoustic models requires roughly 60 hours of material and hundreds of different

speakers[17].

Acoustic models sourced by the Sphinx-4 team were used with a 64KB dictionary[6]. The language model used in both systems was provided by CMU Sphinx on setup and was adapted to be used in the C language for Julius. The source API for CMU Sphinx-4 is readily available on their site[5], as well as Julius' source code[13]. Two separate models were used for testing Julius. They were obtained from the Voxforge repository for acoustic models[18]. Julius-DNN stands for Julius Dynamic Neural Network. Dynamic neural networks allow training an acoustic model with a sense of time, and while the training was done with neural networks, the system focuses on HMM. Julius-GMM was trained using Gaussian mixture models. This extracts the procedure of the second pass of the system to increase the acoustic model's ability to better predict frames on the first pass. Both models tested in Julius used a 256KB dictionary, and testing was done on Windows 10 64-bit. Code was adapted from an online source[20] to calculate word error rate given two strings. Word error rate is calculated using 4 variables. The formula is as follows:

$$WER = \frac{S + D + I}{N}$$

S is the number of substitutions made. A substitution is where the system replaces the word spoken with a different word. D is the number of deletions, where a deletion is defined as a word being removed from sequence. I is the number of insertions, where an insertion is defined as a word being added that was not spoken in sequence. N is the total number of words spoken.

4.3 Data Analysis

Initially looking at figure 1, it is noticeable that file length does not determine compile times. Using Sphinx, a file length of 6 seconds resulted in an average run time of 8.37 seconds while the file with 1 less second of speech had a compile time of 10.16 seconds. This can be a result of many factors. One of these factors is the number of words, as more words means the algorithm needs to search more often. This can come down to how fast the speaker talks or the sampling rate of the audio file. Another factor is the complexity of words. In figure 3, the actual speech is given. Row 4 contains the word "shirk," which may cause the systems to search more of the space than is needed. Complex words can be found with a larger dictionary, but it shows that smaller dictionaries yield poor results when more obscure words become present. This was also true in Julius-DNN and Julius-GMM, where compile times are relatively close. It should also be noted that both of these systems are not intended for use in the Windows environment. This may help explain why we see similar compile times for every file except "ar-05.wav" in Julius-DNN and "ar-01.wav" in Julius-GMM.

System	File	File Length (seconds)	Processing Length	WER
Sphinx-4	ar-01.wav	6	10.11	0
	ar-02.wav	5	5.83	3
	ar-03.wav	6	8.37	2
	ar-04.wav	5	10.16	5
	ar-05.wav	4	6.85	0
	a0012.wav	43.5	102.77	34
Julius-DNN	ar-01.wav	6	17.85	0
	ar-02.wav	5	16.35	2
	ar-03.wav	6	17.66	1
	ar-04.wav	5	8.12	7
	ar-05.wav	4	16.31	2
	a0012.wav	43.5	56.49	29
Julius-GMM	ar-01.wav	6	8.45	5
	ar-02.wav	5	15.88	1
	ar-03.wav	6	16.11	3
	ar-04.wav	5	16.55	11
	ar-05.wav	4	16.21	11
	a0012.wav	43.5	57.39	49

Figure 1: word error rate and compilation time given file length using 16KB *.wav files

Low quality is shown to have a heavy impact on these systems. This is seen in figure 1 and 2, as a higher sampling rate tends to higher correctness and lower processing rates. This is because of the way waveforms appear in .wav files. Figure 4 shows the appearance of waveforms to the machine. In 16 KB sampling rate files, waves are much closer and speech is defined much more clearly. This allows Julius and Sphinx to reduce the amount of guessing, or heuristical searching, it has to do in order to transcribe the file. This also means that these systems have more data in a 16KB .wav to parse through, resulting in reasonably close processing lengths for some samples.

File	File Length (seconds)	Processing Length	WER
ar-01.wav	6	11.32	3
ar-02.wav	5	7.16	3
ar-03.wav	6	8.83	3
ar-04.wav	5	16.69	11
ar-05.wav	4	8.60	1

Figure 2: word error rate and compilation time given file length using 8KB sampling rate *.wav files for CMU Sphinx-4

For most of the data used, Sphinx-4 outperformed Julius-GMM and Julius-DNN in execution time. This makes sense because Sphinx is made in Java to be highly portable and easily used as a back end for projects incorporating speech recognition. It fails, however, at beating either Julius engine when files get longer, as with "a0012.wav". This is contrary to what we expect, because of the two pass system incorporated in Julius. Because of the larger vocabulary, it seems that Julius is able to create reasonable guesses for each word without having to explore more of the search space than Sphinx-4 would need. With a larger dictionary, Guassian pruning would become more effective as the system can relax more of the constraints on diction as search space expands. While both systems use this pruning method, this increase in knowledge base gives Julius an edge over Sphinx-4 for longer files. This data also shows that the execution time of Sphinx increases at a faster rate, given speech length, than Julius. This trend is seen in the high fluctuation of execution seen in figure 1. While Julius-DNN and Julius-GMM have a reasonably consistent processing length, Sphinx has a high variance in files of the same length.

Hypothesis	**Actual**
what if there was a young rat named arthur never could make up his mind	once there was a young rat named arthur who never could make up his mind
whenever his friend asked him if he would like to call with them	whenever his friends asked him if he would like to go out with them
you would only answer... i don't know... you wouldn't a yes or no either	he would only answer i don't know he wouldn't say yes or no either
you would all and her make it right.. this article on that and i'm	he would always shirk making a choice his aunt ellen said to him
now look here... no one is going to care for you if you carry on like that	now look here no one is going to care for you if you carry on like this

Figure 3: examples of output and actual speech to text. A series of periods demonstrates pauses in the speech detected by the system. Results found from CMU Sphinx-4 on 8KB sample data

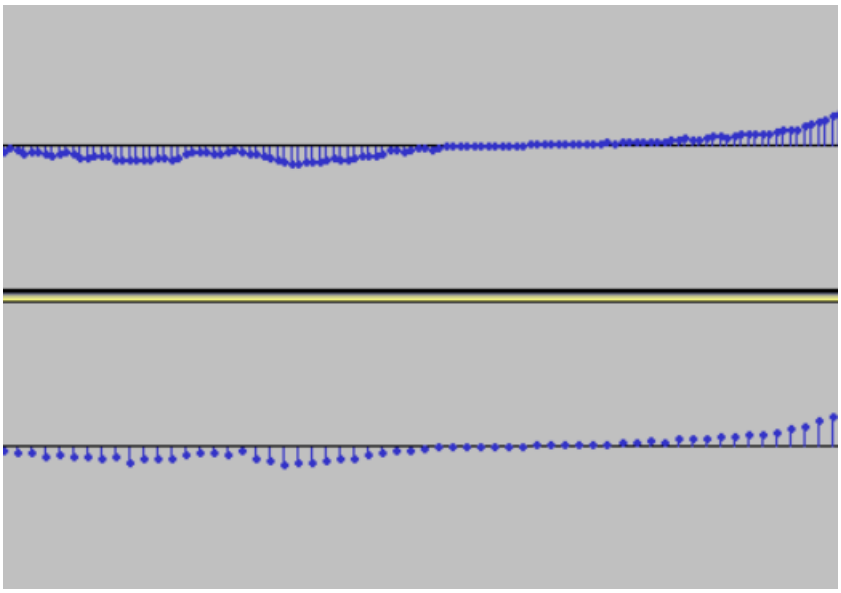

Figure 4: Waveform images in a 100 millisecond interval. The top waveform is 16KB sampling rate, while the bottom is 8KB.

5 Conclusion

This paper has covered topics on the history and development of ASR systems. The implementation of many of these systems are results of advancements made during certain time period that allowed for ASR to be applied to many different environments. It is amazing to see the breadth of ASR engines applications. Through differing methods, we can see that these systems excel in environments that are unique to each system. The CMU Sphinx-4 system makes use of the advancements in probabilistic theories to create a system with high portability, easy to use functionality, and high accuracy with low commitment. Julius is a much more compact system that allows for developers to take on high scale projects with great success. The results of this experiment show that in basic scenarios, Julius tends to a much more consistent runtime that scales to longer files well, while Sphinx-4 is best used in shorter speeches. It would be interesting to see how these systems perform when trained to different levels. An observation may be that, with better training, Sphinx-4 can maintain higher efficiency than Julius on larger files. Another topic to be considered for future work would be the possible loss in efficiency when these systems are used as applications. An application such as Simon is an example of this. Simon is an open source application that allows for simple implementation of Sphinx and Julius. While layering this technology under an application may give more interactivity to clients, an interesting case would be to see if any efficiency is lost by using mediums such as Simon.

References

[1] J. Baker. "The DRAGON system–An overview". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.1 (Feb. 1975), pp. 24–29. ISSN: 0096-3518. DOI: 10.1109/TASSP.1975.1162650.

[2] Leonard E. Baum and Ted Petrie. "Statistical Inference for Probabilistic Functions of Finite State Markov Chains". In: *Ann. Math. Statist.* 37.6 (Dec. 1966), pp. 1554–1563. DOI: 10.1214/aoms/1177699147. URL: https://doi.org/10.1214/aoms/1177699147.

[3] H. Bourlard and N. Morgan. "Continuous speech recognition by connectionist statistical methods". In: *IEEE Transactions on Neural Networks* 4.6 (Nov. 1993), pp. 893–909. ISSN: 1045-9227. DOI: 10.1109/72.286885.

[4] Herve A Bourlard and Nelson Morgan. *Connectionist speech recognition: a hybrid approach*. Vol. 247. Springer Science & Business Media, 2012.

[5] *CMUSphinx Open Source Speech Recognition Toolkit*. https://cmusphinx.github.io/.

[6] *English Gigaword language model*. http://www.keithv.com/software/giga/.

[7] Lee D. Erman et al. "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty". In: *ACM Comput. Surv.* 12 (1980), pp. 213–253.

[8] Alex Graves and Navdeep Jaitly. "Towards end-to-end speech recognition with recurrent neural networks". In: *International Conference on Machine Learning*. 2014, pp. 1764–1772.

[9] Paul Lamere et al. "The CMU SPHINX-4 speech recognition system". In: *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2003), Hong Kong*. Vol. 1. 2003, pp. 2–5.

[10] Akinobu Lee, Tatsuya Kawahara, and Kiyohiro Shikano. *Julius — An Open Source Real-Time Large Vocabulary Recognition Engine*. Sept. 2001. URL: http://library.naist.jp/dspace/handle/10061/7954.

[11] Kai-Fu Lee. *Automatic speech recognition: the development of the SPHINX system*. Vol. 62. Springer Science & Business Media, 1988.

[12] Bruce T Lowerre. *The HARPY speech recognition system*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1976.

[13] *Open-Source LBCSR Engine Julius*. http://julius.osdn.jp/en_index.php.

[14] L. R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (Feb. 1989), pp. 257–286. ISSN: 0018-9219. DOI: 10.1109/5.18626.

[15] Lawrence Rabiner and Biing-Hwang Juang. "Historical Perspective of the Field of ASR/NLU". In: *Springer Handbook of Speech Processing*. Ed. by Jacob Benesty, M. Mohan Sondhi, and Yiteng Arden Huang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 521–538. ISBN: 978-3-540-49127-9. DOI: 10.1007/978-3-540-49127-9_26. URL: https://doi.org/10.1007/978-3-540-49127-9_26.

[16] D. R. Reddy et al. "The Hearsay-I Speech Understanding System: An Example of the Recognition Process". In: *IEEE Transactions on Computers* C-25.4 (Apr. 1976), pp. 422–431. ISSN: 0018-9340. DOI: 10.1109/TC.1976.1674624.

[17] *Training an acoustic model*. https://cmusphinx.github.io/wiki/tutorialam/.

[18] *Voxforge Repository*. http://www.repository.voxforge1.org/downloads/Main/Trunk/AcousticModels/Julius/.

[19] *Voxforge Speech Corpus*. http://www.repository.voxforge1.org/downloads/SpeechCorpus/Trunk/Audio/Main/.

[20] *Word Error Rate Calculations*. https://martin-thoma.com/word-error-rate-calculation/.