Large-Vocabulary Continuous Speech Recognition

Kelvin Gu, Naftali Harris

December 9, 2013

Contents

1 Summary

In this project, we deployed a fully functioning Large-Vocabulary Continuous Speech Recognition (LVCSR) system, by extending CMU Sphinx-4. Our efforts are summarized by the following key tasks:

1. **Understanding the speech recognition pipeline from end to end**: in order to understand our system's errors and optimize its performance, we studied the key components needed for a modern speech recognition system. We document what we learned in **Section 2**.

2. **Analyzing and optimizing parameter settings**: Sphinx-4 has many interchangeable modules and a wealth of parameters. We tested its performance under various configurations and algorithmically attempted to find optimal settings. We interpret our results in **Section 3**.

3. **Software design decisions and obstacles**: due to the large amount of configuration necessary to run Sphinx-4, we wrapped it in a much simpler Python interface, which can be easily used by future researchers wishing to experiment with it. We also addressed audio formatting issues which can cause Sphinx-4 to silently fail and perform poorly. These and other design choices are summarized in **Section 4**.

2 Overview of a speech recognition system

In this section, we describe a full speech recognition system, using Sphinx-4 as our example. We provide a broad overview of all the components, then focus on describing the **decoder**, **phonetic dictionary** and **acoustic model** in more detail, because they are perhaps most different from what we have covered in class. Before we start, it is helpful to define important concepts in speech.

2.1 Speech concepts

- speech can be thought of as a sequence of **utterances**
    - utterances are separated by pauses in speech (periods of silence)
- an **utterance** is composed of **words** and **fillers**
- **fillers** are non-linguistic sounds, such as breathing, coughing, laughing
- a **word** is composed of **pronunciation states**
- a **pronunciation state** is composed of **phones**
- a **phone** is composed of **subphones**: typically a beginning, middle and end
    - **phones** can sound different depending on their neighboring phones (context) and speaker
- a **subphone** is characterized by certain waveform features of raw audio

2.2 Broad overview

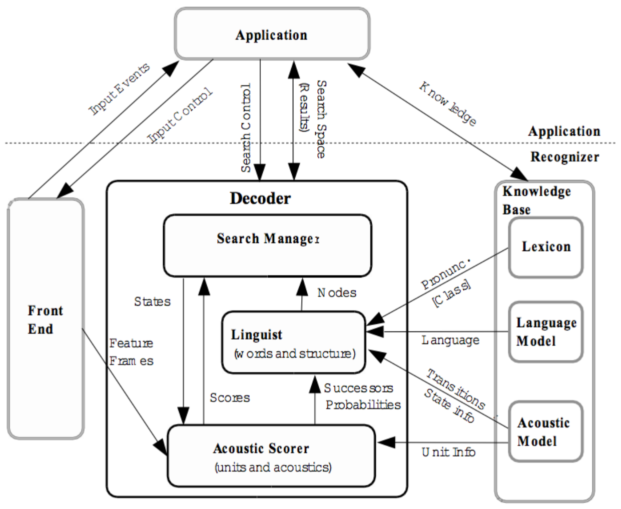


Figure 1: The Sphinx-4 system architecture, taken from [3]

Front end

- takes the raw audio stream and segments it into regions of silence and non-silence. Each non-silent segment is then processed independently and considered to be a single utterance.
- The audio segment for each utterance is discretized into 10 ms bins, known as **frames**.
- Each frame is boiled down into a feature vector summarizing waveform information in the bin.

Knowledge base

Contains key information about how the different levels of abstraction in speech relate to each other, and prior knowledge about language.

- **language model**
  - this is exactly the kind of language model we have been discussing in class. For example a trigram model, or a PCFG.
- **lexicon / phonetic dictionary**, maps:
  - word $\mapsto$ phones
- **acoustic model**, maps:
  - subphones $\mapsto$ feature vector (will be elaborated on in Section 2.4)
  - phone $\mapsto$ subphones

Decoder

This component does the heavy lifting. It attempts to find the sequence of words which best matches the observed audio stream. This is achieved using beam search. It extensively uses the models in the knowledge base to score possible sequences of words.

2.3 Decoder

We describe in detail the decoder's beam search / Viterbi algorithm. For ease of explanation, we will assume the language model is a bigram model.

The setup

- Decoding can be visualized as the construction of a speech lattice $(\mathcal{V}, \mathcal{E})$ and finding an optimal path through that lattice. (The edges we construct represent candidate paths that we consider while searching for the optimal path.)
- A speech lattice is a special kind of directed graph with words on the edges. An example is shown below. (WARNING: the picture is slightly deficient, in that not all nodes are shown. Also, the edges should be directed.)
  - Each **node** corresponds to a point in time. In our particular case, they are regularly spaced apart by 10 milliseconds (a single **frame**). In other words, time$(v_i) = 10 \cdot i$ milliseconds.
  - Each **edge** is represented by a triple $e = (v_i, v_j, w)$ where $w \in \mathcal{W}$ is the word associated with that edge ($\mathcal{W}$ is the set of all words). Also, edges must only move forward in time. In other words, $i < j$.
    - **Interpretation:** the edge $e = (v_1, v_5, \text{"hello"})$ would indicate that the word "hello" was uttered over the time interval $[10, 50]$.
- We start with a set of nodes $\mathcal{V} = \{v_0, v_1, v_2, \ldots, v_n\}$ and no edges, $\mathcal{E} = \emptyset$. We will construct directed edges as part of the decoding procedure.

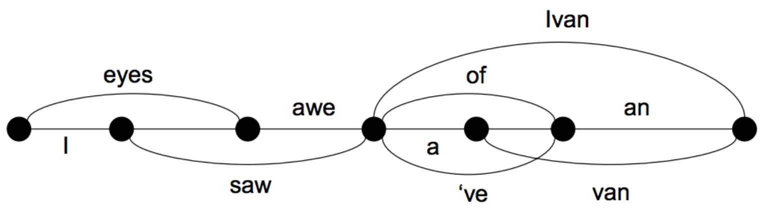


Figure 2: A speech lattice, taken from [1] (**note that not all nodes are shown!**)

Objective

- Our goal is to find an optimal path from v_1 to v_n.
  - The score of a path, $p = \{e_0, e_1, \ldots, e_m\}$ is defined as $\text{score}(p) = A(e_0) + \sum_{i=1}^{m} L(e_i, e_{i+1}) + A(e_i)$ (A and L can be thought of as log-probabilities).
  - $A(\cdot)$ is a score assigned by the **acoustic model** and **phonetic dictionary**. For a given $e = (v_i, v_j, w)$, $A(e)$ measures how well the audio properties in the interval $[\text{time}(v_i), \text{time}(v_j)]$ match up with the word w. We describe the computation of $A(\cdot)$ in the Phonetic dictionary and Acoustic model sections.
  - $L(\cdot,\cdot)$ is a score assigned by the bigram **language model**. $L(f, e) = \log P(\text{word}(e) \mid \text{word}(f))$

Procedure

- To find this optimal path, we will visit the nodes sequentially, $i = 0, 1, 2, \ldots$
- At each node v_j that we visit, we will construct a set of directed edges, $\mathcal{E}_j = \{(v_i, v_j, w) \mid i < j, w \in \mathcal{W}\}$
  - In words, $\mathcal{E}_j$ is the set of edges starting from a node previous to v_j and ending at v_j.
  - Each edge will get a score, $\text{score}(e) = \max_{i<j} \max_{f \in \mathcal{E}_i} \text{score}(f) + L(f, e) + A(e)$
  - **Note that score(e) is actually the score of the best path from v_0 to v_j which has e as its last edge.**
- When we visit the final node v_n we can look at $e^* = \arg\max_{e \in \mathcal{E}_n} \text{score}(e)$ (the highest scoring edge to enter v_n). Note that $\text{score}(e^*) =$ the score of the best path from v_0 to v_n.
- If we keep track of arg max instead of just max in the above steps, we can also recover the actual best path, rather than just the score.

Modification for efficiency

- In practice, each $\mathcal{E}_j$ is enormous. So, we prune many possibilities by replacing $\mathcal{E}_j$ with $\mathcal{E}_j^B = \{e \in \mathcal{E}_j \mid \text{score}(e) \geq B\}$
  - We are filtering $\mathcal{E}_j$, keeping only those edges that achieve a score higher than B
  - B can be chosen at each step j such that $\mathcal{E}_j^B$ is a fixed size. That size is the "beam size".

2.4 Phonetic dictionary

- In a simplified world, a word would just be a unique sequence of phones. In reality, a word can be spoken in several ways.
- We can represent the various ways a word can be spoken using another lattice where this time the edges are now labeled with subphones instead of words.

- Each edge has a score which reflects how well that particular subphone matches the audio in the segment spanned by that edge. This score is computed by the **acoustic model**.
- The overall score for the word is then just the score of the highest-scoring path through the "subphone-lattice" representing this word. The same algorithm as above is used for finding this path.

2.5 Acoustic model

As discussed earlier, each 10 ms frame of audio is boiled down into a feature vector ϕ. For a given subphone s, and observed ϕ, the acoustic model computes the score $\log L_s(\phi)$, where L is a likelihood function parameterized by s. The two most common models used for L are Gaussian mixture models and neural networks.

Gaussian mixture model

- under this model, each $\phi_i \sim N(\mu_{z_i}.\Sigma_{z_i})$, where $z_i \sim \text{Categorical}(p)$
- $p \in \mathbb{R}^K$ is a vector of mixture probabilities with $\sum_{i=1}^{K} p_i = 1$ and $p_i \geq 0$, where K is the total number of Gaussians

Neural network

- the input layer of the neural network takes a vector representing the spectrum within the 10 ms frame (alternatively, the input spectrum vector may have already undergone certain transformations/featurization).
- the final layer of the network is a single unit (a scalar) which produces a value between 0 and 1

3 Analyzing and optimizing parameter settings

3.1 Evaluation metrics

- To evaluate the quality of our Speech Recognition System, we manually transcribed a segment of audio from NPR. Our audio segment was 2 minutes 55 seconds in length, totaling 546 words. Manually transcribing the audio segment was actually an interesting endeavor; it gave us an appreciation for some of the challenges and ambiguities associated with real audio text. We made three observations: First of all, there are a number of "um"'s and "uh"'s in the raw text. Secondly, oftentimes a speaker will repeat a sequence of words, like, "she just she just was shocked and dismayed". We expect this to violate the language model somewhat. Thirdly, occasionally in our text there are some non-words, like when a speaker changes their mind about a word halfway through saying it, or when profanity is bleeped out, (as occured several times in our audio sample). To be true to the text, note that we included all of these features in our manual transcription, despite the fact that we expect this to lower the numerical quality of the speech transcription that we have.
- As is standard in Speech Recognition, we use the Word Error Rate (WER) as our measure of transcription quality. The WER is computed by determining the edit (Levenshtein) distance between the estimated transcription and our manual transcription, and then dividing by the length of the manual transcription. Of course, before computing the WER we strip punctuation and lower-case to make the token streams comparable.

3.2 Parameter optimization procedure

- There are a number of parameters one can use to configure CMU Sphinx. The most interesting ones govern the beam width and the language model weight, which, respectively, determine how exhaustively to search for transcriptions and much fidelity vs. fluency the transcription exhibits. However, there are in total seven different numerical parameters that can be optimized, not to mention that one can swap in one's favorite language or phonetic model. We initialized our configuration of Sphinx with the best regarded models we could find, and values of the parameters that seemed reasonable to us. This initial configuration achieved a WER of 0.549 in 108s, with a beam-width of 1000 and a language model weight of 10.5.

- Of course, we were not confident that these were the optimal values of these parameters. So we decided to optimize the performance of our system by varying the parameters. We did this with a blocky version of coordinate descent–since each run of the code can take up to two or three minutes, it was very important that we minimized the number of times we had to rerun the speech recognition routine with a given set of parameters. So for each parameter, we looked at 60%, 80%, 100%, 120%, and 140% of the parameter, and selected the value that achieved the best WER. We then cycled through each of the parameters 10 times, after starting at our initial configuration guess.

3.3 Results

- This optimization routine yielded values of the parameters that achieved a WER of 0.478 in 115 seconds. This is a 13% decrease in WER over our initial guess. The values of beam-width and language model weight that ended up being selected were 3387 and 6.3 respectively, showing that we had not been aggressive enough with our initial value for beam-width, and had put too much emphasis on fluency in our original configuration. This analysis also shows that configuration is a serious issue with the potential to make a lot of difference in performance.

- This WER is comparable to the values achieved in an earlier transcription analysis of some Yale Open Course lectures, by Stephen Marquard, (see references). In this analysis, Marquard achieved WERs ranging from 32% to 61%. This shows that our performance is relatively typical of CMU Sphinx. However, these WERs really are not very good, and the corresponding transcriptions are frankly not good enough to be used for human-consumed captions.

3.4 Error analysis

- Many of the errors made by the system were of words that were accoustically similar to the correct ones, but did not make much sense in the context. For example, the audio file opens with:

  > And it was just too kidlike for uh the opening, so we decided on something more menacing and ganglike, and this is almost like a rumble song.

  This is transcribed by our optimally configured system as:

  > then it was just to get like a fuhrer ah the opening sibley cited on something more menacing enduring like and this is almost like a rumble sullen

  Notice how "too kidlike" aligns with "to get like", "for uh" aligns with "fuhrer", and "so we decided" aligns with "sibley cited". Phonetically, these transcriptions are reasonable, but they are not true to the audio and are not fluent English.

  As another example, consider this section from the middle:

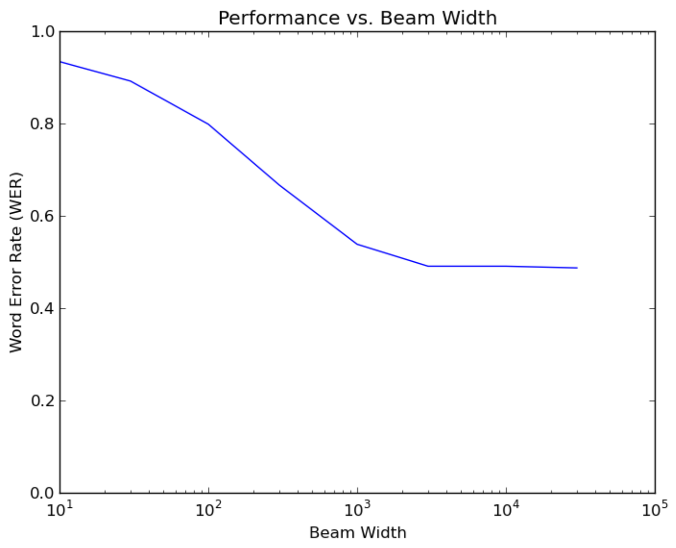
  > And then there's another song that you wrote lyrics for called this turf is ours.

  which is transcribed as:

  > n. n. is another suddenly wrote lyrics for call disturbs is ours

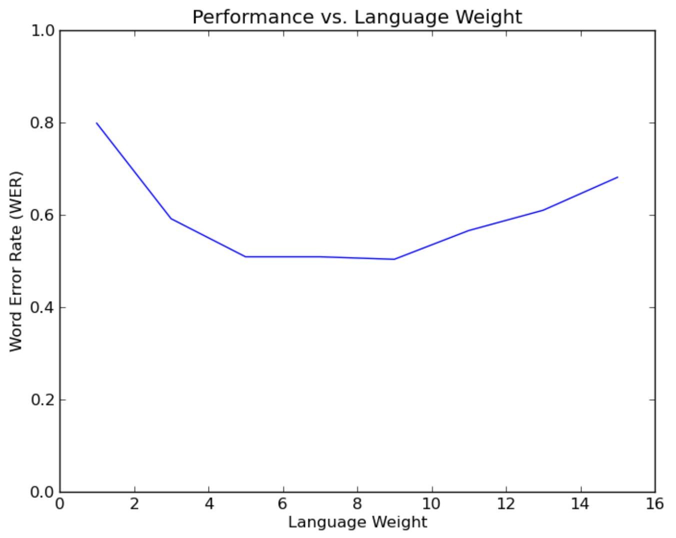
  Note how "And then there's" aligns with "n. n. is", and "called this turf is" aligns with "call disturbs".

- At first, we thought that these problems may have arisen from not searching exhaustively enough for transcriptions. So we decided to see how the performance varied with the beam-width at around our set of optimal parameters, (when all of the other parameters are held fixed at the values we determined optimal):



  Note how the WER begins high, when we are not searching exhaustively enough, but then converges to an asymptote as we begin to see largely dimished returns for more and more exhaustive searches. We see from this plot that the beam-width parameter we found to be optimal, ($3387 = 10^{3.53}$), is already at the asymptote, leading us to conclude that our system was already searching exhaustively enough.

- The second idea we had about why we were making these sorts of phonetically similar errors was that the language weight wasn't tuned high enough. So we decided to troubleshoot this by plotting the performance against the language model weight, all other paramters held constant:



  Note how when the weight is too high or too low, the system performs poorly, and only does well when it's in the proper sweet spot. However, the value we determined optimal, 6.3, is in this sweet spot, so tuning up the language weight more would not improve performance.

Interestingly, in some places Sphinx does not seem to be tripped up by the presense of non-English utterances like "um" and "uh". For example, it transcribes the following:

> sort of felt that maybe it was a little too gentle so uh i we wrote something

as

> sort of felt that maybe it was a little too gentle so uh uh oh we wrote something

From the experience of transcribing the audio file ourselves, we discovered that there really are some places in the audio where there is some ambiguity, so this section should be considered as a place where the ASR system actually did quite well, despite not exactly matching our transcription of the speaker's stuttering.

4 Software design decisions and issues

4.1 Choosing which existing system to extend

- We considered several other speech recognition systems before choosing CMU Sphinx-4, including:
  - Kaldi (http://kaldi.sourceforge.net)
  - Julius (http://julius.sourceforge.jp/en_index.php)
  - CMU Pocketsphinx (http://cmusphinx.sourceforge.net)
- As newcomers to speech recognition, we ultimately chose CMU Sphinx-4 because the documentation was more suitably targeted at non-experts.
- If we had more time, we would have chosen Kaldi. This would have allowed us to extend recent research into deep learning-based acoustic models [2].

4.2 Identifying key parameters and creating a simpler interface to vary them

- To configure Sphinx-4, users need a roughly 200 line XML file with values scattered throughout. As their programmer's guide notes, this is probably the least enjoyable part of setting up Sphinx-4.
- To make our lives easier, and to make it possible to programatically search over parameter space, we wrapped the system in a simple Python call (see `demo.py`), which future researchers can use.

4.3 Audio formatting issues

- The front-end module of Sphinx-4 is responsible for taking each 10 ms frame of audio and computing a feature vector that summarizes its spectral features. However, this module only works with a specific audio format: 16khz 16bit mono files in MS WAV format.
- As a word of warning to future researchers experimenting with audio processing in Java: Java does not offer built in audio encoding. It is unusually difficult to find bug-free third-party encoders capable of translating between diverse formats. The best options we were able to find were:
  - JAVE, a Java wrapper for ffmpeg (http://www.sauronsoftware.it/projects/jave/manual.php)
  - MP3 SPI (http://www.javazoom.net/mp3spi/mp3spi.html)

5 References

- [1] Statistical NLP, Spring 2007 Lecture Slides, Klein.
  - http://www.cs.berkeley.edu/~klein/cs294-7/SP07%20cs294%20lecture%2017%20–%20Lexicalized%20Parsing%20
- [2] Rectier Nonlinearities Improve Neural Network Acoustic Models, Maas et al.
  - http://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf
- [3] The CMU Sphinx-4 Speech Recognition System, Lamere et al.
  - http://www.cs.cmu.edu/~rsingh/homepage/papers/icassp03-sphinx4_2.pdf
- [4] Speech and Language Processing, Jurafsky and Martin.
- [5] Sphinx4 speech recognition results for selected lectures from Open Yale Courses
  - http://trulymadlywordly.blogspot.com/2011/12/sphinx4-speech-recognition-results-for.html