Chapter 9

Speech

Speech recognition has been an intense area of research for decades, and is now a commonplace feature in PCs and mobile devices. Recently, the field has undergone rapid change with the introduction of neural networks, but before that, the standard approach to speech recognition could be formulated in terms of finite transducers. We give only a brief overview here. For a more detailed treatment, see the survey by Mohri, Pereira, and Riley (2002).

9.1 Preprocessing

Given a sample of speech, we can convert it into a sequence of real-valued 39-dimensional vectors, called the *mel-frequency cepstral coefficients*:

1. Slice the signal into overlapping frames, typically 25 ms wide and starting every 10 ms.
2. Perform a Fourier transform on each frame, which is the amount of power at each frequency.
3. Compute how much power there is in each of several frequency bands arranged according to the *mel scale*, which is supposed to match human perception of pitch.
4. Take the log of each power, which matches human perception of loudness.
5. Take a *discrete cosine transform*, which removes correlations between the components.
6. Also compute the change of the components over time ($\Delta c_t = c_{t+2} - c_{t-2}$), and the change of the change ($\Delta^2 c_t = \Delta c_{t+1} - \Delta c_{t-1}$), for a total of 39 components.

9.2 Model

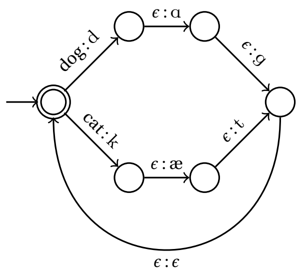
We want to find the most likely sequence of words that this speech came from. To do this, we build a cascade of transducers that maps from text to feature vectors:

1. Generate words using a language model, typically an n-gram model.
2. Map words to their pronunciations, which are strings of *phones*.

3. Divide each phone into *subphones*, and stretch out each subphone to last for one or more frames.

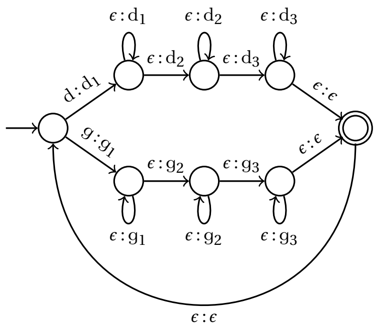
4. For each frame, map the subphone to a feature vector.

We've seen automata for n-gram models already. To maps from words to their pronunciations (shown just for two words), we can use a transducer like this:

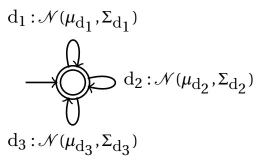


Hand-built dictionaries are available (e.g., the CMU pronunciation dictionary), or a grapheme-to-phoneme model could be used or even learned automatically.

The next stage divides each phone into *subphones* and decides the duration of each subphone. Typically, each phone is divided into three subphones; for example, phone d would be divided into d_1, d_2, and d_3. The duration of each subphone is one or more time slices. So the transducer looks like this (shown just for two phones):



Finally, we map from the sequence of subphone symbols to the observed sequence of feature vectors. The problem is that the vectors have real-valued components, but we only know how to define transducers with a finite output alphabet. So we have to generalize our notation. We need something like this (shown just for one phone, d):



This means that the transducer can read symbol d_1, and outputs a 39-dimensional vector drawn from the multivariate normal distribution, $\mathcal{N}(\mu_{\mathrm{d}_1}, \Sigma_{\mathrm{d}_1})$. The μ's and Σ's are parameters to be learned.

Because the sound of a subphone might not be modeled well by a multivariate normal distribution, we can include multiple transitions for each subphone, each with a multivariate normal distribution with different parameters.

9.3 Training

The first two transducers (language model and lexicon) can be created separately: the first is trained on text data, and the second is generally created by hand.

The third and fourth transducers (phone to subphone and subphone to feature vectors) are trained together. Compose them into a single transducer that maps phones to feature vectors. Then, given data that consists of speech together with phonetic transcriptions, there are many paths through the transducer, corresponding to different ways of aligning the phones and subphones to the speech signal (and different choices of normal distributions, if the model allows them).

As always, we train using maximum-likelihood estimation. The probability of a training example requires us to sum over all the paths through the transducer, which we can do efficiently using the Forward algorithm, just as in unsupervised tagging and in word alignment. Then, we can maximize the log-likelihood using gradient ascent.

Bibliography

Mohri, Mehryar, Fernando Pereira, and Michael Riley (2002). “Weighted finite-state transducers in speech recognition”. In: *Computer Speech and Language* 16, pp. 69–88.